**UNIVERSIDADE FEDERAL DE SANTA CATARINA**

**GABRIEL DA LUZ SIMONETTI SOUZA**

# APLICAÇÃO DOS CONCEITOS DE ENGENHARIA DE SOFTWARE NO DESENVOLVIMENTO DE UM PRODUTO PARA GESTÃO DE CLUBES DE TÊNIS

**FLORIANÓPOLIS**

**2022**

**GABRIEL DA LUZ SIMONETTI SOUZA**

# APLICAÇÃO DOS CONCEITOS DE ENGENHARIA DE SOFTWARE NO DESENVOLVIMENTO DE UM PRODUTO PARA GESTÃO DE CLUBES DE TÊNIS

Monografia submetida ao Programa de Graduação em Ciências da Computação, da Universidade Federal de Santa Catarina, como requisito para obtenção do Grau de Bacharel.

Orientador: Raul Sidnei Wazlawick

UNIVERSIDADE FEDERAL DE SANTA CATARINA

FLORIANÓPOLIS

2022

**GABRIEL DA LUZ SIMONETTI SOUZA**

**APLICAÇÃO DOS CONCEITOS DE ENGENHARIA DE SOFTWARE NO DESENVOLVIMENTO DE UM PRODUTO PARA GESTÃO DE CLUBES DE TÊNIS**

Esta Monografia foi julgada adequada para obtenção do Título de “Bacharel” e aprovada em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 26 de dezembro de 2022.

_____

Prof. Dr. Jean Everson Martina

Coordenador do Curso

**Banca Examinadora:**

_____

Prof. Dr. Raul Sidnei Wazlawick

Orientador

_____

Prof. Me. José Eduardo De Lucca

Universidade Federal de Santa Catarina

_____

Prof. Me. Antonio Carlos Mariani

Universidade Federal de Santa Catarina

# AGRADECIMENTOS

A Deus, pela força, determinação e energia que tive para superar os desafios encontrados durante todos esses anos e para a realização deste trabalho.

Aos meus familiares, por terem dado o melhor suporte que puderam para que eu estivesse aqui hoje finalizando este trabalho.

À minha esposa, por estar ao meu lado dando suporte e carinho, além de ter colaborado diretamente neste trabalho com a revisão de texto dele e desenhos que me ajudaram a chegar ao final.

À professora, Dra. Ingrid Nunes, por ter me despertado um fortíssimo interesse pela engenharia e arquitetura de software quando estudei na UFRGS.

Ao meu orientador, Prof. Dr. Raul Sidnei Wazlawick, pelo acompanhamento e conselhos valiosos que tornaram este trabalho possível.

Meus sinceros agradecimentos.

# RESUMO

O tênis não profissional é um esporte atendido de forma limitada no meio tecnológico nos quesitos de gestão, integração de jogadores e acompanhamento técnico. Com o intuito de exercitar diversas disciplinas do desenvolvimento de software, este trabalho apresenta o estudo, projeto e desenvolvimento de um protótipo para um produto mínimo viável (MVP) de um sistema multiplataforma para gestão de clubes de tênis e integração de seus afiliados. É apresentada a fundamentação teórica dos conceitos ligados à proposta, como também uma revisão tecnológica de trabalhos relacionados disponíveis no mercado, analisando critérios de propósito, *onboarding*, interface, métricas de negócio, personalização e completude de funcionalidades.

O produto proposto tem como público-alvo jogadores, gestores e treinadores de tênis, cujas necessidades foram estudadas e apresentadas dentro de uma matriz utilizando *RICE Score* para definição do escopo. O desenvolvimento do sistema contempla um aplicativo para Android, utilizando a tecnologia .NET MAUI e arquitetura *offline-first*; uma aplicação de *backend*, utilizando a plataforma .NET 6; infraestrutura, utilizando a AWS como provedor de nuvem, e algumas práticas de DevOps para otimizar o processo de desenvolvimento e implantação do software.

**Palavras-chave:** aplicativo para gestão de clubes de tênis, aplicativo para tênis, software multiplataforma, desenvolvimento de software, arquitetura de software, engenharia de software.

# ABSTRACT

Non-professional tennis is a sport that has limited access to technology in terms of management, player integration and technical support. In order to exercise several software development disciplines, this work presents the study, design and development of a prototype for a minimum viable product (MVP) of a multiplatform system for managing tennis clubs and integrating their affiliates. The theoretical basis of the concepts related to the proposal is presented, as well as a technological review of related works available in the market, analyzing criteria of purpose, onboarding, interface, business metrics, customization and completeness of functionalities.

The proposed product is aimed at players, managers and tennis coaches, whose needs were studied and presented within a matrix using RICE Score to define the scope. The development of the system includes an application for Android, using .NET MAUI technology and offline-first architecture, a backend application, using the .NET 6 platform, the infrastructure, using AWS as a cloud provider, and some DevOps practices to optimize the software development and deployment process.



**Keywords**: tennis club management application, tennis application, cross-platform software, software development, software architecture, software engineering.

## LISTA DE ILUSTRAÇÕES

### FIGURAS

Figura 1. Arquitetura genérica para serverless, baseada em (AWS Lambda, 2022).18
Figura 2. Captura da tela de escolha de funcionalidades no Springly.……25
Figura 3. Captura da tela de estatísticas da comunidade no Springly.……26
Figura 4. Captura de tela com problemas de layout do software ACES.....28
Figura 5. Captura de tela com listagem no software ACES.....29
Figura 6. Captura da tela de métricas do jogador no ACES…….....29
Figura 7. Captura da tela do painel do software Club Manager 365.....32
Figura 8. Captura da tela de agendamentos do Club Manager 365.....32
Figura 9. Svelte e .NET MAUI……42
Figura 10. Diagrama ER para o EasyTennis……45
Figura 11. Diagrama de classes para modelos de dados do MVP……46
Figura 12. Diagrama de classes com BaseEntity e BaseAuditableEntity……47
Figura 13. Camadas da arquitetura…….....50
Figura 14. Versão final da arquitetura do sistema……52
Figura 15. Fluxograma da arquitetura offline em edição de registro……53
Figura 16. Biblioteca de regras de negócio.....54
Figura 17. Camadas da arquitetura das aplicações……55
Figura 18. Estrutura de pastas do backend……..56
Figura 19. Estrutura de pastas da camada Application do backend……58
Figura 20. Estrutura de pastas do aplicativo para smartphone……61
Figura 21. Captura de tela do aplicativo EasyTennis: tela de login……...62
Figura 22. Captura de tela do aplicativo EasyTennis: tela de início……..63
Figura 23. Captura de tela do aplicativo EasyTennis: tela de minhas reservas…....64
Figura 24. Captura de tela do aplicativo EasyTennis: tela de configurações...……...65
Figura 25. Diagrama de infraestrutura……...66
Figura 26. Diagrama de sequência para geração de URL pré-assinada…….67
Figura 27. Diagrama do processo de implantação de código em homologação……69

### QUADROS

Quadro 1. Motivos de desistência da análise de alguns produtos….....23

Quadro 2. Módulos de funcionalidades do Springly……24
Quadro 3. Definições adotadas para realização do cálculo de priorização…….38
Quadro 4. Definições adotadas para o critério impact do modelo RICE score……...38
Quadro 5. Definições adotadas para o critério confidence do modelo RICE score…39
Quadro 6. Tecnologias escolhidas.....43
Quadro 7. Serviços de nuvem escolhidos.....44
Quadro 8. Ferramentas de monitoramento escolhidas.....44
Quadro 9. Definições dos papéis em um clube……48

**TABELAS**

Tabela 1. RICE Score aplicado nas funcionalidades……..37

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| MVP | Minimum viable product |
| VPC | Virtual private cloud |
| ORM | Object–relational mapping |
| EF | Entity framework |
| ALB | Application load balancer |
| JWT | JSON web token |
| MFA | Multi-factor authentication |
| SDK | Software development kit |
| UUID | Universally unique identifier |
| DBMS | Database management system |
| EC2 | Amazon Elastic Compute Cloud |
| IDE | Integrated development environment |
| APK | Android package |
| MOM | Message-oriented middleware |

## SUMÁRIO


**1. INTRODUÇÃO 14**
1.1 CONTEXTUALIZAÇÃO DO TEMA 14
1.2 MOTIVAÇÃO 15
1.3 OBJETIVOS 16
1.3.1 Objetivo Geral 16
1.3.2 Objetivos Específicos 16
1.4 ESTRUTURA DO TRABALHO 16

**2. FUNDAMENTAÇÃO TEÓRICA 17**
2.1 ARQUITETURA DE SOFTWARE 17
2.1.1 Composição de API 17
2.1.1.1 Backend For Frontend 18
2.1.2 Serverless Cloud Computing 18
2.2 MIDDLEWARE ORIENTADO A MENSAGENS 19
2.3 SOFTWARE WHITE LABEL 19
2.4 CUSTOMER RELATIONSHIP MANAGEMENT 20
2.5 .NET 6 20
2.5.1 .NET MAUI 20
2.7 SVELTE 20
2.8 MYSQL 21
2.9 OFFLINE-FIRST 21
2.10 RICE SCORE 21

**3. TRABALHOS RELACIONADOS 22**
3.1 SPRINGLY 23
3.1.1 Onboarding 24
3.1.2 Interface 25
3.1.3 Métricas de Negócio 25
3.1.4 Personalização 26
3.1.5 Completude de Funcionalidades 27
3.2 ACES TENNIS MANAGEMENT 27
3.2.1 Onboarding 27
3.2.2 Interface 27
3.2.3 Métricas de Negócio 29
3.2.4 Personalização 30
3.2.5 Completude de Funcionalidades 30
3.3 CLUB MANAGER 365 31
3.3.1 Onboarding 31
3.3.2 Interface 31
3.3.3 Métricas de Negócio 33
3.3.4 Personalização 33
3.3.5 Completude de Funcionalidades 33

**4. DESENVOLVIMENTO 33**

4.1 PÚBLICO-ALVO 34
4.1.1 Clube de Tênis 34
4.1.2 Persona 1: Hugo Andrade - Jogador de Tênis 34
4.1.3 Persona 2: Alexandre de Moraes - Supervisor Técnico 35
4.1.4 Persona 3: Augusto Ribeiro Souza - Gestor de Clube 35
4.2 ESCOPO DO MVP 36
4.2.1 Reserva de Espaços (mobile) 39
4.2.2 Agendamento de Aulas (mobile) 39
4.2.3 Perfil de Usuário (mobile) 40
4.2.4 Gerenciamento de Usuários (somente backend) 40
4.2.5 Gerenciamento de Clubes (somente backend) 40
4.2.6 Gerenciamento de Espaços (somente backend) 40
4.3 DIFERENCIAIS 40
4.3.1 Multiplataforma 41
4.3.2 Otimização de Agendas 41
4.3.3 Dados Financeiros para Treinadores 41
4.3.4 Funcionamento em Locais sem Internet 42
4.4 TECNOLOGIAS, SERVIÇOS E FERRAMENTAS 42
4.4.1 Tecnologias Aplicadas 42
4.4.2 Infraestrutura 44
4.4.3 Monitoramento das Aplicações 44
4.4.4 Procedimentos Demorados 44
4.5 MODELO DE DADOS 45
4.5.1 Entidades 47
4.5.1.1 User 47
4.5.1.2 UserSettings 47
4.5.1.3 DominantSide 47
4.5.1.4 Club 48
4.5.1.5 ClubUser 48
4.5.1.6 ClubRole 48
4.5.1.7 Space 48
4.5.1.8 SpaceUse 49
4.5.1.9 SpaceUser 49
4.6 ARQUITETURA E IMPLEMENTAÇÃO 49
4.6.1 Composição de API 50
4.6.2 Offline-first 52
4.6.3 Aplicação da Arquitetura Limpa 54
4.6.4 Monitoramento e Métricas 55
4.6.5 Backend 55
4.6.5.1 Camadas da Arquitetura 56
4.6.5.2 Autenticação e Autorização 58
4.6.6 Aplicativo para Android 61

4.6.6.1 Login 62
4.6.6.2 Início 63
4.6.6.3 Minhas Reservas 64
4.6.6.4 Configurações 64
4.7 INFRAESTRUTURA 65
4.8 PROCESSO DE DESENVOLVIMENTO 68
4.9 AVALIAÇÃO DO PRODUTO 69
4.9.1 Onboarding 69
4.9.2 Interface 70
4.9.3 Métricas de Negócio 70
4.9.4 Personalização 70
4.9.5 Completude de Funcionalidades 70
**5. CONSIDERAÇÕES FINAIS 70**
**REFERÊNCIAS 72**

# 1. INTRODUÇÃO

Este capítulo apresenta o contexto do tema, a motivação para elaboração deste trabalho, os objetivos a serem alcançados e a estrutura do presente trabalho.

## 1.1 CONTEXTUALIZAÇÃO DO TEMA

O desenvolvimento de um produto de software é composto por diversas etapas e envolve diversas áreas. Dominar o processo e compreender as particularidades de cada uma dessas áreas não é uma tarefa trivial. De forma bem resumida, dentro do contexto de produto de software, encontra-se um problema, identifica-se uma oportunidade e então se inicia a definição de uma solução. Essa solução não se trata somente da implementação de um software, mas de um modelo de negócio que se prove viável.

Para que a construção de um produto de software seja uma solução viável, é necessário definir adequadamente o escopo da solução, o processo de desenvolvimento, implantação, escalabilidade, segurança, organização da equipe, entre outros. A boa tomada de decisão é fundamental para o sucesso de um projeto em cada uma dessas áreas. Ela não se trata de escolher fazer algo de forma perfeita, mas de considerar os diversos recursos disponíveis ou não disponíveis para atingir o resultado desejado da melhor forma. Por vezes, são tomadas decisões de implementação de artefatos não adequados para longo prazo, mas que solucionam um problema em curto prazo. Dentro desse contexto, surge o que chama-se de “dívida técnica” (KRUCHTEN, 2012), que pode, ou não, ser uma escolha consciente da equipe de desenvolvimento.

O exercício de fazer escolhas e lidar com as consequências acrescenta a experiência necessária para lidar melhor com as próximas escolhas. Fazer escolhas faz parte do cotidiano de uma equipe de desenvolvimento de software. Para exercitar o processo de desenvolvimento de um produto de software e adquirir mais conhecimento na área, optou-se por encontrar uma oportunidade de construção de solução real para o mercado e aplicar no desenvolvimento deste trabalho de conclusão de curso. Nesse contexto, encontrou-se a oportunidade de implementação de um sistema para gestão de clubes de tênis por conta da vivência do autor dentro desse esporte durante alguns anos.

Considerando as opções que um negócio tem de adquirir um software, optou-se pela construção de um modelo que atenda tanto o grande negócio quanto o pequeno negócio. A idealização da solução foi no sentido de um software que pode ser acessado por usuários de um clube de tênis assinante do produto. A exploração do modelo de negócio para o produto de software proposto neste trabalho está fora do escopo deste trabalho.

Os desafios de gestão de um clube de tênis foram estudados através da experimentação e análise de alguns produtos disponíveis no mercado. Essa análise é apresentada neste trabalho junto a uma proposta de solução inicial, completa e com diferenciais em relação aos concorrentes. Entretanto, por limitações de tempo e equipe, foi planejada e executada uma versão menor, simplificando a arquitetura, escopo de funcionalidades e infraestrutura do produto.

Neste trabalho, pretendeu-se aplicar as técnicas disponíveis na literatura e mercado para tomada de decisão e desenvolvimento do protótipo de um MVP de um produto de software multiplataforma para gestão de clubes de tênis.

## 1.2 MOTIVAÇÃO

O tênis é um esporte com cerca de 2,2 milhões de praticantes no Brasil e cerca de 4500 clubes (TENIS BRASIL, 2019). Esses clubes fornecem o espaço para a prática do esporte e promovem a conexão de pessoas com esse interesse em comum. Após a realização de pesquisas nas lojas de aplicativos e mecanismos de busca, percebeu-se que o esporte tênis é atendido na área de gestão e integração dos jogadores da comunidade. Entretanto, os produtos disponíveis deixam a desejar em alguns critérios, como usabilidade, layout moderno e completude de funcionalidades, conforme é apresentado na seção de trabalhos relacionados. A partir disso, abre-se a possibilidade de atender o público desse esporte fornecendo um produto de software para gestão dos clubes de tênis e integração de seus afiliados.

O desenvolvimento de um produto de software vendido por assinatura também precisa levar em consideração a sua evolução. Essa evolução tem um custo e precisa ser adequadamente definida. Dessa forma, é importante que sejam incluídos recursos para compreensão de uso dos usuários e da saúde do sistema, como é o caso das ferramentas de monitoramento de software. A introdução desses

recursos facilita a tomada de decisões, pois possibilita que essas sejam baseadas em dados.

Dentro do tema proposto para este trabalho, encontra-se a oportunidade de exercitar as disciplinas de projeto e desenvolvimento de um software para lidar com usuários reais. As decisões de projeto e implementação, como funcionalidades e tecnologias adotadas, estão fundamentadas nos conceitos apresentados no texto, visando entregar um software real que resolva problemas reais, e não pautado em preferências pessoais. O texto deste trabalho contempla importantes conceitos como engenharia de software, arquitetura de software, boas práticas de programação e métricas. Esses conceitos auxiliam no bom processo de desenvolvimento do software e sua evolução, colaborando para que o produto seja mais confiável e de fácil manutenção.

## 1.3 OBJETIVOS

### 1.3.1 Objetivo Geral

Aplicar os conceitos de engenharia de software com a criação da infraestrutura de desenvolvimento e implementação do protótipo de um produto de software para gestão de clubes de tênis.

### 1.3.2 Objetivos Específicos

- Ter uma análise de concorrência, definição do público-alvo e escopo da solução;
- Ter a definição da arquitetura, escolha de tecnologias, projeto e implantação da infraestrutura, e práticas de DevOps;
- Ter uma arquitetura extensível e escalável, visando a possibilidade de expandir para além do escopo de clubes de tênis;
- Implementar um protótipo do sistema para gestão de clubes de tênis.

## 1.4 ESTRUTURA DO TRABALHO

O restante deste documento está organizado em mais 4 seções. O capítulo 2 contém a fundamentação teórica, apresentando os principais conceitos relacionados a este trabalho. O capítulo 3 apresenta uma revisão tecnológica de trabalhos relacionados disponíveis no mercado de gestão de clubes de tênis. O capítulo 4

contempla o desenvolvimento do trabalho, abordando o público-alvo, necessidades do usuário, decisões de tecnologia e a implementação do protótipo. Por fim, o capítulo 5 apresenta algumas considerações finais sobre o estudo produzido e sugestões de trabalhos futuros.

# 2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda alguns conceitos que entregam o embasamento teórico necessário para o melhor entendimento deste trabalho. São descritos conceitos do desenvolvimento de software, de produto e algumas tecnologias mencionadas no trabalho. Não foi dada uma explicação aprofundada para cada um dos conceitos, pois isso foge ao escopo deste trabalho.

## 2.1 ARQUITETURA DE SOFTWARE

Em seu conceito mais puro, a arquitetura se trata da seleção de elementos arquitetônicos, suas interações e as restrições sobre esses elementos, bem como suas interações necessárias para fornecer uma estrutura que satisfaça os requisitos e sirva de base para o projeto (PERRY, 1992). A arquitetura de um sistema de software define esse sistema em termos de componentes e conexões entre esses componentes (HASSELBRING, 2018).

### 2.1.1 Composição de API

Composição de API é uma estratégia para evitar a necessidade de lidar com um alto tempo de latência devido a várias chamadas para o servidor. Essa estratégia consiste em migrar a composição de dados para o backend. Cria-se uma aplicação intermediária, dentro da estrutura de nuvem onde os outros serviços estão, para que essa camada faça a consulta aos serviços, componha os dados e responda para a aplicação. Migrar essa composição de dados para o backend é o que define o padrão de composição de API, ou API Gateway (OLIVEIRA, 2020). Dessa forma, a aplicação além de não precisar lidar com a latência de múltiplas requisições, porque agora é feita somente uma, ela também não precisa conhecer cada um dos serviços necessários para compor os dados, passando essa responsabilidade para o backend.

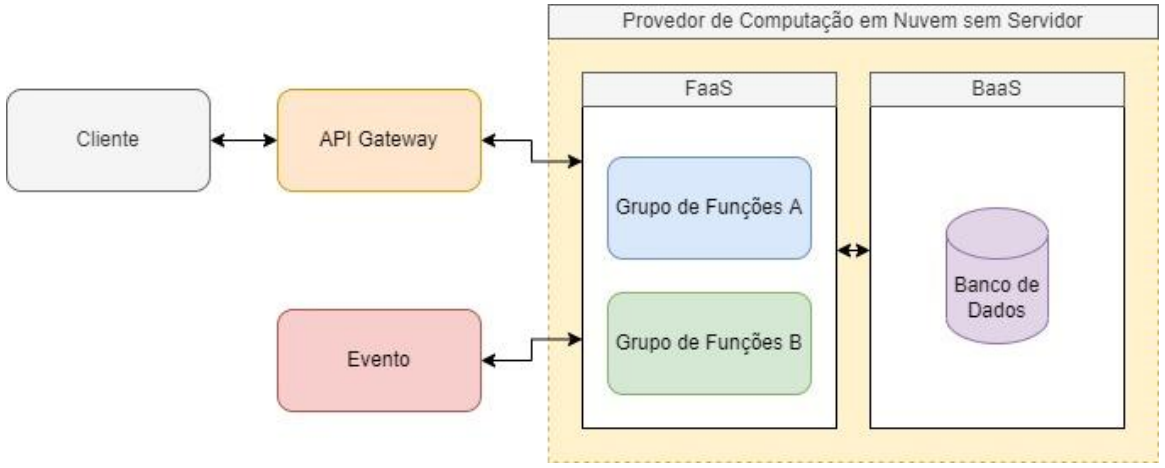
#### 2.1.1.1 Backend For Frontend

À medida que um produto evolui, podem surgir mais aplicações de frontend. Dentre essas novas aplicações, algumas podem consumir serviços de formas bem diferentes de outras. Essa situação pode levar ao questionamento de criar um outro API Gateway, específico para essas aplicações. Normalmente, isso leva a uma variação do padrão de Composição de API, chamada *Backend For Frontend* (*BFF*). O nome foi apresentado pela primeira vez por Phil Calçado, enquanto trabalhava na SoundCloud. A ideia é ter uma única interface de backend (API Gateway) para cada aplicação frontend. (OLIVEIRA, 2020).

### 2.1.2 Serverless Cloud Computing

O conceito de serverless cloud computing, também conhecido somente por serverless, oferece uma combinação entre *backend as a service* (BaaS) e *function as a service* (FaaS), como mostrado na Figura 1. O FaaS possibilita aos profissionais de TI a implantação e execução de código em plataformas de computação. Ele depende de serviços providos pelo BaaS, como banco de dados, mensageria (MOM), autenticação de usuários etc (HASSAN, 2021).

Figura 1. Arquitetura genérica para serverless, baseada em (AWS Lambda, 2022).




Fonte: elaborado pelo autor (2022)

Seu objetivo é permitir que desenvolvedores possam focar na lógica da aplicação ao invés de configurações e gerenciamento de servidor, que pode consumir bastante tempo. O provedor de nuvem fica com a responsabilidade de gerenciar, escalar e prover outros recursos para garantir a perfeita execução dessas funções (DOCS AWS, 2022).

Atualmente, o FaaS é considerado o modelo dominante de serverless, também sendo conhecido como “funções orientadas a eventos” (*event-driven functions*). Nele, o desenvolvedor realiza a implantação de suas aplicações como funções (HASSAN, 2021). Utilizando o FaaS, desenvolvedores podem expressar computações arbitrárias como funções simples que são automaticamente invocadas e gerenciadas por uma plataforma em nuvem como resposta a algum evento (WOLSKI, 2019). Por questões de comodidade e uso mais comum, será utilizado o termo “serverless” referenciando principalmente o FaaS.

## 2.2 MIDDLEWARE ORIENTADO A MENSAGENS

O middleware orientado a mensagens (Message Oriented Middleware - MOM) é um middleware usado para comunicação entre aplicações. Ele permite que aplicações diferentes se comuniquem sem se preocupar com as diferenças de tecnologia, como linguagens de programação ou sistemas operacionais. Esse tipo de middleware pode fornecer comunicação assíncrona e fracamente acoplada através do enfileiramento de mensagens em um intermediário (chamado de broker). Esse tipo de comunicação é mais flexível do que a comunicação ponto a ponto síncrona tradicional (NILSSON, 2020). Atualmente, existem implantações de MOMs em nuvem usando o modelo SaaS. Este tipo de serviço chamamos de serviço de enfileiramento de mensagens (message queuing service). Usuários desses serviços acessam filas para trocar dados usando padrões ponto a ponto ou *publish and subscribe*.

## 2.3 SOFTWARE WHITE LABEL

O software white label é um sistema onde a marca e os recursos do cliente são personalizados por meio de arquivos de configuração. A abordagem de white label é uma técnica de adaptação da identidade visual de um produto para vender para outra corporação (SILVA, 2020). Empresas que não desenvolvem um determinado produto, mas precisam ter ele com a sua marca, optam por esse caminho para atingir um maior nível de credibilidade com seus clientes.

## 2.4 CUSTOMER RELATIONSHIP MANAGEMENT

O Customer Relationship Management (CRM) é um sistema que permite registrar e organizar todos os marcos de contato que um consumidor tem com o vendedor de uma empresa. Assim, quando o profissional de vendas entra em contato com um possível cliente, pode armazenar dados como nome, endereço, telefone, visitas ao site e descrição de cada uma das interações no mesmo lugar (SIQUEIRA, 2021). Em português, chamamos de gestão de relacionamento com o cliente, mas neste trabalho, será utilizada a sigla “CRM”.

## 2.5 .NET 6

.NET é uma plataforma de código aberto, multiplataforma e gratuita para desenvolvimento de aplicações, utilizando as linguagens de programação C#, F# ou Visual Basic. Independente da linguagem escolhida, o código será executado nativamente em qualquer sistema operacional compatível. O .NET providencia um conjunto de bibliotecas de base que são comuns a todas as aplicações .NET. Entretanto, cada modelo de aplicação pode expor APIs adicionais que são específicas ao sistema operacional alvo (MICROSOFT, 2022).

### 2.5.1 .NET MAUI

O .NET Multi-platform App UI (.NET MAUI) é um framework cross-platform para criação de aplicações mobile e desktop nativas, utilizando as linguagens C# e XAML. Esse framework unifica as APIs dos sistemas operacionais Android, iOS, macOS e Windows em uma única API que permite a escrita de um único código que executa em qualquer um desses sistemas. Como é construído em .NET 6, também aproveita o recurso de possuir um acesso mais aprofundado às APIs específicas de cada sistema operacional desejado. Para Android, iOS e macOS, o ambiente é implementado pelo Mono, uma implementação do .NET runtime. No Windows, o .NET CoreCLR provê o ambiente de execução (BRITCH, 2022).

## 2.7 SVELTE

O Svelte é um framework open source para implementação de aplicações web. Diferentemente de opções mais tradicionais no mercado, como React e Vue, o Svelte realiza boa parte do trabalho que é destinado ao navegador já no processo de

compilação da aplicação. Ao invés de utilizar técnicas como a diferenciação do DOM (Document Object Model) virtual, ele escreve código que altera exatamente o que é necessário no DOM real quando o estado da aplicação é alterado (SVELTE, 2022). Isso é feito para alcançar a reatividade de componentes esperada nos produtos web mais modernos.

Em 2021, o Svelte recebeu uma posição de destaque na 2021 Developer Survey realizada pelo Stack Overflow, onde recebeu a maior porcentagem de avaliações positivas como o framework web mais amado (STACK OVERFLOW, 2021).

## 2.8 MYSQL

O MySQL é um sistema de gerenciamento de banco de dados, ou *database management system* (DBMS), open source para bancos de dados relacionais. Ele entrega um servidor rápido, multithread e multiusuário para sistemas críticos, de alta carga ou para software embarcado em massa (MYSQL, 2022). Atualmente, é o DBMS mais utilizado no mundo (STACK OVERFLOW, 2021) e possui uma extensa documentação.

## 2.9 OFFLINE-FIRST

Capacidade de funcionamento sem conexão com a internet é uma característica chave para aplicações web progressivas modernas (OFFLINE FIRST ORG, 2022). O termo *offline-first* refere-se ao foco de projetar e construir um sistema para funcionar sem depender da conexão com a internet já desde o início. Esta não é a prática mais comum, porém a conexão de rede móvel não é sempre confiável porque o sinal varia de um local para o outro. Portanto, aplicações mobile deveriam estar preparadas para lidar com problemas relacionados a falha de conexão e maiores latências do que o esperado. É neste ponto que entra a abordagem *offline-first* (OFFLINE, 2020).

## 2.10 RICE SCORE

O modelo de pontuação RICE é um framework de priorização projetado para ajudar gerentes a definir quais produtos, funcionalidades ou outras iniciativas devem ser colocadas no roteiro de produto através da pontuação de quatro critérios:

alcance, impacto, confiança e esforço (RICE, 2022). Os termos em inglês formam o acrônimo de RICE, sendo ***r**each*, ***i**mpact*, ***c**onfidence* e ***e**ffort*. A pontuação final do RICE obtém-se pela fórmula $RICE\ Score = \frac{R \cdot I \cdot C}{E}$.

O primeiro critério é o alcance. Nele, devem ser estimadas quantas pessoas serão atingidas pela iniciativa em questão. Deve ser definida uma janela de tempo para tal estimativa. A pontuação do alcance será o número estimado. O impacto pode ser um critério quantitativo, como quantas pessoas serão convertidas, ou qualitativo, como uma melhoria de usabilidade vai impactar na felicidade do cliente, respectivamente. A confiança indica o quanto o time tem base para afirmar as pontuações anteriores. O objetivo é equilibrar as pontuações de iniciativas baseadas em dados e outras baseadas mais na intuição. Por fim, para pontuar o esforço, estima-se o total de recursos (produtos, pessoas, ferramentas etc) para completar a iniciativa dentro de um espaço de tempo. Normalmente, utiliza-se algo como "pessoa-mês" (RICE, 2022).

## 3. TRABALHOS RELACIONADOS

Neste capítulo, são apresentados alguns dos trabalhos relacionados ao produto de software proposto neste trabalho. A seleção dos trabalhos relacionados foi feita através de pesquisas nos mecanismos de busca das lojas de aplicativos para Android e iOS, bem como o Google. Vários termos de busca foram testados, como: "*software gestão tênis*", "*gestão tênis*", "*tennis management*", "*tennis club manager*" e "*tennis management software*". Nas lojas de aplicativos, somente jogos apareceram. No Google, o último termo utilizado trouxe vários resultados, incluindo artigos que apresentam várias opções de produtos para gestão de clubes de tênis. A partir dessa busca, foi possível encontrar alguns aplicativos nas lojas de aplicativos pesquisando diretamente pelo nome do produto.

Alguns dos trabalhos relacionados encontrados são produtos com pouca divulgação e sem testes abertos, onde seria possível se cadastrar e utilizar o software por um período determinado. Por conta disso, fazer uma análise desses produtos demandaria mais tempo e optou-se por aqueles em que não havia necessidade de agendar uma demonstração com a empresa antes de iniciar seu uso. O Quadro 1 apresenta essas dificuldades junto aos produtos que não foram

analisados mais profundamente. Por conta disso, optou-se por 3 produtos analisados, conforme apresentados adiante.

Quadro 1. Motivos de desistência da análise de alguns produtos.

| Produto | Motivo de Desistência |
|---|---|
| TPC - Matchpoint | Necessário agendamento de apresentação. |
| EZFacility | Necessário agendamento de apresentação. |
| Activesports | Site com erro 404. |
| USTA - Serve Tennis | Necessário comprovar residência nos EUA. |
| DaySmart Appointments | Necessário agendamento de apresentação. |
| Club Manager Central | Site não abre. |
| clube-manager.org | Não foi possível fazer o cadastro. |
| onlineclubmanager.com | Não foi possível fazer o cadastro. |
| Tennis Bookings | Processo de cadastro muito extenso. |
| LetzPlay | Necessário agendamento de apresentação para a parte de gestão de clubes. |

Fonte: elaborado pelo autor (2022)

Cada um dos produtos foi analisado em termos de propósito, *onboarding* (conjunto de processos para integração de novo usuário ao sistema), interface, métricas de negócio, personalização e completude de funcionalidades para atender a um clube de tênis. Vale ressaltar que o critério de personalização envolve a capacidade do produto fazer com que o usuário sinta que aquele sistema é dele ou de seu clube, semelhante à abordagem de um produto white label. É um critério subjetivo que foi descoberto à medida que as análises avançaram. Foram identificados alguns pontos de personalização já mais populares, como a utilização de uma logo da organização, alteração do esquema de cores e URL personalizada.

## 3.1 SPRINGLY

Springly (SPRINGLY, 2022) é um software exclusivo para web para gestão de afiliados para organizações sem fins lucrativos e conta com alguns recursos (Quadro 2) para auxiliar na retenção do público-alvo do usuário do software. O site da empresa apresenta páginas específicas para alguns esportes, dando a entender que o software atende àquele esporte de forma específica, como é o caso do tênis

(SPRINGLY TENNIS, 2022), mas o software em si trata-se de um modelo genérico capaz de atender a diversas situações. Os módulos de funcionalidades, que podem ser ativados ou desativados pela interface, estão apresentados no Quadro 2.

Quadro 2. Módulos de funcionalidades do Springly.

| **Nome do Módulo** |
|---|
| Contabilidade |
| Gestão de afiliados |
| Doações |
| Eventos |
| Loja online |
| Campanhas de e-mails |
| Construtor de website |
| Lista de presença |

Fonte: elaborado pelo autor (2022)

### 3.1.1 Onboarding

O processo de onboarding no software é muito simples para o cadastro e busca ter uma linguagem amigável para o primeiro acesso em algumas funcionalidades. Poucos minutos são necessários para iniciar o uso. Entretanto, para o usuário que não está acostumado com ferramentas de gestão, alguns termos podem soar um pouco estranhos. Um exemplo disso é que ao acessar a seção de filiados, a primeira pergunta que o software faz é "*Que tipo de campanha você está executando?*" para que seja construído um formulário para que os futuros filiados preencham. Não necessariamente o termo "campanha" é facilmente compreendido por qualquer usuário do software sem uma explicação prévia. Apesar de ser um termo amplamente usado em ferramentas de CRM,, não existe uma expectativa de que o público-alvo do software compreenda isso.

De forma geral, o onboarding é satisfatório, sendo possível compreender o funcionamento do software realizando alguns registros de teste em cada funcionalidade. A maior dificuldade encontrada foi entender para que servem algumas das funcionalidades, pois são tão genéricas que encontrar seu caso de uso para o clube de tênis pode exigir certo esforço.

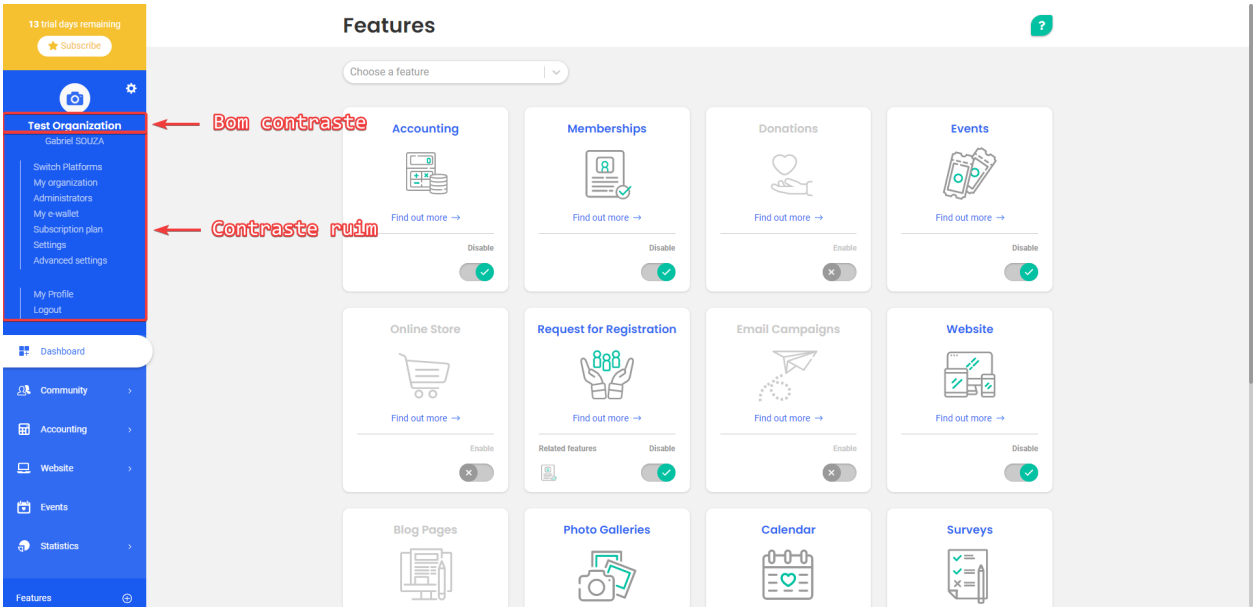
### 3.1.2 Interface

Em geral, a interface é simples e segue o padrão com menu principal na lateral esquerda. Nos formulários, algumas dicas surgem à medida que os campos são preenchidos, facilitando a compreensão da ferramenta e colaborando no aprendizado sobre gerenciamento de filiados. Um exemplo disso é o texto informativo que surge quando é criado um formulário para cadastro de pessoas em um evento, indicando que um menor número de campos obrigatórios aumenta o engajamento em inscrições.

Um ponto interessante sobre a interface é que ela é altamente personalizável em nível de funcionalidade, permitindo ao usuário escolher quais deseja utilizar. Dessa forma, evita a poluição visual de recursos que não deseja utilizar. Isso torna o uso do software mais simples.

Por fim, notou-se que existem pequenos problemas de contraste no menu lateral, onde o fundo azul com a fonte cinza claro dificulta um pouco a leitura, como mostrado na Figura 2.

Figura 2. Captura da tela de escolha de funcionalidades no Springly.



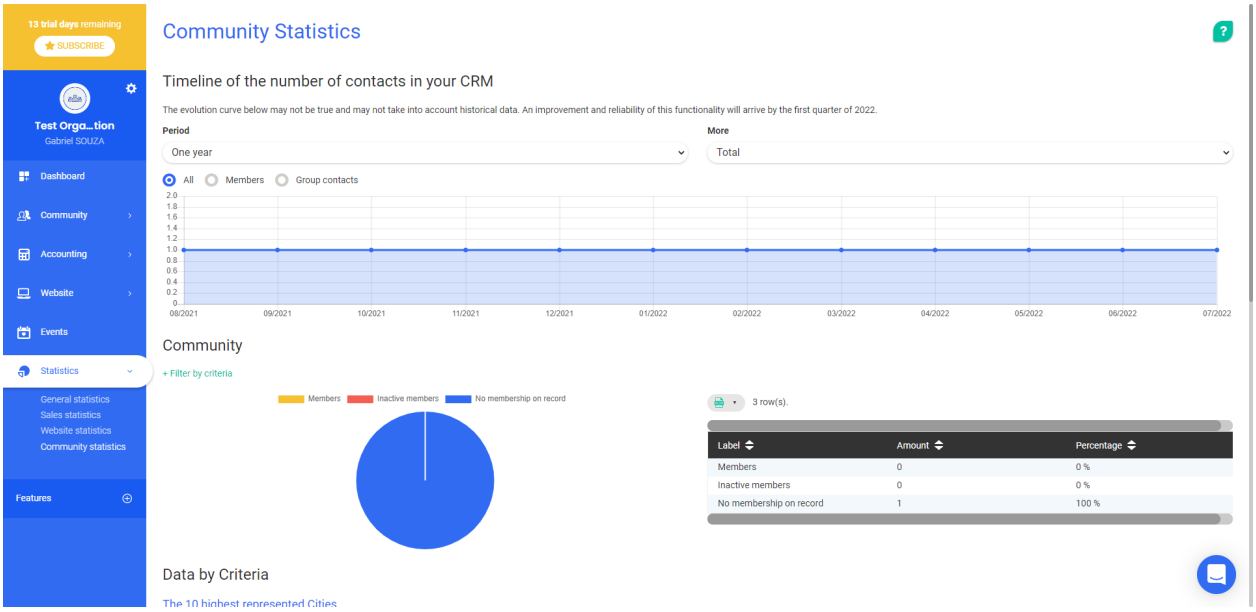

Fonte: elaborada pelo autor (2022).

### 3.1.3 Métricas de Negócio

Em geral, bons produtos de gestão trazem consigo funcionalidades para visualização de métricas de negócio. O Springly atende a esse critério fornecendo 4 visualizações diferentes:

- Estatísticas gerais;
- Estatísticas de vendas;
- Estatísticas do website;
- Estatísticas da comunidade, mostrado na figura 3.

Figura 3. Captura da tela de estatísticas da comunidade no Springly.




Fonte: elaborada pelo autor (2022).

Por se tratar de uma ferramenta que combina foco na gestão de filiados, CRM e vendas, as métricas são limitadas a esse escopo, podendo fazer com que um clube de tênis sinta falta de informações como número de jogos por quadra, manutenções, etc.

### 3.1.4 Personalização

O nível de personalização é suficiente para pequenos negócios, pois a ferramenta permite o upload da logo da empresa e entrega um editor de site simples. Entretanto, não é possível realizar alterações das cores do painel de administração, ficando sempre na combinação de azul com branco. Além disso, os formulários que envolvem recursos financeiros estão adaptados somente para lidar com o dólar e com taxas em porcentagem fixadas que não se adaptam à realidade do Brasil.

### 3.1.5 Completude de Funcionalidades

O Springly é um produto que possui várias funcionalidades, sendo que todas elas podem atender a um clube de tênis, mas não possui todas as funcionalidades que um clube de tênis pode precisar. Isso porque ele atende muito bem às necessidades relacionadas à comunicação e finanças. O software permite a fácil criação de eventos, com formulários em que o usuário já pode realizar pagamentos no momento da submissão da inscrição, de forma muito simples. Recursos como disparo de comunicados via e-mail e lembretes de pagamento são acionáveis por caixas de seleção, sem a necessidade de maiores configurações.

Apesar do Springly entregar muito bem alguns recursos, ele não é uma ferramenta que entrega todos os recursos necessários para a administração de um clube de tênis. Uma das principais funcionalidades, que é a reserva da quadra, está fora do escopo do software. Dessa forma, não é um produto que atende sozinho a um clube de tênis de forma integral.

## 3.2 ACES TENNIS MANAGEMENT

ACES TM (ACES, 2022) é um produto composto pela aplicação web e aplicativos para smartphones Android e iOS. Ele tem um foco maior no próprio esporte em si, trazendo principalmente funcionalidades de agendamento de horários para quadras, aulas e eventos, e análises de performance dos jogadores. O ACES também promove um ambiente de integração entre pessoas afiliadas ao clube através do compartilhamento de mídia, funcionando como uma rede social no estilo Instagram.

### 3.2.1 Onboarding

O processo de onboarding do ACES é bem simples, tendo somente alguns registros já criados no momento que o ambiente abre. Não foi encontrado nenhum outro recurso explicando ou facilitando o primeiro uso em nenhuma das funcionalidades.

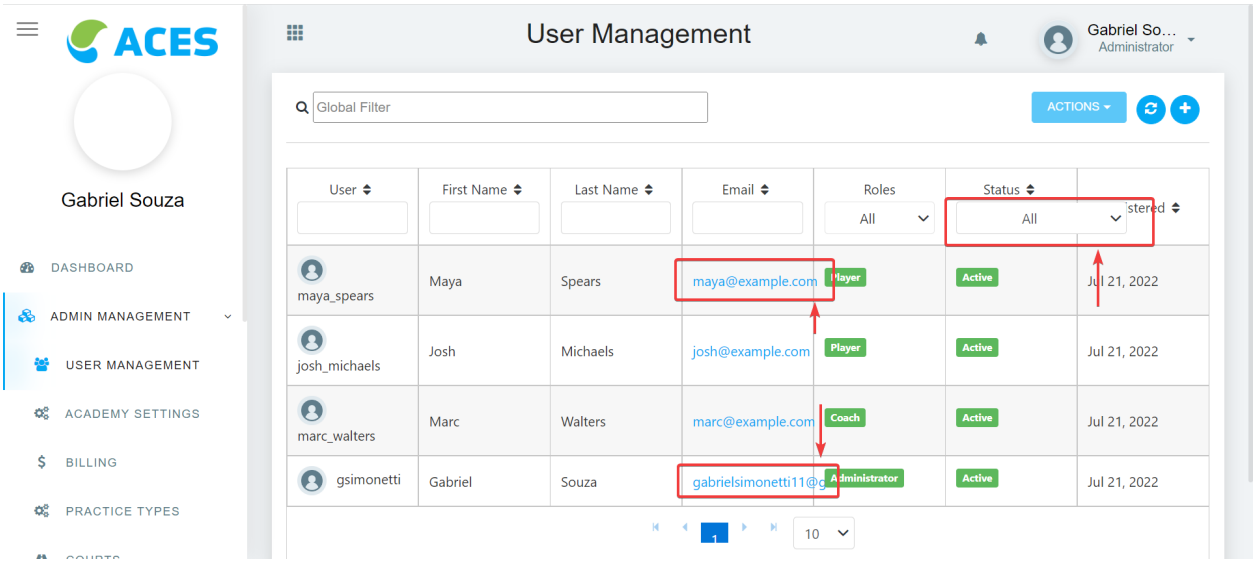
### 3.2.2 Interface

A interface tem aparência moderna, porém com falhas facilmente perceptíveis ao usuário final. É possível notar partes do layout vindas de templates conhecidos

na comunidade de desenvolvimento. A organização dos elementos na tela foi basicamente empilhada na vertical ou horizontal, fazendo com que telas de cadastro facilmente fiquem maiores do que a altura disponível no navegador ou que a quantidade de informações dispostas na horizontal fique grande e com espaçamento pequeno. Isso causa uma poluição visual.

A responsividade não é muito bem tratada, possuindo vários pequenos problemas de layout. Nenhum deles inviabiliza o uso da aplicação, mas trazem a sensação de falta de capricho, como pode ser observado na Figura 4.

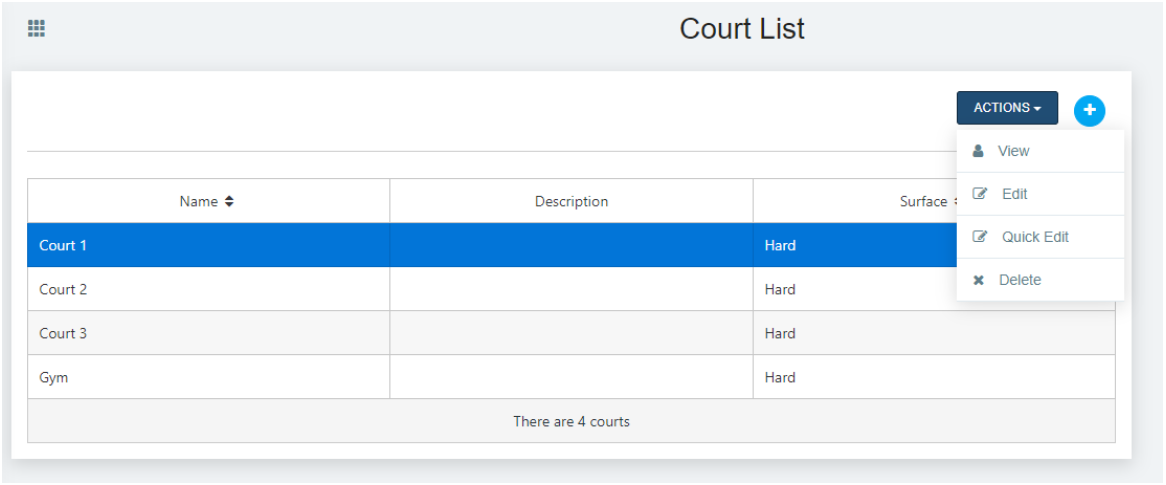
Figura 4. Captura de tela com problemas de layout do software ACES.




Fonte: elaborada pelo autor (2022).

Pode-se observar também que a usabilidade das listagens não é tão agradável, pois sempre é necessário selecionar um elemento para então abrir um seletor de ações e selecionar qual ação deve ser executada. Esse comportamento é mostrado na Figura 5.

Figura 5. Captura de tela com listagem no software ACES.

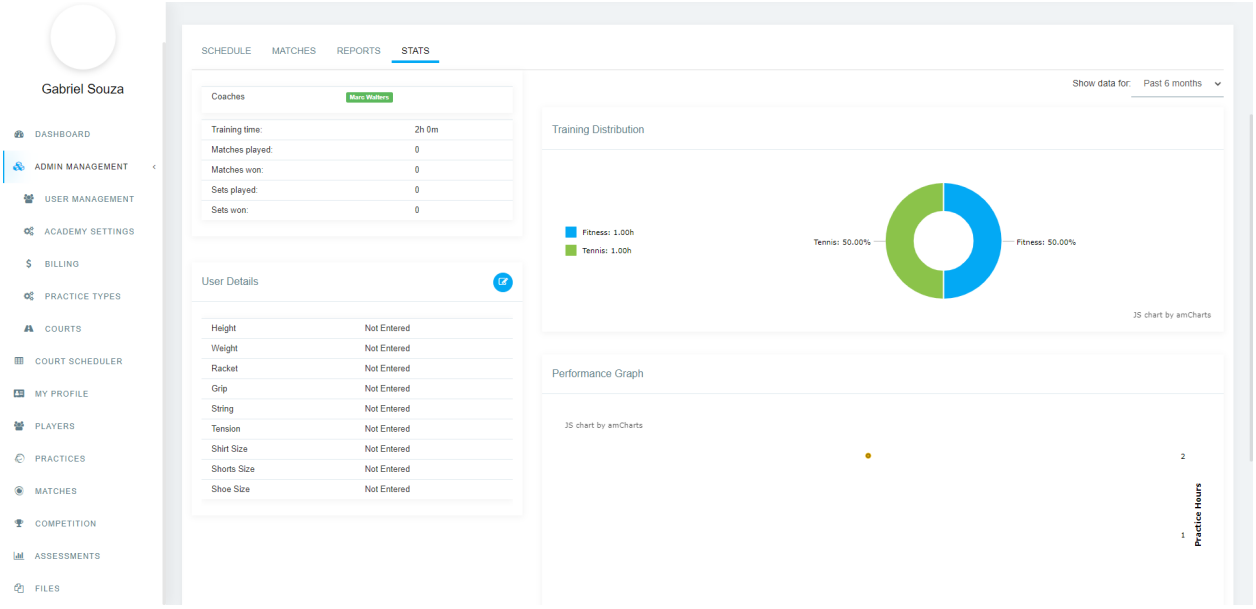



Fonte: elaborada pelo autor (2022).

### 3.2.3 Métricas de Negócio

O ACES possui uma seção de métricas dos jogadores, como pode ser observado na Figura 6. Ela apresenta dados gerais, como o total de horas de prática e horas jogadas, bem como o número de partidas jogadas e vencidas. O ponto mais interessante das métricas desse produto é o gráfico de performance da pessoa, que apresenta horas praticadas, partidas vencidas e partidas perdidas. Além disso, o software possui uma seção destinada a relatórios, onde os treinadores podem pontuar cada tenista e seu progresso pode ser acompanhado.

Figura 6. Captura da tela de métricas do jogador no ACES.




Fonte: elaborada pelo autor (2022).

O produto tem um ambiente rico em informações que podem ser inseridas, mas possui poucos recursos de visualização. Um exemplo disso é que os técnicos podem avaliar todos os jogadores em nível de comprometimento, mas não é possível extrair a informação de nível de comprometimento vs número de vitórias. Também pode ser interessante saber quais treinadores têm tido jogadores com melhores resultados, quais os efeitos das vitórias e derrotas no mental e emocional dos jogadores etc. Na seção de completude de funcionalidades, são explicadas algumas métricas que podem ser registradas pelos treinadores e é possível perceber a grande quantidade de possibilidades de análise em cima delas.

### 3.2.4 Personalização

O ACES entrega um nível básico de personalização, permitindo ativação ou desativação de módulos do sistema, configuração de subdomínio específico para o clube e algumas configurações de horário de funcionamento e fuso-horário. É possível inserir a foto de perfil para cada usuário, mas não é possível inserir uma logo do clube, ficando sempre a logo do ACES à mostra. Também não é possível realizar alteração de cores, configurar uma URL específica ou ter um site do clube.

Um ponto curioso percebido logo na tela de login da ferramenta foi que existe um seletor de clube, mostrando todos os clubes que já foram criados na base de dados deles. Isso mostra-se como uma falha de segurança porque todos os subdomínios ficam possíveis de encontrar, facilitando ataques de força bruta ou engenharia social, por exemplo. Uma análise mais aprofundada na segurança do ACES não será feita aqui porque foge do escopo do trabalho.

### 3.2.5 Completude de Funcionalidades

De todos os produtos analisados, o ACES é o que mais destacou-se em termos de funcionalidades por ser bem completo. Ele possui as funcionalidades básicas necessárias para administração de um clube, como o agendamento de quadras e gerenciamento de filiados, mas vai além por ter um rico ambiente de integração entre treinadores e jogadores. O software também conta com histórico de partidas, organização de competições e submissão de arquivos de mídia, onde é possível compartilhar com a comunidade do clube em um ambiente de rede social no estilo do Instagram.

Um treinador pode escolher um jogador que esteja treinando com ele e registrar um relatório indicando como a pessoa está evoluindo. Esse relatório é dividido em três partes: tênis, mental e emocional, e físico. Na parte de tênis, o treinador dá notas de 0 a 100 em intensidade, foco e treino. Na parte mental e emocional, também dá notas dentro das mesmas medidas para rotina de competição, atitude positiva e mentalidade. Por fim, na seção de físico, dá notas para intensidade, foco e comprometimento. Esses relatórios possuem opções de visualização, permitindo que outros treinadores vejam, que o jogador veja e/ou que os pais do jogador vejam. Além disso, esse ambiente de integração entre treinador e jogador permite a criação de planos semanais, planos de desenvolvimento, notas de observação da prática dos treinos e avaliação da prática do jogador nas qualidades de energia, esforço e atitude.

## 3.3 CLUB MANAGER 365

O Clube Manager 365 (CLUB MANAGER 365, 2022) é um software exclusivo para web para clubes de esporte com funcionalidades focadas em reservas, torneios e histórico de jogos.

### 3.3.1 Onboarding

O processo de onboarding resume-se à criação de um ambiente de testes com alguns registros de jogadores já criados para que o usuário possa conhecer a ferramenta. Entretanto, não possui nenhum tipo de recurso que guie ou ensine o usuário sobre qualquer funcionalidade do sistema.

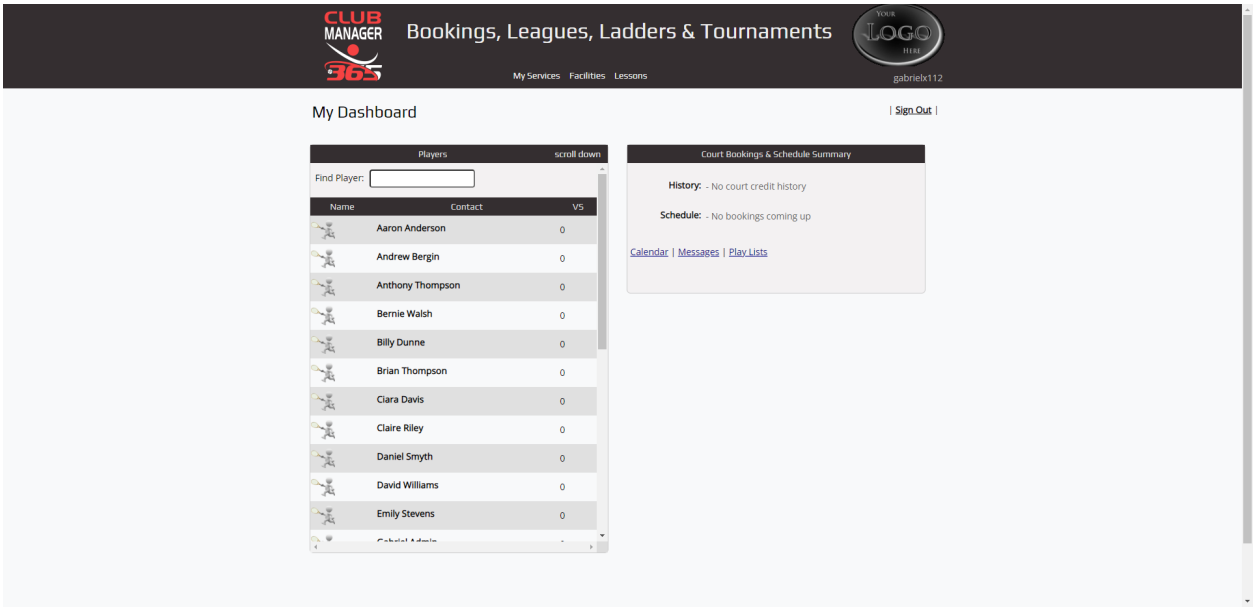
### 3.3.2 Interface

A interface desse produto é bem simples, trazendo elementos de tamanho fixo e não responsivos. O uso é complicado, pois demanda que o usuário vá descobrindo cada uma das funcionalidades, já que o posicionamento dos elementos e a navegação não seguem as práticas comuns de mercado. O calendário somente é acessível pelo link encontrado no dashboard, tornando mais confuso caso o usuário esteja acostumado com produtos que centralizam a navegação principal em um mesmo local. Pela formatação da aplicação, é comum pensar que uma funcionalidade importante como o calendário seja acessível pelo menu principal, posicionado na parte de cima. Alguns elementos visuais não são facilmente

compreendidos, como é o caso da coluna “VS” na lista apresentada no painel principal da aplicação, mostrado na Figura 7.

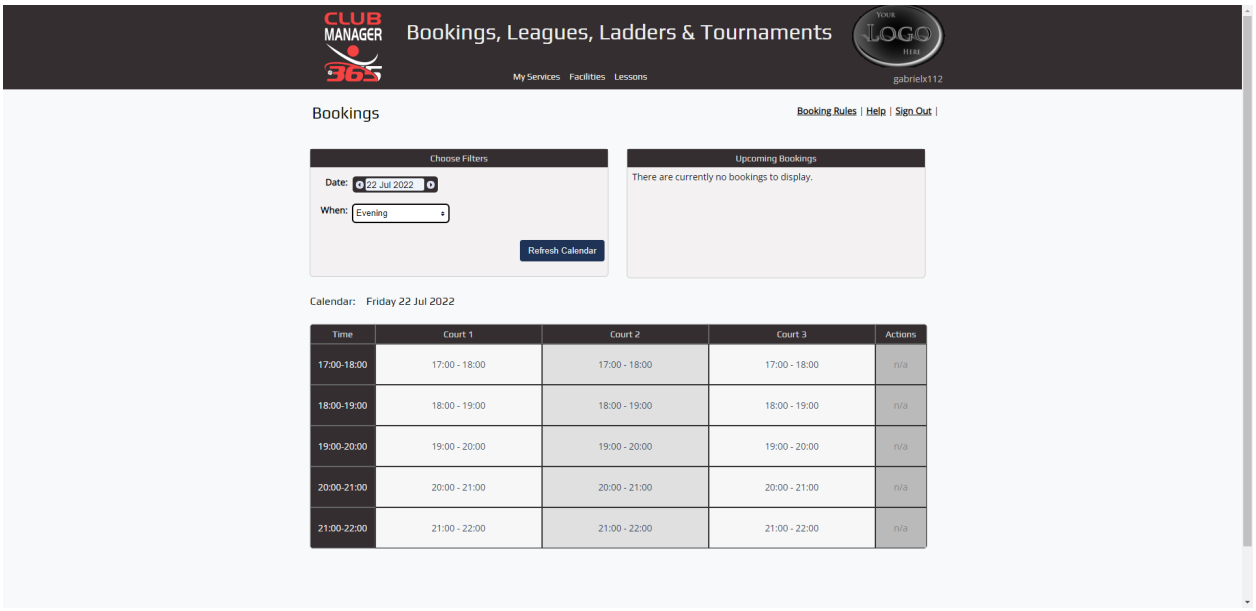
Nota-se que o software não segue padrões mais modernos de interface, como: o uso da largura total da tela, menu de navegação centralizado, melhor contraste das cores e layout dos elementos, tamanho e posicionamento. Podemos observar essas características nas figuras 7 e 8 a seguir:

Figura 7. Captura da tela do painel do software Club Manager 365.




Fonte: elaborada pelo autor (2022).

Figura 8. Captura da tela de agendamentos do Club Manager 365.




Fonte: elaborada pelo autor (2022).

### 3.3.3 Métricas de Negócio

O software não apresenta métricas de negócio, mas é possível visualizar o histórico de jogos com poucos detalhes.

### 3.3.4 Personalização

A única característica personalizável no Club Manager 365 é a logo da empresa. Entretanto, esse nível de personalização é uma prática muito comum no mercado atualmente e não é digno de um destaque. Dessa forma, atribui-se neste trabalho que esse produto não é personalizável.

### 3.3.5 Completude de Funcionalidades

Apesar do Club Manager 365 ser um software fora dos padrões do mercado de hoje, ele possui algumas funcionalidades não encontradas nos outros produtos explorados, como é o caso da “play list” e “slot watching”. A funcionalidade “play list” serve para que o usuário possa marcar quais são os jogadores dentro de uma organização que ele mais gostaria de jogar regularmente. Não ficou muito claro o que o software faz com isso, mas encontrar esse tipo de funcionalidade abriu espaço para pensar sobre possibilidades geradas com ela. Por conta disso, essa funcionalidade foi incluída para futura análise da solução proposta neste trabalho. O recurso “slot watching” serve para avisar o usuário que um determinado dia e horário uma quadra foi liberada. Basicamente, ele pode registrar que têm interesse em um horário já agendado para saber se o agendamento foi cancelado.

O pacote de funcionalidades do produto está dentro do mínimo desejado, pois é possível realizar o agendamento de quadras, visualizar a lista de jogadores filiados, organizar eventos e enviar mensagens. Entretanto, seu uso é difícil e limitado, tornando-o pouquíssimo atrativo. Considerando que boa usabilidade é um pré-requisito para este tipo de software, conclui-se que não atende às necessidades de um clube de tênis moderno.

## 4. DESENVOLVIMENTO

Nesta seção, é apresentado o estudo do problema, a definição do produto de software proposto e a implementação da solução até onde foi possível. Foi definido o nome EasyTennis para o produto. Inicialmente, é apresentado o público-alvo,

utilizando a definição de três personas. Em sequência, fala-se sobre as necessidades dos usuários, diferenciais propostos e como o escopo do MVP foi elaborado. Por fim, são apresentadas a definição e a implementação da solução, comentando sobre as decisões de arquitetura de software e tecnologia.

Este projeto do software foi desenvolvido com o objetivo de balancear a extensibilidade, escalabilidade, complexidade e redução de escopo para definição de um MVP. Portanto, alguns itens, como a modelagem de dados, ficaram mais simples. Outros itens, como a arquitetura do aplicativo para dispositivo móvel ou a arquitetura utilizada na API tornaram-se mais complexos do que o necessário para um MVP desse porte. Cada um desses pontos é discutido nesta seção do trabalho.

## 4.1 PÚBLICO-ALVO

O público-alvo do MVP proposto neste trabalho são jogadores e treinadores afiliados a clubes de tênis, bem como seus gestores. Foram elaboradas algumas descrições de personas para compreender melhor o público-alvo.

### 4.1.1 Clube de Tênis

Um clube de tênis, dentro do escopo deste trabalho, pode ser definido como um ambiente físico, administrado por pessoas, que contém quadras de tênis e outros espaços, permitindo somente a pessoas afiliadas o uso desses espaços. Esses clubes podem conter professores de tênis contratados ou autônomos que utilizam os espaços do clube para dar aulas. O EasyTennis contempla os dois casos, mas nenhuma funcionalidade de gestão de funcionários foi incluída no MVP.

Além dessas atividades do cotidiano, os clubes de tênis muitas vezes também são a sede de campeonatos de diversos tamanhos. Os campeonatos às vezes são promovidos pelo próprio clube e às vezes são feitos em parceria ou sob locação do espaço.

### 4.1.2 Persona 1: Hugo Andrade - Jogador de Tênis

Hugo tem 24 anos, nasceu em Florianópolis e mora com seus pais. Está no quarto ano do curso de Direito na universidade e tem o sonho de tornar-se juiz, assim como seu pai. Conheceu o tênis aos 11 anos por influência de seu pai, praticou e progrediu no esporte. Aos 15, participava de campeonatos infanto juvenis com patrocínio.

Atualmente, treina religiosamente por duas horas todas as manhãs com seu treinador, de segunda a sábado. Seu passatempo aos finais de semana é jogar com os amigos no clube ou participar dos campeonatos da região. Hugo tem preferência por quadras de saibro, seu ídolo é o espanhol Rafael Nadal e ele gostaria de saber como está sua progressão para atingir um alto nível no esporte.

Hugo treina em um clube pequeno com pouco mais de 20 jogadores filiados e mais alguns treinadores. O clube possui um sistema de administração, acessível por navegador, onde é possível realizar a reserva das quadras, evitando conflitos com os demais jogadores. Costumam compartilhar as novidades do clube por aplicativo de mensagens e avisar sobre o cancelamento de reservas de quadras também por ali, já que o sistema não envia notificações sobre isso.

### 4.1.3 Persona 2: Alexandre de Moraes - Supervisor Técnico

Alexandre é paulista e treina tênis desde os 8 anos de idade. Já alcançou o nível profissional, lesionou-se e passou a atuar como treinador após um tempo de recuperação. Ele é graduado em educação física e acompanha desde atletas juvenis até profissionais. Acredita que uma boa performance do jogador é atingida com muita dedicação e acompanhamento personalizado. Por isso, ele vende uma modalidade de treino onde realiza avaliações periódicas em vários aspectos para seus alunos, incluindo planos de treino e até dicas de alimentação.

Ultimamente, os resultados dos alunos de Alexandre têm se destacado e os pedidos para os treinos personalizados têm aumentado. Como ele costuma fazer todo esse trabalho de análise em uma planilha impressa junto a uma prancheta, a quantidade de informação espalhada começou a aumentar. Eventualmente, esquece de imprimir a planilha de algum aluno. Esse trabalho unido ao controle do calendário de aulas e pagamento isso tem dificultado seu trabalho. Alexandre quer evoluir seus tenistas, não ficar administrando eles.

### 4.1.4 Persona 3: Augusto Ribeiro Souza - Gestor de Clube

Apaixonado pelo tênis há mais de 30 anos, Augusto decidiu abrir um clube de tênis para possibilitar que mais pessoas pudessem usufruir dessa paixão e ainda ganhar dinheiro com isso. Ele não é um gestor experiente, mas tem noções de como organizar as finanças de um negócio. Entretanto, acaba tendo mais trabalho do que gostaria porque precisa garantir a manutenção das quadras, organizar

campeonatos, promover melhorias no clube, cobrar o pagamento dos filiados e ainda ouvir os pedidos e reclamações de quem frequenta o local. Augusto optou por ter um clube para nunca distanciar-se das quadras, mas a resolução desses problemas de gestão acabaram o distanciando um pouco. Os filhos de Augusto também jogam tênis e seu sonho é ter uma renomada academia de tênis.

## 4.2 ESCOPO DO MVP

Utilizando o Svelte e o .NET MAUI, é possível construir aplicações para vários navegadores ou plataformas, mas isso também exige maior dedicação para a configuração de cada caso. Dessa forma, o escopo inicial do desenvolvimento da solução neste trabalho havia sido definido como um MVP com aplicação web para navegador Chrome e aplicação nativa para dispositivos Android. Basicamente, além da infraestrutura e APIs, foi proposta a implementação de duas aplicações para o usuário final, sendo uma com Svelte (web) e outra com .NET MAUI (mobile). Entretanto, por limitações de tempo e equipe, optou-se por não realizar a implementação para web neste trabalho, ficando somente a aplicação para Android e backend.

Para elencar as necessidades do público-alvo, foi realizado um estudo buscando quais são as principais funcionalidades desejadas por gestores de clubes de tênis. Inicialmente, foram identificadas algumas necessidades pela própria vivência do estudante frequentando clubes de tênis de diversos tamanhos. Nessa parte, surgiram funcionalidades como reserva de quadras, histórico de jogos e encontrar parceria para jogar. O próximo passo foi pesquisar em mecanismos de busca quais são as maiores necessidades de gestão de clubes de tênis ou clubes de esporte, no geral. A partir disso, foram encontradas outras funcionalidades como pagamentos, comunicação automática, finanças e organização de eventos (campeonatos, por exemplo). Por fim, foram realizadas as análises dos trabalhos relacionados, levantando mais algumas funcionalidades como perfil de usuário, histórico de jogos, fórum e notificações de horários vagos. Os resultados dessa análise foram colocados na Tabela 1.

Além das funcionalidades, notou-se que o *onboarding* dos produtos analisados não recebem muita atenção, conforme pontuado na seção de trabalhos relacionados, o que pode indicar uma barreira de entrada para o produto. De acordo

com as personas descritas neste trabalho, não espera-se que tenham uma vasta experiência com software de gestão. Dessa forma, foi considerado que a facilidade de uso é uma necessidade do usuário dentro do público-alvo.

Para definir o escopo de quais funcionalidades seriam atendidas na solução proposta neste trabalho, optou-se por aplicar o modelo de priorização RICE Score por tratar-se de um framework de aplicação simples e por introduzir um nível de confiança maior nas decisões envolvendo a definição do escopo do MVP do produto em relação a uma simples escolha pessoal. A Tabela 1 apresenta todas as funcionalidades analisadas, em ordem decrescente de pontuação RICE.

Tabela 1. RICE Score aplicado nas funcionalidades.

| Funcionalidade | R | I | C | E | RICE | MVP |
|---|---|---|---|---|---|---|
| Reserva de espaços | 950 | 3 | 100% | 5 | 570.00 | Sim |
| Agendamento de aulas | 800 | 2 | 80% | 5 | 256.00 | Sim |
| Perfil do usuário | 600 | 1 | 80% | 3 | 160.00 | Sim |
| Gerenciamento de usuários | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Gerenciamento de clubes | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Gerenciamento de espaços | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Pagamentos (mensalidades, aulas e quadras) | 800 | 1 | 80% | 6 | 106.67 | Não |
| Histórico de jogos | 150 | 1 | 80% | 3 | 40.00 | Não |
| Análise de performance do jogador | 100 | 1 | 80% | 3 | 26.67 | Não |
| Notificação de horário vago | 100 | 1 | 80% | 3 | 26.67 | Não |
| Recomendações de jogos | 200 | 1 | 50% | 5 | 20.00 | Não |
| Comunicados (sem automação) | 20 | 2 | 80% | 3 | 10.67 | Não |
| Organização de campeonatos | 30 | 3 | 80% | 8 | 9.00 | Não |
| Fórum | 100 | 0.5 | 50% | 3 | 8.33 | Não |
| Finanças (registros e análises) | 20 | 2 | 80% | 8 | 4.00 | Não |
| Comunicadas (com automação) | 10 | 2 | 80% | 5 | 3.20 | Não |
| Espaço de feedbacks para o clube | 50 | 0.25 | 50% | 3 | 2.08 | Não |
| Manutenção das quadras | 20 | 0.25 | 50% | 3 | 0.83 | Não |

Fonte: elaborada pelo autor (2022).

As colunas R, I, C e E indicam os critérios *reach* (alcance), *impact* (impacto), *confidence* (confiança) e *effort* (esforço), respectivamente. A coluna RICE indica a pontuação RICE calculada e a coluna MVP indica se a funcionalidade foi escolhida para ser parte do MVP, considerando a nota de corte em 150.

Para calcular o RICE Score, normalmente define-se um período de tempo e um escopo de usuários para determinarmos o primeiro critério (*reach*). Como este é um produto que ainda não foi desenvolvido e faz parte de um trabalho acadêmico, foram escolhidos alguns números arbitrários. O Quadro 3 apresenta as definições adotadas para realizar o cálculo da priorização das funcionalidades.

Quadro 3. Definições adotadas para realização do cálculo de priorização.

| **Definição** | **Valor** |
|---|---|
| Usuários ativos | 1000 pessoas |
| Tempo de aderência | 3 meses |
| Unidade de esforço | Pessoa-semana |

Fonte: elaborado pelo autor (2022).

Portanto, se o critério *reach* tiver o valor 200, significa que foi estimado que 200 dos 1000 usuários usariam a funcionalidade após 3 meses do lançamento dela. Uma unidade de esforço equivale a uma pessoa com os mesmos conhecimentos do autor deste trabalho trabalhando por uma semana, considerando uma carga horária de 40 horas por semana.

Para o critério *impact*, a definição de valores é apresentada no Quadro 4.

Quadro 4. Definições adotadas para o critério *impact* do modelo *RICE score*.

| **Valor** | **Descrição** |
|---|---|
| 3 | Impacto massivo |
| 2 | Impacto alto |
| 1 | Impacto médio |
| 0.5 | Impacto baixo |
| 0.25 | Impacto mínimo |

Fonte: elaborado pelo autor (2022)

Para o critério *confidence*, a definição de valores é apresentada no Quadro 5.

Quadro 5. Definições adotadas para o critério *confidence* do modelo *RICE score*.

| Valor | Descrição |
|---|---|
| 100% | Confiança alta |
| 80% | Confiança média |
| 50% | Confiança baixa |

Fonte: elaborado pelo autor (2022).

A seguir, são apresentadas as descrições das funcionalidades que fazem parte do escopo do MVP.

### 4.2.1 Reserva de Espaços (mobile)

Para atender à necessidade de reserva de quadras de forma mais genérica, optou-se por definir a funcionalidade como uma reserva de espaços. Um espaço pode ser uma quadra ou outro espaço qualquer. Dessa forma, o software não fica limitado às reservas de quadras. O objetivo desta funcionalidade é permitir que usuários realizem a reserva de horários dos espaços de um determinado clube para evitar conflitos de utilização deles. Cada espaço possui um número máximo de usos simultâneos, definido pela pessoa gestora do clube. O usuário poderá criar, visualizar ou editar uma reserva sua. Não será possível excluir, apenas cancelar (que é considerado como uma edição do registro aqui). Também poderá visualizar as reservas feitas por outras pessoas, porém não poderá editar. Não será possível criar uma reserva para um espaço cujos horários conflitem com outra reserva já existente no sistema.

Para clubes com horários específicos de funcionamento, define-se um horário mínimo e um horário máximo para a reserva de cada espaço. O tempo mínimo de reserva de um espaço também pode ser personalizado para cada espaço.

### 4.2.2 Agendamento de Aulas (mobile)

Semelhante ao item anterior, o agendamento de aulas serve para que os treinadores possam gerenciar suas agendas e facilitar a marcação de horários para treinos. Um agendamento de aula é um tipo de reserva de espaço. A diferença é que deve ser criado ou editado somente por treinadores. Essa condição deve-se ao tipo de vínculo que treinadores têm com o clube, permitindo a realização de atividades

profissionais dentro dos espaços do clube. Cada aula tem um valor específico, sendo somente visível para as pessoas envolvidas no agendamento.

### 4.2.3 Perfil de Usuário (mobile)

Trata-se de uma seção no software dedicada para manutenção das informações do perfil do usuário, como nome, e-mail, telefone e foto, semelhante a uma rede social. Uma característica importante para o tênis é o lado dominante, indicando se a pessoa é destra, canhota ou ambidestra. Um usuário pode visualizar e editar os dados de seu perfil, mas somente visualizar os dados de perfil de outros usuários afiliados ao mesmo clube.

### 4.2.4 Gerenciamento de Usuários (somente backend)

Esta é uma simples funcionalidade de gerenciamento de usuários, permitindo que pessoas donas de clubes possam cadastrar, visualizar e editar o perfil e vínculos dos afiliados de seus clubes para que eles possam acessar o sistema.

### 4.2.5 Gerenciamento de Clubes (somente backend)

Esta é uma simples funcionalidade de gerenciamento de clubes, permitindo que pessoas donas de clubes possam registrar, visualizar e editar seus clubes.

### 4.2.6 Gerenciamento de Espaços (somente backend)

Esta é uma simples funcionalidade de gerenciamento de espaços dos clubes, permitindo que pessoas donas de clubes possam registrar, visualizar e editar os espaços de seus clubes.

## 4.3 DIFERENCIAIS

A partir do estudo dos trabalhos relacionados, foi possível definir alguns diferenciais para a proposta de desenvolvimento do EasyTennis, apresentados a seguir. Dos diferenciais propostos, somente o item multiplataforma foi atendido no quesito de tecnologia, pois existe a possibilidade de compilar facilmente para outras plataformas, e o funcionamento em ambiente sem conexão com a internet.

### 4.3.1 Multiplataforma

A maioria dos produtos encontrados não são multiplataforma. Dessa forma, coloca-se este como um diferencial em relação à maioria. Pelas tecnologias definidas, é possível produzir o software para os principais navegadores e nativamente para Windows, macOS, Android e iOS. Entretanto, o escopo do MVP limitou-se ao sistema operacional Android. Compilar para outras plataformas ficou como sugestão de trabalho futuro.

### 4.3.2 Otimização de Agendas

Em geral, os produtos estudados contam com um gerenciamento simples de reservas de quadras. O diferencial proposto aqui trata-se da otimização das agendas de treinadores para evitar horários vagos, principalmente em decorrência de cancelamentos.

O funcionamento ocorre através de quatro gatilhos: quando um novo agendamento é criado, quando o horário de um agendamento é alterado, quando um agendamento é cancelado ou ao tocar no botão para sugerir horário no formulário de agendamento de aula. Para qualquer um desses casos, é disparado um processo que analisa os horários agendados para sugerir encaixes melhores, evitando tempo ocioso. Para o caso de uma aula ser cancelada, uma notificação é disparada notificando o treinador com uma sugestão de encaixe, se pertinente.

### 4.3.3 Dados Financeiros para Treinadores

Dentro dos produtos estudados, as informações financeiras exibidas costumam ter uma relação direta com o clube, independente de ser na visão do gestor ou do afiliado. Normalmente, gestores visualizam receitas e despesas do clube, enquanto afiliados visualizam seus pagamentos. Esta proposta é um diferencial por trazer informações financeiras para uma gestão mais profissional por parte dos treinadores, exibindo um histórico de receitas e previsão de fechamento do mês para suas aulas. A ideia é que, além das informações comuns que atendem bem a gestores e afiliados em geral, sejam disponibilizadas informações financeiras para afiliados treinadores.

### 4.3.4 Funcionamento em Locais sem Internet

A solução implementada neste trabalho traz a abordagem de *offline-first* para o aplicativo mobile por conta da variação de sinal e latência em diferentes locais. Isso porque não existe a garantia de que os clubes de tênis estejam localizados em regiões onde o sinal da rede móvel seja forte. Um estudo mais aprofundado sobre a localização dos clubes de tênis no Brasil poderia ser feito para ajudar a decidir se essa abordagem realmente vale o esforço, mas tal pesquisa sai do escopo deste trabalho. Dessa forma, optou-se pelo *offline-first* por conta do benefício claro em relação à falta de conexão e da robustez em manter as aplicações funcionando, mesmo que haja uma degradação no servidor. Além disso, é uma vantagem competitiva em relação aos concorrentes que não tem.

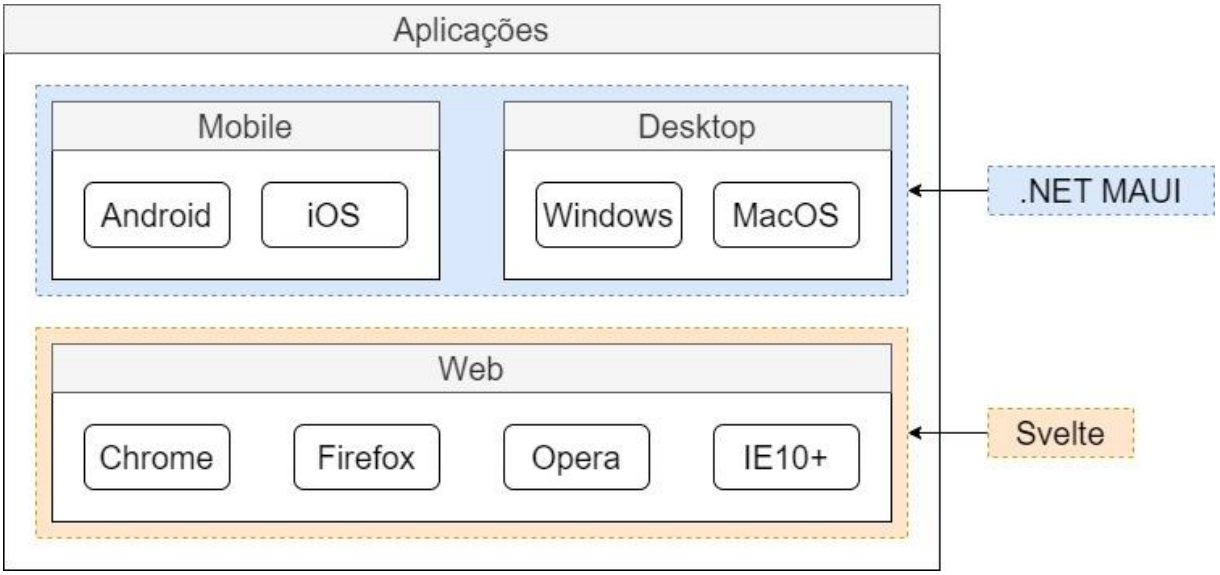
## 4.4 TECNOLOGIAS, SERVIÇOS E FERRAMENTAS

Esta seção do trabalho apresenta algumas definições de tecnologias, plataformas, serviços e ferramentas aplicadas em nível de projeto e/ou implementação.

### 4.4.1 Tecnologias Aplicadas

A Figura 9 ilustra qual é a abrangência da atuação das tecnologias .NET MAUI e Svelte para este projeto.

Figura 9. Svelte e .NET MAUI.




Fonte: elaborada pelo autor (2022).

O Quadro 6 apresenta o contexto, as tecnologias escolhidas e a motivação por trás das escolhas. A coluna de motivação não é exaustiva, trazendo apenas algumas bases para as escolhas. Todas as tecnologias escolhidas são de código aberto e uma descrição mais completa pode ser encontrada na seção de fundamentação teórica deste trabalho.

Quadro 6. Tecnologias escolhidas.

| **Contexto** | **Tecnologia** | **Motivação** |
|---|---|---|
| Aplicações desktop e mobile | .NET MAUI | ● Conhecimento prévio nas linguagens de programação utilizadas (C# e XAML);<br>● Facilidade para implantação em diferentes plataformas: Android, iOS, Windows e macOS;<br>● Grande quantidade de profissionais que trabalham com .NET. |
| Aplicação web | Svelte | ● Menor quantidade de código necessário para atingir o mesmo resultado que outros frameworks populares;<br>● Maior desempenho em relação a frameworks populares, como o React;<br>● Alto grau de satisfação dos desenvolvedores. |
| APIs | .NET 6 | ● Conhecimento prévio da tecnologia;<br>● É uma versão LTS (com suporte de longo prazo);<br>● Simples para exportar bibliotecas que podem ser utilizadas pelo .NET MAUI;<br>● Grande quantidade de profissionais que trabalham com .NET. |
| Banco de dados | MySQL | ● Conhecimento prévio da tecnologia;<br>● Menor custo para manter em cloud;<br>● Capaz de sustentar aplicações de larga escala. |

Fonte: elaborado pelo autor (2022).

A escolha do MySQL, como sistema de gerenciamento de banco de dados,deu-se após a comparação das tecnologias do MySQL, PostgreSQL e SQL Server feitas em (ILYUKHA, 2020), (IBM CLOUD EDUCATION, 2021) e (CHEN, 2021). Os pontos fortes dele são a simplicidade, tamanho da comunidade, documentação, ter uma empresa por trás (Oracle) e atender aplicações de todos os portes.

### 4.4.2 Infraestrutura

No Quadro 7, são apresentados alguns dos serviços escolhidos para sustentar o sistema durante seu desenvolvimento e evolução. No caso deste trabalho, será utilizada a AWS como provedor de serviços na nuvem por conhecimento prévio do autor e referência no mercado (STACK OVERFLOW, 2021). Como a utilização de servidores da AWS localizados nos Estados Unidos costuma ser muito mais barata do que no Brasil, os servidores do EasyTennis estão alocados nos Estados Unidos.

Quadro 7. Serviços de nuvem escolhidos.

| Serviço | Objetivo |
|---|---|
| AWS RDS | Operar bancos de dados na nuvem. |
| AWS Elastic Beanstalk | Facilitar a operação de instâncias de máquinas virtuais para aplicações na nuvem. |
| AWS S3 | Armazenar arquivos na nuvem. |

Fonte: elaborado pelo autor (2022).

### 4.4.3 Monitoramento das Aplicações

Para garantir boa rastreabilidade de eventos e erros que ocorrem no produto, foram escolhidas algumas ferramentas para auxiliar no monitoramento e resolução de problemas. O Quadro 8 apresenta elas.

Quadro 8. Ferramentas de monitoramento escolhidas.

| Serviço | Objetivo |
|---|---|
| Sentry | Monitoramento de performance e erros. |
| App Center | Monitoramento de uso, erros e eventos para aplicações mobile. Além disso, também provê integração e entrega contínua para aplicações mobile desenvolvidas em .NET. |

Fonte: elaborado pelo autor (2022).

### 4.4.4 Procedimentos Demorados

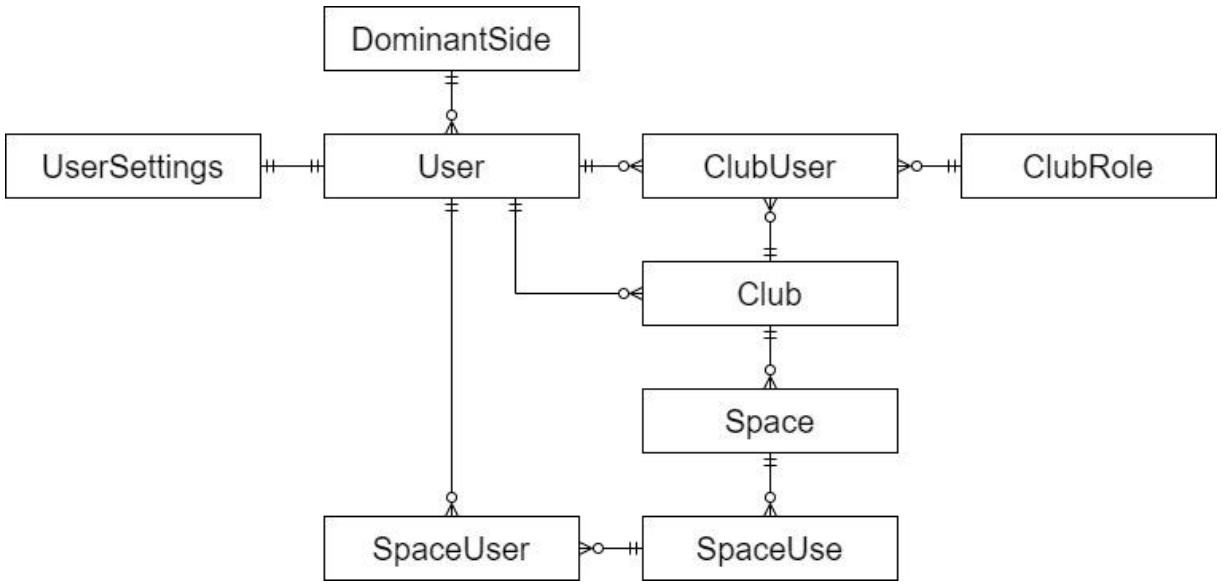
Na solução proposta, procedimentos mais demorados, como geração de relatórios, por exemplo, são trabalhados de forma assíncrona e via *serverless*. Assim, o backend fica mais aliviado, sem correr o risco desses processamentos maiores sobrecarregarem sua execução. A geração de relatórios é um recurso que

pode estar contido dentro de várias funcionalidades, mas não faz parte do escopo do MVP, pois foi cortado. Portanto, não será discutido com maior profundidade aqui.

## 4.5 MODELO DE DADOS

A modelagem de dados do software gira em torno da ideia de que um usuário pode ter zero ou mais clubes, um clube pode ter zero ou mais espaços e um espaço pode ter zero ou mais usos. Para o MVP, o diagrama ER simplificado é mostrado na Figura 10.

Figura 10. Diagrama ER para o EasyTennis.




Fonte: elaborada pelo autor (2022).

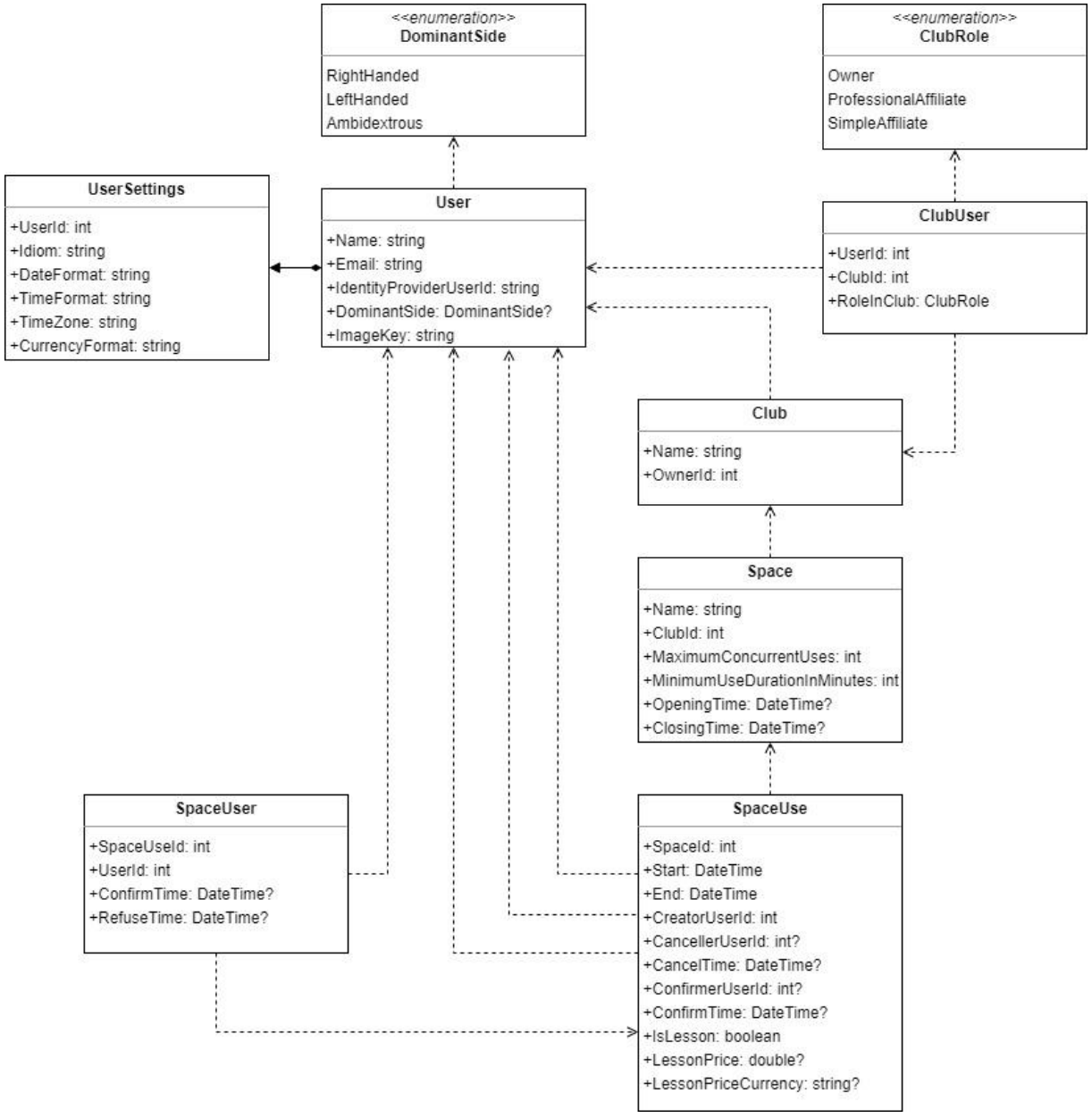
Um primeiro modelo criado utilizava os nomes *TennisClub*, *Court* e *CourtScheduling* ao invés de *Club*, *Space* e *SpaceUse*, respectivamente. Como isso tornava o modelo de dados limitado ao esporte tênis e agendamento de quadras, optou-se pela alteração dos nomes. Dessa forma, o modelo de dados não se limita ao escopo do tênis, podendo ser utilizado para outros esportes em momento futuro, e também não se limita a agendamento de quadras, permitindo o registro de usos passados de qualquer tipo de espaço.

As entidades *DominantSide* e *ClubRole* foram definidas como enumerações quando as classes foram definidas. As classes *User*, *Club*, *ClubUser*, *Space*, *SpaceUse*, e *SpaceUser* são filhas da classe *BaseAuditableEntity*, que segue a hierarquia apresentada na Figura 12. Dessa forma, a herança garante que existam os atributos *Id*, *CreateTime* e *UpdateTime* nessas classes.

Em termos de modelagem ER, não há necessidade das classes *ClubUser* e *SpaceUser* herdar de *BaseEntity* por conta de representarem um relacionamento e não precisarem de um ID, já que podem utilizar uma chave primária composta. Essa decisão deu-se por uma maior simplicidade para utilizar a biblioteca de *object–relational mapping* (ORM) escolhido.

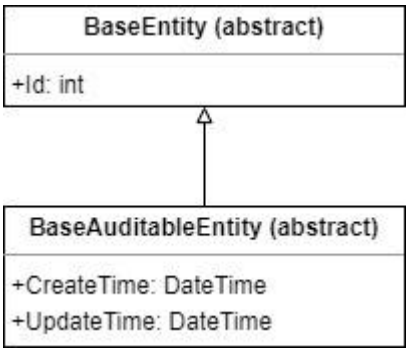
O diagrama de classes contendo todas as classes que definem os modelos de dados definidos para o MVP é apresentado na Figura 11.

Figura 11. Diagrama de classes para modelos de dados do MVP.




Fonte: elaborada pelo autor (2022).

Figura 12. Diagrama de classes com *BaseEntity* e *BaseAuditableEntity*.




Fonte: elaborada pelo autor (2022).

### 4.5.1 Entidades

Aqui são definidas todas as entidades apresentadas nos diagramas anteriores.

#### 4.5.1.1 User

Entidade que define o usuário do sistema. Possui um atributo chamado *IdentityProviderUserId* como identificador, além do próprio *ID*. A diferença entre os dois atributos é que o *ID* é utilizado para relações internas do banco de dados e o outro para fazer referência ao registro do usuário que estará em um provedor de identidade, como o AWS Cognito, que foi escolhido para este trabalho. Dessa forma, as credenciais do usuário não precisam ficar dentro do banco de dados da aplicação, ficando apenas no banco de dados do provedor de identidade. Essa escolha é discutida posteriormente neste trabalho.

#### 4.5.1.2 UserSettings

Entidade que define as configurações do usuário. No MVP, é utilizada para armazenar dados relacionados à internacionalização e localização do software. É possível que em uma evolução do software, o contexto dessa entidade seja expandido. Seus atributos contém os dados que possibilitam a aplicação definir o idioma, fuso-horário, formatos de data, horário e moeda.

#### 4.5.1.3 DominantSide

Entidade que define o lado dominante do jogador. Em geral, o lado dominante é uma característica importante para jogadores de tênis. Como só existem três

opções, sendo elas pessoa destra, canhota ou ambidestra, optou-se por torná-la uma enumeração.

#### 4.5.1.4 Club

Entidade que define o clube de tênis para o MVP. Inicialmente, esta entidade é extremamente simples, contendo apenas os dados derivados de sua hierarquia (*Id*, *CreateTime* e *UpdateTime*) e o nome do clube.

#### 4.5.1.5 ClubUser

Entidade que define um relacionamento entre o clube e um usuário. Cada relacionamento estará vinculado a um papel do usuário desempenhado no clube através da entidade *ClubRole*.

#### 4.5.1.6 ClubRole

Entidade que define um papel que um usuário pode desempenhar dentro de um clube. Atualmente, foram definidos três papéis possíveis, conforme mostrado no Quadro 9.

Quadro 9. Definições dos papéis em um clube.

| **Papel** | **Descrição** |
|---|---|
| Owner | Define uma pessoa dona de um clube. |
| Professional Affiliate | Define uma pessoa que pode exercer atividades profissionais nas dependências do clube em que é afiliada. |
| Simple Affiliate | Define uma pessoa afiliada ao clube. |

Fonte: elaborado pelo autor (2022).

#### 4.5.1.7 Space

Entidade que define um espaço físico do clube. Nela são definidas as limitações utilizadas para registros de usos, como número máximo de usos simultâneos, tempo de uso mínimo, horário de abertura e horário de fechamento. O número máximo de usos simultâneos serve como bloqueio para evitar que um determinado espaço seja agendado para o mesmo horário mais vezes do que o determinado por quem gere o clube. Entretanto, o software não considera nenhum

limitante de pessoas por espaço e isso ficou definido como sugestão de trabalho futuro.

#### 4.5.1.8 SpaceUse

Entidade que define o uso de um espaço. Um uso com data futura é tratado como um agendamento e com data passada, é tratado como histórico. Um uso não pode ser excluído, mas pode ser editado e marcado como cancelado, por exemplo. O intuito dessa limitação é permitir que usuários, principalmente gestores, compreendam como está a taxa de cancelamento de agendamentos. Uma vez que o uso do espaço está com uma data passada, é possível confirmar que o espaço foi utilizado por quem criou o evento ou gestores do clube. Possui três dependências com a classe *User* por conta dos atributos *CreatorUserId*, *CancellerUserId* e *ConfirmerUserId*.

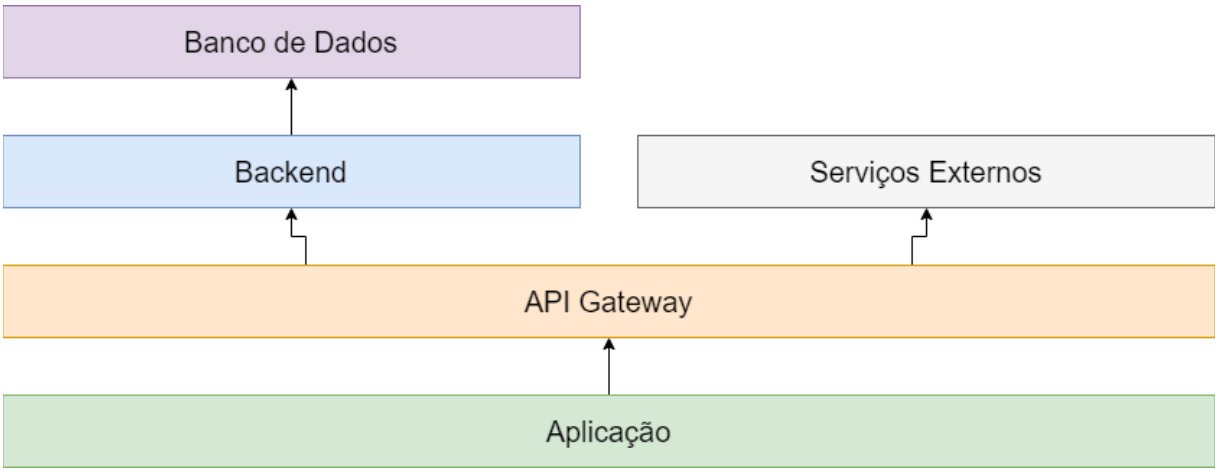
#### 4.5.1.9 SpaceUser

Entidade que define um relacionamento entre o uso de um espaço e um usuário. Todo usuário marcado para participar de um determinado uso pode confirmar essa participação, onde o software mapeia para o atributo *ConfirmTime*.

## 4.6 ARQUITETURA E IMPLEMENTAÇÃO

A arquitetura inicialmente proposta para o sistema é dividida nas camadas de aplicação, API Gateway, backend e serviços externos, e banco de dados, como pode ser observado na Figura 13. A camada “Aplicação” é composta por todas as aplicações para mobile, desktop e web. A camada “API Gateway” define um intermediário entre as aplicações e o backend ou serviços externos. Sempre que uma aplicação necessitar de dados do banco de dados ou algum serviço externo, precisará comunicar-se com a camada “API Gateway”. A camada “Backend” define uma interface de comunicação com o banco de dados do sistema. A camada “Serviços Externos” não será desenvolvida neste trabalho, pois ela contempla qualquer serviço externo a ser consultado, como previsão de clima, por exemplo. Por fim, a camada “Banco de Dados” contém todos os bancos de dados acessíveis pelo backend.

Figura 13. Camadas da arquitetura.




Fonte: elaborada pelo autor (2022).

### 4.6.1 Composição de API

O produto de software apresentado neste trabalho foi idealizado de forma que serviços internos, ou de terceiros, possam ser chamados para fornecer novos recursos para o sistema. Quando o usuário abre uma determinada tela, a aplicação precisa carregar as informações para apresentá-las ao usuário e é possível que essas informações venham de fontes diferentes, podendo ser necessário consultar diferentes serviços. Para realizar a composição de dados vindos de diferentes serviços para uma determinada tela, cada um dos serviços precisaria ser chamado pela própria aplicação.

Como neste trabalho foi utilizada a AWS como provedor de serviços na nuvem e seus servidores são mantidos nos Estados Unidos, é necessário lidar com chamadas que levam centenas de milissegundos entre a requisição e a resposta. Nesse caso, a latência pode acabar sendo um problema para múltiplas chamadas ao servidor porque o somatório dos tempos de resposta pode tornar-se desconfortável para os usuários.

Supondo-se uma tela que carrega informações de reservas de quadras de tênis e previsão do tempo, sendo que os dados das reservas de quadras são solicitados por um serviço e os dados de previsão do tempo por outro, pode-se começar a observar os primeiros problemas de performance na aplicação. Dentro dessa suposição, estamos considerando que cada serviço conta com sua própria estrutura de API e banco de dados, e ambos estão na mesma estrutura de nuvem. A realização de duas chamadas, sendo uma para cada serviço, para compor os dados

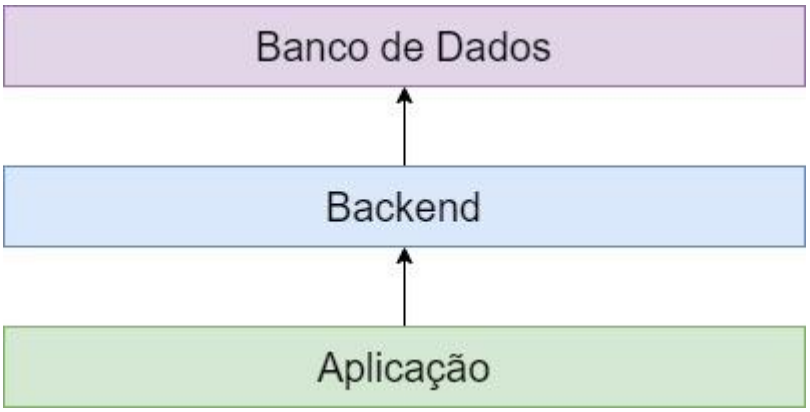
desta tela precisa lidar com a latência de cada chamada. Na prática, essa latência também é mais tempo do usuário esperando a tela carregar. Quanto mais serviços sendo acionados, mais vezes a latência ocorre e maior o tempo de espera.

Para evitar os problemas relacionados à latência devido à consulta em vários serviços, optou-se por introduzir uma camada de API Gateway na arquitetura do sistema. Essa escolha baseou-se em uma abordagem de separação de responsabilidades, performance e extensibilidade. Decidiu-se atribuir a responsabilidade de conhecer os serviços específicos utilizados pelo sistema ao API Gateway, mantendo a camada de aplicação mais responsável pela apresentação das informações ao usuário e servir como interface para interação dele com o sistema. Dessa forma, as aplicações não precisarão chamar múltiplos serviços para compor os dados para apresentar ao usuário porque a camada de API Gateway faz as várias chamadas necessárias, compõe a resposta e devolve para a aplicação. Isso traz um ganho de performance porque será necessário lidar apenas com a latência de uma única chamada ao servidor. Por fim, também é desejado que a arquitetura seja extensível. Considerando que existe a possibilidade de migrar a solução proposta aqui para uma arquitetura de microsserviços no futuro, ou seja, potencialmente aumentar o número de requisições a serviços diferentes, a aplicação de um API Gateway foi considerada adequada pelo autor, apesar de não estar no escopo atual do projeto.

Existem outras estratégias para lidar com problemas relacionados à latência, como o CDN e o padrão CQRS, por exemplo. Essas soluções não fazem parte da proposta inicial de solução, apesar de que existe muito espaço para introduzi-las no futuro.

Para reduzir o escopo do MVP, por limitações de tempo e equipe, optou-se por não implementar a camada de API Gateway neste trabalho, ficando como sugestão de trabalho futuro. Como os dispositivos móveis trabalham com o modelo *offline-first*, não precisam fazer múltiplas chamadas para o servidor para carregar a visualização das páginas, uma vez que possuem uma versão local dos dados do servidor para um determinado usuário. Dessa forma, a utilização de um API Gateway somente para esta aplicação não teria grandes vantagens. Como o escopo também foi reduzido, optando-se por não implementar a aplicação web, a não utilização do API Gateway encaixou bem com as propostas de redução de escopo. A versão final de arquitetura produzida neste trabalho está representada na Figura 14.

Figura 14. Versão final da arquitetura do sistema.




Fonte: elaborada pelo autor (2022).

### 4.6.2 Offline-first

Pela experiência do autor, deve-se considerar que existe a possibilidade de não haver conexão com a internet nos ambientes de treino de tênis. Dessa forma, buscou-se por opções de como fazer com que as aplicações continuassem funcionando em caso de problemas com o servidor ou falta de conexão com a internet. Para solucionar isso, optou-se por utilizar uma abordagem *offline-first* nas aplicações para smartphones. Também será feita a referência para essa abordagem como “arquitetura offline”.

Além de resolver o problema da falta de conexão com a internet, essa abordagem também já resolve parte do problema relacionado a uma degradação do servidor, pois garante que os dispositivos terão uma versão local dos dados do servidor para um determinado usuário. Somente quando houver conexão com a internet e o servidor estiver funcionando corretamente, as operações feitas nos dispositivos serão sincronizadas.

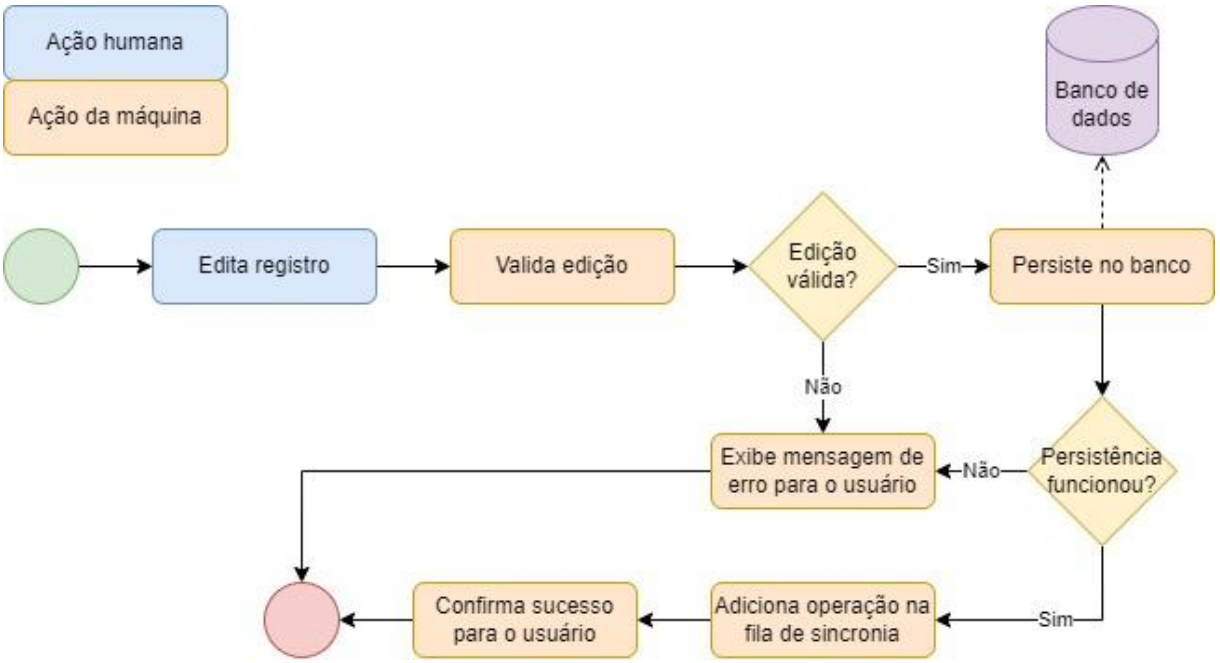
Poderia-se optar pela utilização de um MOM para mitigar o risco de perda de funcionamento das aplicações por conta de alguma degradação do lado do servidor, por exemplo. O problema desse tipo de abordagem é o custo de implantação e de adesão de novos componentes de infraestrutura. Por conta disso, optou-se por não utilizar essa abordagem na implementação do MVP do sistema proposto neste trabalho. Outro ponto positivo do MOM é que múltiplas aplicações podem consumir a mesma informação da fila de mensagens para usar de formas diferentes, sem a necessidade de realizar várias solicitações para a aplicação que enviou a

informação. O aprofundamento dessa análise fica como sugestão de próximos passos.

De maneira geral, a arquitetura offline também oferece vantagens como redução de custos de *roaming* durante viagens, minimização do uso de dados em mapas, economia de energia da bateria e tempo de carregamento rápido (OFFLINE, 2022). Isso porque a necessidade de utilizar a rede fica reduzida somente aos momentos de sincronizar dados.

Basicamente, o funcionamento dessa arquitetura consiste em executar operações, validações e persistências de dados em cada dispositivo onde o software estiver instalado, sem depender de um servidor para isso. Essas operações são armazenadas em alguma fila para que, em um momento determinado, possam ser feitas as chamadas para o servidor para executar cada uma das operações que haviam sido feitas sem conexão com a internet. Uma ilustração desse funcionamento, simulando o caso em que um usuário edita um registro, é apresentada na Figura 15, onde todo o fluxo ocorre sem comunicação com o servidor.

Figura 15. Fluxograma da arquitetura offline em edição de registro.




Fonte: elaborada pelo autor (2022).

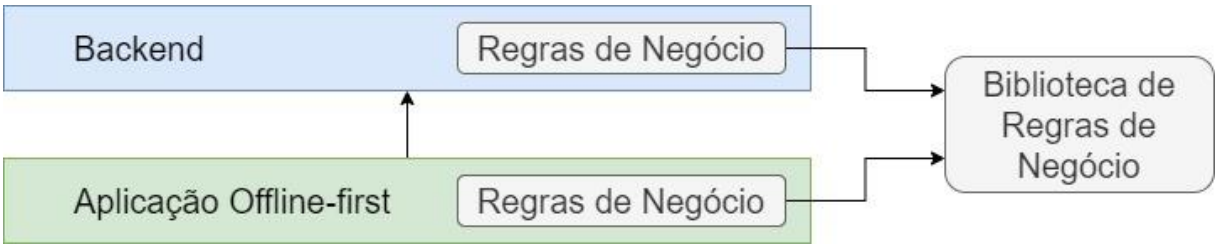
Uma das desvantagens dessa arquitetura é que existe a necessidade de lidar com conflitos de regras de negócio, já que todos os dispositivos podem operar sobre os mesmos registros de forma independente e sem uma validação central do

servidor. Se dois usuários reservam um espaço do clube para o mesmo horário em locais onde ambos estão sem conexão com a internet, fica a dúvida de quem teria direito à reserva no momento em que os dados forem para o servidor.

Para solucionar esse problema, adotou-se a abordagem de que a solicitação que chegar primeiro no servidor é a que será válida. As outras serão descartadas e o usuário que fez a solicitação de reserva será notificado pela aplicação. É importante mencionar que trabalhar com esse tipo de abordagem exige um maior cuidado em termos de experiência de usuário. Uma vez que o usuário executou o comando, é comum pensar que aquilo está válido no sistema. Se não houver conexão com a internet, é importante que a aplicação avise o usuário que aquela ação só terá validade quando os dados forem sincronizados com o servidor.

Outra das consequências do *offline-first* é que as aplicações precisam conhecer as regras de negócio para funcionar sem conexão com a internet, o que pode entrar em conflito com a separação de responsabilidades desejada na arquitetura proposta. Para solucionar esse problema, foi construída uma biblioteca com as regras de negócio do sistema, podendo ser utilizada tanto no backend quanto nas aplicações de frontend, sem necessidade de duplicação do código. Um exemplo ilustrativo pode ser visto na Figura 16.

Figura 16. Biblioteca de regras de negócio.



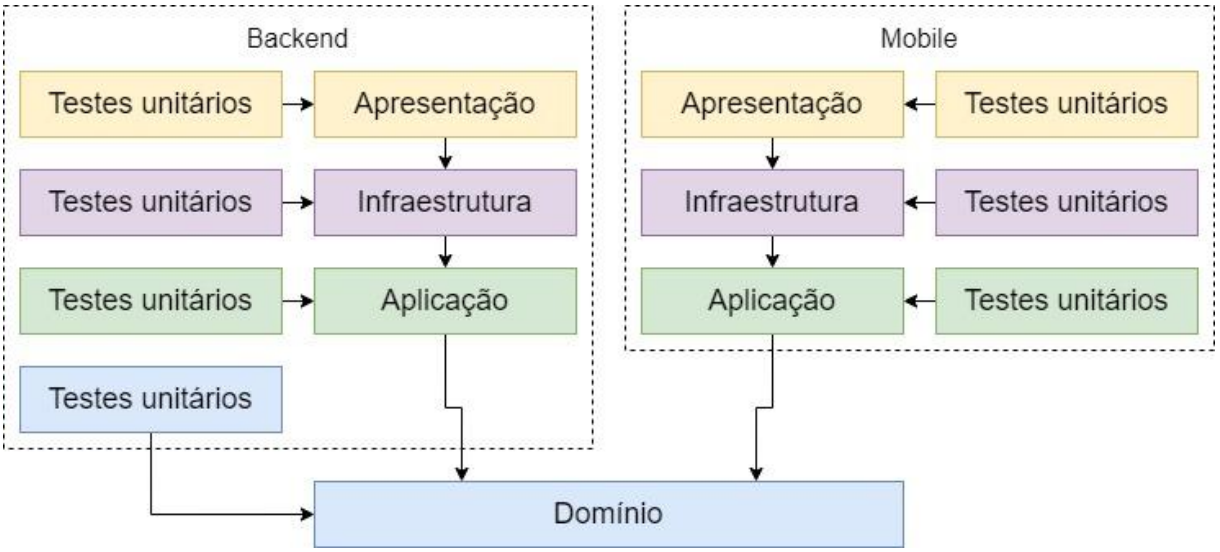

Fonte: elaborada pelo autor (2022).

### 4.6.3 Aplicação da Arquitetura Limpa

Para as aplicações, optou-se por implementar o conceito de design de software chamado de arquitetura limpa ou *clean architecture* (MARTIN, 2018). Como apresentado na Figura 17, as camadas foram definidas como apresentação, infraestrutura, aplicação e domínio. Esta última é compartilhada entre todas as aplicações do sistema, pois contém artefatos comuns a todas elas, facilitando a reusabilidade. Para cada uma das camadas, há uma camada equivalente de testes

unitários que testam exclusivamente a respectiva camada. A definição dos testes unitários para a camada de domínio estão no backend, pois é onde o código dessa camada é mantido. No mobile, é consumido como um pacote. Um exemplo de como ficaram organizadas as pastas é apresentado na Figura 18. A estrutura de pastas para os projetos foi fortemente baseada em (TAYLOR, 2022).

Figura 17. Camadas da arquitetura das aplicações.




Fonte: elaborada pelo autor (2022).

### 4.6.4 Monitoramento e Métricas

Para facilitar a tomada de decisão nas etapas de evolução do software, é importante ter conhecimento de como o usuário se comporta com o software e como o software se comporta com o usuário. Com o objetivo de aumentar esse conhecimento, algumas ferramentas são utilizadas para monitorar o uso do software e centralizar esses dados. Para a aplicação de backend, utilizou-se o Sentry como principal ferramenta de monitoramento de erros. Para a aplicação mobile, utilizou-se o Sentry e o App Center.

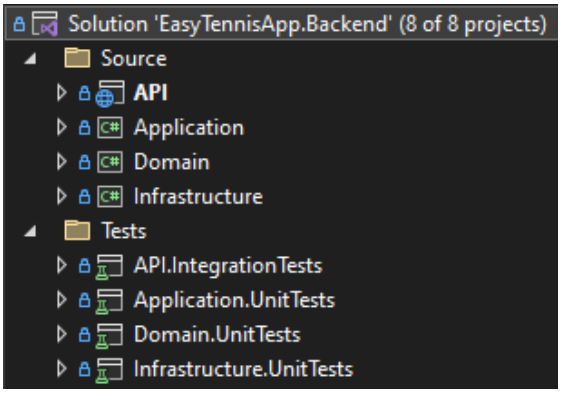
### 4.6.5 Backend

O backend foi implementado utilizando a linguagem de programação C# com a plataforma .NET 6. Utilizou-se o modelo de arquitetura em camadas *Clean Architecture*, separando os escopos de apresentação, aplicação, infraestrutura e domínio. Cada uma das camadas é representada por um projeto .NET independente dentro da solução criada. A estrutura de pastas é apresentada na Figura 18.

#### 4.6.5.1 Camadas da Arquitetura

A camada de domínio é comum a todas as aplicações do sistema e poderia ser mantida por qualquer uma das aplicações. Entretanto, o backend foi a aplicação escolhida como responsável pela manutenção e evolução dessa camada. O motivo é que, como o backend faz o contato mais próximo do banco de dados, o autor do trabalho julgou mais prático e de senso comum ser o primeiro local onde criam-se as classes que definem os modelos de dados do sistema. Dessa forma, para que uma nova funcionalidade seja criada, as classes de domínio devem ser criadas no backend. Uma vez que estiverem funcionando corretamente, um pacote NuGet pode ser exportado do projeto para que possa ser adicionado em outras aplicações.

Figura 18. Estrutura de pastas do backend.




Fonte: elaborada pelo autor (2022).

A camada definida como API (Figura 18) é a camada de apresentação da *clean architecture*. Nela foram implementados os *controllers*, responsáveis por receberem as requisições e responderem à aplicação cliente. Um exemplo para lidar com requisições de dados de usuário é mostrado abaixo:

```
[Authorize]
[HttpGet("{id}")]
public async Task<ActionResult<UserDto>> GetById(int id)
{
    UserDto dto = await Mediator.Send(new GetUserByIdQuery(id));
    return Ok(dto);
}
```

O padrão de projeto *Command Query Responsibility Segregation* (CQRS) foi utilizado no backend. Para que esse padrão seja adequadamente empregado dentro do contexto dessa aplicação, também foi feito uso dos padrões de projeto *mediator* e

*command*. Além disso, a injeção de dependência foi um recurso bastante utilizado para facilitar o baixo acoplamento entre os módulos do software desenvolvido.

No exemplo de código mostrado acima, a classe *GetUserByIdQuery* define um *command* e o atributo *Mediator* é uma instância da interface *ISender*, que envia uma requisição para as classes manipuladoras (*handlers*). *Handlers* são classes que realizam a execução do comando em si, chamando a consulta ao banco de dados e retornando o valor, por exemplo. Para cada *command*, foi implementado um *handler* específico, mantido dentro da camada *Application*. Um exemplo de *handler* para o mesmo código de lidar com requisições de dados de usuário é mostrado abaixo:

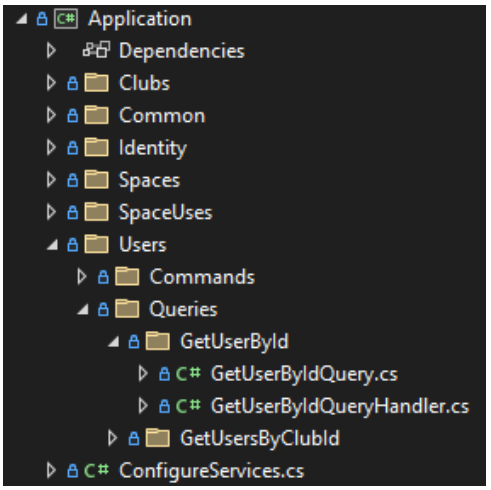
```
internal class GetUserByIdQueryHandler : IRequestHandler<GetUserByIdQuery,
UserDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetUserByIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<UserDto> Handle(GetUserByIdQuery request,
CancellationToken cancellationToken)
    {
        User? user = await context.Users
            .Include(user => user.LinkedClubs)
            .Include(user => user.Settings)
            .SingleOrDefaultAsync(user => user.Id.Equals(request.Id),
cancellationToken);
        return mapper.Map<UserDto>(user);
    }
}
```

Pode-se observar a injeção de dependências ocorrendo no construtor da classe, onde são injetadas as instâncias de *IApplicationDbContext* e *IMapper*, responsáveis por acessar o banco de dados e mapear os modelos de dados para instâncias de *data transfer objects* (DTO's), respectivamente. O método *Handle* executa exatamente o que é esperado que aquele comando execute: buscar os dados do usuário no banco de dados e retornar um DTO para o controlador poder retornar para o cliente. A classe foi declarada como *internal* ao invés de *public* porque isso a define como acessível publicamente somente para a camada em que

ela foi definida. Dessa forma, somente classes internas da camada *Application* podem acessá-la.

A estrutura de pastas da camada *Application*, contendo todos os *handlers* e *commands* ficou como apresentada na Figura 19. Para cada entidade principal definida no modelo de dados, são criadas pastas que vão conter os *commands*, *queries* e *handlers*. Por isso, é possível observar que foram criadas as pastas *Users*, *Clubs*, *Spaces* e *SpaceUses*. A pasta *Common* contém algumas classes comuns a outras classes dessa mesma camada e a pasta *Identity* contém alguns DTO's relativos às chamadas de autenticação e autorização.

Figura 19. Estrutura de pastas da camada Application do backend.




Fonte: elaborada pelo autor (2022).

A camada de domínio, mostrada como *Domain* na Figura 18, contém a definição das classes que representam os modelos de dados, DTO's e outras características comuns ao domínio do sistema, como enumerações e validações, por exemplo.

A camada *Infrastructure* contém as classes que implementam características específicas de infraestrutura da aplicação do backend, como acesso ao banco de dados e integração com provedor de identidade. As outras camadas da aplicação não devem conhecer qualquer particularidade da infraestrutura. No caso do backend, o banco de dados é MySQL e o provedor de identidades é o AWS Cognito.

#### 4.6.5.2 Autenticação e Autorização

A autenticação no EasyTennis foi implementada utilizando *JSON Web Token* (JWT) e o serviço AWS Cognito como provedor de identidade. Para que qualquer

aplicação autentique um usuário, é necessário que esse usuário forneça suas credenciais de acesso: e-mail e senha. Essas credenciais são enviadas ao Cognito para verificação de validade. Os motivos que levaram à decisão de utilizar um provedor de identidade ao invés de armazenar as credenciais de acesso dos usuários no banco de dados estão listados a seguir:

- Manter o banco de dados separado do banco com as credenciais evita que um acesso indevido a um destes bancos comprometa os dados do outro;
- A segurança do serviço contratado fica a cargo da AWS, bastando que o autor seja responsável por garantir a segurança da chave de API utilizada para acessar o Cognito via backend e das credenciais de acesso ao console da AWS;
- Mesmo que ocorresse um vazamento da chave de API, ainda não seria possível visualizar todo o banco de dados de usuários sem saber previamente os e-mails ou ID's de cada um por conta das restrições dessa chave de API;
- A implementação do fluxo de autenticação e autorização fica mais simples de fazer porque esses recursos já estão prontos no Cognito, bastando configurar. Como exemplo desses recursos, temos *multi-factor authentication* (MFA), confirmação de e-mail, política de senha e integração com logins de redes sociais. Inclusive, é possível utilizar as telas de login do próprio Cognito.

Para garantir um baixo acoplamento, criou-se a interface *IIdentityService*, que é referenciada como o serviço de identidades adotado, independente de qual seja. Para utilizar o AWS Cognito como provedor, a implementação dessa interface foi através da classe *AmazonCognitoIdentityService*. Para ilustrar a simplicidade da implementação, é apresentado o trecho de código que corresponde ao cadastro do usuário no sistema a seguir, passando pelo *controller*, *handler* e *service*.

Inicialmente, temos o *controller*, que recebe a requisição de criação de usuário:

```
[Authorize]
[HttpPost]
public async Task<ActionResult<int>> Create([FromBody] CreateUserCommand
command)
{
    return await Mediator.Send(command);
}
```

A chamada chega até o *handler*, que solicita à instância de *IIdentityService* que o usuário seja criado com as credenciais informadas. O retorno recebido é o UUID do usuário criado. Esse valor é salvo no banco de dados da aplicação para que exista a conexão com o registro criado dentro do Cognito. Após isso, o usuário é criado no banco de dados do EasyTennis. O trecho de código do *handler* é exibido abaixo:

```
public async Task<int> Handle(CreateUserCommand request, CancellationToken
cancellationToken)
{
    UserCredentialsDto credentials = new()
    {
        Email = request.Email,
        Password = request.Password
    };

    string identityProviderUserId = await
identityService.CreateUserAsync(credentials);

    User user = new()
    {
        Name = request.Name,
        Email = request.Email,
        IdentityProviderUserId = identityProviderUserId,
        LinkedClubs = new List<ClubUser>()
        {
            new ClubUser()
            {
                RoleId = request.RoleId,
                ClubId = request.ClubId
            }
        },
        Settings = new UserSettings()
    };

    await context.Users.AddAsync(user, cancellationToken);
    await context.SaveChangesAsync(cancellationToken);

    return user.Id;
}
```

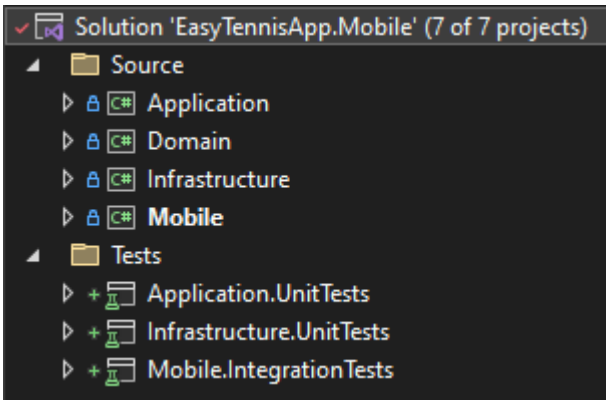
A interface *IIdentityService* define somente dois métodos: *CreateUserAsync* e *TryLoginAsync*. Conforme mostrado acima, o método *CreateUserAsync* é chamado para realizar a criação do usuário. Para tal, é instanciado um objeto do tipo *SignUpRequest*, que é enviado pelo SDK do Cognito. Abaixo é mostrado o trecho de código com essa lógica:

```
public async Task<string> CreateUserAsync(UserCredentialsDto dto)
{
    SignUpRequest request = new()
    {
        ClientId = configuration.AppClientId,
        Username = dto.Email,
        Password = dto.Password,
        UserAttributes = new List<AttributeType>()
        {
            new AttributeType()
            {
                Name = "email",
                Value = dto.Email
            }
        }
    };

    SignUpResponse response = await provider.SignUpAsync(request);

    return response.UserSub;
}
```

### 4.6.6 Aplicativo para Android

O aplicativo para smartphone foi implementado utilizando a linguagem de programação C# com a plataforma .NET MAUI. Da mesma forma que no backend, utilizou-se o modelo de arquitetura em camadas Clean Architecture, separando os escopos de apresentação, aplicação, infraestrutura e domínio. Cada uma das camadas é representada por um projeto .NET independente dentro da solução criada. A estrutura de pastas é apresentada na Figura 20.

Figura 20. Estrutura de pastas do aplicativo para smartphone.




Fonte: elaborada pelo autor (2022).

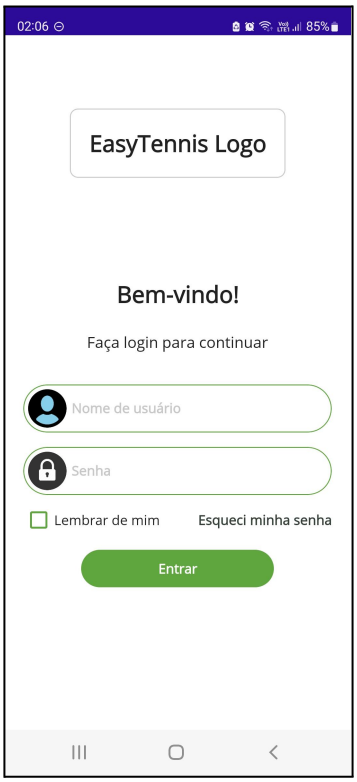
Buscou-se implementar um layout moderno e simples para o aplicativo EasyTennis, no qual o usuário pode encontrar os principais recursos da aplicação com poucos toques. O critério de quantidade de toques não foi aprofundado neste trabalho. O aplicativo foi projetado considerando a necessidade de, no máximo, 2 toques para acessar cada funcionalidade principal, uma vez que o aplicativo está aberto e com o usuário autenticado. O aplicativo é internacionalizado e possui suporte às culturas Português (Brasil) e Inglês (Estados Unidos).

Dadas as limitações de tempo e equipe, o desenvolvimento ficou limitado a quatro principais telas: login, início, minhas reservas e configurações. Todas essas telas são descritas a seguir. Os usuários de nome "John Connor" e "T-800" são fictícios.

#### 4.6.6.1 Login

A tela do login, mostrada na Figura 21, permite que um usuário já registrado no sistema possa se autenticar mediante fornecimento de e-mail e senha. O cadastro de usuários diretamente pelo aplicativo não foi projetado ou implementado.

Figura 21. Captura de tela do aplicativo EasyTennis: tela de login.




Fonte: elaborada pelo autor (2022).

#### 4.6.6.2 Início

A tela de início, mostrada na Figura 22, é a tela visualizada pelo usuário logo após ele se autenticar ou ao acessar o aplicativo já estando autenticado previamente. Nesta tela, são exibidas as principais notificações e avisos para o usuário. Os avisos de cadastro incompleto e de convites pendentes foram os únicos implementados neste trabalho, porém outras notificações de diversas naturezas podem ser implementados.

O aviso de cadastro incompleto tem o objetivo de incentivar o usuário a inserir mais informações no seu perfil para que a base de dados do sistema fique mais rica. Um exemplo disso é o registro de uma imagem de perfil, para facilitar o reconhecimento entre as pessoas usuárias do sistema.

O aviso de convites pendentes se trata das reservas de espaços nas quais a pessoa ainda não confirmou sua presença. No exemplo da Figura 22, o usuário "John Connor" foi convidado para uma reserva de espaço pelo usuário "T-800" (terceira reserva da Figura 23). Como ele ainda não confirmou sua presença, o convite fica como pendente.

Figura 22. Captura de tela do aplicativo EasyTennis: tela de início.


Fonte: elaborada pelo autor (2022).

#### 4.6.6.3 Minhas Reservas

A tela "Minhas Reservas", mostrada na Figura 23, permite a visualização das reservas do dia corrente em diante. As reservas são agrupadas por datas e, para cada uma delas, são mostradas as informações de nome do espaço, responsável pela reserva, horário de início e horário de final. O botão com símbolo "+" foi introduzido para levar o usuário para o formulário de criação de reserva, mas a funcionalidade não foi implementada.

Figura 23. Captura de tela do aplicativo EasyTennis: tela de minhas reservas.


Fonte: elaborada pelo autor (2022).

#### 4.6.6.4 Configurações

A tela de configurações, mostrada na Figura 24, permite ao usuário configurar o idioma, junto às definições de cultura de um país, e o fuso-horário. Essas opções são importantes para aumentar o alcance e melhorar a usabilidade do aplicativo. A decisão de implementar esse recurso como uma das primeiras funcionalidades se deu porque, em geral, é mais barato construir o suporte à internacionalização desde o início do desenvolvimento de uma aplicação (BREAKING, 2022).

Figura 24. Captura de tela do aplicativo EasyTennis: tela de configurações.

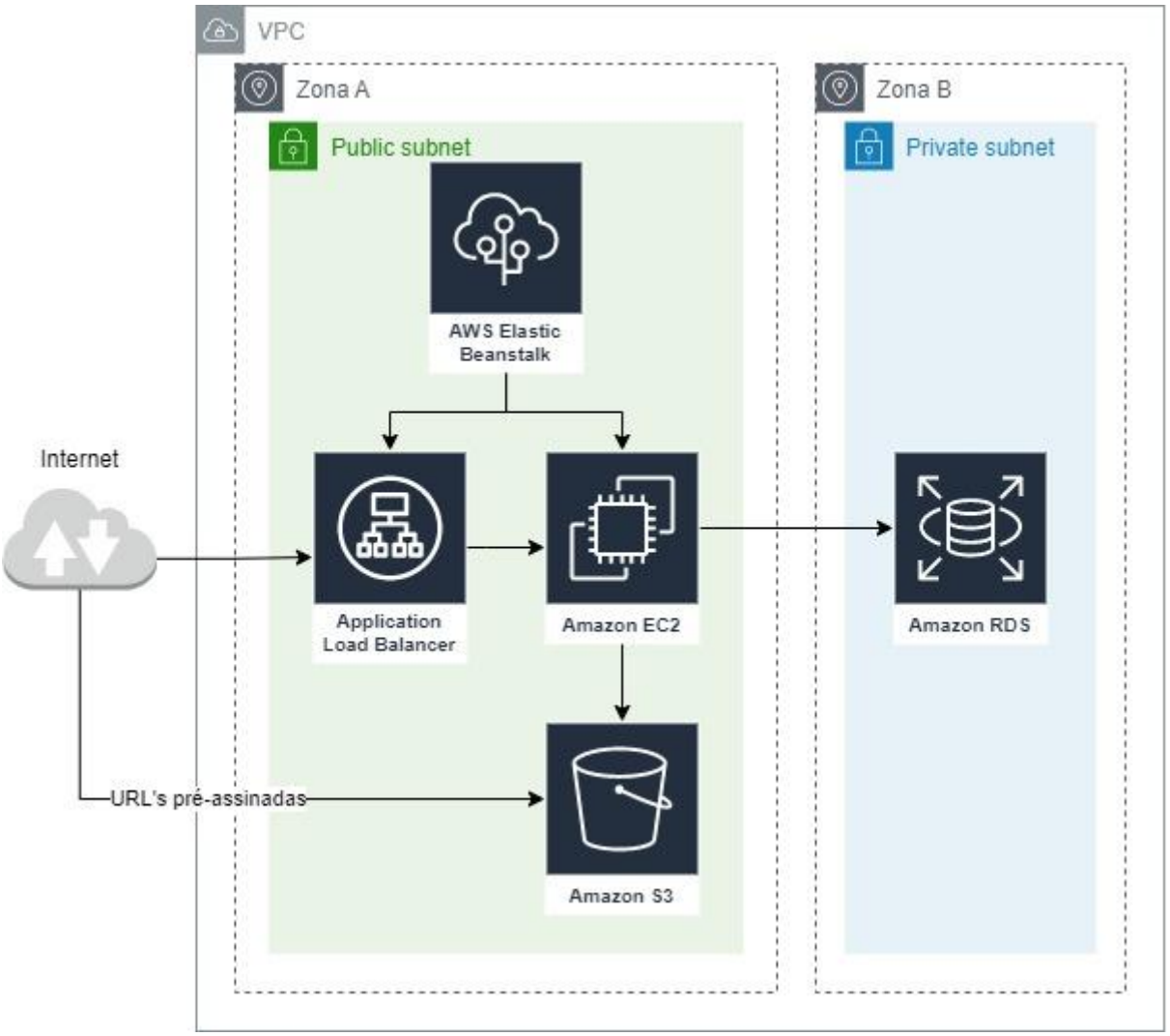

Fonte: elaborada pelo autor (2022).

## 4.7 INFRAESTRUTURA

A infraestrutura do sistema foi implantada em nuvem, utilizando a plataforma AWS, e é composta principalmente por banco de dados, aplicação de backend e repositório de arquivos. A Figura 25 apresenta o diagrama da infraestrutura com os serviços que foram utilizados, estando todos eles na camada gratuita da AWS.

Foi definida uma *virtual private cloud* (VPC) com duas zonas de disponibilidade para implantação dos recursos de infraestrutura. A zona A contém uma sub-rede pública, sendo acessível de fora da VPC somente por meio do *application load balancer* (ALB) ou repositório de arquivos S3 com URL pré-assinada. A zona B contém uma sub-rede privada contendo somente o banco de dados da aplicação. A única forma de acessá-lo é através de aplicação de backend.

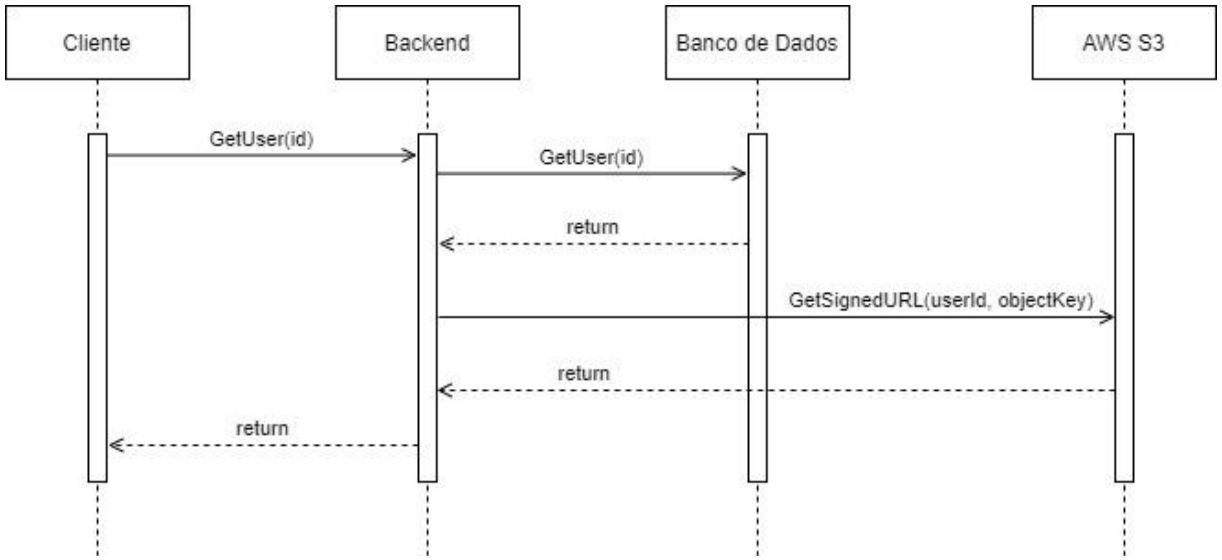
Figura 25. Diagrama de infraestrutura.




Fonte: elaborada pelo autor (2022).

O ALB faz o papel de interface com a internet. Toda aplicação cliente se comunica diretamente com ele, evitando a necessidade de expor a máquina EC2 com a aplicação de backend para a internet. O único caso em que uma aplicação cliente se comunica com outro recurso, além do ALB, é quando precisa ler uma determinada imagem do repositório de arquivos S3. Isso é feito através de uma URL pré-assinada, gerada com as credenciais do usuário, garantindo acesso por tempo limitado para download do recurso (SHARING, 2022). A Figura 26 mostra um diagrama de sequência exemplificando o momento em que ocorre a geração da URL pré-assinada.

Figura 26. Diagrama de sequência para geração de URL pré-assinada.




Fonte: elaborada pelo autor (2022).

Quando uma aplicação cliente solicita o registro de um usuário para o backend, o registro é buscado no banco de dados e o atributo *ImageKey* vem com o valor da chave do objeto no repositório S3. As credenciais e *role* do usuário são utilizadas para gerar a URL de acesso de leitura para a imagem. Esta é retornada dentro da resposta para a aplicação cliente no campo *ImageSignedUrl*. Essa URL tem um tempo de expiração, limitando a aplicação cliente a fazer o acesso dentro desse tempo para baixar a imagem. Para evitar que esse processo seja realizado com frequência, as aplicações poderiam utilizar a estratégia de cache para imagens, mas isso não foi implementado.

O serviço Elastic Beanstalk é utilizado para gerenciar as máquinas EC2 e o ALB de forma mais simples. Como o autor do trabalho não tem grande domínio sobre recursos de infraestrutura, optou-se por utilizar essa plataforma gerenciada pela própria AWS. Dessa forma, responsabilidades como atualização de sistema operacional, implantação de atualizações da aplicação, gerenciamento de variáveis de ambiente, ciclo de vida das máquinas virtuais e outras ficam a cargo do serviço contratado.

Por questões de simplicidade para o desenvolvimento do trabalho, algumas brechas de segurança foram aceitas, como o acesso direto ao banco de dados por meio de software gerenciador de banco de dados, como o MySQL Workbench (MYSQL WORKBENCH, 2022), por exemplo. O diagrama da Figura 25 está representando o modelo que seria colocado em produção para o MVP.

## 4.8 PROCESSO DE DESENVOLVIMENTO

O processo de desenvolvimento do software foi feito utilizando o Git Flow como base para integração do código ao repositório. Os repositórios ficam armazenados no GitHub. Sempre que um *pull request* é criado, utilizou-se o serviço GitHub Actions para disparar a execução de análise estática de código, bloqueando o *merge* no caso de alguma inconsistência ser encontrada. Além da análise estática de código, são executados os testes automatizados da aplicação para garantir que *branches* só sejam unidas ao código principal sem que haja regressão de padrões de estilos de código e qualidade coberta pelos testes.

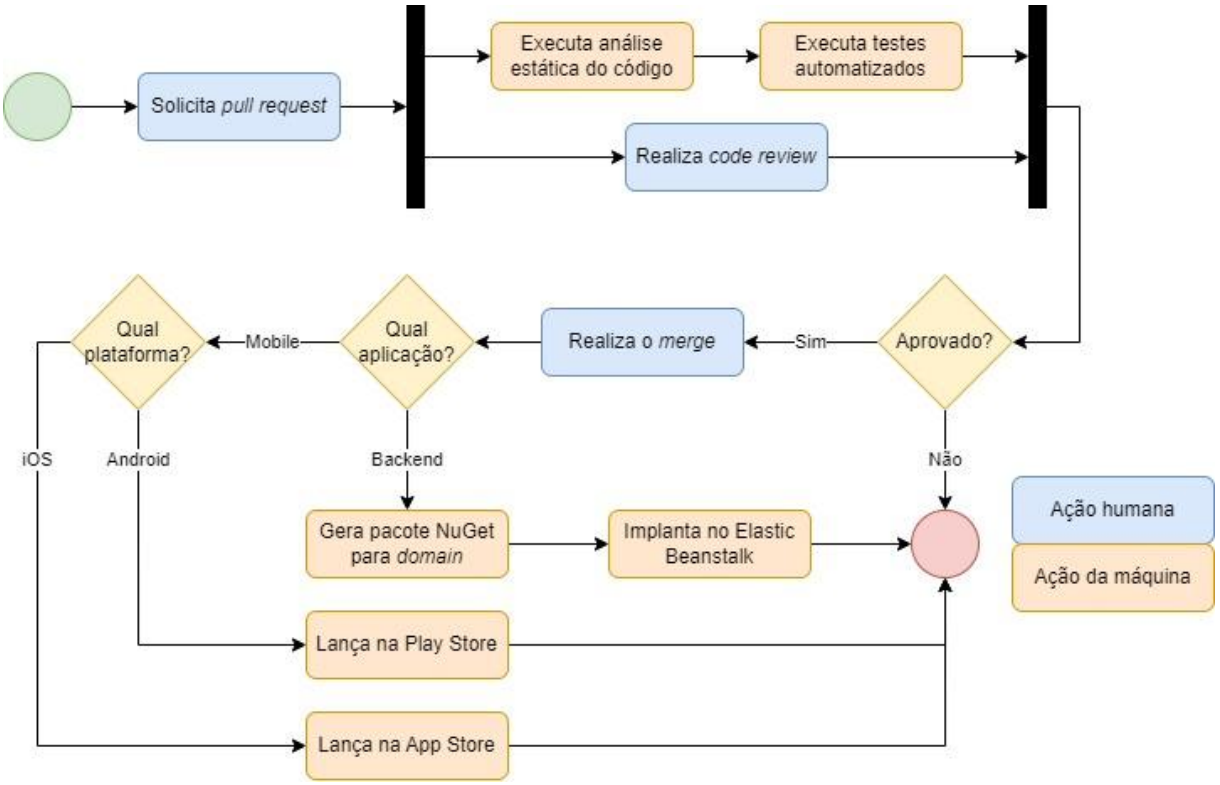
Para o caso da aplicação de backend, após a realização do *merge* com a *branch* de desenvolvimento, a aplicação é implantada em ambiente de homologação. Como não foi realizada a implantação de um ambiente de produção, não foi criada a automação para essa implantação para o *merge* com a *branch* principal.

Para o caso da aplicação *mobile*, o processo de deploy em ambiente de homologação não seria feito pelo GitHub Actions, mas pelo App Center. Essa parte não foi implementada, pois a publicação em lojas de aplicativos exige um trabalho burocrático e custos que estão fora do escopo deste trabalho. Para instalar a aplicação em diversos dispositivos, utilizou-se a geração de uma APK gerada diretamente com a IDE Visual Studio e SDK Android.

O diagrama da Figura 27 ilustra o processo de implantação de código em ambiente de homologação. Este é um diagrama com propósito didático, pois as aplicações estão em repositórios separados e nos fluxos implementados não há uma bifurcação de “Qual aplicação?”, como mostrado na imagem, por exemplo.

Para que as aplicações possam utilizar as mesmas classes para os modelos de dados sem a necessidade de reescrevê-las, definiu-se que um pacote NuGet é exportado a partir do código da aplicação do backend para ser utilizado na aplicação mobile. Para isso, foi adicionada a criação do pacote dentro do processo de implantação, logo após o merge. Esse pacote fica disponível para ser baixado no próprio GitHub. Uma sugestão de trabalho futuro é automatizar a atualização desse pacote diretamente na aplicação mobile, buscando ele no repositório do backend quando o comando *nuget restore* for utilizado, por exemplo. Esse comando é utilizado para baixar os pacotes referenciados no projeto de uma aplicação .NET.

Figura 27. Diagrama do processo de implantação de código em homologação.




Fonte: elaborada pelo autor (2022).

## 4.9 AVALIAÇÃO DO PRODUTO

A avaliação do produto de software desenvolvido segue os mesmos critérios utilizados na análise dos trabalhos relacionados, sendo eles: o *onboarding*, interface, métricas de negócio, personalização e completude de funcionalidades. Entretanto, o desenvolvimento executado não alcançou o que foi proposto por limitações de tempo e equipe, como explicado no texto. Portanto, a avaliação pessoal descrita aqui considera o trabalho proposto, deixando claro qual foi a parte não executada.

### 4.9.1 Onboarding

O *onboarding* da aplicação é muito simples, contendo somente uma tela de login. O aplicativo ainda não permite que uma pessoa se cadastre, obrigando que o autor, ou outra pessoa com credenciais administrativas do sistema, cadastre os usuários diretamente via chamada de API. O único destaque é para a possibilidade de alterar o idioma, facilitando para pessoas que não compreendem o idioma nativo da aplicação, e fuso-horário, facilitando a interpretação de horários do uso de espaços.

### 4.9.2 Interface

Poucos elementos visuais foram desenvolvidos para o aplicativo. O intuito foi garantir uma usabilidade simples e conhecida, baseada no senso comum de posicionamento de botões e uso de ícones. Um breve estudo de cores foi realizado para garantir que o contraste de elementos textuais sejam agradáveis aos olhos, diferente de algumas das aplicações analisadas.

### 4.9.3 Métricas de Negócio

Nenhuma funcionalidade que inclui algum tipo de métrica de negócio foi adicionada ao escopo do MVP ou desenvolvida durante este trabalho.

### 4.9.4 Personalização

O sistema não permite nenhum tipo de personalização visual ou de infraestrutura (URL's personalizadas, por exemplo). Os únicos elementos personalizáveis são os formatos de data, horário, idioma e moeda.

### 4.9.5 Completude de Funcionalidades

O MVP já proposto possui um escopo bem reduzido e mesmo assim não foi possível implementá-lo por inteiro. Dessa forma, o resultado deixa bastante a desejar em termos de completude de funcionalidades. Todas as funcionalidades do MVP foram implementadas no backend, mas somente o login no aplicativo.

## 5. CONSIDERAÇÕES FINAIS

Este trabalho trouxe o objetivo de aplicar os conceitos de engenharia de software no desenvolvimento do protótipo de um produto, utilizando o tema de gestão de clubes de tênis como um laboratório para experiências. As atividades desenvolvidas foram: revisão tecnológica dos produtos disponíveis no mercado, definição do público-alvo, definição do escopo de desenvolvimento, modelagem dos dados, definição da arquitetura, preparo do monitoramento das aplicações, implementação do backend, implementação do aplicativo Android, construção da infraestrutura em nuvem e aplicação de práticas de DevOps para otimizar o desenvolvimento.

O objetivo original era construir um MVP completo com aplicativo para smartphone (iOS e Android) e aplicação web, passando pelas atividades descritas acima. Entretanto, durante a execução, foi possível enxergar que alguns dos tópicos exigiram mais tempo de estudo do que o estimado, algumas provas de conceito precisaram ser construídas para facilitar o entendimento e o escopo em si realmente estava muito grande para um trabalho de conclusão de curso individual.

Foi possível executar um pouco de todas as atividades desejadas, mesmo que em um nível superficial. Todas as funcionalidades propostas foram criadas no backend, mas não foi possível implementá-las inteiramente no aplicativo, mesmo com a redução de escopo e simplificações aplicadas.

Mesmo que não tenha sido possível construir um MVP, a proposta e implementação da arquitetura, infraestrutura e processo de desenvolvimento foram de extremo valor para ganho e compartilhamento de conhecimento. Dessa forma, observa-se que este trabalho serve como um compilado de discussões sobre as decisões no início do desenvolvimento do MVP de um produto de software.

Algumas sugestões de trabalhos futuros foram descritas durante o trabalho. Todas elas e mais algumas estão compiladas a seguir:

- A ampliação das plataformas disponíveis, incluindo web, macOS, iOS e Windows;
- Implementar um limite máximo de pessoas por espaço reservável do clube;
- Automatizar o processo de atualização do pacote NuGet EasyTennis.Domain nas aplicações diferentes do backend;
- Implementação do API Gateway, que foi cortado do escopo;
- Aprofundamento da análise de incorporação de um MOM na arquitetura;
- Implantação dos recursos de infraestrutura *serverless* para procedimentos custosos, como relatórios;
- Introduzir funcionalidades de personalização, como cores e URL, indo na direção de um produto white label, pelo menos para a versão web.

## REFERÊNCIAS

AWS Lambda. **AWS Lambda**, 2022. Disponível em: <https://aws.amazon.com/lambda/>. Acesso em: 10 de julho de 2022.

BRITCH, David, Gechev, Ivan e Conrey, J. What is .NET MAUI? **Microsoft Docs**, 2022. Disponível em: <https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui>. Acesso em: 24 de julho de 2022.

CHEN, Brandon. PostgreSQL vs. MySQL: What you need to know. **Fivetran Blog**, 2021. Disponível em: <https://www.fivetran.com/blog/postgresql-vs-mysql>. Acesso em: 24 de julho de 2022.

GARLAN, David. "**Software architecture: a roadmap.**" Proceedings of the Conference on the Future of Software Engineering. 2000.

HASSAN, Hassan B., Saman A. Barakat, and Qusay I. Sarhan. "**Survey on serverless computing.**" Journal of Cloud Computing 10.1 (2021): 1-29.

HASSELBRING, Wilhelm. "**Software architecture: Past, present, future.**" The Essence of Software Engineering. Springer, Cham, 2018. 169-184.

HEIJS, Tom, M. Aerts-Veenstra, and K. J. Roodbergen. "**Strengths and weaknesses of parcel locker concepts: a comparison between white label lockers, public transport lockers and exclusive lockers.**" (2021).

IBM CLOUD EDUCATION. PostgreSQL vs. MySQL: What's the Difference? **IBM Blog**, 2021. Disponível em: <https://www.ibm.com/cloud/blog/postgresql-vs-mysql-whats-the-difference>. Acesso em: 11 de julho de 2022.

ILYUKHA, Vitaliy. Choosing between MySQL vs PostgreSQL vs SQL Server. **Jelvix Blog**, 2020. Disponível em: <https://jelvix.com/blog/mysql-postgresql-sql-server>. Acesso em: 24 de julho de 2022.

JAMES, Nefe. Should you switch from React to Svelte? **LogRocket**, 2022. Disponível em: <https://blog.logrocket.com/should-you-switch-react-svelte/>. Acesso em: 23 de julho de 2022.

MICROSOFT. Microsoft .NET, 2022. Página What is .NET? Disponível em: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>. Acesso em: 24 de julho de 2022.

MYSQL. MySQL 8.0 Reference Manual, 2022. Chapter 1 General Information. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/introduction.html>. Acesso em: 24 de julho de 2022.

NILSSON, Erik, and Victor Pregén. "**Performance evaluation of message-oriented middleware.**" (2020).

OFFLINE app architecture: Why you should build offline-first apps. TechAhead, 2020. Disponível em: <https://www.techaheadcorp.com/blog/offline-app-architecture/>. Acesso em: 24 de julho de 2022.

OFFLINE FIRST ORG. Offline First, 2022. Página inicial. Disponível em: <https://offlinefirst.org/>. Acesso em: 24 de julho de 2022.

OLIVEIRA, Allan. Backend For Frontend: Uma estratégia sob medida para a entrega de microsserviços. **Medium**, 2020. Disponível em: <https://medium.com/jeitosanar/backend-for-frontend-uma-estrat%C3%A9gia-sob-demanda-para-a-entrega-de-microsservi%C3%A7os-2f12d4cb9e3f>. Acesso em: 23 de julho de 2022.

PERRY, Dewayne E., and Alexander L. Wolf. "**Foundations for the study of software architecture.**" ACM SIGSOFT Software engineering notes 17.4 (1992): 40-52.

RICE Scoring Model. ProductPlan, 2022. Disponível em: <https://www.productplan.com/glossary/rice-scoring-model/>. Acesso em: 24 de julho de 2022.

SILVA, Franklin, Renata Souza, and Ivan Machado. "**Taming and unveiling software reuse opportunities through white label software in startups.**" 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2020.

TAYLOR, Jason. Clean Architecture Solution Template. GitHub, 2022. Disponível em: <https://github.com/jasontaylordev/CleanArchitecture>. Acesso em: 27 de novembro de 2022.

SIQUEIRA, André. CRM. **Resultados Digitais**, 2021. Disponível em: <https://resultadosdigitais.com.br/crm/>. Acesso em: 24 de julho de 2022.

STACK OVERFLOW. 2021 Developer Survey, 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021>. Acesso em: 23 de julho de 2022.

SVELTE. Svelte, 2022. Página inicial. Disponível em: <https://svelte.dev/>. Acesso em: 24 de julho de 2022.

TENIS BRASIL. Tenistas brasileiros representam 2,6% do mundo, 2019. Disponível em: <https://tenisbrasil.uol.com.br/noticias/72168/Tenistas-brasileiros-representam-26-do-mundo/>. Acesso em: 24 de julho de 2022.

THE Shared Responsibility Model. **DOCS AWS**, 2022. Disponível em: <https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/the-shared-responsibility-model.html>. Acesso em: 11 de julho de 2022.

WOLSKI, Rich; KRINTZ, Chandra; BAKIR, Fatih; GEORGE, Gareth; LIN, Wei-Tsung. **CSPOT: Portable, Multi-scale Functions-as-a-Service for IoT**. Proceedings of the

4th ACM/IEEE Symposium on Edge Computing (SEC ‘19), New York, NY, USA, p. 236–249, 2019.

SPRINGLY. Springly, 2022. Página inicial. Disponível em: <https://www.springly.org/>. Acesso em: 26 de novembro de 2022.

SPRINGLY TENNIS. Springly Tennis, 2022. Bring your A-game with an all-in-one tennis club management software. Disponível em: <https://www.springly.org/en-us/nonprofit/tennis-club-software/>. Acesso em: 26 de novembro de 2022.

ACES. Aces TM, 2022. Página inicial. Disponível em: <https://aces-tm.com/>. Acesso em: 26 de novembro de 2022.

CLUB MANAGER 365. Club Manager 365, 2022. Página inicial. Disponível em: <https://clubmanager365.com/>. Acesso em: 26 de novembro de 2022.

MARTIN, Robert C., et al. Clean architecture: a craftsman's guide to software structure and design. No. s 31. Prentice Hall, 2018.

SHARING objects using presigned URLs. AWS, 2022. Disponível em: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html>. Acesso em: 27 de novembro de 2022.

MYSQL WORKBENCH. MySQL Workbench, 2022. Página inicial. Disponível em: <https://www.mysql.com/products/workbench/>. Acesso em: 27 de novembro de 2022.

KRUCHTEN, Philippe, Robert L. Nord, and Ipek Ozkaya. "Technical debt: From metaphor to theory and practice." Ieee software 29.6 (2012): 18-21.

BREAKING it Down: The Cost of Localization. DayTranslations, 2022. Disponível em: <https://www.daytranslations.com/blog/cost-of-localization/>. Acesso em: 18 de dezembro de 2022.

# APÊNDICE A - Artigo

# Aplicação dos Conceitos de Engenharia de Software no Desenvolvimento de um Produto para Gestão de Clubes de Tênis

**Gabriel da Luz Simonetti Souza**

Universidade Federal de Santa Catarina
Florianópolis, Brasil

**Abstract.** Non-professional tennis is a sport that has limited access to technology in terms of management, player integration and technical support. In order to exercise several software development disciplines, this work presents the study, design and development of a prototype for a minimum viable product (MVP) of a multiplatform system for managing tennis clubs and integrating their affiliates. The development of the system includes an application for Android, using .NET MAUI technology and offline-first architecture; a backend application, using the .NET 6 platform; infrastructure, using AWS as a cloud provider, and some DevOps practices to optimize the software development and deployment process.

**Resumo.** O tênis não profissional é um esporte atendido de forma limitada no meio tecnológico nos quesitos de gestão, integração de jogadores e acompanhamento técnico. Com o intuito de exercitar diversas disciplinas do desenvolvimento de software, este trabalho apresenta o estudo, projeto e desenvolvimento de um protótipo para um produto mínimo viável (MVP) de um sistema multiplataforma para gestão de clubes de tênis e integração de seus afiliados. O desenvolvimento do sistema contempla um aplicativo para Android, utilizando a tecnologia .NET MAUI e arquitetura offline-first; uma aplicação de backend, utilizando a plataforma .NET 6; infraestrutura, utilizando a AWS como provedor de nuvem, e algumas práticas de DevOps para otimizar o processo de desenvolvimento e implantação do software.

## 1. Introdução

O desenvolvimento de um produto de software é composto por diversas etapas e envolve diversas áreas. Dominar o processo e compreender as particularidades de cada uma dessas áreas não é uma tarefa trivial. De forma bem resumida, dentro do contexto de produto de software, encontra-se um problema, identifica-se uma oportunidade e então se inicia a definição de uma solução. Essa solução não se trata somente da implementação de um software, mas de um modelo de negócio que se prove viável.

O exercício de fazer escolhas e lidar com as consequências acrescenta a experiência necessária para lidar melhor com as próximas escolhas. Fazer escolhas faz parte do cotidiano de uma equipe de desenvolvimento de software. Para exercitar o processo de desenvolvimento de um produto de software e adquirir mais conhecimento na área, optou-se por encontrar uma oportunidade de construção de solução real para o mercado e aplicar no desenvolvimento deste trabalho.

Dentro do tema proposto para este trabalho, encontra-se a oportunidade de exercitar as disciplinas de projeto e desenvolvimento de um software para lidar com usuários reais. As decisões de projeto e implementação, como funcionalidades e tecnologias adotadas, estão fundamentadas nos conceitos apresentados no texto, visando entregar um software real que resolva problemas reais, e não pautado em preferências pessoais. O texto deste trabalho contempla importantes conceitos como engenharia de software, arquitetura de software, boas práticas de programação e métricas. Esses conceitos auxiliam no bom processo de desenvolvimento do software e sua evolução, colaborando para que o produto seja mais confiável e de fácil manutenção.

Neste trabalho, pretendeu-se aplicar as técnicas disponíveis na literatura e mercado para tomada de decisão e desenvolvimento do protótipo de um MVP de um produto de software multiplataforma para gestão de clubes de tênis. Este trabalho apresenta os seguintes objetivos que devem ser alcançados para satisfazer seu escopo:

1. Ter uma análise de concorrência, definição do público-alvo e escopo da solução;
2. Ter a definição da arquitetura, escolha de tecnologias, projeto e implantação da infraestrutura, e práticas de DevOps;
3. Ter uma arquitetura extensível e escalável, visando a possibilidade de expandir para além do escopo de clubes de tênis;
4. Implementar um protótipo do sistema para gestão de clubes de tênis.

## 2. Desenvolvimento

Este projeto do software foi desenvolvido com o objetivo de balancear a extensibilidade, escalabilidade, complexidade e redução de escopo para definição de um MVP. Portanto, alguns itens, como a modelagem de dados, ficaram mais simples. Outros itens, como a arquitetura do aplicativo para dispositivo móvel ou a arquitetura utilizada na API tornaram-se mais complexos do que o necessário para um MVP desse porte. Cada um desses pontos é discutido nesta seção do trabalho.

### 2.1. Escopo do MVP

O público-alvo do MVP proposto neste trabalho são jogadores e treinadores afiliados a clubes de tênis, bem como seus gestores. Para elencar as necessidades do público-alvo, foi realizado um estudo buscando quais são as principais funcionalidades desejadas por gestores de clubes de tênis. Inicialmente, foram identificadas algumas necessidades pela própria vivência do estudante frequentando clubes de tênis de diversos tamanhos. Nessa parte, surgiram funcionalidades como reserva de quadras, histórico de jogos e encontrar parceria para jogar. O próximo passo foi pesquisar em mecanismos de busca quais são

as maiores necessidades de gestão de clubes de tênis ou clubes de esporte, no geral. A partir disso, foram encontradas outras funcionalidades como pagamentos, comunicação automática, finanças e organização de eventos (campeonatos, por exemplo). Por fim, foram realizadas as análises dos trabalhos relacionados, levantando mais algumas funcionalidades como perfil de usuário, histórico de jogos, fórum e notificações de horários vagos. Os resultados dessa análise foram colocados na Tabela 1.

Para definir o escopo de quais funcionalidades seriam atendidas na solução proposta neste trabalho, optou-se por aplicar o modelo de priorização RICE Score por tratar-se de um framework de aplicação simples e por introduzir um nível de confiança maior nas decisões envolvendo a definição do escopo do MVP do produto em relação a uma simples escolha pessoal. A Tabela 1 apresenta todas as funcionalidades analisadas, em ordem decrescente de pontuação RICE.

**Tabela 1. RICE Score aplicado nas funcionalidades.**

| Funcionalidade | R | I | C | E | RICE | MVP |
|---|---|---|---|---|---|---|
| Reserva de espaços | 950 | 3 | 100% | 5 | 570.00 | Sim |
| Agendamento de aulas | 800 | 2 | 80% | 5 | 256.00 | Sim |
| Perfil do usuário | 600 | 1 | 80% | 3 | 160.00 | Sim |
| Gerenciamento de usuários | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Gerenciamento de clubes | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Gerenciamento de espaços | 50 | 3 | 100% | 1 | 150.00 | Sim |
| Pagamentos (mensalidades, aulas e quadras) | 800 | 1 | 80% | 6 | 106.67 | Não |
| Histórico de jogos | 150 | 1 | 80% | 3 | 40.00 | Não |
| Análise de performance do jogador | 100 | 1 | 80% | 3 | 26.67 | Não |
| Notificação de horário vago | 100 | 1 | 80% | 3 | 26.67 | Não |
| Recomendações de jogos | 200 | 1 | 50% | 5 | 20.00 | Não |
| Comunicados (sem automação) | 20 | 2 | 80% | 3 | 10.67 | Não |
| Organização de campeonatos | 30 | 3 | 80% | 8 | 9.00 | Não |
| Fórum | 100 | 0.5 | 50% | 3 | 8.33 | Não |
| Finanças (registros e análises) | 20 | 2 | 80% | 8 | 4.00 | Não |
| Comunicadas (com automação) | 10 | 2 | 80% | 5 | 3.20 | Não |
| Espaço de feedbacks para o clube | 50 | 0.25 | 50% | 3 | 2.08 | Não |
| Manutenção das quadras | 20 | 0.25 | 50% | 3 | 0.83 | Não |

## 2.2. Tecnologias, Serviços e Ferramentas

A Tabela 2 apresenta o contexto, as tecnologias escolhidas e a motivação por trás das escolhas. A coluna de motivação não é exaustiva, trazendo apenas algumas bases para as escolhas.

**Tabela 2. Tecnologias escolhidas.**

| Contexto | Tecnologia | Motivação |
|---|---|---|
| Aplicações desktop e mobile | .NET MAUI | • Conhecimento prévio nas linguagens de programação utilizadas (C# e XAML);<br>• Facilidade para implantação em diferentes plataformas: Android, iOS, Windows e macOS;<br>• Grande quantidade de profissionais que trabalham com .NET. |
| Aplicação web | Svelte | • Menor quantidade de código necessário para atingir o mesmo resultado que outros frameworks populares;<br>• Maior desempenho em relação a frameworks populares, como o React;<br>• Alto grau de satisfação dos desenvolvedores. |
| APIs | .NET 6 | • Conhecimento prévio da tecnologia;<br>• É uma versão LTS (com suporte de longo prazo);<br>• Simples para exportar bibliotecas que podem ser utilizadas pelo .NET MAUI;<br>• Grande quantidade de profissionais que trabalham com .NET. |
| Banco de dados | MySQL | • Conhecimento prévio da tecnologia;<br>• Menor custo para manter em cloud;<br>• Capaz de sustentar aplicações de larga escala. |

## 2.3. Modelo de Dados

A modelagem de dados do software gira em torno da ideia de que um usuário pode ter zero ou mais clubes, um clube pode ter zero ou mais espaços e um espaço pode ter zero ou mais usos. Para o MVP, o diagrama ER simplificado é mostrado na Figura 1.

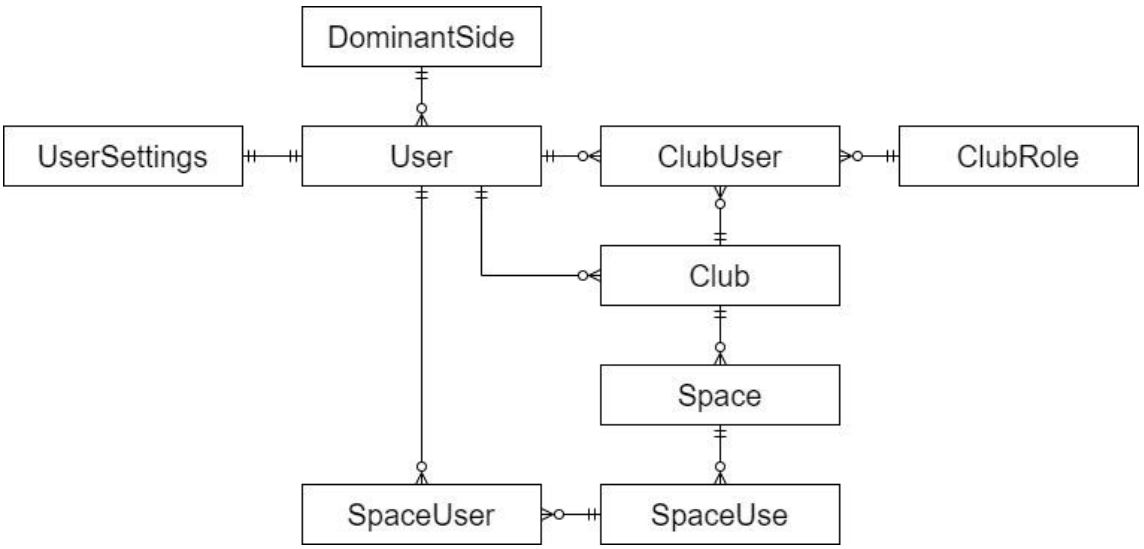



**Figura 1. Diagrama ER para o EasyTennis.**

As entidades *DominantSide* e *ClubRole* foram definidas como enumerações quando as classes foram definidas. As classes *User*, *Club*, *ClubUser*, *Space*, *SpaceUse*, e *SpaceUser* são filhas da classe *BaseAuditableEntity*, que segue a hierarquia apresentada na Figura 3. Dessa forma, a herança garante que existam os atributos *Id*, *CreateTime* e *UpdateTime* nessas classes.

Em termos de modelagem ER, não há necessidade das classes *ClubUser* e *SpaceUser* herdar de *BaseEntity* por conta de representarem um relacionamento e não precisarem de um ID, já que podem utilizar uma chave primária composta. Essa decisão deu-se por uma maior simplicidade para utilizar a biblioteca de *object–relational mapping* (ORM) escolhido.

O diagrama de classes contendo todas as classes que definem os modelos de dados definidos para o MVP é apresentado na Figura 2.

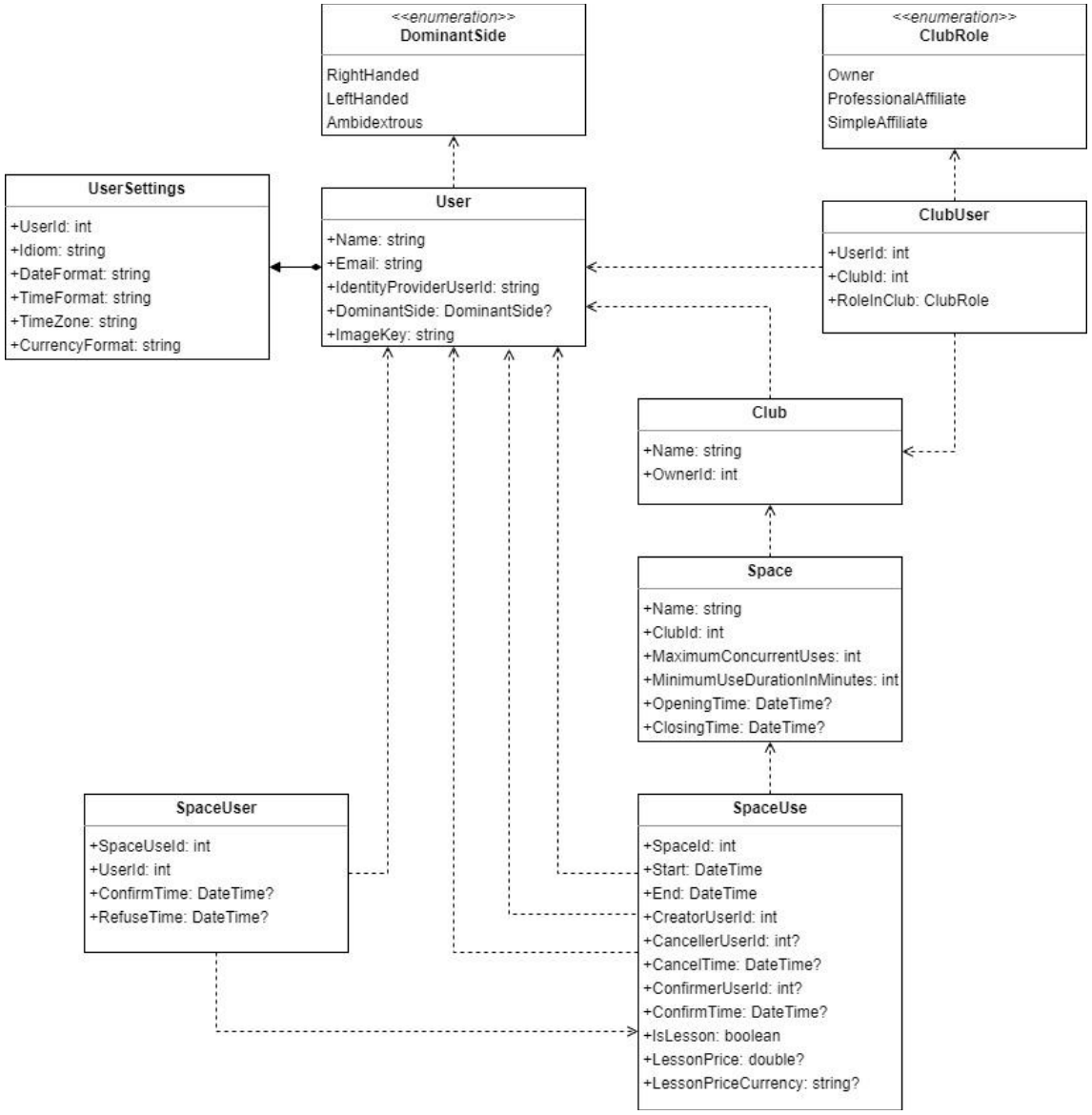



**Figura 2. Diagrama de classes para modelos de dados do MVP.**

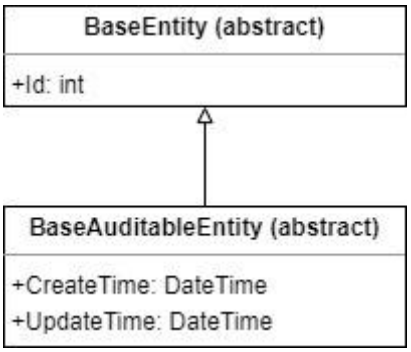



**Figura 3. Diagrama de classes com *BaseEntity* e *BaseAuditableEntity*.**

### 2.4. Arquitetura do Backend

O backend foi implementado utilizando a linguagem de programação C# com a plataforma .NET 6. Utilizou-se o modelo de arquitetura em camadas *Clean Architecture*, separando os escopos de apresentação, aplicação, infraestrutura e domínio. Cada uma das camadas é representada por um projeto .NET independente dentro da solução criada. A estrutura de pastas é apresentada na Figura 4.

#### 2.4.1. Camadas da Arquitetura

A camada de domínio é comum a todas as aplicações do sistema e poderia ser mantida por qualquer uma das aplicações. Entretanto, o backend foi a aplicação escolhida como responsável pela manutenção e evolução dessa camada. O motivo é que, como o backend faz o contato mais próximo do banco de dados, o autor do trabalho julgou mais prático e de senso comum ser o primeiro local onde criam-se as classes que definem os modelos de dados do sistema. Dessa forma, para que uma nova funcionalidade seja criada, as classes de domínio devem ser criadas no backend. Uma vez que estiverem funcionando corretamente, um pacote NuGet pode ser exportado do projeto para que possa ser adicionado em outras aplicações.

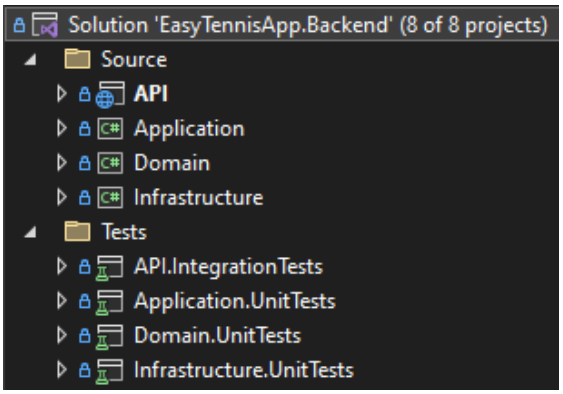



**Figura 4. Estrutura de pastas do backend.**

A camada definida como API (Figura 4) é a camada de apresentação da *clean architecture*. Nela foram implementados os *controllers*, responsáveis por receberem as requisições e responderem à aplicação cliente. Um exemplo para lidar com requisições de dados de usuário é mostrado abaixo:

```
[Authorize]
[HttpGet("{id}")]
public async Task<ActionResult<UserDto>> GetById(int id)
{
    UserDto dto = await Mediator.Send(new GetUserByIdQuery(id));
    return Ok(dto);
}
```

O padrão de projeto *Command Query Responsibility Segregation* (CQRS) foi utilizado no backend. Para que esse padrão seja adequadamente empregado dentro do contexto dessa aplicação, também foi feito uso dos padrões de projeto *mediator*, *command* e a estratégia de injeção de dependência.

No exemplo de código mostrado acima, a classe *GetUserByIdQuery* define um *command* e o atributo *Mediator* é uma instância da interface *ISender*, que envia uma requisição para as classes manipuladoras (*handlers*). *Handlers* são classes que realizam a execução do comando em si, chamando a consulta ao banco de dados e retornando o valor, por exemplo. Para cada *command*, foi implementado um *handler* específico, mantido dentro da camada *Application*. Um exemplo de *handler* para o mesmo código de lidar com requisições de dados de usuário é mostrado abaixo:

```
internal class GetUserByIdQueryHandler : IRequestHandler<GetUserByIdQuery,
UserDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetUserByIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<UserDto> Handle(GetUserByIdQuery request,
CancellationToken cancellationToken)
    {
        User? user = await context.Users
            .Include(user => user.LinkedClubs)
            .Include(user => user.Settings)
            .SingleOrDefaultAsync(user => user.Id.Equals(request.Id),
cancellationToken);
        return mapper.Map<UserDto>(user);
    }
```

```
}
```

Pode-se observar a injeção de dependências ocorrendo no construtor da classe, onde são injetadas as instâncias de *IApplicationDbContext* e *IMapper*, responsáveis por acessar o banco de dados e mapear os modelos de dados para instâncias de *data transfer objects* (DTO's), respectivamente. O método *Handle* executa exatamente o que é esperado que aquele comando execute: buscar os dados do usuário no banco de dados e retornar um DTO para o controlador poder retornar para o cliente. A classe foi declarada como *internal* ao invés de *public* porque isso a define como acessível publicamente somente para a camada em que ela foi definida. Dessa forma, somente classes internas da camada *Application* podem acessá-la.

A estrutura de pastas da camada *Application*, contendo todos os *handlers* e *commands* ficou como apresentada na Figura 5. Para cada entidade principal definida no modelo de dados, são criadas pastas que vão conter os *commands*, *queries* e *handlers*. Por isso, é possível observar que foram criadas as pastas *Users*, *Clubs*, *Spaces* e *SpaceUses*. A pasta *Common* contém algumas classes comuns a outras classes dessa mesma camada e a pasta *Identity* contém alguns DTO's relativos às chamadas de autenticação e autorização.

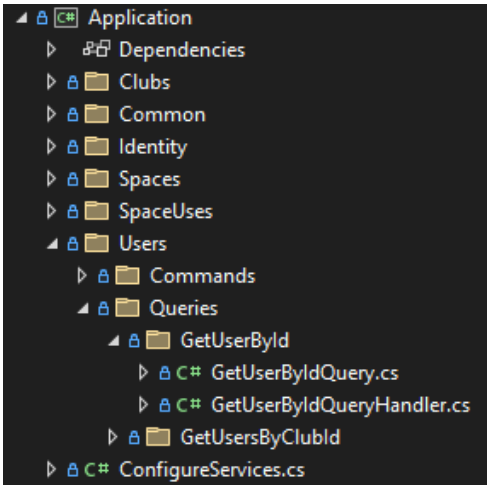



**Figura 5. Estrutura de pastas da camada Application do backend.**

A camada de domínio, mostrada como *Domain* na Figura 4, contém a definição das classes que representam os modelos de dados, DTO's e outras características comuns ao domínio do sistema, como enumerações e validações, por exemplo.

A camada *Infrastructure* contém as classes que implementam características específicas de infraestrutura da aplicação do backend, como acesso ao banco de dados e integração com provedor de identidade. As outras camadas da aplicação não devem conhecer qualquer particularidade da infraestrutura. No caso do backend, o banco de dados é MySQL e o provedor de identidades é o AWS Cognito.

### 2.5. Aplicativo para Android

O aplicativo para smartphone foi implementado utilizando a linguagem de programação C# com a plataforma .NET MAUI. Da mesma forma que no backend, utilizou-se o modelo de arquitetura em camadas Clean Architecture, separando os escopos de apresentação, aplicação, infraestrutura e domínio. Cada uma das camadas é representada por um projeto .NET independente dentro da solução criada. A estrutura de pastas é apresentada na Figura 6.

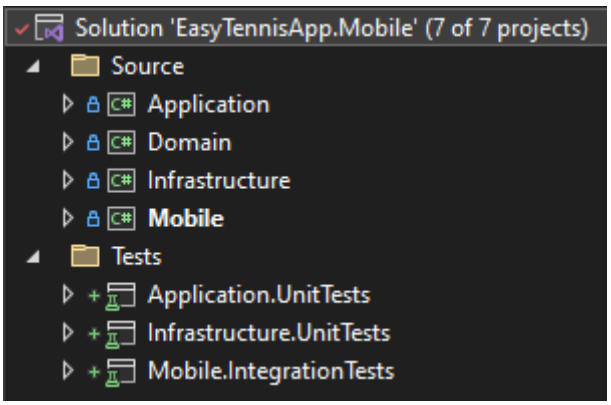



**Figura 6. Estrutura de pastas do aplicativo para dispositivos móveis.**

Buscou-se implementar um layout moderno e simples para o aplicativo EasyTennis, no qual o usuário pode encontrar os principais recursos da aplicação com poucos toques. O critério de quantidade de toques não foi aprofundado neste trabalho. O aplicativo foi projetado considerando a necessidade de, no máximo, 2 toques para acessar cada funcionalidade principal, uma vez que o aplicativo está aberto e com o usuário autenticado. O aplicativo é internacionalizado e possui suporte às culturas Português (Brasil) e Inglês (Estados Unidos).

Dadas as limitações de tempo e equipe, o desenvolvimento ficou limitado a quatro principais telas: login, início, minhas reservas e configurações. Todas essas telas são descritas a seguir. Os usuários de nome "John Connor" e "T-800" são fictícios.

### 2.5.1. Login

A tela do login, mostrada na Figura 7, permite que um usuário já registrado no sistema possa se autenticar mediante fornecimento de e-mail e senha. O cadastro de usuários diretamente pelo aplicativo não foi projetado ou implementado.

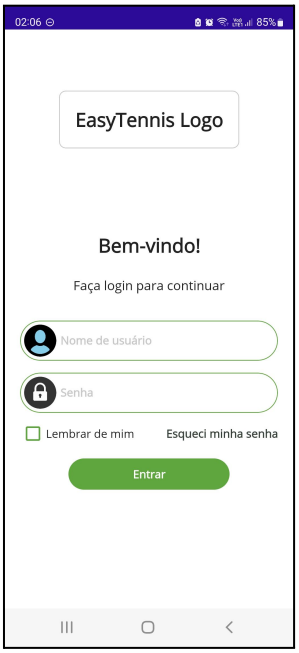



**Figura 7. Captura de tela do aplicativo EasyTennis: tela de login.**

### 2.5.2. Início

A tela de início, mostrada na Figura 8, é a tela visualizada pelo usuário logo após ele se autenticar ou ao acessar o aplicativo já estando autenticado previamente. Nesta tela, são exibidas as principais notificações e avisos para o usuário. Os avisos de cadastro incompleto e de convites pendentes foram os únicos implementados neste trabalho, porém outras notificações de diversas naturezas podem ser implementados.

O aviso de cadastro incompleto tem o objetivo de incentivar o usuário a inserir mais informações no seu perfil para que a base de dados do sistema fique mais rica. Um exemplo disso é o registro de uma imagem de perfil, para facilitar o reconhecimento entre as pessoas usuárias do sistema.

O aviso de convites pendentes se trata das reservas de espaços nas quais a pessoa ainda não confirmou sua presença. No exemplo da Figura 8, o usuário "John Connor" foi convidado para uma reserva de espaço pelo usuário "T-800" (terceira reserva da Figura 23). Como ele ainda não confirmou sua presença, o convite fica como pendente.


**Figura 8. Captura de tela do aplicativo EasyTennis: tela de início.**

### 2.5.3. Minhas Reservas

A tela "Minhas Reservas" (Figura 9) permite a visualização das reservas do dia corrente em diante. As reservas são agrupadas por data e, para cada uma delas, são mostradas as informações de nome do espaço, responsável pela reserva, horários de início e fim. O botão "+" leva o usuário para o formulário de criação de reserva.

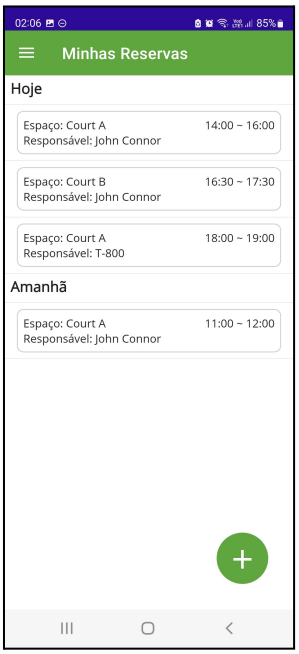



**Figura 9. Captura de tela do aplicativo EasyTennis: tela de minhas reservas.**

### 2.5.4. Configurações

A tela de configurações, mostrada na Figura 10, permite ao usuário configurar o idioma, junto às definições de cultura de um país, e o fuso-horário. Essas opções são importantes para aumentar o alcance e melhorar a usabilidade do aplicativo. A decisão de implementar esse recurso como uma das primeiras funcionalidades se deu porque, em geral, é mais barato construir o suporte à internacionalização desde o início do desenvolvimento de uma aplicação (BREAKING, 2022).


**Figura 10. Captura de tela do aplicativo EasyTennis: tela de configurações.**

## 2.6. Infraestrutura

A infraestrutura do sistema foi implantada em nuvem, utilizando a plataforma AWS, e é composta principalmente por banco de dados, aplicação de backend e repositório de arquivos. A Figura 11 apresenta o diagrama da infraestrutura com os serviços que foram utilizados, estando todos eles na camada gratuita da AWS.

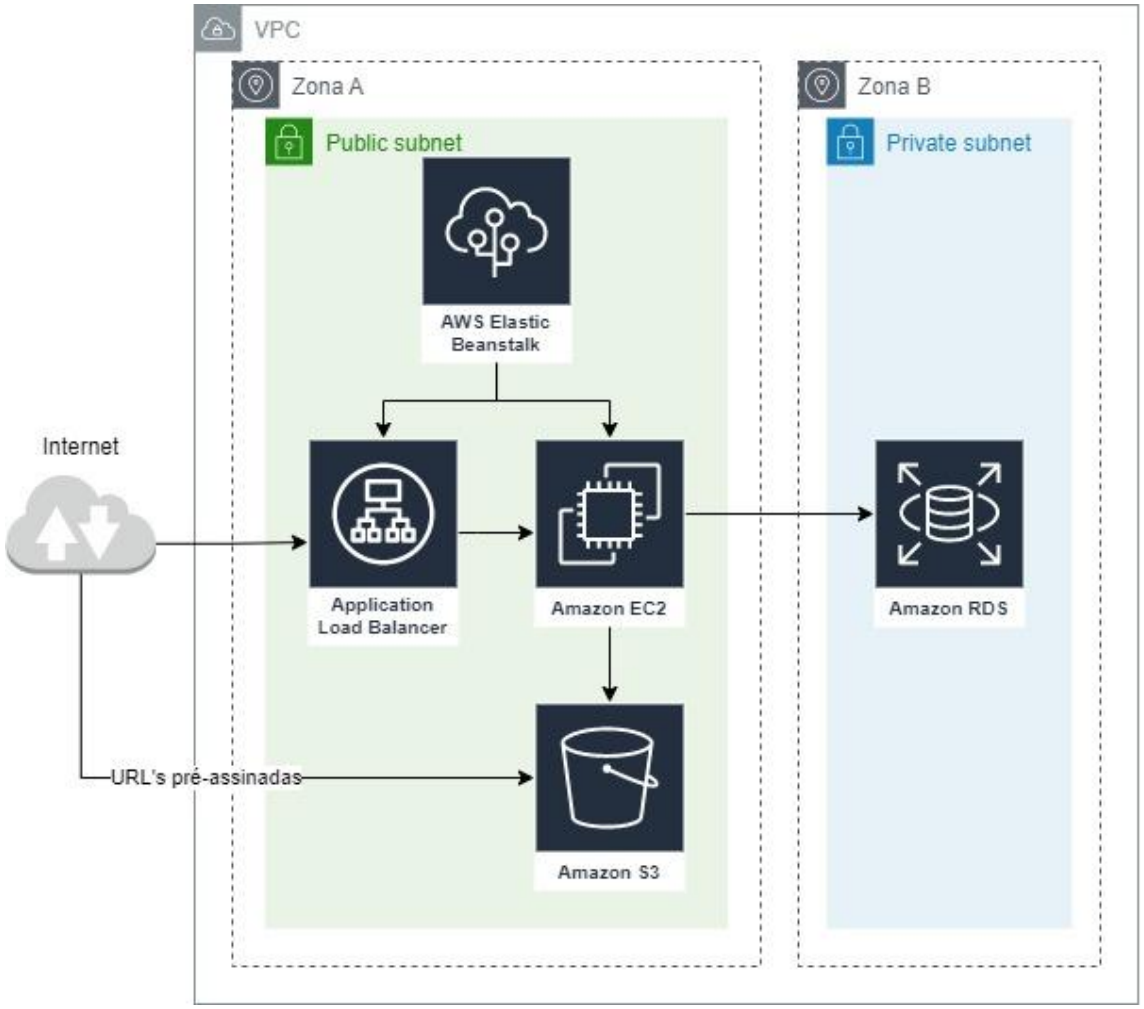



**Figura 11. Diagrama de infraestrutura.**

Foi definida uma *virtual private cloud* (VPC) com duas zonas de disponibilidade para implantação dos recursos de infraestrutura. A zona A contém uma sub-rede pública, sendo acessível de fora da VPC somente por meio do *application load balancer* (ALB) ou repositório de arquivos S3 com URL pré-assinada. A zona B contém uma sub-rede privada contendo somente o banco de dados da aplicação. A única forma de acessá-lo é através de aplicação de backend.

O ALB faz o papel de interface com a internet. Toda aplicação cliente se comunica diretamente com ele, evitando a necessidade de expor a máquina EC2 com a aplicação de backend para a internet. O único caso em que uma aplicação cliente se comunica com outro recurso, além do ALB, é quando precisa ler uma determinada imagem do repositório de arquivos S3. Isso é feito através de uma URL pré-assinada, gerada com as credenciais do usuário, garantindo acesso por tempo limitado para download do recurso (SHARING, 2022). A Figura 12 mostra um diagrama de sequência exemplificando o momento em que ocorre a geração da URL pré-assinada.

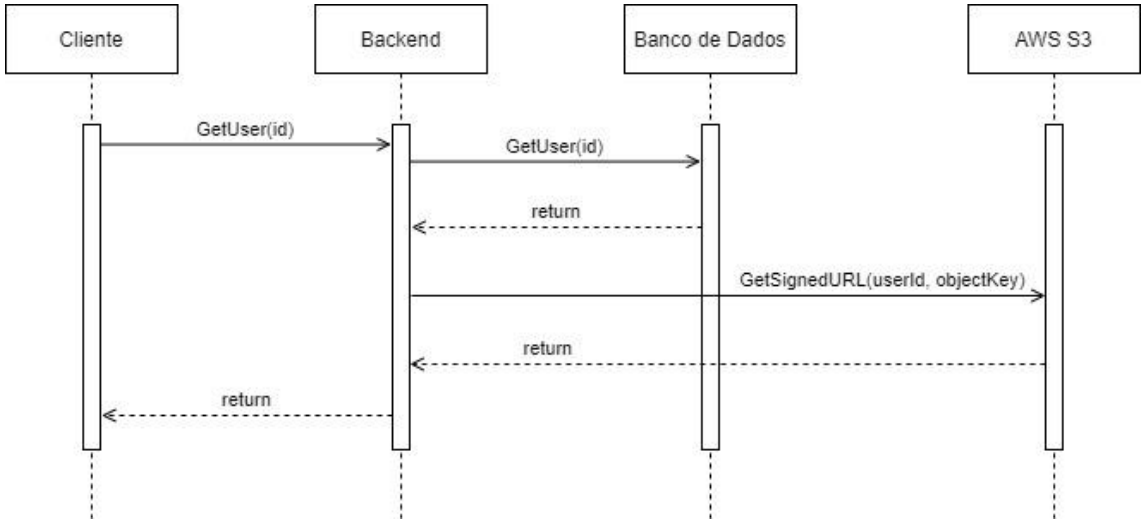



**Figura 12. Diagrama de sequência para geração de URL pré-assinada.**

Quando uma aplicação cliente solicita o registro de um usuário para o backend, o registro é buscado no banco de dados e o atributo *ImageKey* vem com o valor da chave do objeto no repositório S3. As credenciais e *role* do usuário são utilizadas para gerar a URL de acesso de leitura para a imagem. Esta é retornada dentro da resposta para a aplicação cliente no campo *ImageSignedUrl*. Essa URL tem um tempo de expiração, limitando a aplicação cliente a fazer o acesso dentro desse tempo para baixar a imagem. Para evitar que esse processo seja realizado com frequência, as aplicações poderiam utilizar a estratégia de cache para imagens, mas isso não foi implementado.

O serviço Elastic Beanstalk é utilizado para gerenciar as máquinas EC2 e o ALB de forma mais simples. Como o autor do trabalho não tem grande domínio sobre recursos de infraestrutura, optou-se por utilizar essa plataforma gerenciada pela própria AWS. Dessa forma, responsabilidades como atualização de sistema operacional, implantação de atualizações da aplicação, gerenciamento de variáveis de ambiente, ciclo de vida das máquinas virtuais e outras ficam a cargo do serviço contratado.

## 2.7. Processo de Desenvolvimento

O processo de desenvolvimento do software foi feito utilizando o Git Flow como base para integração do código ao repositório. Os repositórios ficam armazenados no GitHub. Sempre que um *pull request* é criado, utilizou-se o serviço GitHub Actions para disparar a execução de análise estática de código, bloqueando o *merge* no caso de alguma inconsistência ser encontrada. Além da análise estática de código, são executados os testes automatizados da aplicação para garantir que *branches* só sejam unidas ao código principal sem que haja regressão de padrões de estilos de código e qualidade coberta pelos testes.

Para o caso da aplicação de backend, após a realização do *merge* com a *branch* de desenvolvimento, a aplicação é implantada em ambiente de homologação. Como não foi realizada a implantação de um ambiente de produção, não foi criada a automação para essa implantação para o *merge* com a *branch* principal.

Para o caso da aplicação *mobile*, o processo de deploy em ambiente de homologação não seria feito pelo GitHub Actions, mas pelo App Center. Essa parte não foi implementada, pois a publicação em lojas de aplicativos exige um trabalho burocrático e custos que estão fora do escopo deste trabalho. Para instalar a aplicação em diversos dispositivos, utilizou-se a geração de uma APK gerada diretamente com a IDE Visual Studio e SDK Android.

O diagrama da Figura 13 ilustra o processo de implantação de código em ambiente de homologação. Este é um diagrama com propósito didático, pois as aplicações estão em repositórios separados e nos fluxos implementados não há uma bifurcação de "Qual aplicação?", como mostrado na imagem, por exemplo.

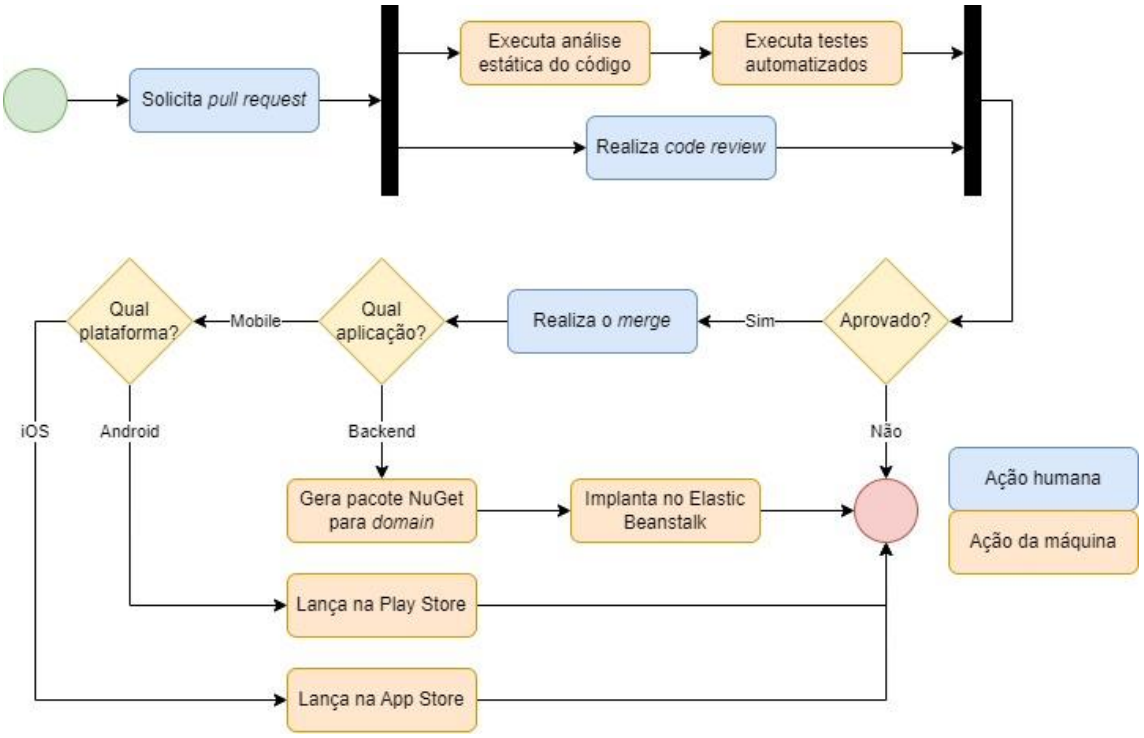



**Figura 13. Diagrama do processo de implantação de código em homologação.**

Para que as aplicações possam utilizar as mesmas classes para os modelos de dados sem a necessidade de reescrevê-las, definiu-se que um pacote NuGet é exportado a partir do código da aplicação do backend para ser utilizado na aplicação mobile. Para isso, foi adicionada a criação do pacote dentro do processo de implantação, logo após o merge. Esse pacote fica disponível para ser baixado no próprio GitHub. Uma sugestão de trabalho futuro é automatizar a atualização desse pacote diretamente na aplicação mobile, buscando ele no repositório do backend quando o comando *nuget restore* for utilizado, por exemplo. Esse comando é utilizado para baixar os pacotes referenciados no projeto de uma aplicação .NET.

## 2.8. Considerações Finais

Este trabalho trouxe o objetivo de aplicar os conceitos de engenharia de software no desenvolvimento do protótipo de um produto, utilizando o tema de gestão de clubes de tênis como um laboratório para experiências. As atividades desenvolvidas foram: revisão tecnológica dos produtos disponíveis no mercado, definição do público-alvo, definição do escopo de desenvolvimento, modelagem dos dados, definição da arquitetura, preparo do monitoramento das aplicações, implementação do backend, implementação do aplicativo Android, construção da infraestrutura em nuvem e aplicação de práticas de DevOps para otimizar o desenvolvimento.

O objetivo original era construir um MVP completo com aplicativo para smartphone (iOS e Android) e aplicação web, passando pelas atividades descritas acima. Entretanto, durante a execução, foi possível enxergar que alguns dos tópicos exigiram mais tempo de estudo do que o estimado, algumas provas de conceito precisaram ser construídas para facilitar o entendimento e o escopo em si realmente estava muito grande para um trabalho de conclusão de curso individual.

Foi possível executar um pouco de todas as atividades desejadas, mesmo que em um nível superficial. Todas as funcionalidades propostas foram criadas no backend, mas não foi possível implementá-las inteiramente no aplicativo, mesmo com a redução de escopo e simplificações aplicadas.

Mesmo que não tenha sido possível construir um MVP, a proposta e implementação da arquitetura, infraestrutura e processo de desenvolvimento foram de extremo valor para ganho e compartilhamento de conhecimento. Dessa forma, observa-se que este trabalho serve como um compilado de discussões sobre as decisões no início do desenvolvimento do MVP de um produto de software.

## Referências

BREAKING it Down: The Cost of Localization. DayTranslations, 2022. Disponível em: <https://www.daytranslations.com/blog/cost-of-localization/>. Acesso em: 18 de dezembro de 2022.

SHARING objects using presigned URLs. AWS, 2022. Disponível em: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html>. Acesso em: 27 de novembro de 2022.

# APÊNDICE B - Código Fonte do Backend

```
[ApiController]
[Route("[controller]")]
public abstract class ApiControllerBase : ControllerBase
{
    protected ISender Mediator => _mediator ??=
HttpContext.RequestServices.GetRequiredService<ISender>();

    private ISender _mediator = null!;
}
```

ApiControllerBase.cs

```
public class ClubsController : ApiControllerBase
{
    [Authorize]
    [HttpPost]
    public async Task<ActionResult<int>> Create(CreateClubCommand command)
    {
        return await Mediator.Send(command);
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult> Update(int id, [FromBody] UpdateClubCommand
command)
    {
        if (id != command.Id)
        {
            return BadRequest();
        }

        await Mediator.Send(command);
        return NoContent();
    }

    [Authorize]
    [HttpGet("{id}")]
    public async Task<ActionResult<ClubDto>> GetById(int id)
    {
        ClubDto dto = await Mediator.Send(new GetClubByIdQuery(id));
        return Ok(dto);
    }
}
```

ClubsController.cs

```
public class SpacesController : ApiControllerBase
{
```

```
    [Authorize]
    [HttpPost]
    public async Task<ActionResult<int>> Create(CreateSpaceCommand command)
    {
        return await Mediator.Send(command);
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult> Update(int id, [FromBody] UpdateSpaceCommand
command)
    {
        if (id != command.Id)
        {
            return BadRequest();
        }

        await Mediator.Send(command);
        return NoContent();
    }

    [Authorize]
    [HttpGet("{id}")]
    public async Task<ActionResult<SpaceDto>> GetById(int id)
    {
        SpaceDto dto = await Mediator.Send(new GetSpaceByIdQuery(id));
        return Ok(dto);
    }

    [Authorize]
    [HttpGet]
    public async Task<ActionResult<IEnumerable<SpaceDto>>>
GetByClubId([FromQuery] int clubId)
    {
        IEnumerable<SpaceDto> dto = await Mediator.Send(new
GetSpacesByClubIdQuery(clubId));
        return Ok(dto);
    }
}
```

SpacesController.cs

```
public class SpaceUsesController : ApiControllerBase
{
    [Authorize]
    [HttpPost]
    public async Task<ActionResult<int>> CreateSpaceUse(CreateSpaceUseCommand
command)
    {
        return await Mediator.Send(command);
    }
```

```
    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult> UpdateSpaceUse(int id, UpdateSpaceUseCommand
command)
    {
        if (id != command.Id)
        {
            return BadRequest();
        }

        await Mediator.Send(command);
        return NoContent();
    }

    [Authorize]
    [HttpGet("{id}")]
    public async Task<ActionResult<SpaceUseDto>> GetSpaceUseById(int id)
    {
        SpaceUseDto dto = await Mediator.Send(new GetSpaceUseByIdQuery(id));
        return Ok(dto);
    }

    [Authorize]
    [HttpGet]
    public async Task<ActionResult<IEnumerable<SpaceUseDto>>>
GetSpaceUsesBySpaceId([FromQuery] int spaceId)
    {
        IEnumerable<SpaceUseDto> dto = await Mediator.Send(new
GetSpaceUsesBySpaceIdQuery(spaceId));
        return Ok(dto);
    }
}
```

SpaceUsesController.cs

```
public class UsersController : ApiControllerBase
{
    private readonly IIdentityService identityService;

    public UsersController(IIdentityService identityService)
    {
        this.identityService = identityService;
    }

    [AllowAnonymous]
    [HttpPost("signin")]
    public async Task<AuthResponseDto> SignIn([FromBody] UserLoginDto dto)
    {
        return await identityService.TryLoginAsync(dto);
    }
```

```
    [Authorize]
    [HttpPost]
    public async Task<ActionResult<int>> Create([FromBody] CreateUserCommand
command)
    {
        return await Mediator.Send(command);
    }

    [Authorize]
    [HttpPut("{id}")]
    public async Task<ActionResult> Update(int id, [FromBody] UpdateUserCommand
command)
    {
        if (id != command.Id)
        {
            return BadRequest();
        }

        await Mediator.Send(command);
        return NoContent();
    }

[Authorize]
[HttpPut("{id}/settings")]
public async Task<ActionResult> UpdateSettings(int id, [FromBody]
UpdateUserSettingsCommand command)
{
    if (id != command.UserId)
    {
        return BadRequest();
    }

    await Mediator.Send(command);
    return NoContent();
}

    [Authorize]
    [HttpGet("{id}")]
    public async Task<ActionResult<UserDto>> GetById(int id)
    {
        UserDto dto = await Mediator.Send(new GetUserByIdQuery(id));
        return Ok(dto);
    }

    [Authorize]
    [HttpGet]
    public async Task<ActionResult<IEnumerable<SimpleUserDto>>>
GetByClubId([FromQuery] int clubId)
    {
        IEnumerable<SimpleUserDto> dto = await Mediator.Send(new
```

```
GetUsersByClubIdQuery(clubId));
        return Ok(dto);
    }
}
```

UserController.cs

```
public class Program
{
    public static void Main(string[] args)
    {
        WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddInfrastructureServices(builder.Configuration);
        builder.Services.AddApplicationServices();

        builder.Services.AddControllers();

        // Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();

        builder.Services.AddHealthChecks();

        WebApplication app = builder.Build();

        app.MapHealthChecks("/health");

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseHttpsRedirection();

        app.UseAuthentication();
        app.UseAuthorization();

        app.MapControllers();

        app.Run();
    }
}
```

Program.cs

```
public class CreateClubCommand : IRequest<int>
```

```
{
    public string Name { get; set; } = null!;
    public int OwnerId { get; set; }
}
```

CreateClubCommand.cs

```
internal class CreateClubCommandHandler : IRequestHandler<CreateClubCommand,
int>
{
    private readonly IApplicationDbContext context;

    public CreateClubCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<int> Handle(CreateClubCommand request, CancellationToken
cancellationToken)
    {
        Club club = new()
        {
            Name = request.Name,
            OwnerId = request.OwnerId,
            ClubUserLinks = new List<ClubUser>()
            {
                new ClubUser()
                {
                    RoleId = (int)Roles.ClubOwner,
                    UserId = request.OwnerId
                }
            }
        };

        await context.Clubs.AddAsync(club, cancellationToken);
        await context.SaveChangesAsync(cancellationToken);

        return club.Id;
    }
}
```

CreateClubCommandHandler.cs

```
public class UpdateClubCommand : IRequest<Unit>
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
}
```

UpdateClubCommand.cs

```
internal class UpdateClubCommandHandler : IRequestHandler<UpdateClubCommand,
Unit>
{
    private readonly IApplicationDbContext context;

    public UpdateClubCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<Unit> Handle(UpdateClubCommand request, CancellationToken
cancellationToken)
    {
        Club club = await context.Clubs.SingleAsync(club =>
club.Id.Equals(request.Id), cancellationToken);

        club.Name = request.Name;

        context.Clubs.Update(club);
        await context.SaveChangesAsync(cancellationToken);

        return Unit.Value;
    }
}
```

UpdateClubCommandHandler.cs

```
public class GetClubByIdQuery : IRequest<ClubDto>
{
    public int Id { get; }

    public GetClubByIdQuery(int id)
    {
        Id = id;
    }
}
```

GetClubByIdQuery.cs

```
internal class GetClubByIdQueryHandler : IRequestHandler<GetClubByIdQuery,
ClubDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetClubByIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }
```

```
    public async Task<ClubDto> Handle(GetClubByIdQuery request,
CancellationToken cancellationToken)
    {
        Club? club = await context.Clubs.SingleOrDefaultAsync(x =>
x.Id.Equals(request.Id), cancellationToken);
        return mapper.Map<ClubDto>(club);
    }
}
```

GetClubByIdQueryHandler.cs

```
public class CreateSpaceCommand : IRequest<int>
{
    public string Name { get; set; } = null!;
    public int ClubId { get; set; }
}
```

CreateSpaceCommand.cs

```
internal class CreateSpaceCommandHandler : IRequestHandler<CreateSpaceCommand,
int>
{
    private readonly IApplicationDbContext context;

    public CreateSpaceCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<int> Handle(CreateSpaceCommand request, CancellationToken
cancellationToken)
    {
        Space clubSpace = new()
        {
            Name = request.Name,
            ClubId = request.ClubId
        };

        await context.Spaces.AddAsync(clubSpace, cancellationToken);
        await context.SaveChangesAsync(cancellationToken);

        return clubSpace.Id;
    }
}
```

CreateSpaceCommandHandler.cs

```
public class UpdateSpaceCommand : IRequest<Unit>
{
    public int Id { get; set; }
```

```
    public string Name { get; set; } = null!;
}
```

UpdateSpaceCommand.cs

```
internal class UpdateSpaceCommandHandler : IRequestHandler<UpdateSpaceCommand,
Unit>
{
    private readonly IApplicationDbContext context;

    public UpdateSpaceCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<Unit> Handle(UpdateSpaceCommand request, CancellationToken
cancellationToken)
    {
        Space clubSpace = await context.Spaces.SingleAsync(space =>
space.Id.Equals(request.Id), cancellationToken);

        clubSpace.Name = request.Name;

        context.Spaces.Update(clubSpace);
        await context.SaveChangesAsync(cancellationToken);

        return Unit.Value;
    }
}
```

UpdateSpaceCommandHandler.cs

```
public class GetSpaceByIdQuery : IRequest<SpaceDto>
{
    public int Id { get; }

    public GetSpaceByIdQuery(int id)
    {
        Id = id;
    }
}
```

GetSpaceByIdQuery.cs

```
internal class GetSpaceByIdQueryHandler : IRequestHandler<GetSpaceByIdQuery,
SpaceDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetSpaceByIdQueryHandler(IApplicationDbContext context, IMapper
```

```
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<SpaceDto> Handle(GetSpaceByIdQuery request,
CancellationToken cancellationToken)
    {
        Space? space = await context.Spaces.SingleOrDefaultAsync(space =>
space.Id.Equals(request.Id), cancellationToken);
        return mapper.Map<SpaceDto>(space);
    }
}
```

GetSpaceByIdQueryHandler.cs

```
public class GetSpacesByClubIdQuery : IRequest<IEnumerable<SpaceDto>>
{
    public int ClubId { get; }

    public GetSpacesByClubIdQuery(int clubId)
    {
        ClubId = clubId;
    }
}
```

GetSpacesByClubIdQuery.cs

```
internal class GetSpacesByClubIdQueryHandler :
IRequestHandler<GetSpacesByClubIdQuery, IEnumerable<SpaceDto>>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetSpacesByClubIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<IEnumerable<SpaceDto>> Handle(GetSpacesByClubIdQuery
request, CancellationToken cancellationToken)
    {
        IEnumerable<Space> spaces = await context.Spaces.Where(space =>
space.ClubId.Equals(request.ClubId)).ToListAsync(cancellationToken);
        return mapper.Map<IEnumerable<SpaceDto>>(spaces);
    }
}
```

GetSpacesByClubIdQueryHandler.cs

```
public class CreateSpaceUseCommand : IRequest<int>
{
    public int SpaceId { get; set; }
    public int ResponsibleUserId { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }
    public IEnumerable<int> SpaceUsersIds { get; set; } = null!;
}
```

CreateSpaceUseCommand.cs

```
internal class CreateSpaceUseCommandHandler :
IRequestHandler<CreateSpaceUseCommand, int>
{
    private readonly IApplicationDbContext context;

    public CreateSpaceUseCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<int> Handle(CreateSpaceUseCommand request,
CancellationToken cancellationToken)
    {
        SpaceUse spaceUse = new()
        {
            SpaceId = request.SpaceId,
            CreatorUserId = request.ResponsibleUserId,
            Start = request.Start,
            End = request.End,
            SpaceUsers = new
List<SpaceUser>(CreateSpaceUsersSequence(request.SpaceUsersIds))
        };

        await context.SpaceUses.AddAsync(spaceUse, cancellationToken);
        await context.SaveChangesAsync(cancellationToken);

        return spaceUse.Id;
    }

    private static IEnumerable<SpaceUser>
CreateSpaceUsersSequence(IEnumerable<int> spaceUsersIds)
    {
        return spaceUsersIds.Select(id => new SpaceUser() { UserId = id });
    }
}
```

CreateSpaceUseCommandHandler.cs

```
public class UpdateSpaceUseCommand : IRequest<Unit>
{
```

```
    public int Id { get; set; }
    public int SpaceId { get; set; }
    public int ResponsibleUserId { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }
    public IEnumerable<int> SpaceUsersIds { get; set; } = null!;
}
```

UpdateSpaceUserCommand.cs

```
internal class UpdateSpaceUseCommandHandler :
IRequestHandler<UpdateSpaceUseCommand, Unit>
{
    private readonly IApplicationDbContext context;

    public UpdateSpaceUseCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<Unit> Handle(UpdateSpaceUseCommand request,
CancellationToken cancellationToken)
    {
        SpaceUse spaceUse = await context.SpaceUses
            .Include(spaceUse => spaceUse.SpaceUsers)
            .SingleAsync(spaceUse => spaceUse.Id.Equals(request.Id),
cancellationToken);

        spaceUse.SpaceId = request.SpaceId;
        spaceUse.CreatorUserId = request.ResponsibleUserId;
        spaceUse.Start = request.Start;
        spaceUse.End = request.End;

        context.SpaceUsers.RemoveRange(spaceUse.SpaceUsers);

        spaceUse.SpaceUsers = new
List<SpaceUser>(CreateSpaceUsersSequence(request.SpaceUsersIds));

        context.SpaceUses.Update(spaceUse);
        await context.SaveChangesAsync(cancellationToken);

        return Unit.Value;
    }

    private static IEnumerable<SpaceUser>
CreateSpaceUsersSequence(IEnumerable<int> spaceUsersIds)
    {
        return spaceUsersIds.Select(id => new SpaceUser() { UserId = id });
    }
}
```

UpdateSpaceUseCommandHandler.cs

```
public class GetSpaceUseByIdQuery : IRequest<SpaceUseDto>
{
    public int Id { get; }

    public GetSpaceUseByIdQuery(int id)
    {
        Id = id;
    }
}
```

GetSpaceUseByIdQuery.cs

```
internal class GetSpaceUseByIdQueryHandler :
IRequestHandler<GetSpaceUseByIdQuery, SpaceUseDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetSpaceUseByIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<SpaceUseDto> Handle(GetSpaceUseByIdQuery request,
CancellationToken cancellationToken)
    {
        SpaceUse? spaceUse = await context.SpaceUses
            .Include(spaceUse => spaceUse.SpaceUsers)
            .ThenInclude(spaceUser => spaceUser.User)
            .SingleOrDefaultAsync(spaceUse => spaceUse.Id.Equals(request.Id),
cancellationToken);

        return mapper.Map<SpaceUseDto>(spaceUse);
    }
}
```

GetSpaceUseByIdQueryHandler.cs

```
public class GetSpaceUsesBySpaceIdQuery : IRequest<IEnumerable<SpaceUseDto>>
{
    public int SpaceId { get; }

    public GetSpaceUsesBySpaceIdQuery(int spaceId)
    {
        SpaceId = spaceId;
    }
}
```

GetSpaceUsesBySpaceIdQuery.cs

```
internal class GetSpaceUsesBySpaceIdQueryHandler :
IRequestHandler<GetSpaceUsesBySpaceIdQuery, IEnumerable<SpaceUseDto>>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetSpaceUsesBySpaceIdQueryHandler(IApplicationDbContext context,
IMapper mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<IEnumerable<SpaceUseDto>>
Handle(GetSpaceUsesBySpaceIdQuery request, CancellationToken cancellationToken)
    {
        IEnumerable<SpaceUse> spaceUses = await context.SpaceUses
            .Include(spaceUse => spaceUse.SpaceUsers)
            .ThenInclude(spaceUser => spaceUser.User)
            .Where(spaceUse => spaceUse.SpaceId.Equals(request.SpaceId))
            .ToListAsync(cancellationToken);

        return mapper.Map<IEnumerable<SpaceUseDto>>(spaceUses);
    }
}
```

GetSpaceUsesBySpaceIdQueryHandler.cs

```
public class CreateUserCommand : IRequest<int>
{
    public string Name { get; set; } = null!;
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
    public int ClubId { get; set; }
    public int RoleId { get; set; }
}
```

CreateUserCommand.cs

```
internal class CreateUserCommandHandler : IRequestHandler<CreateUserCommand,
int>
{
    private readonly IApplicationDbContext context;
    private readonly IIdentityService identityService;

    public CreateUserCommandHandler(IApplicationDbContext context,
IIdentityService identityService)
    {
        this.context = context;
        this.identityService = identityService;
    }
```

```
    public async Task<int> Handle(CreateUserCommand request, CancellationToken
cancellationToken)
    {
        UserCredentialsDto credentials = new()
        {
            Email = request.Email,
            Password = request.Password
        };

        string identityProviderUserId = await
identityService.CreateUserAsync(credentials);

        User user = new()
        {
            Name = request.Name,
            Email = request.Email,
            IdentityProviderUserId = identityProviderUserId,
            LinkedClubs = new List<ClubUser>()
            {
                new ClubUser()
                {
                    RoleId = request.RoleId,
                    ClubId = request.ClubId
                }
            },
            Settings = new UserSettings()
        };

        await context.Users.AddAsync(user, cancellationToken);
        await context.SaveChangesAsync(cancellationToken);

        return user.Id;
    }
}
```

CreateUserCommandHandler.cs

```
public class UpdateUserCommand : IRequest<Unit>
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
}
```

UpdateUserCommand.cs

```
internal class UpdateUserCommandHandler : IRequestHandler<UpdateUserCommand,
Unit>
{
    private readonly IApplicationDbContext context;
```

```
    public UpdateUserCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<Unit> Handle(UpdateUserCommand request, CancellationToken
cancellationToken)
    {
        User user = await context.Users.SingleAsync(user =>
user.Id.Equals(request.Id), cancellationToken);

        user.Name = request.Name;

        context.Users.Update(user);
        await context.SaveChangesAsync(cancellationToken);

        return Unit.Value;
    }
}
```

UpdateUserCommandHandler.cs

```
public class UpdateUserSettingsCommand : IRequest<Unit>
{
    public int UserId { get; set; }
    public string Idiom { get; set; } = null!;
    public string DateFormat { get; set; } = null!;
    public string TimeFormat { get; set; } = null!;
    public string TimeZone { get; set; } = null!;
}
```

UpdateUserSettingsCommand.cs

```
internal class UpdateUserSettingsCommandHandler :
IRequestHandler<UpdateUserSettingsCommand, Unit>
{
    private readonly IApplicationDbContext context;

    public UpdateUserSettingsCommandHandler(IApplicationDbContext context)
    {
        this.context = context;
    }

    public async Task<Unit> Handle(UpdateUserSettingsCommand request,
CancellationToken cancellationToken)
    {
        UserSettings settings = await context.UserSettings.SingleAsync(settings
=> settings.UserId.Equals(request.UserId), cancellationToken);

        settings.Idiom = request.Idiom;
        settings.DateFormat = request.DateFormat;
```

```
        settings.TimeFormat = request.TimeFormat;
        settings.TimeZone = request.TimeZone;

        context.UserSettings.Update(settings);
        await context.SaveChangesAsync(cancellationToken);

        return Unit.Value;
    }
}
```

UpdateUserSettingsCommandHandler.cs

```
public class GetUserByIdQuery : IRequest<UserDto>
{
    public int Id { get; }

    public GetUserByIdQuery(int id)
    {
        Id = id;
    }
}
```

GetUserByIdQuery.cs

```
internal class GetUserByIdQueryHandler : IRequestHandler<GetUserByIdQuery,
UserDto>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetUserByIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<UserDto> Handle(GetUserByIdQuery request,
CancellationToken cancellationToken)
    {
        User? user = await context.Users
            .Include(user => user.LinkedClubs)
            .Include(user => user.Settings)
            .SingleOrDefaultAsync(user => user.Id.Equals(request.Id),
cancellationToken);
        return mapper.Map<UserDto>(user);
    }
}
```

GetUserByIdQueryHandler.cs

```
public class GetUsersByClubIdQuery : IRequest<IEnumerable<SimpleUserDto>>
{
    public int ClubId { get; }

    public GetUsersByClubIdQuery(int clubId)
    {
        ClubId = clubId;
    }
}
```

GetUsersByClubIdQuery.cs

```
internal class GetUsersByClubIdQueryHandler :
IRequestHandler<GetUsersByClubIdQuery, IEnumerable<SimpleUserDto>>
{
    private readonly IApplicationDbContext context;
    private readonly IMapper mapper;

    public GetUsersByClubIdQueryHandler(IApplicationDbContext context, IMapper
mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

    public async Task<IEnumerable<SimpleUserDto>> Handle(GetUsersByClubIdQuery
request, CancellationToken cancellationToken)
    {
        IEnumerable<User> users = await context.Users
            .Include(user => user.LinkedClubs)
            .Where(user => user.LinkedClubs != null &&
user.LinkedClubs.Any(clubUser => clubUser.ClubId.Equals(request.ClubId)))
            .ToListAsync(cancellationToken);

        return mapper.Map<IEnumerable<SimpleUserDto>>(users);
    }
}
```

GetUsersByClubIdQueryHandler.cs

```
public interface IApplicationDbContext
{
    DbSet<Club> Clubs { get; }
    DbSet<Space> Spaces { get; }
    DbSet<SpaceUse> SpaceUses { get; }
    DbSet<SpaceUser> SpaceUsers { get; }
    DbSet<User> Users { get; }
    DbSet<ClubUser> ClubUsers { get; }
    DbSet<UserSettings> UserSettings { get; }

    Task<int> SaveChangesAsync(CancellationToken cancellationToken);
```

```
}
```

IApplicationDbContext.cs

```
public interface IDateTime
{
    DateTime Now { get; }
}
```

IDateTime.cs

```
public interface IIdentityService
{
    Task<string> CreateUserAsync(UserCredentialsDto dto);
    Task<AuthResponseDto> TryLoginAsync(UserLoginDto dto);
}
```

IIdentityService.cs

```
public class ApplicationMappingProfile : Profile
{
    public ApplicationMappingProfile()
    {
        CreateMap<Club, ClubDto>();

        CreateMap<Space, SpaceDto>();

        CreateMap<User, UserDto>()
            .ForMember(dto => dto.LinkedClubs, action => action.MapFrom(user =>
user.LinkedClubs))
            .ForMember(dto => dto.Settings, action => action.MapFrom(user =>
user.Settings));

        CreateMap<User, SimpleUserDto>();

        CreateMap<ClubUser, ClubUserDto>();

        CreateMap<SpaceUse, SpaceUseDto>()
            .ForMember(dto => dto.Users, action => action.MapFrom(spaceUse =>
spaceUse.SpaceUsers));

        CreateMap<SpaceUser, SimpleUserDto>()
            .ForMember(dto => dto.Id, action => action.MapFrom(spaceUser =>
spaceUser.UserId))
            .ForMember(dto => dto.Name, action => action.MapFrom(spaceUser =>
spaceUser.User.Name));

        CreateMap<SpaceUser, SpaceUserDto>();

        CreateMap<UserSettings, UserSettingsDto>();
    }
```

```
}
```

ApplicationMappingProfile.cs

```
public abstract class BaseResponseDto
{
    public bool IsSuccess { get; set; }
    public string? Message { get; set; }
}
```

BaseResponseDto.cs

```
public class AuthResponseDto : BaseResponseDto
{
    public AuthTokensDto? Tokens { get; set; }
}
```

AuthResponseDto.cs

```
public class AuthTokensDto
{
    public string? IdToken { get; set; }
    public string? AccessToken { get; set; }
    public int ExpiresIn { get; set; }
    public string? RefreshToken { get; set; }
}
```

AuthTokensDto.cs

```
public class UserCredentialsDto
{
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
}
```

UserCredentialsDto.cs

```
public class UserLoginDto
{
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
}
```

UserLoginDto.cs

```
public static class ConfigureServices
{
    public static IServiceCollection AddApplicationServices(this
IServiceCollection services)
    {
        services.AddAutoMapper(typeof(ApplicationMappingProfile));
```

```
        services.AddMediatR(Assembly.GetExecutingAssembly());

        return services;
    }
}
```

ConfigureServices.cs

```
public class UserSettings
{
    [Key]
    public int UserId { get; set; }
    public string Idiom { get; set; } = "English (US)";
    public string DateFormat { get; set; } = "YYYY-MM-DD";
    public string TimeFormat { get; set; } = "h:mm a";
    public string TimeZone { get; set; } = "+0000";

    public User User { get; set; } = null!;
}
```

UserSettings.cs

```
public class User : BaseAuditableEntity
{
    public string Name { get; set; } = null!;
    public string Email { get; set; } = null!;
    public string IdentityProviderUserId { get; set; } = null!;
    public string? DominantSide { get; set; }

    public UserSettings Settings { get; set; } = null!;
    public IEnumerable<ClubUser>? LinkedClubs { get; set; }
    public IEnumerable<Club>? OwnedClubs { get; set; }
    public IEnumerable<SpaceUse>? SpaceUses { get; set; }
    public IEnumerable<SpaceUser>? SpaceUsers { get; set; }
}
```

User.cs

```
public class SpaceUser : BaseAuditableEntity
{
    public int SpaceUseId { get; set; }
    public int UserId { get; set; }

    public SpaceUse SpaceUse { get; set; } = null!;
    public User User { get; set; } = null!;
}
```

SpaceUser.cs

```
public class SpaceUse : BaseAuditableEntity
{
```

```
    public int SpaceId { get; set; }
    public int CreatorUserId { get; set; }
    public int? CancellerUserId { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }
    public bool IsLesson { get; set; }
    public double? LessonPrice { get; set; }
    public string? LessonPriceCurrency { get; set; }

    public Space Space { get; set; } = null!;
    public User ResponsibleUser { get; set; } = null!;
    public IEnumerable<SpaceUser> SpaceUsers { get; set; } = null!;
}
```

SpaceUse.cs

```
public class Space : BaseAuditableEntity
{
    public string Name { get; set; } = null!;
    public int ClubId { get; set; }
    public int MaximumConcurrentUses { get; set; } = 1;
    public int MinimumUseDurationInMinutes { get; set; } = 30;
    public DateTime? OpeningTime { get; set; }
    public DateTime? ClosingTime { get; set; }

    public Club Club { get; set; } = null!;
    public IEnumerable<SpaceUse>? SpaceUses { get; set; }
}
```

Space.cs

```
public class ClubUser : BaseAuditableEntity
{
    public int UserId { get; set; }
    public int RoleId { get; set; }
    public int ClubId { get; set; }

    public User User { get; set; } = null!;
    public ClubRole Role { get; set; } = null!;
    public Club Club { get; set; } = null!;
}
```

ClubUser.cs

```
public class ClubRole
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public string Description { get; set; } = null!;
    public string ScopeType { get; set; } = null!;
```

```
    public IEnumerable<ClubUser>? UserRoles { get; set; }
}
```

ClubRole.cs

```
public class Club : BaseAuditableEntity
{
    public string Name { get; set; } = null!;
    public int OwnerId { get; set; }

    public User Owner { get; set; } = null!;
    public IEnumerable<Space>? Spaces { get; set; }
    public IEnumerable<ClubUser> ClubUserLinks { get; set; } = null!;
}
```

Club.cs

```
public class UserSettingsDto
{
    public int UserId { get; set; }
    public string Idiom { get; set; } = null!;
    public string DateFormat { get; set; } = null!;
    public string TimeFormat { get; set; } = null!;
    public string TimeZone { get; set; } = null!;
}
```

UserSettingsDto.cs

```
public class UserDto
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public string? ImageSignedUrl { get; set; }
    public UserSettingsDto Settings { get; set; } = null!;
    public IEnumerable<ClubUserDto>? LinkedClubs { get; set; }
}
```

UserDto.cs

```
public class SpaceUserDto : BaseAuditableEntity
{
    public int SpaceUseId { get; set; }
    public int UserId { get; set; }
    public SimpleUserDto User { get; set; } = null!;
}
```

SpaceUserDto.cs

```
public class SpaceUseDto : BaseAuditableEntity
{
    public int SpaceId { get; set; }
```

```
    public int ResponsibleUserId { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }
    public IEnumerable<SimpleUserDto> Users { get; set; } = null!;
}
```

SpaceUseDto.cs

```
public class SpaceDto
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public int ClubId { get; set; }
}
```

SpaceDto.cs

```
public class SimpleUserDto
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
}
```

SimpleUserDto.cs

```
public class ClubUserDto
{
    public int UserId { get; set; }
    public int RoleId { get; set; }
    public int ClubId { get; set; }
}
```

ClubUserDto.cs

```
public class ClubDto
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public int OwnerId { get; set; }
}
```

ClubDto.cs

```
public enum Roles
{
    SystemAdministrator = 1,
    SystemUser = 2,
    ClubOwner = 3,
    ClubProfessionalAffiliate = 4,
    ClubSimpleAffiliate = 5
}
```

Roles.cs

```
public abstract class BaseEntity
{
    public int Id { get; set; }
}
```

BaseEntity.cs

```
public abstract class BaseAuditableEntity : BaseEntity
{
    public DateTime CreateTime { get; set; }
    public DateTime UpdateTime { get; set; }
}
```

BaseAuditableEntity.cs

```
internal class AmazonCognitoConfigurationDto
{
    public string? AccessKeyID { get; set; }
    public string? SecretAccessKey { get; set; }
    public string? Region { get; set; }
    public string? UserPoolID { get; set; }
    public string? AppClientId { get; set; }
}
```

AmazonCognitoConfigurationDto.cs

```
internal class AmazonCognitoIdentityService : IIdentityService
{
    private readonly AmazonCognitoIdentityProviderClient provider;
    private readonly CognitoUserPool userPool;
    private readonly AmazonCognitoConfigurationDto configuration;

    public AmazonCognitoIdentityService(AmazonCognitoConfigurationDto
configuration)
    {
        provider = new
AmazonCognitoIdentityProviderClient(configuration.AccessKeyID,
configuration.SecretAccessKey,
RegionEndpoint.GetBySystemName(configuration.Region));
        userPool = new(configuration.UserPoolID, configuration.AppClientId,
provider);

        this.configuration = configuration;
    }

    public async Task<string> CreateUserAsync(UserCredentialsDto dto)
    {
        SignUpRequest request = new()
        {
```

```
            ClientId = configuration.AppClientId,
            Username = dto.Email,
            Password = dto.Password,
            UserAttributes = new List<AttributeType>()
            {
                new AttributeType()
                {
                    Name = "email",
                    Value = dto.Email
                }
            }
        };

        SignUpResponse response = await provider.SignUpAsync(request);

        return response.UserSub;
    }

    public async Task<AuthResponseDto> TryLoginAsync(UserLoginDto userLoginDto)
    {
        CognitoUser user = new(userLoginDto.Email, configuration.AppClientId,
userPool, provider);

        InitiateSrpAuthRequest authRequest = new()
        {
            Password = userLoginDto.Password
        };

        AuthFlowResponse authResponse = await
user.StartWithSrpAuthAsync(authRequest);

        return new AuthResponseDto()
        {
            IsSuccess = true,
            Tokens = new AuthTokensDto()
            {
                IdToken = authResponse.AuthenticationResult.IdToken,
                AccessToken = authResponse.AuthenticationResult.AccessToken,
                ExpiresIn = authResponse.AuthenticationResult.ExpiresIn,
                RefreshToken = authResponse.AuthenticationResult.RefreshToken
            }
        };
    }
}
```

AmazonCognitoIdentityService.cs

```
internal class ClubConfiguration : IEntityTypeConfiguration<Club>
{
    public void Configure(EntityTypeBuilder<Club> builder)
    {
```

```
        builder.Property(x => x.Name)
            .HasMaxLength(255)
            .IsRequired();
    }
}
```

ClubConfiguration.cs

```
internal class RoleConfiguration : IEntityTypeConfiguration<ClubRole>
{
    public void Configure(EntityTypeBuilder<ClubRole> builder)
    {
        builder.Property(x => x.Name)
            .HasMaxLength(255)
            .IsRequired();

        builder.Property(x => x.Description)
            .HasMaxLength(255)
            .IsRequired();

        builder.Property(x => x.ScopeType)
            .HasMaxLength(255)
            .IsRequired();
    }
}
```

RoleConfiguration.cs

```
internal class SpaceConfiguration : IEntityTypeConfiguration<Space>
{
    public void Configure(EntityTypeBuilder<Space> builder)
    {
        builder.Property(x => x.Name)
            .HasMaxLength(255)
            .IsRequired();
    }
}
```

SpaceConfiguration.cs

```
internal class UserConfiguration : IEntityTypeConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.Property(x => x.Name)
            .HasMaxLength(255)
            .IsRequired();

        builder.Property(x => x.Email)
            .HasMaxLength(255)
            .IsRequired();
```

```
        builder.Property(x => x.IdentityProviderUserId)
            .HasMaxLength(255)
            .IsRequired();
    }
}
```

UserConfiguration.cs

```
internal class UserSettingsConfiguration :
IEntityTypeConfiguration<UserSettings>
{
    public void Configure(EntityTypeBuilder<UserSettings> builder)
    {
        builder.Property(x => x.Idiom)
            .HasMaxLength(32)
            .IsRequired();

        builder.Property(x => x.DateFormat)
            .HasMaxLength(32)
            .IsRequired();

        builder.Property(x => x.TimeFormat)
            .HasMaxLength(32)
            .IsRequired();

        builder.Property(x => x.TimeZone)
            .HasMaxLength(5)
            .IsRequired();
    }
}
```

UserSettingsConfiguration.cs

```
internal class ApplicationDbContextFactory :
IDesignTimeDbContextFactory<ApplicationDbContext>
{
    public ApplicationDbContext CreateDbContext(string[] args)
    {
        ArgumentsDictionary arguments = new(args);

        DbContextOptionsBuilder<ApplicationDbContext> optionsBuilder = new();
        optionsBuilder.UseMySQL(CreateConnectionString(arguments));

        AuditableEntitySaveChangesInterceptor interceptor = new(new
DateTimeService());

        return new ApplicationDbContext(optionsBuilder.Options, interceptor);
    }

    private static string CreateConnectionString(ArgumentsDictionary arguments)
```

```
    {
        return
$"Server={arguments["dbHost"]};DataBase={arguments["dbSchema"]};Uid={arguments["
dbUser"]};Pwd={arguments["dbPassword"]}";
    }
}
```

ApplicationDbContextFactory.cs

```
internal class ArgumentsDictionary : Dictionary<string, string>
{
    public ArgumentsDictionary(string[] args) : base()
    {
        if (args.Length % 2 != 0)
        {
            throw new Exception("Wrong number of arguments... Must be
\"--argumentName\" followed by \"argumentValue\"");
        }

        for (int i = 0; i < args.Length; i += 2)
        {
            string key = args[i].Split("--",
StringSplitOptions.RemoveEmptyEntries)[0];
            Add(key, args[i + 1]);
        }
    }
}
```

ArgumentsHolder.cs

```
internal class MysqlEntityFrameworkDesignTimeServices : IDesignTimeServices
{
    public void ConfigureDesignTimeServices(IServiceCollection
serviceCollection)
    {
        serviceCollection.AddEntityFrameworkMySQL();
        new EntityFrameworkRelationalDesignServicesBuilder(serviceCollection)
            .TryAddCoreServices();
    }
}
```

MysqlEntityFrameworkDesignTimeServices.cs

```
internal class AuditableEntitySaveChangesInterceptor : SaveChangesInterceptor
{
    private readonly IDateTime dateTime;

    public AuditableEntitySaveChangesInterceptor(IDateTime dateTime)
    {
        this.dateTime = dateTime;
    }
```

```
    public override InterceptionResult<int> SavingChanges(DbContextEventData
eventData, InterceptionResult<int> result)
    {
        UpdateEntities(eventData.Context);

        return base.SavingChanges(eventData, result);
    }

    public override ValueTask<InterceptionResult<int>>
SavingChangesAsync(DbContextEventData eventData, InterceptionResult<int> result,
CancellationToken cancellationToken = default)
    {
        UpdateEntities(eventData.Context);

        return base.SavingChangesAsync(eventData, result, cancellationToken);
    }

    public void UpdateEntities(DbContext? context)
    {
        if (context == null)
        {
            return;
        }

        foreach
(Microsoft.EntityFrameworkCore.ChangeTracking.EntityEntry<BaseAuditableEntity>
entry in context.ChangeTracker.Entries<BaseAuditableEntity>())
        {
            if (entry.State == EntityState.Added)
            {
                entry.Entity.CreateTime = dateTime.Now.ToUniversalTime();
            }

            if (entry.State == EntityState.Added || entry.State ==
EntityState.Modified || entry.HasChangedOwnedEntities())
            {
                entry.Entity.UpdateTime = dateTime.Now.ToUniversalTime();
            }
        }
    }
}
```

AuditableEntitySaveChangesInterceptor.cs

```
internal static class EntityEntryExtensions
{
    public static bool HasChangedOwnedEntities(this EntityEntry entry)
    {
        return entry.References.Any(r =>
                r.TargetEntry != null &&
```

```
                r.TargetEntry.Metadata.IsOwned() &&
                (r.TargetEntry.State == EntityState.Added || r.TargetEntry.State
== EntityState.Modified));
    }
}
```

EntityEntryExtesions.cs

```
internal class ApplicationDbContext : DbContext, IApplicationDbContext
{
    public DbSet<Club> Clubs => Set<Club>();
    public DbSet<Space> Spaces => Set<Space>();
    public DbSet<SpaceUse> SpaceUses => Set<SpaceUse>();
    public DbSet<SpaceUser> SpaceUsers => Set<SpaceUser>();
    public DbSet<ClubRole> Roles => Set<ClubRole>();
    public DbSet<User> Users => Set<User>();
    public DbSet<ClubUser> ClubUsers => Set<ClubUser>();
    public DbSet<UserSettings> UserSettings => Set<UserSettings>();

    private readonly AuditableEntitySaveChangesInterceptor
auditableEntitySaveChangesInterceptor;

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options,
AuditableEntitySaveChangesInterceptor interceptor) : base(options)
    {
        auditableEntitySaveChangesInterceptor = interceptor;
    }

    public override async Task<int> SaveChangesAsync(CancellationToken
cancellationToken = default)
    {
        return await base.SaveChangesAsync(cancellationToken);
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {

builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
        builder.ToSnakeCase();

        base.OnModelCreating(builder);
    }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.AddInterceptors(auditableEntitySaveChangesInterceptor);
    }
}
```

ApplicationDbContext.cs

```
internal class DateTimeService : IDateTime
{
    public DateTime Now => DateTime.Now;
}
```

DateTimeService.cs

```
public static class ConfigureServices
{
    public static IServiceCollection AddInfrastructureServices(this
IServiceCollection services, IConfiguration configuration)
    {
        string connectionString = CreateConnectionString(configuration);
        services.AddDbContext<IApplicationDbContext,
ApplicationDbContext>(options => options.UseMySQL(connectionString));

        AmazonCognitoConfigurationDto cognitoConfiguration =
CreateCognitoConfiguration(configuration);
        services.AddScoped<IIdentityService,
AmazonCognitoIdentityService>((provider) => new
AmazonCognitoIdentityService(cognitoConfiguration));
        services.AddScoped<AuditableEntitySaveChangesInterceptor>();

        services.AddTransient<IDateTime, DateTimeService>();

        services.AddCognitoIdentity();

        services.AddAuthentication(options =>
        {
            options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
        }).AddJwtBearer(options =>
        {
            options.Authority =
$"https://cognito-idp.{cognitoConfiguration.Region}.amazonaws.com/{cognitoConfig
uration.UserPoolID}";
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                ValidateAudience = false
            };
        });

        return services;
    }

    private static string CreateConnectionString(IConfiguration configuration)
    {
        string dbHost = configuration["Database:Host"];
```

```
        string dbSchema = configuration["Database:Schema"];
        string dbUser = configuration["Database:User"];
        string dbPassword = configuration["Database:Password"];

        return
$"Server={dbHost};DataBase={dbSchema};Uid={dbUser};Pwd={dbPassword}";
    }

    private static AmazonCognitoConfigurationDto
CreateCognitoConfiguration(IConfiguration configuration)
    {
        return new AmazonCognitoConfigurationDto()
        {
            AccessKeyID = configuration["AWS:AccessKeyID"],
            SecretAccessKey = configuration["AWS:SecretAccessKey"],
            Region = configuration["AWS:Region"],
            UserPoolID = configuration["AWS:Cognito:UserPoolID"],
            AppClientId = configuration["AWS:Cognito:AppClientId"]
        };
    }
}
```

ConfigureServices.cs

# APÊNDICE C - Código Fonte do Aplicativo

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        CultureService.LoadCulture();

        MauiAppBuilder builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

        builder.Services.AddSingleton<LoginPage>();
        builder.Services.AddSingleton<HomePage>();
        builder.Services.AddSingleton<MyReservationsPage>();
        builder.Services.AddSingleton<SettingsPage>();

        builder.Services.AddSingleton<LoginPageViewModel>();
        builder.Services.AddSingleton<HomePageViewModel>();
        builder.Services.AddSingleton<MyReservationsPageViewModel>();
        builder.Services.AddSingleton<SettingsPageViewModel>();

        return builder.Build();
    }
}
```

MauiProgram.cs

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="Mobile.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:resources="clr-namespace:Mobile.Resources.Strings"
    xmlns:views="clr-namespace:Mobile.Views"
    Shell.TabBarIsVisible="False">

    <ShellContent
        Title="Home"
        ContentTemplate="{DataTemplate views:LoginPage}"
        FlyoutItemIsVisible="False"
        Shell.FlyoutBehavior="Disabled"
        Route="LoginPage"/>

    <FlyoutItem
        Route="HomePage"
```

```
        FlyoutDisplayOptions="AsMultipleItems">
        <ShellContent
            Title="{x:Static resources:Strings.HomePageTitle}"
            ContentTemplate="{DataTemplate views:HomePage}"/>
        <ShellContent
            Title="{x:Static resources:Strings.MyReservationsPageTitle}"
            ContentTemplate="{DataTemplate views:MyReservationsPage}"/>
        <ShellContent
            Title="{x:Static resources:Strings.SettingsPageTitle}"
            ContentTemplate="{DataTemplate views:SettingsPage}"/>
    </FlyoutItem>

</Shell>
```

AppShell.xaml

```
public partial class AppShell : Shell
{
    public AppShell()
    {
        InitializeComponent();
    }
}
```

AppShell.xaml.cs

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Mobile"
             x:Class="Mobile.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
                <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

App.xaml

```
public partial class App : IApplication
{
    public App()
    {
        InitializeComponent();

        MainPage = new AppShell();
    }
```

```
}
```

App.xaml.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:resources="clr-namespace:Mobile.Resources.Strings"
             Shell.NavBarIsVisible="False"
             x:Class="Mobile.Views.LoginPage"
             Title="{x:Static resources:Strings.LoginPageTitle}">
    <Grid BackgroundColor="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="50"/>
        </Grid.RowDefinitions>

        <Frame
            HorizontalOptions="Center"
            VerticalOptions="Center">
            <Label
                Text="EasyTennis Logo"
                HorizontalTextAlignment="Center"
                VerticalTextAlignment="Center"
                FontAttributes="Bold"
                FontSize="Title"/>
        </Frame>

        <Grid
            RowSpacing="5"
            Grid.Row="1"
            Margin="20,20,20,0">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="40"/>
                <RowDefinition Height="40"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Label
                Grid.Row="1"
                Text="{x:Static resources:Strings.LoginWelcome}"
                HorizontalOptions="Center"
                FontSize="Title"
                FontAttributes="Bold"
                Padding="0"
                Margin="0"/>
            <Label
```

```
                Grid.Row="2"
                Text="{x:Static resources:Strings.SignInMessage}"
                HorizontalOptions="Center"
                FontSize="Subtitle"/>

            <StackLayout
                Grid.Row="3"
                HorizontalOptions="CenterAndExpand"
                Orientation="Vertical">
                <Frame
                    BackgroundColor="Transparent"
                    BorderColor="{StaticResource Primary}"
                    Padding="0"
                    HorizontalOptions="Fill"
                    CornerRadius="30">
                    <StackLayout
                        Orientation="Horizontal">
                        <Frame
                            BackgroundColor="SkyBlue"
                            HeightRequest="40"
                            WidthRequest="40"
                            CornerRadius="30"
                            Padding="0"
                            Margin="5">
                            <Image

Source="https://cdn4.iconfinder.com/data/icons/glyphs/24/icons_user2-256.png"
                                Aspect="AspectFill"
                                Margin="0"/>
                        </Frame>
                        <Entry
                            Text="{Binding UserName}"
                            Placeholder="{x:Static resources:Strings.Username}"
                            TextColor="Black"
                            FontAttributes="Bold"
                            VerticalOptions="Center"
                            HorizontalOptions="FillAndExpand"
                            Margin="0,0,20,0"/>
                    </StackLayout>
                </Frame>
                <Frame
                    BackgroundColor="Transparent"
                    BorderColor="{StaticResource Primary}"
                    Padding="0"
                    HorizontalOptions="Fill"
                    CornerRadius="30"
                    Margin="0,15,0,0">
                    <StackLayout
                        Orientation="Horizontal">
                        <Frame
                            BackgroundColor="SkyBlue"
```

```
                            HeightRequest="40"
                            WidthRequest="40"
                            CornerRadius="30"
                            Padding="0"
                            Margin="5">
                            <Image

Source="https://cdn0.iconfinder.com/data/icons/basic-ui-elements-round/700/09_lo
ck-256.png"
                                Aspect="AspectFill"
                                Margin="0"/>
                        </Frame>
                        <Entry
                            Text="{Binding Password}"
                            Placeholder="{x:Static resources:Strings.Password}"
                            TextColor="Black"
                            FontAttributes="Bold"
                            VerticalOptions="Center"
                            HorizontalOptions="FillAndExpand"
                            Margin="0,0,20,0"
                            IsPassword="True"/>
                    </StackLayout>
                </Frame>

                <StackLayout
                    Margin="0,10,0,0"
                    Padding="0"
                    Orientation="Horizontal">
                    <CheckBox IsChecked="False"/>
                    <Label
                        Text="{x:Static
resources:Strings.RememberMeCheckboxLabel}"
                        FontSize="Small"
                        VerticalTextAlignment="Center"
                        HorizontalTextAlignment="Center"/>
                    <Label
                        Text="{x:Static resources:Strings.ForgotPasswordLabel}"
                        TextColor="{StaticResource Tertiary}"
                        FontAttributes="Bold"
                        HorizontalOptions="EndAndExpand"
                        VerticalTextAlignment="Center"
                        HorizontalTextAlignment="Center"/>
                </StackLayout>

                <Button
                    Text="{x:Static resources:Strings.SignInButtonText}"
                    BackgroundColor="{StaticResource Primary}"
                    TextColor="{StaticResource White}"
                    FontAttributes="Bold"
                    CornerRadius="30"
                    WidthRequest="200"
```

```
                    Margin="0,15,0,0"
                    Command="{Binding LoginCommand}"/>
            </StackLayout>
        </Grid>
    </Grid>
</ContentPage>
```

LoginPage.xaml

```
public partial class LoginPage : ContentPage
{
    public LoginPage(LoginPageViewModel loginPageViewModel)
    {
        InitializeComponent();
        BindingContext = loginPageViewModel;
    }
}
```

LoginPage.xaml.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:resources="clr-namespace:Mobile.Resources.Strings"
             x:Class="Mobile.Views.HomePage"
             Title="{x:Static resources:Strings.HomePageTitle}">
    <ScrollView
        Orientation="Vertical">
        <VerticalStackLayout>
            <Image
                HeightRequest="120"
                Source="https://cdn-icons-png.flaticon.com/512/847/847969.png"
                HorizontalOptions="Center"
                VerticalOptions="Center"
                Margin="16,24,16,8"/>
            <Label
                Text="Hello, John Connor!"
                HorizontalOptions="Center"
                HorizontalTextAlignment="Center"
                Margin="16,16,16,32"
                FontSize="Medium"/>
            <Frame
                Style="{StaticResource InfoCard}">
                <VerticalStackLayout
                    Style="{StaticResource InfoCardStack}">
                    <Label
                        Text="{x:Static resources:Strings.IncompleteProfile}"
                        Style="{StaticResource FrameHeaderLabel}"/>
                    <BoxView
                        Style="{StaticResource FramedHorizontalSeparator}"/>
                    <Button
```

```
                        Text="{x:Static resources:Strings.CompleteButtonText}"
                        Style="{StaticResource InfoCardButton}"/>
                </VerticalStackLayout>
            </Frame>
            <Frame
                Style="{StaticResource InfoCard}">
                <VerticalStackLayout
                    Style="{StaticResource InfoCardStack}">
                    <Label
                        Text="{Binding PendingInvites, StringFormat={x:Static
resources:Strings.PendingInvitesCardTitle}}"
                        Style="{StaticResource FrameHeaderLabel}"/>
                    <BoxView
                        Style="{StaticResource FramedHorizontalSeparator}"/>
                    <Button
                        Text="{x:Static resources:Strings.VisualizeButtonText}"
                        Style="{StaticResource InfoCardButton}"/>
                </VerticalStackLayout>
            </Frame>
        </VerticalStackLayout>
    </ScrollView>
</ContentPage>
```

HomePage.xaml

```
public partial class HomePage : ContentPage
{
    public HomePage(HomePageViewModel homePageViewModel)
    {
        InitializeComponent();
        BindingContext = homePageViewModel;
    }
}
```

HomePage.xaml.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:resources="clr-namespace:Mobile.Resources.Strings"
             x:Class="Mobile.Views.MyReservationsPage"
             Title="{x:Static resources:Strings.MyReservationsPageTitle}">
    <Grid>
        <ListView
            ItemsSource="{Binding Reservations}"
            IsGroupingEnabled="True"
            HasUnevenRows="True"
            SelectionMode="None">
            <ListView.GroupHeaderTemplate>
                <DataTemplate>
                    <ViewCell>
```

```
                        <Label
                            Text="{Binding Name}"
                            Style="{StaticResource ListSectionHeaderLabel}"/>
                    </ViewCell>
                </DataTemplate>
            </ListView.GroupHeaderTemplate>
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <Frame
                            Style="{StaticResource ListCardFrame}">
                            <VerticalStackLayout>
                                <StackLayout
                                    Orientation="Horizontal">
                                    <Label
                                        Text="{Binding SpaceName,
StringFormat={x:Static resources:Strings.SpaceNameFieldWithValue}}"/>
                                    <Label
                                        HorizontalOptions="EndAndExpand">
                                        <Label.Text>
                                            <MultiBinding StringFormat="{}{0:t}
~ {1:t}">
                                                <Binding Path="StartTime" />
                                                <Binding Path="EndTime" />
                                            </MultiBinding>
                                        </Label.Text>
                                    </Label>
                                </StackLayout>
                                <Label
                                    Text="{Binding ResponsibleName,
StringFormat={x:Static resources:Strings.ResponsibleNameFieldWithValue}}"/>
                            </VerticalStackLayout>
                        </Frame>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
        <Button
            Grid.Row="0"
            HeightRequest="64"
            WidthRequest="64"
            CornerRadius="32"
            Padding="0"
            Text="+"
            FontSize="48"
            HorizontalOptions="End"
            VerticalOptions="End"
            Margin="0,0,32,32"/>
    </Grid>
</ContentPage>
```

MyReservationsPage.xaml

```
public partial class MyReservationsPage : ContentPage
{
    public MyReservationsPage(MyReservationsPageViewModel viewModel)
    {
        InitializeComponent();
        BindingContext = viewModel;
    }
}
```

MyReservationsPage.xaml.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:resources="clr-namespace:Mobile.Resources.Strings"
             x:Class="Mobile.Views.SettingsPage"
             Title="{x:Static resources:Strings.SettingsPageTitle}">
    <ContentPage.ToolbarItems>
        <ToolbarItem
            Text="{x:Static resources:Strings.SaveToolbarItemText}"
            Command="{Binding SaveCommand}"/>
    </ContentPage.ToolbarItems>
    <VerticalStackLayout
        Style="{StaticResource FormStack}">
        <Label
            Text="{x:Static resources:Strings.CultureFormFieldLabel}"
            Style="{StaticResource FormFieldLabel}"/>
        <Frame
            Style="{StaticResource FormFieldFrame}">
            <Picker
                x:Name="CulturePicker"
                Title="{x:Static resources:Strings.CulturePickerTitle}"
                SelectedItem="{Binding SelectedCulture}"
                ItemsSource="{Binding Cultures}"
                ItemDisplayBinding="{Binding DisplayName}">
            </Picker>
        </Frame>
        <BoxView
            Style="{StaticResource FormFieldSeparator}"/>
        <Label
            Text="{x:Static resources:Strings.TimeZoneFormFieldLabel}"
            Style="{StaticResource FormFieldLabel}"/>
        <Frame
            Style="{StaticResource FormFieldFrame}">
            <Entry
                Text="-0300"/>
        </Frame>
    </VerticalStackLayout>
</ContentPage>
```

SettingsPage.xaml

```
public partial class SettingsPage : ContentPage
{
    public SettingsPage(SettingsPageViewModel settingsPageViewModel)
    {
        InitializeComponent();
        BindingContext = settingsPageViewModel;
    }
}
```

SettingsPage.xaml.cs

```
public partial class BaseViewModel : ObservableObject
{
    [ObservableProperty]
    public bool _isBusy;
    [ObservableProperty]
    public bool _title;
}
```

BaseViewModel.cs

```
public partial class LoginPageViewModel : BaseViewModel
{
    [RelayCommand]
    public async Task Login()
    {
        await Task.Delay(1000);
        await Shell.Current.GoToAsync($"//{nameof(HomePage)}");
    }
}
```

LoginPageViewModel.cs

```
public partial class HomePageViewModel : BaseViewModel
{
    [ObservableProperty]
    private int pendingInvites = 1;
}
```

HomePageViewModel.cs

```
public partial class MyReservationsPageViewModel : BaseViewModel
{
    [ObservableProperty]
    private List<SpaceUseGroup> reservations;

    public MyReservationsPageViewModel()
    {
        string todayString = CultureService.GetString("Today");
        string tomorrowString = CultureService.GetString("Tomorrow");
```

```
        SpaceUseGroup todayGroup = new(todayString, new List<SpaceUseListItem>()
        {
            new SpaceUseListItem()
            {
                SpaceUseId = 1,
                SpaceName = "Court A",
                ResponsibleName = "John Connor",
                StartTime = DateTime.Today.AddHours(14),
                EndTime = DateTime.Today.AddHours(16)
            },
            new SpaceUseListItem()
            {
                SpaceUseId = 2,
                SpaceName = "Court B",
                ResponsibleName = "John Connor",
                StartTime = DateTime.Today.AddHours(16.5),
                EndTime = DateTime.Today.AddHours(17.5)
            },
            new SpaceUseListItem()
            {
                SpaceUseId = 3,
                SpaceName = "Court A",
                ResponsibleName = "T-800",
                StartTime = DateTime.Today.AddHours(18),
                EndTime = DateTime.Today.AddHours(19)
            }
        });

        SpaceUseGroup tomorrowGroup = new(tomorrowString, new
List<SpaceUseListItem>()
        {
            new SpaceUseListItem()
            {
                SpaceUseId = 4,
                SpaceName = "Court A",
                ResponsibleName = "John Connor",
                StartTime = DateTime.Today.AddHours(24 + 11),
                EndTime = DateTime.Today.AddHours(24 + 12)
            }
        });

        reservations = new List<SpaceUseGroup> { todayGroup, tomorrowGroup };
    }
}
```

MyReservationsPageViewModel.cs

```
public partial class SettingsPageViewModel : BaseViewModel
{
    [ObservableProperty]
    private List<CultureInfo> cultures;
```

```
    [ObservableProperty]
    private CultureInfo selectedCulture;

    public SettingsPageViewModel()
    {
        cultures = new List<CultureInfo>(CultureService.AllowedCultures);
    }

    [RelayCommand]
    public void Save()
    {
        CultureService.SetCulture(selectedCulture);
    }
}
```

SettingsPageViewModel.cs

```
public class CultureService
{
    private const string ResourcesPath = "Mobile.Resources.Strings.Strings";

    private static readonly ResourceManager resourceManager = new(ResourcesPath,
Assembly.GetExecutingAssembly());

    internal static List<CultureInfo> AllowedCultures => new()
    {
        CultureInfo.GetCultureInfo("pt-BR"),
        CultureInfo.GetCultureInfo("en-US")
    };

    internal static void LoadCulture()
    {
        CultureInfo selectedCulture = GetConfiguredCulture();
        SetCulture(selectedCulture);
    }

    internal static void SetCulture(CultureInfo culture)
    {
        CultureInfo.CurrentCulture = culture;
        CultureInfo.CurrentUICulture = culture;
        CultureInfo.DefaultThreadCurrentCulture = culture;
        CultureInfo.DefaultThreadCurrentUICulture = culture;
    }

    private static CultureInfo GetConfiguredCulture()
    {
        return AllowedCultures[0];
    }

    internal static string GetString(string name)
```

```
    {
        return resourceManager.GetString(name);
    }
}
```

CultureService.cs

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string"
msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
```

```
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="CompleteButtonText" xml:space="preserve">
    <value>Complete</value>
  </data>
  <data name="CultureFormFieldLabel" xml:space="preserve">
    <value>Culture</value>
  </data>
  <data name="CulturePickerTitle" xml:space="preserve">
    <value>Select a culture</value>
  </data>
  <data name="ForgotPasswordLabel" xml:space="preserve">
    <value>Forgot password</value>
  </data>
  <data name="HomePageTitle" xml:space="preserve">
    <value>Home</value>
  </data>
  <data name="IncompleteProfile" xml:space="preserve">
    <value>Incomplete profile</value>
  </data>
  <data name="LoginPageTitle" xml:space="preserve">
    <value>Login</value>
  </data>
  <data name="LoginWelcome" xml:space="preserve">
    <value>Welcome!</value>
  </data>
  <data name="MyReservationsPageTitle" xml:space="preserve">
    <value>My Reservations</value>
  </data>
```

```
  <data name="Password" xml:space="preserve">
    <value>Password</value>
  </data>
  <data name="PendingInvitesCardTitle" xml:space="preserve">
    <value>Pending invites: {0}</value>
  </data>
  <data name="RememberMeCheckboxLabel" xml:space="preserve">
    <value>Remember me</value>
  </data>
  <data name="ResponsibleNameFieldWithValue" xml:space="preserve">
    <value>Responsible: {0}</value>
  </data>
  <data name="SaveToolbarItemText" xml:space="preserve">
    <value>Save</value>
  </data>
  <data name="SettingsPageTitle" xml:space="preserve">
    <value>Settings</value>
  </data>
  <data name="SignInButtonText" xml:space="preserve">
    <value>Sign in</value>
  </data>
  <data name="SignInMessage" xml:space="preserve">
    <value>Sign in to continue</value>
  </data>
  <data name="SpaceNameFieldWithValue" xml:space="preserve">
    <value>Space: {0}</value>
  </data>
  <data name="TimeZoneFormFieldLabel" xml:space="preserve">
    <value>Time zone</value>
  </data>
  <data name="Today" xml:space="preserve">
    <value>Today</value>
  </data>
  <data name="Tomorrow" xml:space="preserve">
    <value>Tomorrow</value>
  </data>
  <data name="Username" xml:space="preserve">
    <value>Username</value>
  </data>
  <data name="VisualizeButtonText" xml:space="preserve">
    <value>Visualize</value>
  </data>
</root>
```

Strings.resx

```
<?xml version="1.0" encoding="utf-8"?>
<root>
   <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
```

```
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string"
msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
```

```
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="CompleteButtonText" xml:space="preserve">
    <value>Complete</value>
  </data>
  <data name="CultureFormFieldLabel" xml:space="preserve">
    <value>Culture</value>
  </data>
  <data name="CulturePickerTitle" xml:space="preserve">
    <value>Select a culture</value>
  </data>
  <data name="ForgotPasswordLabel" xml:space="preserve">
    <value>Forgot password</value>
  </data>
  <data name="HomePageTitle" xml:space="preserve">
    <value>Home</value>
  </data>
  <data name="IncompleteProfile" xml:space="preserve">
    <value>Incomplete profile</value>
  </data>
  <data name="LoginPageTitle" xml:space="preserve">
    <value>Login</value>
  </data>
  <data name="LoginWelcome" xml:space="preserve">
    <value>Welcome!</value>
  </data>
  <data name="MyReservationsPageTitle" xml:space="preserve">
    <value>My Reservations</value>
  </data>
  <data name="Password" xml:space="preserve">
    <value>Password</value>
  </data>
  <data name="PendingInvitesCardTitle" xml:space="preserve">
    <value>Pending invites: {0}</value>
  </data>
  <data name="RememberMeCheckboxLabel" xml:space="preserve">
    <value>Remember me</value>
  </data>
  <data name="ResponsibleNameFieldWithValue" xml:space="preserve">
    <value>Responsible: {0}</value>
  </data>
```

```
  <data name="SaveToolbarItemText" xml:space="preserve">
    <value>Save</value>
  </data>
  <data name="SettingsPageTitle" xml:space="preserve">
    <value>Settings</value>
  </data>
  <data name="SignInButtonText" xml:space="preserve">
    <value>Sign in</value>
  </data>
  <data name="SignInMessage" xml:space="preserve">
    <value>Sign in to continue</value>
  </data>
  <data name="SpaceNameFieldWithValue" xml:space="preserve">
    <value>Space: {0}</value>
  </data>
  <data name="TimeZoneFormFieldLabel" xml:space="preserve">
    <value>Time zone</value>
  </data>
  <data name="Today" xml:space="preserve">
    <value>Today</value>
  </data>
  <data name="Tomorrow" xml:space="preserve">
    <value>Tomorrow</value>
  </data>
  <data name="Username" xml:space="preserve">
    <value>Username</value>
  </data>
  <data name="VisualizeButtonText" xml:space="preserve">
    <value>Visualize</value>
  </data>
</root>
```

Strings.en-US.resx

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
```

```
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string"
msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
```

```
<data name="CompleteButtonText" xml:space="preserve">
  <value>Completar</value>
</data>
<data name="CultureFormFieldLabel" xml:space="preserve">
  <value>Cultura</value>
</data>
<data name="CulturePickerTitle" xml:space="preserve">
  <value>Selecione uma cultura</value>
</data>
<data name="ForgotPasswordLabel" xml:space="preserve">
  <value>Esqueci minha senha</value>
</data>
<data name="HomePageTitle" xml:space="preserve">
  <value>Início</value>
</data>
<data name="IncompleteProfile" xml:space="preserve">
  <value>Cadastro incompleto</value>
</data>
<data name="LoginPageTitle" xml:space="preserve">
  <value>Login</value>
</data>
<data name="LoginWelcome" xml:space="preserve">
  <value>Bem-vindo!</value>
</data>
<data name="MyReservationsPageTitle" xml:space="preserve">
  <value>Minhas Reservas</value>
</data>
<data name="Password" xml:space="preserve">
  <value>Senha</value>
</data>
<data name="PendingInvitesCardTitle" xml:space="preserve">
  <value>Convites pendentes: {0}</value>
</data>
<data name="RememberMeCheckboxLabel" xml:space="preserve">
  <value>Lembrar de mim</value>
</data>
<data name="ResponsibleNameFieldWithValue" xml:space="preserve">
  <value>Responsável: {0}</value>
</data>
<data name="SaveToolbarItemText" xml:space="preserve">
  <value>Salvar</value>
</data>
<data name="SettingsPageTitle" xml:space="preserve">
  <value>Configurações</value>
</data>
<data name="SignInButtonText" xml:space="preserve">
  <value>Entrar</value>
</data>
<data name="SignInMessage" xml:space="preserve">
  <value>Faça login para continuar</value>
</data>
```

```
  <data name="SpaceNameFieldWithValue" xml:space="preserve">
    <value>Espaço: {0}</value>
  </data>
  <data name="TimeZoneFormFieldLabel" xml:space="preserve">
    <value>Fuso-horário</value>
  </data>
  <data name="Today" xml:space="preserve">
    <value>Hoje</value>
  </data>
  <data name="Tomorrow" xml:space="preserve">
    <value>Amanhã</value>
  </data>
  <data name="Username" xml:space="preserve">
    <value>Nome de usuário</value>
  </data>
  <data name="VisualizeButtonText" xml:space="preserve">
    <value>Visualizar</value>
  </data>
</root>
```

Strings.pt-BR.resx

```
public class SpaceUseListItem
{
    public int SpaceUseId { get; set; }
    public string SpaceName { get; set; } = null!;
    public string ScheduleTime => $"{StartTime:t} ~ {EndTime:t}";
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public string ResponsibleName { get; set; } = null!;
}
```

SpaceUseListItem.cs

```
public class SpaceUseGroup : List<SpaceUseListItem>
{
    public string Name { get; set; } = null!;

    public SpaceUseGroup(string name, List<SpaceUseListItem> uses) : base(uses)
    {
        Name = name;
    }
}
```

SpaceUseGroup.cs

```
public class LoggedUser
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public string Email { get; set; } = null!;
```

```
}
```

LoggedUser.cs