UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Thales Porto Mendes

Sistema de gestão de Zonas de Auto-Salvamento no entorno de barragens de hidroelétricas

Florianópolis
2021

Thales Porto Mendes

Sistema de gestão de Zonas de Auto-Salvamento no entorno de barragens de hidroelétricas

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Rodrigo Castelan Carlson, Dr.
Supervisor: Marcos Hollerweger, Eng.

Florianópolis
2021

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Thales Porto Mendes

Sistema de gestão de Zonas de Auto-Salvamento no entorno de barragens de hidroelétricas

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 22 de fevereiro de 2021.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Rodrigo Castelan Carlson, Dr.
Orientador
UFSC/CTC/DAS

Marcos Meyer Hollerweger, Eng.
Supervisor

Diego Sales, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Fabio Luis Baldissera, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

Inicio agradecendo aos meus pais e minha irmã, que desde de sempre me apoiaram em toda a minha jornada. Vocês sempre foram uma fonte de inspiração e motivação, muito obrigado por toda a confiança depositada em mim e nos meus sonhos. Sou eternamente grato.

Agradeço também, aos meus amigos e colegas que caminharam comigo nessa jornada de erros e acertos. Compartilhamos risadas e noites em claro. Sem vocês tudo seria muito mais difícil.

À empresa parceira e todo o seu time de desenvolvimento, agradeço pela oportunidade e pelo grande apoio. Aprendi muito trabalhando com vocês e admiro bastante a dedicação e competência do trabalho realizado pela empresa.

Por fim, agradeço à Universidade Federal de Santa Catarina e ao Departamento de Automação e Sistemas por me proporcionar tantas experiências incríveis e oportunidades.

RESUMO

Desde a antiguidade as barragens são estruturas que simbolizam o avanço tecnológico das sociedades. A partir delas é possível aprimorar atividades primárias como a agricultura e pecuária, garantir o abastecimento de água e gerar energia elétrica. Deve-se considerar também que existem diversos riscos associados à construção dessas obras; entre eles, o rompimento da barragem que pode causar grandes perdas ambientais, sociais e econômicas. Nestas situações de emergência, a maior fragilidade ocorre nas Zonas de Auto-Salvamento, regiões a jusante das barragens, em que não há tempo hábil para uma intervenção das autoridades competentes. Assim, seus habitantes devem providenciar seus próprio salvamento. Nesse contexto, com o intuito de aumentar a segurança de tais regiões, desenvolve-se um aplicativo mobile com as funcionalidades de alertar os moradores sobre eventuais situações de risco e auxiliá-los na sua evacuação através da sugestão de rotas seguras de fuga. Utiliza-se diversas ferramentas tecnológicas atuais na elaboração do aplicativo, como os frameworks de desenvolvimento React-Native e Django, método de gerenciamento Scrum e diferentes tipos de arquitetura: Cliente-Servidor, Model-Template-View e Redux. Ao final do desenvolvimento o sistema foi avaliado de acordo como padrão internacional de qualidade de software, o que o torna hábil a contribuir na segurança das Zonas de Auto-Salvamento.

Palavras-chave: Barragens. Zonas de Auto-Salvamento. Alertas. Roteamento. Cliente-Servidor. React-Native. Django. Scrum.

ABSTRACT

Since ancient times dams symbolize the technological advancement of societies. By their use, it is possible to improve primary activities such as agriculture and livestock, guarantee water supply and generate electricity. However, it must also be considered that there are several risks associated with their construction. For instance, the rupture of a dam can cause great environmental, social and economic impact. In these emergency situations, the most vulnerable regions are the Self-Rescue Zones, i.e, regions downstream the dams, where there is not enough time for intervention by the competent authorities. Thus, the inhabitants of this zones are responsible for their own rescue. In this context, in order to increase the safety of such regions, a mobile application was developed with the functionality of alerting residents about possible risks and assisting them in their evacuation process through suggestions of safe escape routes. Several current technological tools are used in the development of the application, such as the React-Native and Django development frameworks, scrum management method and different types of architecture: client-server, model-template-view and redux. At the end of the development, the system was evaluated according to an international software quality standard, which affirm its capacity to contribute to the safety of the Self-Rescue Zones.

Keywords: Dams. Self-Rescue Zones. Alerts. Routing. Client-Server. React-Native. Django. Scrum.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANA	Agência Nacional de Águas
API	*Application Programming Interface*
CD	*Continuous Delivery*
CI	*Continuous Integration*
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
MTV	*Model-Template-Views*
PAE	Plano de Ação de Emergência
PFC	Projeto de Fim de Curso
PNSB	Política Nacional de Segurança de Barragens
REST	*Representational State of Transfer*
SMS	*Short Message Service*
SNISB	Sistema Nacional de Informações sobre Segurança de Barragens
UI	*User Interface*
UML	*Unified Modelling Language*
URI	*Uniform Resource Identifiers*
UX	*User Experience*
ZAS	Zonas de Auto-Salvamento

SUMÁRIO

1 INTRODUÇÃO

Desde a antiguidade, dispor de água doce de boa qualidade é uma condição básica para o desenvolvimento da sociedade. Ao aprender a construir barragens com seus reservatórios, a humanidade encontrou uma forma de melhorar sua relação com água. Isso possibilitou uma melhor qualidade de vida e condições para o crescimento das cidades, e aprimorou o desenvolvimento de atividades essenciais, como a agricultura e a pecuária (SNISB, 2020). Com o tempo, outros usos incentivaram a construção de vários tipos de barragens, não somente para retenção de água, mas também de alguns resíduos líquidos. Sua importância virou de extrema relevância para o desenvolvimento da região.

Nos dias atuais, as barragens possuem uma grande relevância para a economia brasileira. O país tem 19.412 barragens cadastradas no Sistema Nacional de Informações sobre Segurança de Barragens (SNISB, 2020), as quais são utilizadas, sobretudo, para o abastecimento de água em zonas residenciais, irrigação em áreas agrícolas, mineração e, principalmente, na produção de energia elétrica.

Não há como negar os benefícios que as barragens geram para o desenvolvimento de uma região. Contudo, não se pode negligenciar os diversos riscos e impactos associados com sua construção e manejo. Ao ser construída, uma barragem sempre representa uma interferência econômica, ambiental e social, que necessita ser pertinentemente estudada e analisada (SNISB, 2020).

Além do impacto de sua construção, é de extrema importância considerar possíveis falhas na estrutura e suas consequências. Uma ruptura de uma barragem pode liberar em segundos uma enorme quantidade de água, causando grandes prejuízos materiais e humanos. No Brasil, o Ministério do Desenvolvimento Regional junto com a ANA e o Sistema Nacional de Informações sobre Segurança de Barragens (SNISB) são responsáveis pela regulamentação e fiscalização das barragens. Tais esforços são de extrema importância visto que dentre todos os tipos de barragens registradas no SNISB, 17,89% possuem um alto dano potencial associado em caso de falha (SNISB, 2020).

Diante de tais riscos, é essencial regular e fiscalizar essas construções. Ações preventivas de acidentes devem ser realizadas, entre eles o monitoramento, registro de dados, elaboração de planos de ações de emergência, inspeções e revisões periódicas de segurança.

Na ocorrência de uma falha, ou seja, o rompimento da barragem, ou em cheias extraordinárias, a área de maior fragilidade é aquela que está a até 10 km ou 30 minutos de distância a jusante da barragem, denominada Zona de Auto-Salvamento (ZAS). Nesta área considera-se não haver tempo suficiente para uma intervenção das autoridades competentes em caso de acidente. Logo, os atores principais no

salvamento passam a ser os próprios moradores.

Há uma responsabilidade enorme junto a essas áreas, uma vez que sua segurança depende fundamentalmente da população, a qual deverá ter conhecimento do plano de segurança estabelecido pelo empreendedor da barragem e estar orientada a seguí-lo passo a passo. A fim de melhorar a segurança das Zonas de Auto-Salvamento (ZAS), soluções tecnológicas para alertar tais regiões e guiar seus residentes para áreas seguras, vem a contribuir de forma decisiva em momento tão crucial de tomada decisão.

Neste contexto, surgiu a possibilidade de desenvolver este trabalho junto a uma empresa especializada em soluções de monitoramento e gestão estratégica para grandes infraestruturas, que propôs-se a desenvolver um sistema de gerenciamento de zonas de auto-salvamento e orientação para casos de enchentes ou rompimento de barragens.

Este trabalho consiste no desenvolvimento de um sistema composto por um aplicativo de *smartphone* e um servidor. Em caso de adoção oficial de uma solução como a que será desenvolvida neste trabalho, ela poderia ser utilizada pelos moradores das ZAS, afim de auxiliar na sua orientação em caso de uma falha de grande impacto venha a ocorrer em uma barragem. O objetivo da adoção desse tipo de sistema é diminuir o impacto de grandes tragédias, como a ocorrida devido ao rompimento da barragem em Brumadinho em 2019 que causou a morte de 259 pessoas e o desaparecimento de outras onze (G1, 2019).

Uma vez que o curso de Engenharia de Controle e Automação promove a automatização de processos industriais e administrativos através de ferramentas tecnológicas, acredita-se que este trabalho contribuirá significativamente para automatização e controle de alertas de evacuações em áreas em torno a barragens em casos de risco, aumentando a segurança de tais infra estruturas. No contexto apresentado, as habilidades adquiridas nas disciplinas de algoritmos, banco de dados, programação, metodologia de sistemas e aspectos de segurança realizadas durante o curso são de importante relevância para o desenvolvimento deste projeto.

1.1 OBJETIVOS

O objetivo do projeto é desenvolver um software que possa ser utilizado para o gerenciamento das atividades relacionadas à infraestrutura e a gestão de processos internos de áreas de auto-salvamento, e possibilite quantificar e definir geograficamente os aspectos emergenciais e não-emergenciais, trazendo, assim, mais segurança às ZAS e seus moradores.

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- desenvolver um aplicativo para *smartphone* que alerte e guie os moradores das ZAS para regiões seguras em situações de emergência;
- desenvolver um servidor para o monitoramento do estado hídrico de barragens e disparo de notificações;
- desenvolver uma *Application Programming Interface* (API) para estabelecer a comunicação entre o aplicativo e o servidor;
- atender com os padrões internacionais de qualidade de software no desenvolvimento do sistema;

1.1.2 Não-escopo

Este projeto não abrange atividades relacionadas a:

- aspectos financeiros, comerciais ou mercadológicos relacionados à solução;
- implantação da solução em um cenário real;
- testes e/ou simulados envolvendo a população ou qualquer outra entidade;
- qualquer outro assunto que não esteja explicitamente citado nos objetivos específicos ou que não esteja sendo abordado neste relatório;

1.2 ESTRUTURA DO DOCUMENTO

O restante deste relatório é composto por seis capítulos.

No capítulo 2 é feita uma descrição mais detalhada e contextualização do projeto. No Capítulo 3 é apresentado a fundamentação teórica. No Capítulo 4 é tratado a modelagem do software para a resolução do problema, sendo a sua implementação efetiva apresentada no Capítulo 5. Por fim, nos capítulos 6 e 7 são apresentados a análise do sistema desenvolvido e as conclusões, respectivamente.

2 CONTEXTUALIZAÇÃO

A fim de contextualizar o projeto e os tópicos abordados ao longo deste documento, neste capítulo é apresentada uma descrição dos papéis das usinas, e aborda-se também algumas questões legais envolvendo a segurança de barragens no contexto brasileiro e o Plano de Ação de Emergência (PAE).

2.1 EMPRESAS ENVOLVIDAS E SEUS RESPECTIVOS PAPÉIS NO PROJETO

2.1.1 Empresa parceira no desenvolvimento do projeto

Este Projeto de Fim de Curso (PFC) foi desenvolvido durante um estágio em uma empresa especializada em soluções de monitoramento e gestão estratégica para grandes infraestruturas. A empresa está situada em Florianópolis, Santa Catarina, e atua no setor de grandes infraestruturas, como setor rodoviário e hidroelétrico, fornecendo soluções digitais para gestão da conservação dessas construções.

A empresa surgiu a partir da ideia de ex-alunos da Universidade federal de Santa Catarina em 2017, que observaram a falta de soluções tecnológicas direcionadas para administração de rodovias. Com o crescimento e sucesso da empresa em meados de 2018, surgiu o interesse em expandir a sua atuação, de modo a atender não apenas rodovias, mas grandes infraestruturas em geral. Assim, iniciou-se a expansão para o setor de meio ambiente de hidrelétricas.

Atualmente a empresa atende diversas concessionárias no Sul e Sudeste do país, além de oferecer soluções para empresas no ramo de geração de energia.

Seus produtos são compostos por plataformas web de gerenciamento de serviços e aplicativos *mobile* para o auxilio de tarefas executadas em campo.

2.1.2 Empresa concessionária da usina

No contexto deste documento, a empresa concessionária da usina é considerada um *stakeholder* do projeto, pois pode ser contratante do sistema desenvolvido. Ela é caracterizada como o agente empreendedor da barragem. Através da Lei nº 12.334/2010 que estabelece a Política Nacional de Segurança de Barragens (PNSB), as empresas concessionárias de usinas são responsáveis pela gestão de segurança da barragem. Está sob sua responsabilidade o desenvolvimento do Plano de Segurança de Barragens, assim como PAE, a ser abordado na Seção 2.1.3, obrigatório em termos legais junto a barragem.

O projeto deste PFC em questão vem auxiliar as empresas concessionárias de usinas na melhoria da gestão sobre as ZAS, áreas que estão a até 10 km ou 30 minutos do possível ponto de rompimento da barragem.

2.1.3 Plano de Ação de Emergência

A ANA classifica as barragens em função da categoria de risco e do dano potencial associado ao risco. Conforme a tabela na Figura 1, todas as barragens com classificação A e B obrigatoriamente devem conter um PAE.

Figura 1 – Tabela de classificação de barragens segunda a ANA.

	DANO POTENCIAL ASSOCIADO - ANA		
CATEGORIA DE RISCO	**ALTO**	**MÉDIO**	**BAIXO**
ALTO	A	B	C
MÉDIO	A	C	D
BAIXO	A	D	D

Fonte – Legislação Federal Brasileira em Segurança de Barragens (NEVES, 2020).

De acordo com a tabela da Figura 1, assumiremos como exemplo para análise uma barragem de usina hidrelétrica que possua categoria de risco BAIXO, e dano potencial ALTO, ou seja, a probabilidade de um acidente ocorrer é baixa, contudo os danos causados por um acidente seriam altos. Uma barragem como essa receberia a classificação A, portanto é obrigatória a elaboração do PAE.

O PAE deverá contemplar:

- i) identificação e análise das possíveis situações de emergência;
- ii) procedimentos para identificação e notificação de mau funcionamento ou de condições potenciais de ruptura da barragem;
- iii) procedimentos preventivos e corretivos a serem adotados em situações de emergência, com indicação do responsável pela ação;
- iv) estratégia e meio de divulgação e alerta para as comunidades potencialmente afetadas em situação de emergência (Art. 12 da Lei 12.334/2010).

No PAE deve estar contida uma síntese do estudo de inundação e mapa de inundação. Esse estudo permite avaliar os danos no vale a jusante e determinar as zonas que ficarão inundadas de acordo com o cenário simulado. Assim é definida Zona de Auto-Salvamento (ZAS), ou seja, a região a jusante da barragem em que se considera não haver tempo suficiente para uma intervenção das autoridades competentes em caso de acidente.

O estudo de inundação também tem a função de avaliar os danos no vale a jusante provocada pela ruptura da barragem ou por cheias em decorrências de altas vazões de saída do reservatório. Trata-se de um estudo que se baseia, essencialmente, na simulação da cheia induzida.

O projeto abordado nesse documento juntamente com PAE serve como mais uma camada de segurança para a proteção da barragem e áreas no seu entorno.

3 FUNDAMENTAÇÃO TEÓRICA E TECNOLOGIAS

Neste capítulo serão discutidos alguns dos conhecimentos técnicos e tecnologias utilizadas na elaboração do projeto.

3.1 ESTRUTURAÇÃO DE DADOS GEOGRÁFICOS

O sistema desenvolvido fundamenta-se em dados geográficos de posicionamento. Estes serão utilizado para a localizar edifícios, pessoas e rotas. Permitem localizar também, com precisão, as áreas de alagamento. Logo, é necessário estabelecer uma formatação adequada para a estruturação desses dados geográficos.

Uma das formatações mais populares é o padrão aberto GeoJSON (H. BUTLER S. GILLIES, 2016). GeoJSON é um formato de intercâmbio de dados geoespaciais baseado em JavaScript Object Notation (JSON), projetado para representar características geográficas simples, juntamente com seus atributos não espaciais, como mostrado na Figura 2.

Optou-se por utilizar o GeoJson devido sua similaridade com o formato JSON, bastante consolidado no ambiente desenvolvimento WEB, e com diversas ferramentas disponíveis de auxílio na sua manipulação. GeoJSON classifica objetos geoespaciais como *features*, que incluem:

- *Points*: podem representar endereços e localizações.
- *Line strings*: podem representar ruas, rodovias, passagens e limites de território.
- *Polygons*: podem representar área de terra, regiões e cidades.

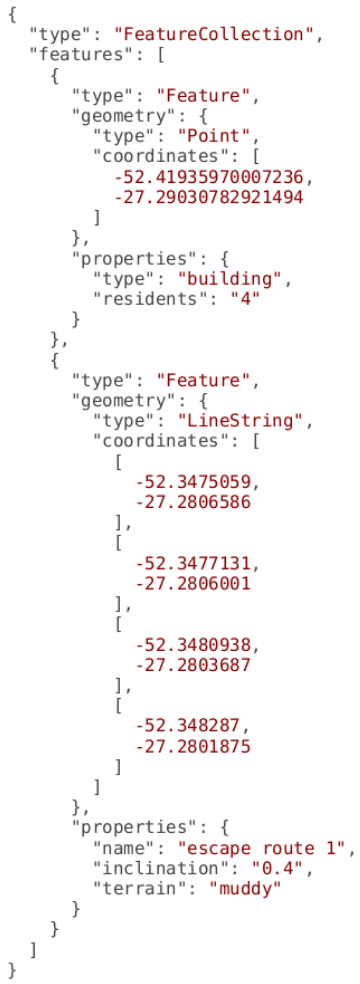
As *features* do GeoJSON não precisam representar entidades apenas do mundo físico; aplicativos de navegação e roteamento móvel, por exemplo, podem descrever sua cobertura de serviço usando GeoJSON.

3.2 ARQUITETURAS

Definir e usar padrões de arquitetura contribui na solução de muitos problemas comuns no desenvolvimento de software. O sistema desenvolvido possui uma arquitetura cliente-servidor, isto é, uma estrutura distribuída que particiona tarefas ou cargas de trabalho entre os provedores de um recurso ou serviço, chamados servidores, e solicitantes de serviços, chamados clientes (REESE, 2000). Geralmente servidores e clientes comunicam-se através de uma rede, utilizando um protocolo pré definido, como representado na Figura 3.

Como cada partição do sistema possui suas particularidades e desafios, utiliza-se diferentes tecnologias e arquiteturas internas para cumprir suas tarefas. Esta seção

Figura 2 – Exemplo GeoJSON.



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -52.41935970007236,
          -27.29030782921494
        ]
      },
      "properties": {
        "type": "building",
        "residents": "4"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            -52.3475059,
            -27.2806586
          ],
          [
            -52.3477131,
            -27.2806001
          ],
          [
            -52.3480938,
            -27.2803687
          ],
          [
            -52.348287,
            -27.2801875
          ]
        ]
      },
      "properties": {
        "name": "escape route 1",
        "inclination": "0.4",
        "terrain": "muddy"
      }
    }
  ]
}
```

Fonte – Autor.

aborda as arquiteturas e tecnologias utilizadas nas partições do cliente e servidor do sistema.

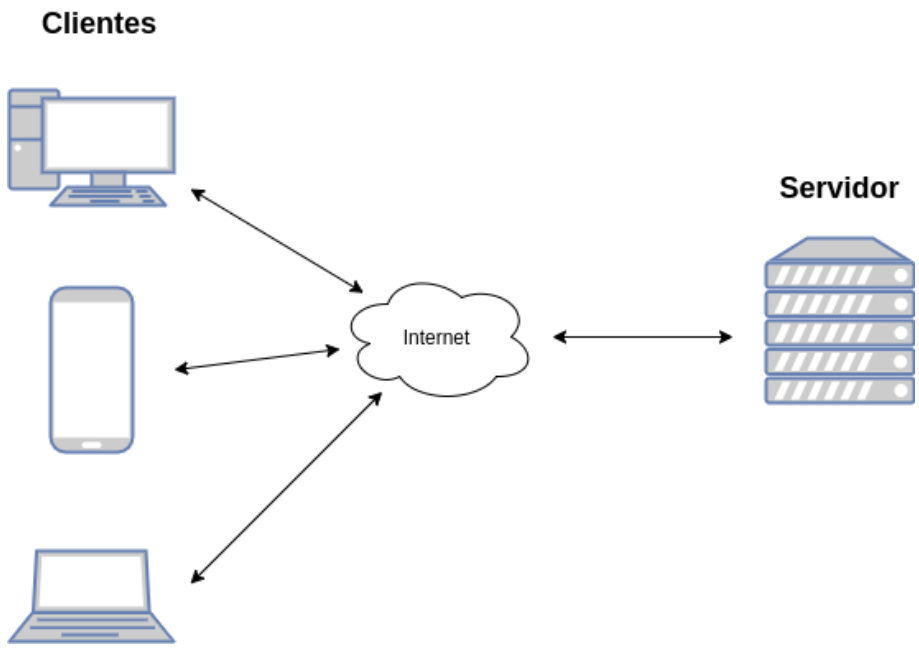
3.2.1 React Native

React Native (FACEBOOK, 2015) é um *framework* de desenvolvimento híbrido para sistemas operacionais Android e iOS, baseado no *framework* de desenvolvimento web criado pelo Facebook, React (FACEBOOK, 2013). O principal objetivo do React Native é diminuir o tempo e o conhecimento necessário para produzir um aplicativo *mobile* com suporte para Android e iOS, o Facebook chama essa abordagem de "aprenda uma vez, escreva em qualquer lugar".

As principais características do React Native são:

- **Declarativo:** Visualizações declarativas tornam o código mais previsível e fácil

Figura 3 – Arquitetura cliente-servidor.



Fonte – Autor.

de depurar;

- **Baseado em componentes:** Componentes encapsulados que gerenciam seu estado, permitindo a criação de interfaces de usuário complexas;
- **Velocidade de desenvolvimento:** React disponibiliza ferramentas como o *hot reloading*, que permite alterações no código serem recarregadas em tempo real, sem precisar reconstruir o aplicativo nativo;
- **Portabilidade:** Reutilização do código em iOS, Android e outras plataformas.

Ao contrário de outras bibliotecas e *frameworks*, React não impõe um padrão de arquitetura. Em termos gerais, o React cria uma árvore de componentes, unidades centrais mínimas, como botões, entradas de textos, ou componentes mais complexos. Cada componente possui o seu estado próprio, os quais podem ser alterados por ações dos usuários ou ações internas do sistema. O estado serve como uma ferramenta de manipulação de dados e pode determinar o que será exibido na interface.

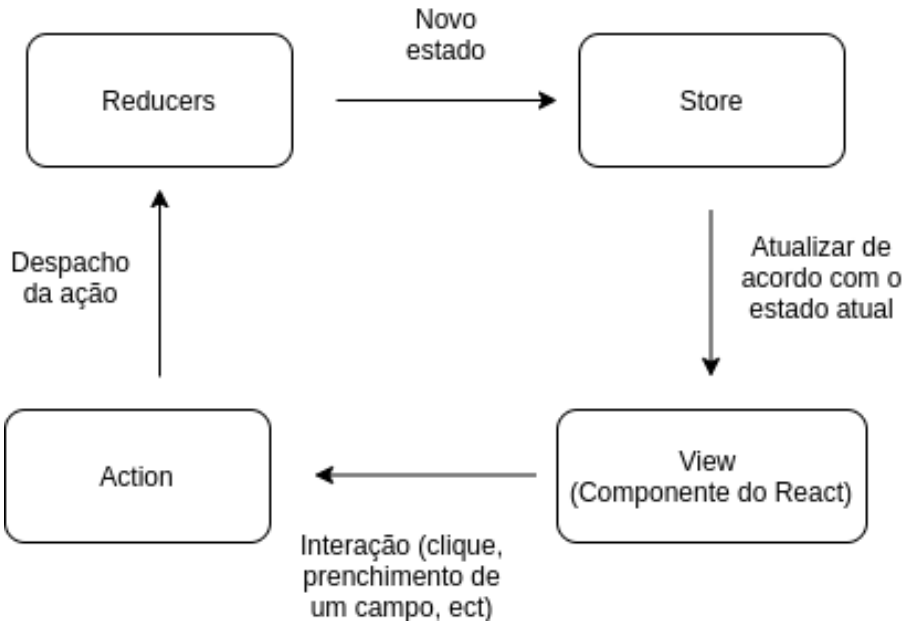
Apesar do React oferecer um mecanismo *built-in* para gerenciamento local do estado de cada componente, como o React Hooks (FACEBOOK, 2019) isso não é o suficiente para aplicações mais complexas, onde é necessário um gerenciamento global do estado da aplicação. Por isso, torna-se essencial para a escalabilidade do aplicativo a implementação de arquiteturas de gerenciamento, como o Redux.

3.2.1.1 Arquitetura Redux

A arquitetura Redux (DAN ABRAMOV, 2015) é uma implementação da arquitetura Flux (FACEBOOK, 2014), reduzindo complexidades através de composições funcionais com mudanças de abstrações, facilitando sua implementação. Essa arquitetura tem o intuito de permitir o gerenciamento do estado global de software diversos.

O fluxo de dados na arquitetura Redux é unidirecional. O estado global da aplicação fica armazenado num *store*, que atualiza as *views*. Nelas *actions* são disparadas, que por sua vez alteram o *store*. As ações passam através de um *hub*, chamado de *reducer*, que altera o estado atual conforme o estado anterior e a ação executada. Após o estado ser atualizado no *store*, todas as *views* são atualizadas conforme o novo estado global. A fluxograma da arquitetura está representada na Figura 4.

Figura 4 – Arquitetura Redux.



Fonte – Autor.

Os principais benefícios do Redux são:

- **Previsibilidade:** Mudanças de estado do sistema se tornam previsíveis e fáceis de depurar, já que apenas *actions* podem alterar o estado e são facilmente rastreadas devido ao fluxo unidirecional de dados.
- **Centralizado:** Centralização do estado e da lógica do sistema permite a utilização de recursos como desfazer, refazer e persistir;
- **Facilidade no *debugger*:** O Redux disponibiliza a ferramenta DevTools que permite o rastreamento do estado da aplicação e de todas as ações realizadas.

- **Flexibilidade:** Redux funciona com qualquer camada de interface do usuário e tem um grande ecossistema de complementos para atender diversas necessidades.

3.2.2 Django

Django (ADRIAN HOLOVATY, 2005) é um *framework open-source* para desenvolvimento web baseado em Python que segue a arquitetura MTV, explixado na Seção 3.2.2.1. Atualmente é mantido e desenvolvido pela Django Software Foundation, uma organização sem fins lucrativos.

O objetivo principal do Django é facilitar a criação de sistemas web complexos e API integrada com banco de dados. A estrutura enfatiza o princípio *DRY* (ANDREW HUNT, 1999), através da capacidade de reutilizar e modular os componentes, reduzindo a quantidade de código. Além disso incentiva o baixo acoplamento e acelera o desenvolvimento da aplicação.

Os principais pontos fortes do Django são:

- promover velocidade no desenvolvimento de aplicações;
- evitar erros comuns de segurança;
- aumentar a velocidade e flexibilidade de desenvolvimento
- fornecer uma interface opcional para criar, ler, atualizar e deletar modelos referenciados no banco de dados.

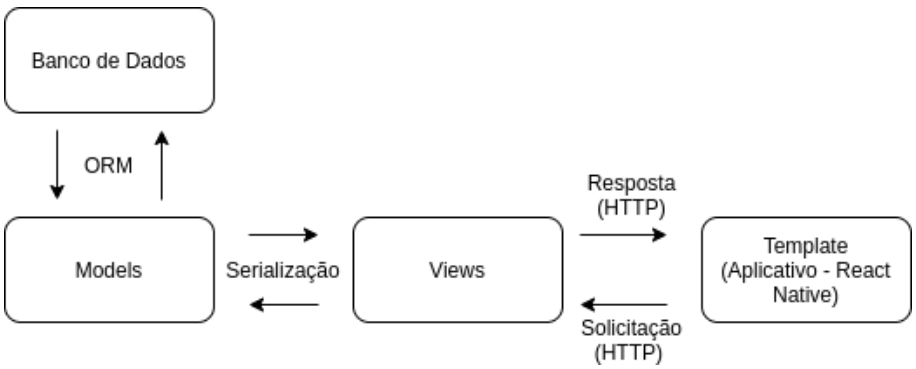
3.2.2.1 Arquitetura MTV

A arquitetura *Model-Template-View* (MTV) é composta por 3 componentes:

- Model: Realiza o mapeamento do banco de dados para o projeto, chamado em inglês de *Object-relational mapping* (ORM);
- Template: Interface para visualização de dados.
- View: Contem a lógica de negócios da aplicação. Realiza a serialização dos *models* para ter acesso as informações presentes no banco de dados e realizar operações CRUD (*Create, read, update and delete*).

Na Figura 5 é apresentado o diagrama básico da arquitetura MTV.

Figura 5 – Arquitetura MTV.



Fonte – Autor.

3.3 RESTFUL API

Devido à arquitetura cliente-servidor do projeto, deve-se estabelecer um protocolo de integração para a comunicação entre servidor e cliente. A arquitetura mais estabelecida para esta integração foi concebida por Roy Fielding (FIELDING, 2000) e é conhecida como REST. *Representational State of Transfer* (REST) consiste em seis conceitos:

- **Cliente–Servidor:** separação de atribuições entre a interface do usuário e o banco de dados. Melhora a portabilidade do sistema em múltiplas plataformas e também a escalabilidade conforme o lado do servidor se torna menos complexo;
- **Sem estado:** não há informações armazenadas no servidor sobre o estado do cliente, todas as sessões devem ser mantidas no lado do cliente;
- **Cache:** os dados das respostas podem ser armazenados em cache, o que permite a utilização de estratégias que diminuam a carga do servidor em solicitações idênticas;
- **Interface Uniforme:** fundamenta-se em quatro restrições: identificação de recursos; manipulação de recursos por meio de representações; mensagens autodescritivas; e hipermídia como o guia do estado da aplicação;
- **Sistema em camadas:** restringe o comportamento de cada componente, através de camadas hierárquicas;
- **Código sob demanda (opcional):** O REST permite que a funcionalidade do cliente seja estendida por meio do download e execução de scripts. Isso simplifica os clientes, reduzindo o número de recursos necessários para serem pré-implementados.

Esses conceitos foram utilizados para implementar uma API entre as partições do servidor e cliente.

3.4 CONTROLE DE VERSÃO

É comum, em projetos desenvolvidos na indústria, que muitos desenvolvedores estejam trabalhando no mesmo projeto, performando diferentes ações em diferentes partes do projeto. Um sistema de controle de versão é utilizado para garantir a integridade do código entre todos os desenvolvedores, facilitar a distribuição de tarefas, alterar códigos, controlar o fluxo das mudanças e permitir reverter para versões anteriores do projeto.

Git é a plataforma mais famosa neste aspecto (TORVALDS, 2005). O Git permite e incentiva os desenvolvedores a ter vários *branches* locais, que são versões da base do código, e podem ser totalmente independentes umas das outras. A criação, fusão e exclusão dessas diferentes versões é bastante facilitada, o que permite um rápido desenvolvimento e integração de funcionalidades.

No projeto em questão, o Git foi usado como a principal ferramenta para armazenar e alterar o código fonte. Todos os projetos desenvolvidos na empresa parceira possuem repositórios no GitLab, plataforma de hospedagem que utiliza o Git como controle de versionamento.

3.5 CONTROLE DE QUALIDADE DE SOFTWARE

O controle de qualidade de software é um conjunto de atividades para garantir a qualidade no desenvolvimento de software. As duas principais técnicas usadas no projeto foram Teste Unitário e Revisões de Software.

3.5.1 Teste Unitário

O teste unitário é um método para validação individual de cada funcionalidade na aplicação. Isola cada parte do programa e localiza facilmente as falhas. Usando-o é possível encontrar erros críticos nas fases iniciais do ciclo de desenvolvimento, reduzindo os custos para corrigi-los (OLAN, 2003).

3.5.2 Revisões de Software

Revisões de software são inspeções no código por outros membros da equipe, após a adição de um novo recurso. Auxilia os engenheiros de software na validação da qualidade, funcionalidade e outros recursos dos componentes vitais do software. O processo inclui testar o aplicativo executando todas as funcionalidades, verificar o código-fonte e revisar a lógica e a limpeza do código (WIEGERS, 2001).

3.6 CI/CD

A *Continuous Integration* (CI) e a *Continuous Delivery* (CD) são um conjunto de princípios operacionais e uma coleção de práticas que permitem que as equipes de desenvolvimento de software forneçam alterações de código com mais frequência e confiabilidade (SWARTOUT, 2018).

CI é uma filosofia de desenvolvimento e um conjunto de práticas que leva as equipes a implementar pequenas mudanças no código e enviá-las frequentemente para repositórios de controle de versão. Como a maioria dos aplicativos modernos requer o desenvolvimento de código em diferentes plataformas e ferramentas, a equipe precisa de um mecanismo para integrar e validar suas mudanças.

O objetivo técnico da CI é estabelecer uma maneira consistente e automatizada de construir, empacotar e testar aplicativos. Assim, as equipes são mais propensas a realizar alterações do código fonte com mais frequência, o que leva a uma melhor colaboração e qualidade no software.

A CD começa onde termina a CI. A *Continuous Delivery* automatiza a entrega da aplicação para ambientes de infraestrutura selecionados. A maioria das equipes trabalha com vários ambientes diferentes de produção, e o CD garante que haja uma maneira automatizada de enviar alterações de código a eles.

Praticas maduras de CI/CD proporcionam:

- *feedback* constante do estado do software para o cliente ou *stakeholder* do projeto;
- elevação de confiabilidade no produto, e
- maior velocidade em disponibilizar novas versões do produto para os usuários.

4 REQUISITOS E MODELAGEM

Apresentado o contexto e os conhecimentos técnicos do problema abordado neste PFC — criação de um sistema de gerenciamento e orientação das ZAS - este capítulo tem como objetivo apresentar os requisitos do sistema, sua modelagem e as escolhas tecnológicas para sua implementação. Serão utilizados diagramas padrões *Unified Modelling Language* (UML) como diagramas de casos de uso e diagramas de sequência.

4.1 REQUISITOS

O problema descrito neste documento - gerenciamento e orientação para ZAS - é bastante amplo e abrange diversos casos e funcionalidades. Reuniões foram feitas entre os stakeholders para especificar o escopo do sistema e definir os seus requisitos. Os principais requisitos estabelecidos são:

- obter leituras de sensores de nível de cota (esses sensores já existem nas usinas);
- gerar rotas de fuga em riscos de cheias;
- identificar populações e construções atingidas de acordo com cada nível de cota;
- alertar moradores sobre eventuais situações de risco através de Notificações *Push* e
- o aplicativo deve ser funcional mesmo não possuindo acesso a internet;

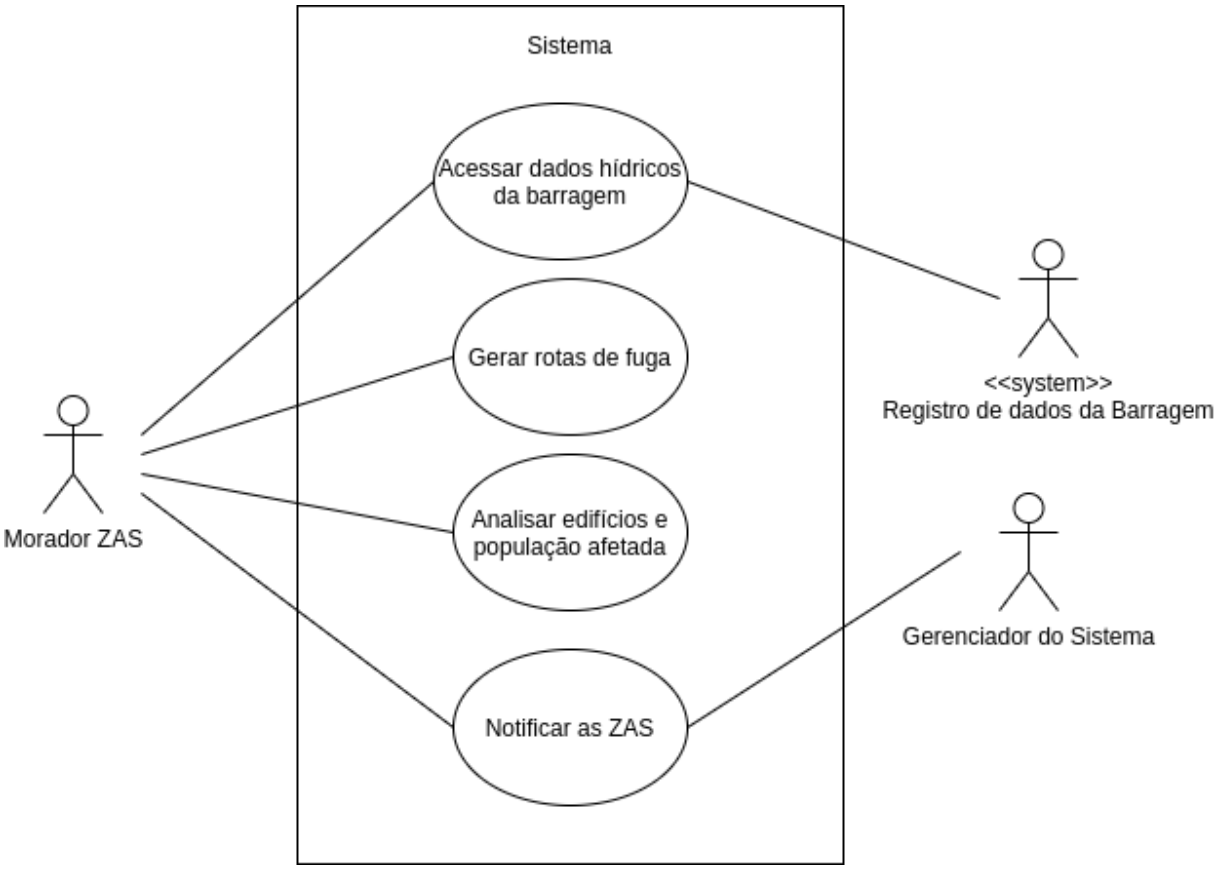
4.2 FUNCIONALIDADES

A partir dos requisitos do projeto, definiu-se as principais funcionalidades do sistema, simplificadas no Diagrama de Casos de Uso mostrado na Figura 6.

O sistema deve atender as seguintes funcionalidades:

- **Acessar dados hídricos da barragem**: Acesso aos dados dos sensores de nível e fluxo da barragem através de uma API;
- **Gerar rotas de fuga**: Geração de rotas de fuga de acordo com o nível da cota da barragem;
- **Analisar edifícios e população afetada**: Análise quantitativa de edifícios e pessoas afetadas em cada nível de cota da barragem;
- **Notificar as ZAS**: Notificação dos moradores sobre eventuais situações de risco através de notificações no sistema.

Figura 6 – Casos de Uso do Sistema.



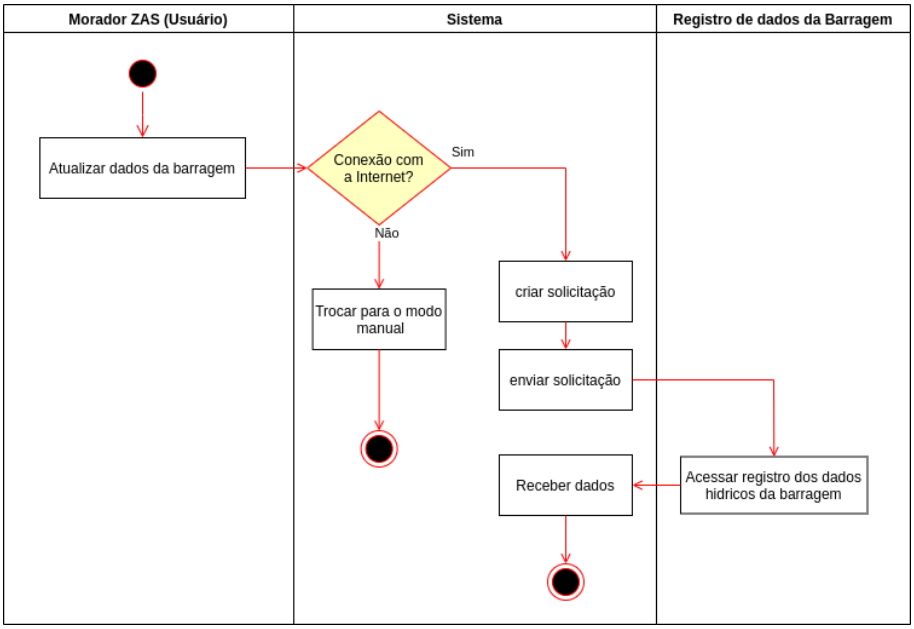
Fonte – Autor.

4.2.1 Acesso aos dados hídricos da barragem

O sistema deve-se conectar com uma API disponibilizada pelo agente empreendedor da barragem, no qual deve obter as informações hídricas atuais como o nível de água e vazão. Esses dados serão utilizados na previsão das regiões inundadas conforme o mapa de inundações presente no PAE e de acordo dados topológicos da área.

Para toda ação executada no sistema, que necessite das informações hídricas da barragem, verifica-se se o usuário possui conexão com a internet. Caso possua, deve-se criar uma solicitação de leitura desses dados, que é então enviada através da API da barragem. Caso não haja conexão o software deve mudar para o modo manual, no qual os dados poderão serem preenchidos manualmente pelo usuário (ver Seção 4.2.5). Esse processo é exibido no diagrama de atividades da Figura 7.

Figura 7 – Acessar dados da barragem - Diagrama de atividades.



Fonte – Autor.

4.2.2 Geração de rotas

Em situações de emergência, garantir a segurança da população das ZAS é indispensável. Guiar corretamente os moradores para pontos seguros da região tem o potencial de salvar vidas. Deve ser planejada com bastante cautela e analisando diversos fatores circunstanciais e topológicos da região.

No PAE são estabelecidos pontos de encontro próximos às regiões povoadas, para onde a população deve se dirigir em casos de emergência. Nesses locais é possível garantir a segurança da população. Entidades competentes orientarão sobre os próximos passos a serem seguidos após o desastre. Neste cenário, o aplicativo deve ser capaz de guiar o usuário de forma segura e rápida da sua localização de origem até um ponto de encontro selecionado. Assim, diversos fatores são considerados na determinação da melhor rota:

- Posição de origem do morador;
- Posição do ponto de encontro selecionado;
- Distância total a ser percorrida;
- Tempo estimado de percurso;
- Elevação do terreno;
- Pavimentação da estrada;

- Cota de inundação do trecho e
- Cota atual registrada.

Não faz parte do escopo do presente trabalho considerar fatores que não sejam geográficos, como faixa etária do usuário ou limitações físicas.

As rotas indicadas são destinadas para pedestres, a utilização de veículos automotores é decisão do morador. O usuário do software poderá habilitar a utilização do GPS para sua posição inicial ser identificada automaticamente ou poderá informa-la manualmente, indicando na interface gráfica.

As características geográficas consideradas acima, devem ser identificas a partir de um levantamento topológico e devidamente registrado num documento no formato GeoJSON. Nesse arquivo constam todas as possíveis rotas que os moradores podem percorrer. É determinado também o grau de dificuldade de utilizá-la, a partir do coeficientes de atrito, ou seja, o tipo de pavimentação da rota e sua elevação. É informado ainda a partir de quais níveis de cotas do reservatório o caminho é obstruído.

Vale ressaltar que o sistema sugere e destaca a rota com o menor custo considerando todos os itens acima, não deixando de exibir todas as outras possíveis rotas. É de responsabilidade do usuário decidir qual utilizará.

4.2.3 Edificações e populações afetadas

O sistema é capaz de identificar e quantificar quais propriedades que serão afetadas para um determinado nível de cota.

Os dados de localização dos edifícios, quantidade de moradores e nível da cota que atinge a construção também serão recolhidos e registrados em arquivo no formato GeoJSON.

A partir desse arquivo o sistema quantifica o número de pessoas e edificações suscetíveis ao risco para as condições atuais do reservatório. As informações são apresentadas graficamente para facilitar a interpretação do usuário.

4.2.4 Notificações

Em casos de emergência o sistema deve alertar todos os usuários sobre o ocorrido. Duas formas de notificações podem ser utilizadas: *Short Message Service* (SMS) e notificações *push*.

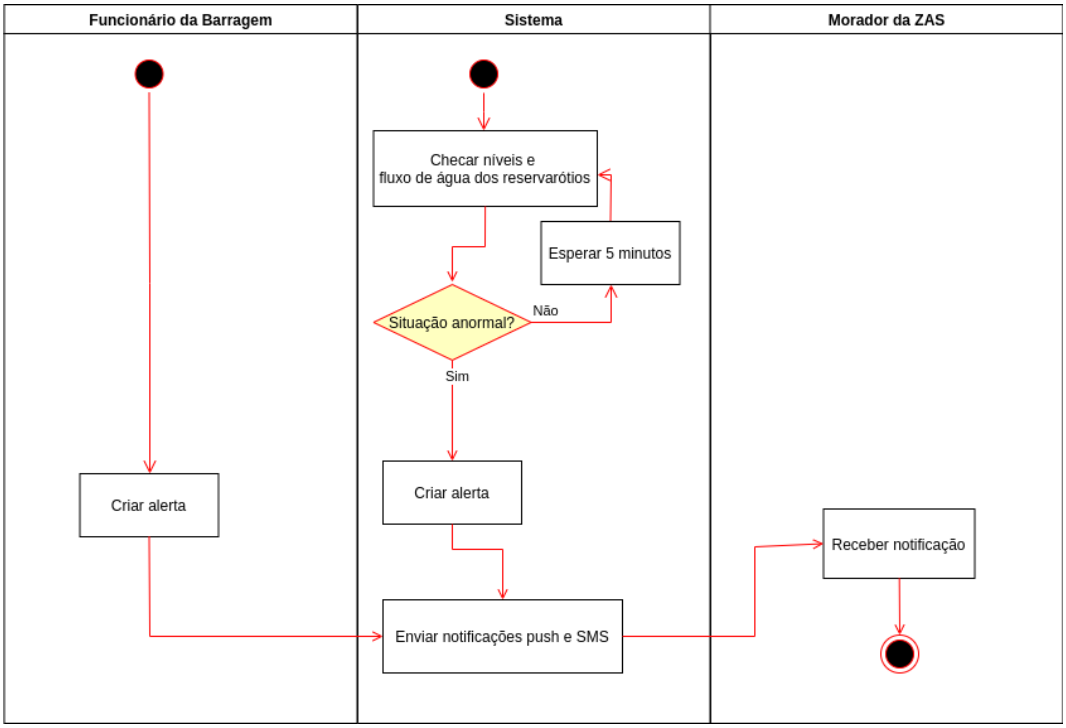
SMS é um serviço disponível em celulares que permite o envio de mensagens curtas (até 160 caracteres) conhecidas popularmente como mensagens de texto. Este serviço pode ser tarifado ou não, dependendo da operadora de telefonia e do plano associado. SMS originalmente foi projetado como parte do GSM (Sistema de comunicação móvel global) padrão digital de telefone celular (HILLEBRAND, 2010), mas está agora disponível num vasto leque de redes, incluindo redes 3G, 4G e até 5G.

As mensagens SMS devem ser encaminhas para moradores registrados no sistema em caso de cheia ou emergência. A mensagem contém um link para redirecionar o usuário para o software desenvolvido neste documento ou para uma página web, onde é possível obter acesso ao software caso não esteja instalado.

Já as notificações *push*, são mensagens de alerta enviadas para aplicativos de celular e sistemas operacionais, em geral. Elas são enviadas e acessadas diretamente no software, apesar de necessitar de acesso à internet para seu uso, ela promove uma interação direta com o sistema, permitindo troca de dados.

As notificações deverão ser disparadas automaticamente caso a barragem apresente alguma irregularidade nos seus dados hídricos. Mas também podem ser disparadas manualmente, por algum operador, em caso de situação não prevista no projeto, não identificado pela rotina automática. A esquematização do disparo de notificações foi modelada no diagrama da Figura 8.

Figura 8 – Criar notificações - Diagrama de atividades.



Fonte – Autor.

4.2.5 Funcionamento off-line

Devido ao caráter emergencial da utilização do aplicativo e a localidade dos usuários principais, que podem estar em áreas afastadas e com baixa ou sem conexão com a internet, é necessário que a maior parte das funcionalidades do aplicativo possam ser usadas sem necessitar o acesso à internet.

Para cumprir tal condição é necessário que informações como os mapas sejam salvas no dispositivo durante o primeiro acesso à internet. Essas informações devem ser salvas na memória interna do celular, não necessitando fazer a requisição destes dados novamente.

Além disso, ainda é necessário que o usuário possa preencher manualmente os valores hídricos da barragem. O residente da ZAS pode ser informado das condições do reservatório por outras formas, desde SMS até por auto-falantes. Assim, a geração das rotas não depende exclusivamente da conexão com a internet. A partir das informações fornecidas pelo usuário as rotas podem ser definidas. Essa funcionalidade pode ser usada tanto para simular diferentes cenários de cotas, ou situação de emergência onde não há acesso à internet.

Já no caso do *Global Positioning System* (GPS), ele é utilizado para apontar a posição inicial na geração da rota, porém essa posição pode ser também especificada manualmente. O que exclui a dependência do uso do GPS pelo sistema.

4.3 TECNOLOGIA

O software desenvolvido deve ser acessado por um dispositivo de resposta individual que tenha uma penetrabilidade no maior número possível de edificações e pessoas. Tal estratégia busca vantagem no "efeito rebanho", onde mesmo que a notificação não tenha 100% de sucesso, o alto número de pessoas que receberam o alerta acaba por produzir um "comportamento de manada".

Assim, optou-se por desenvolver um aplicativo para plataformas *mobile* com sistemas operacionais *Android* e *iOS*. Dispositivos *mobile* também permitem a utilização de tecnologias como GPS, SMS e notificações *push* que podem ser usadas para melhor orientar e alertar cada morador.

O aplicativo se comunicará com um servidor, estabelecendo uma arquitetura cliente-servidor, permitindo uma maior modulação do sistema. O servidor fornece e atualiza dados referentes às barragens registradas no sistema e cria alertas em casos de emergência.

4.3.1 *Stack*

Após definir a arquitetura cliente-servidor e suas plataformas físicas, é necessário estabelecer quais serão as ferramentas tecnológicas utilizadas para facilitar o desenvolvimento do aplicativo *mobile* e do servidor.

Optou-se por utilizar ferramentas e *frameworks* já bem estabelecidos dentro da empresa parceira. Assim, a curva de aprendizado seria menor, pois teria um grande auxílio de outros funcionários e poderiam ser reaproveitados códigos fonte de outros projetos.

A arquitetura final com as principais linguagens e *frameworks* utilizados é composta pelo aplicativo *mobile*, desenvolvido utilizando os *frameworks* React-Native e Redux em Typescript, linguagem baseada em Javascript, e o servidor construído a partir do *framework* Django em Python. O servidor se comunica com um banco de dados relacional administrado pela ferramenta PostgreSQL. E a comunicação cliente-servidor é realizada através de uma RESTful API.

Abaixo estão listadas as ferramentas mais relevantes utilizadas no lado do cliente e no lado do servidor.

4.3.1.1 Cliente

As tecnologias usadas no aplicativo *mobile* são:

- React-Native: *Framework* para desenvolvimento de aplicativos *mobile* em Javascript;
- Redux: Gerenciador do estado da aplicação;
- Redux-Saga: Auxilia o sequenciamento de ações do Redux;
- React-Native-Paper: Biblioteca de interface gráfica;
- Jest: Biblioteca de teste para códigos em Javascript;
- Mapbox: Ferramenta de criação de mapas interativos; e
- Turf: Biblioteca para análises geoespaciais.

4.3.1.2 Servidor

No servidor, as seguintes tecnologias são usadas:

- Django: *Framework* para desenvolvimento do servidor em Python;
- DjangoRestFramework: *Framework* para criação de uma RESTful API;
- PostgreSQL: Gerenciador do banco dados;
- Django-Scarface: Gerenciador de notificações *push* e
- Zappa: *Wrapper* para arquitetura *serveless*. Utilizado para gerenciar rotinas de tarefas programadas;

4.3.1.3 GitLab

O GitLab foi usado para controle de código, gerenciamento de projeto e CI/CD. Ele oferece recursos interessantes, como *task board*, *issue tracking*, *time tracking* e *built-in continuous integration*.

5 IMPLEMENTAÇÃO

Este capítulo apresenta os detalhes sobre a implementação do projeto, junto com as tecnologias, processos e técnicas escolhidos para apoiar os requisitos do projeto discutidos no Capítulo 4.

5.1 METODOLOGIA SCRUM

A metodologia de gerenciamento de projeto escolhida para ser utilizada foi a Scrum (SUTHERLAND, 2015). Consiste em uma estrutura para organizar as tarefas e planejar sua execução com agilidade e flexibilidade.

O tempo do projeto é dividido em ciclos denominados *sprints* que normalmente duram entre uma a quatro semanas dependendo do projeto. Na empresa parceira, a duração é de duas semanas.

Cada demanda do projeto é adicionada a uma lista chamada de *Project Backlog*. No início de cada *sprint* há uma reunião chamada planejamento de *sprint*, onde a equipe de desenvolvimento decide quais tarefas do *Project Backlog* devem ser atribuídas para a próxima *sprint*. Todos os membros votam no número de pontos que cada tarefa merece de acordo com a dificuldade e estimativa de tempo para concluí-la. Estes pontos seguem uma sequência de Fibonacci de forma a representar claramente a dificuldade da tarefa. Cada membro deve ponderar quantos pontos executarão e assim as tarefas são atribuídas na *sprint*.

Durante a *sprint*, a equipe realiza reuniões diárias denominadas *daily scrum*. São reuniões bem curtas, duração de cerca de 15 minutos, considerando um time de 6 pessoas. Cada membro da equipe relata o que trabalhou no dia anterior, o que está planejado para o dia atual e quais dificuldades apareceram. Isso é importante para que o líder (*Scrum Master*) ou outros colegas de equipe saibam quando a ajuda é necessária para remover obstáculos e aumentar a produtividade.

Quando uma *sprint* termina, a equipe se reúne novamente para a retrospectiva do ciclo, onde cada membro apresenta o que foi desenvolvido. Eventualmente, melhorias ou sugestões podem ser propostas.

A ferramenta utilizada para gerenciar a metodologia Scrum foi a plataforma web GitLab, que além de servir como repositório para o código do projeto, também oferece diferentes painéis de criação de tarefas e acompanhamento de atividades.

5.2 SERVIDOR E BANCO DE DADOS

O sistema é dividido em duas partes principais, o servidor e o cliente.

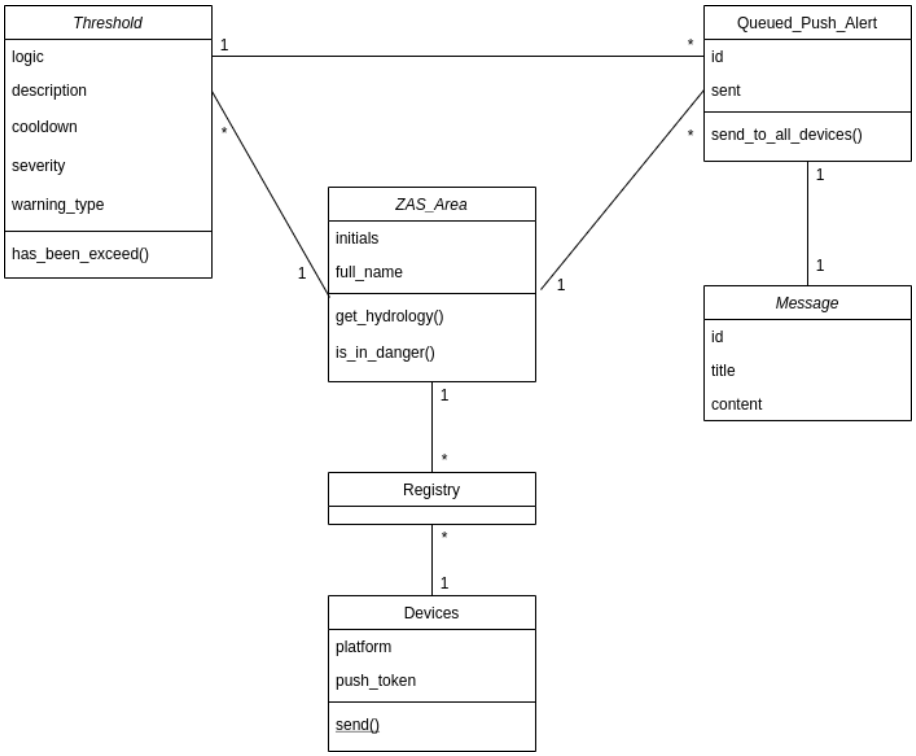
Primeiro projetou-se o servidor do sistema, assim como o seu banco de dados. Utilizou-se o *framework* Django (Seção 3.2.2) para a construção do servidor e um

banco de dados relacional gerenciado pela ferramenta PostgreSQL.

O servidor se baseia na arquitetura MTV (Seção 3.2.2.1), construída em cima do diagrama de classes elaborado da Figura 9.

Vale apontar, que não existe a necessidade de elaborar um diagrama relacional de entidades para o banco de dados, pois sua estrutura já é estabelecida e abstraída a partir do diagramas de classes da Figura 9.

Figura 9 – Diagrama de Classes.



Fonte – Autor.

No diagrama são modelados 6 classes representas por tabelas relacionais no banco de dados. Cada classe representa uma entidade física ou um conceito abstrato dentro do sistema.

- *ZAS Area*: Abstração das Zonas de Auto-Salvamento;
- *Queued Push Alert*: Conjunto de alertas criados caso um *Threshold* de uma ZAS Area tenha sido alcançado;
- *Message*: Mensagens que serão enviadas através de notificações *push* e SMS;
- *Threshold*: Condição limite para criar uma situação de risco em uma ZAS;
- *Device*: Registro de dispositivos móveis para o envio de notificações e

- *Registry*; Objeto intermediário entre *ZAS Area* e *Device*: criado para simplificar uma relação de muitos-para-muitos no diagrama de classes.

A partir dessas seis classes é possível executar as principais funções do servidor. A identificação de situações de emergência é feita através de uma interface de comunicação entre o servidor desenvolvido e outras APIs. Essas APIs disponibilizam as informações hídricas dos reservatórios e das barragens referentes às ZAS presentes no sistema. Assim é possível fazer o monitoramento e detecção caso haja alguma cheia ou estado de emergência.

Uma rotina de verificação é criada para o monitoramento de todas as ZAS (*ZAS Area*). Solicita-se continuamente os dados hídricos de todas as barragens, caso algum valor desses valores ultrapasse os limites estabelecidos (*Threshold*), alertas são criados (*Queued Push Alert*) e mensagens (*Message*) são enviadas a todos dispositivos móveis (*Device*) referentes a essa ZAS (*ZAS Area*).

Ainda, através do painel de administração provido pelo Django é possível criar, deletar e atualizar instâncias no banco de dados. Desse modo é possível criar alertas (*Queued Push Alert*) manualmente através desse painel. Algumas funções e informações do servidor são acessadas através de uma API criada para este fim (ver Seção 5.3).

5.3 *APPLICATION PROGRAMMING INTERFACE (API)*

Para estabelecer a comunicação efetiva entre cliente-servidor, desenvolveu-se uma RESTful API (Seção 3.3). Desta forma, o cliente pode solicitar informações e registrar dados no banco de dados do sistema com facilidade.

A arquitetura REST utiliza o protocolo *Hypertext Transfer Protocol* (HTTP) na construção de mensagens. Existem dois tipos de mensagens: requisições e respostas, cada uma com seu próprio formato. As mensagens de requisição devem conter as seguintes informações:

- o método HTTP utilizado: GET, POST, PUT ou DELETE. Eles indicam respectivamente as seguintes operações: leitura, criação, atualização e exclusão;
- a versão do protocolo HTTP;
- cabeçalhos contendo informações adicionais para o servidor e
- o corpo de dados que contém a mensagem a ser transmitida.

Já as mensagens de respostas devem conter:

- um código de *status*, indicando se a requisição foi bem sucedida, ou não, e por quê;

- a versão do protocolo HTTP;
- uma mensagem de status, isto é, uma pequena descrição informal sobre o código de status;
- cabeçalhos HTTP, assim como aqueles das requisições; e
- opcionalmente, um corpo com dados do recurso requisitado.

RESTful APIs usam uma *Uniform Resource Identifiers* (URI) para mapear os recursos e funções do sistema, também chamados de *endpoints*. Para o sistema desenvolvido, os principais *endpoints* que serão utilizados no sistema estão explicados no Quadro 1.

Quadro 1 – *Endpoints*.

URI	/zas_area
Descrição	Lista de todas as ZAS registradas no sistema.
Método HTTP	GET
Mensagem de resposta	Formato JSON com a lista de todas as ZAS e suas informações.
URI	/zas_area/:id/hydrology
Descrição	Valores mais recentes das variáveis hídricas do reservatório referente à ZAS indicada.
Método HTTP	GET
Parâmetros	Id da ZAS
Mensagem de resposta	Formato JSON com os valores hídricos da barragem refente à ZAS indicada.
URI	/zas_area/:id/registerpush
Descrição	Registrar o dispositivo móvel para envio de alertas futuros.
Método HTTP	POST
Parâmetros	Id da ZAS
Mensagem enviada	Formato JSON contendo a identificação do sistema operacional do dispositivo móvel utilizado e um *token* único de identificação.

Fonte – Autor.

Vale ressaltar, que a função de obtenção dos valores hídricos é uma informação essencial na geração de rotas. Primordialmente é provida pela API e em casos de falta comunicação com o servidor, uma segunda opção deve ser oferecida. Essa opção é explicada na Seção 5.4.1.

5.4 APLICATIVO MÓVEL

O lado do cliente é constituído por um aplicativo para *smartphone*, desenvolvido utilizando o *framework* React-Native (Seção 3.2.1) e arquitetura Redux (Seção 3.2.1.1)

para o gerenciamento de estados.

Diferente de arquiteturas mais tradicionais, o desenvolvimento de um software em React-Native não utiliza-se de diagramas de classes. Cada componente é desenvolvido independentemente e em geral não há uma necessidade de diagramas. Isto fica ainda mais evidente quando se trabalha com metodologias ágeis, como o Scrum, que priorizam o desenvolvimento sobre a documentação.

Logo, não se apresentará nenhum diagrama de desenvolvimento nessa seção. Será apenas descrito o detalhamento de como ocorre a geração da rota de fuga e o conteúdo das telas principais com as suas respectivas funcionalidades no aplicativo.

Durante a duração prevista para este projeto o objetivo era lançar uma primeira versão completa do aplicativo. Porém tal meta não foi alcançada, devido a pandemia da COVID-19, que impossibilitou a obtenção dos dados topológicos e demográficos da região. Todas as funcionalidades e telas especificadas foram criadas com sucesso, entretanto não foram testadas com dados reais ou em campo.

5.4.1 Roteamento

A principal funcionalidade do sistema é guiar os moradores de forma segura até um ponto de encontro selecionado pelo usuário. Essa orientação é feita através da exibição de uma rota com o menor tempo de percurso, boa pavimentação e sem riscos de inundação.

É de extrema importância que essa funcionalidade esteja também disponível sem acesso à internet. Assim, o arquivo GeoJSON contendo todos os dados topológicos é baixado juntamente com o aplicativo e os dados hídricos podem ser adquiridos através da API ou adicionados manualmente pelo usuário. Uma vez que tais informações podem ser transmitidas por outros meios, através de SMS, auto-falantes ou medidores locais.

Vários algoritmos poderiam ser utilizados para resolver este problema de caminho mínimo, como Dijkstra (DIJKSTRA, 1959), Bellman-Ford (BELLMAN, 1958) ou Floyd-Warshall (FLOYD, 1962).

Optou-se pelo algoritmo de Dijkstra por possuir a menor complexidade no tempo dentre eles, além de ser achar o caminho ótimo para apenas o nodo referenciado.

Desse modo, é construído um grafo a partir da topologia da região e utiliza-se o algoritmo de Dijkstra para encontrar o caminho mais curto dentro do grafo.

5.4.1.1 Grafo

É construído um grafo ponderado e dirigido para representar matematicamente a topologia presente no arquivo GeoJSON. Os vértices e arcos do grafo são construídos a partir das *line strings* presentes no GeoJSON. O peso de cada arco é determinado

conforme a distância entre os seus vértices, o tipo de pavimentação, a inclinação da superfície e o risco da área.

O risco da área é recalculado sempre que o usuário solicita a geração de uma nova rota. O sistema verifica os valores hídricos atuais da barragem referente e define se a cota limite para a inundação da região foi alcançada ou não. Caso o arco conecte um vértice de partida considerado como não inundado até um vértice inundado, seu peso atribuído é infinito. Caso os dois nodos sejam considerado como inundados, o peso atribuído é muito alto porém diferente de infinito. Assim, possibilita a geração de uma rota partindo de um ponto já inundado.

5.4.1.2 Algoritmos de caminhos mínimos - Dijkstra

O algoritmo de Dijkstra resolve o problema de encontrar um caminho mínimo de fonte única em um grafo $G = (V, E, w)$ ponderado dirigido ou não. Para esse algoritmo, as arestas/arcos não devem ter pesos negativos. Então, a função de pesos é redefinida como $w : E \rightarrow R_+$. O algoritmo é denominado como guloso, pois em cada passo ele repetidamente seleciona o vértice de menor custo estimado até então. Quando esse vértice é selecionado, ele não é mais atualizado e sua distância é propagada para suas adjacências.

O pseudo-código do algoritmo é mostrado na Figura 10. As estruturas de dados D, A e C no pseudo-código são utilizadas, respectivamente, para armazenar o custo mínimo a partir do vértice de origem, definir os vértices antecessores e definir se um vértice foi visitado ou não. A função *arg min*, na linha 6, encontra o vértice adjacente não visitado com a menor distância. Se o algoritmo de Dijkstra mantiver uma fila de prioridades mínimas para mapear a distância estimada no lugar de D, utilizando um *heap* a complexidade computacional miníma é de $O((|V| + |E|) \log_2 |V|)$. O que lhe torna hábil o suficiente a ser processado num dispositivo móvel.

5.4.2 Interface gráfica

A interface gráfica é uma importante característica de softwares. Está estritamente ligada à usabilidade do sistema e sua adesão pelos usuários. Logo, projetar uma interface coerente e de fácil utilização é de grande importância para uma maior adoção e uso correto do aplicativo.

Vários conceitos de *User Experience* (UX) *Design* e *User Interface* (UI) *Design* foram aplicadas no desenvolvimento da interface (KRUG, 2005). Dentre eles pode-se destacar os seguintes princípios:

- Consistência: um design único, em conformidade com a expectativa do usuário.
- Clareza: o conteúdo da informação é veiculado com rapidez e precisão. É auto-descritivo.

Figura 10 – Pseudo código do algoritmo de Disjkstra.

Algoritmo 11: Algoritmo de Dijkstra.

Input : um grafo $G = (V, E, w : E \to \mathbb{R}_*^+)$, um vértice de origem $s \in V$

```
1  D_v ← ∞ ∀v ∈ V
2  A_v ← null ∀v ∈ V
3  C_v ← false ∀v ∈ V
4  D_s ← 0
5  while ∃v ∈ V(C_v = false) do
6      u ← arg min_{v∈V} {D_v | C_v = false}
7      C_u ← true
8      foreach v ∈ N(u) : C_v = false do
9          if D_v > D_u + w((u, v)) then
10             D_v ← D_u + w((u, v))
11             A_v ← u
12 return (D, A)
```

Fonte – Rafael de Santiago, Professor UFSC/INE.

- Legibilidade: as informações são fáceis de ler.
- Compreensibilidade: o significado é claramente compreensível, inequívoco, interpretável e reconhecível.
- *Feedback*: Informações de status que indicam o estado contínuo do aplicativo.
- Discriminabilidade: as informações exibidas podem ser distinguidas com precisão.

Aplicando esses conceitos desenvolveu-se três telas principais descritas na subseções seguintes. Todas as telas podem ser observadas no Apêndice B.

5.4.2.1 Mapa da região

A tela principal do aplicativo é a tela de visualização do mapa. Nela é possível interagir com o mapa de diversas formas e ligar o modo manual do aplicativo. Quando o modo manual é ativado, um campo é habilitado ao usuário. Nele é possível preencher as variáveis hídricas da barragem.

No mapa é possível executar as seguintes ações:

- Visualizar os pontos de encontro próximos estabelecidos em conjunto com a defesa civil;

- Visualizar todas as rotas disponíveis na região, que não foram inundadas de acordo com os valores hídricos atuais da barragem;
- Gerar uma rota segura até um ponto de encontro, a partir da sua localização ou indicando um ponto no mapa; e
- Visualizar as construções localizadas nas ZAS e seus status (afetada pela inundação ou não).

5.4.2.2 Informações hídricas da barragem

Nessa tela o usuário pode visualizar as informações mais atuais das condições hídricas da barragem selecionada. As variáveis disponíveis são:

- Nível do reservatório;
- Vazão defluente; e
- Nível do canal de fuga;

As informações são atualizadas assim que o usuário entra na página, mas também pode-se forçar a atualização através de um botão.

5.4.2.3 Configurações

Tela de configurações e informações gerais do aplicativo. No momento do desenvolvimento deste projeto existe apenas uma ZAS registrada, porém prevendo uma futura expansão do aplicativo, nessa tela será possível selecionar outras ZAS. Também é possível visualizar informações como atribuições de ferramentas *open-source* utilizadas e a versão do aplicativo.

6 ANÁLISE

Nesse capítulo é apresentada uma avaliação do software desenvolvido. Com a finalidade de mensurar a qualidade do software produzido, utilizou o padrão internacional de avaliação de software ISO/IEC 25010:2011 (SYSTEMS. . . , 2011). O padrão estabelece oito características principais que um sistema deve ter:

- Adequação funcional;
- Eficiência de desempenhos;
- Compatibilidade;
- Usabilidade;
- Confiabilidade;
- Segurança;
- Capacidade de manutenção; e
- Portabilidade.

Todas essas características são abordadas nas seções seguintes.

6.1 ADEQUAÇÃO FUNCIONAL

A adequação funcional representa o grau em que um produto ou sistema fornece funções que atendem às necessidades declaradas e implícitas quando usado sob condições especificadas.

Tal característica é alcançada, visto que o sistema possui uma completude funcional, já que contém as quatro funcionalidades principais modeladas na Seção 4.2 e que garante os requisitos estalecidos na Seção 4.1.

6.2 EFICIÊNCIA DE DESEMPENHO

Eficiência de desempenho representa a relação da quantidade de recursos usados nas condições estabelecidas.

Foram realizados diversos testes de desempenho para *smartphones* com sistema operacional Android. Os testes foram realizados pela Google Play, plataforma responsável por distribuir aplicativos Android. O aplicativo foi testado em diferentes modelos de *smartphones* e diferentes versões do Android. Os principais resultados com relação ao uso de memória e de CPU foram compilados na Tabela 1.

Notou-se uma pequena lentidão na renderização de *frames* no *smartphone* Nokia Nokia 1, mas nada que afetasse a usabilidade ou a funcionalidade do aplicativo.

Tabela 1 – Performance de processamento em diversos *smartphones* com sistema operacional Android.

Smartphone	**Versão Android**	**Uso médio da CPU**	**Uso médio de memoria**	**Tempo de inicialização**
Nokia Nokia 1	Android 8.1	21,8%	107 MB	3180 ms
Motorola Moto E5 Play	Android 8.1	22,6%	105 MB	768 ms
Samsung Galaxy S9	Android 8.0	8,6%	196 MB	406 ms
Google Pixel 3	Android 9.0	6,3%	245 MB	-

Fonte – Autor.

6.3 COMPATIBILIDADE

Compatibilidade é o grau em que um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes ou executar suas funções necessárias enquanto compartilha o mesmo ambiente de hardware ou software.

É possível utilizar o aplicativo desenvolvido juntamente com qualquer outro software ou funcionalidade presentes nos sistemas operacionais Android e iOS, o que garante a co-existência do sistema. O sistema também possui uma interoperabilidade de dados, qualquer outro software pode trocar dados com o servidor através da API criada.

6.4 USABILIDADE

Usabilidade é o grau em que um produto ou sistema pode ser usado por usuários específicos para atingir objetivos específicos com eficácia, eficiência e satisfação em um contexto de uso especificado.

A construção da interface gráfica seguiu princípios de UX/UI, mostrado na Seção 5.4.2. Isso facilita a aprendizagem, a operabilidade e acessibilidade do software.

6.5 CONFIABILIDADE

Confiabilidade é o grau com que um sistema, produto ou componente executa funções especificadas sob condições especificadas por um período de tempo especificado.

A fim de garantir a corretude de diversas funcionalidades, aplicou-se teste unitários (Seção 3.5.1) durante o processo de CI (Seção 3.6) utilizado no desenvolvimento do software. As Tabela 2 e Tabela 3 mostram a cobertura dos testes no servidor e no gerenciamento de estado do aplicativo respectivamente. Optou-se por apenas tes-

tar o funcionamento da arquitetura Redux no aplicativo em um primeiro momento, e futuramente implementar testes nas funcionalidades da interface gráfica.

Tabela 2 – Cobertura de testes do servidor.

Arquivo	***Statements***	**Não testados**	**Cobertura**
apps/zas/admin.py	3	0	100%
apps/zas/models.py	17	5	71%
apps/zas/notifications.py	9	9	0%
apps/zas/serializers.py	6	0	100%
apps/zas/views.py	32	2	94%
Total	**67**	**16**	**76%**

Fonte – Autor.

Tabela 3 – Cobertura de testes no gerenciamento de estados do aplicativo.

Arquivo	***Statements* testados**	**Funções testadas**
src/stateReducers.ts	100%	100%
src/stateStore.tsx	100%	100%
src/state/general/generalActions.ts	100%	83,33%
src/state/general/generalReducers.ts	100%	100%
src/state/general/generalSaga.ts	96,88%	66,67%
src/state/hydrology/hydrologyActions.ts	100%	71,43%
src/state/hydrology/hydrologyReducers.ts	87,5%	100%
src/state/hydrology/hydrologySaga.ts	88,89%	85,71%
src/state/route/routeActions.ts	100%	60%
src/state/route/routeReducers.ts	81,82%	100%
src/state/route/routeSaga.ts	92,59%	75%
src/state/route/routeActions.ts	100%	60%
src/state/route/routeReducers.ts	81,82%	100%
src/state/route/routeSaga.ts	92,59%	75%
src/state/user/userActions.ts	100%	100%
src/state/user/userReducers.ts	100%	100%
src/state/user/userSaga.ts	100%	100%
Cobertura Total	**95,73%**	**80,65%**

Fonte – Autor.

Além dos testes, toda nova funcionalidade programada passava por uma revisão de software (Seção 3.5.2) por outro membro experiente na empresa.

6.6 SEGURANÇA

Segurança é o grau com que um produto ou sistema protege informações e dados para que pessoas ou outros produtos ou sistemas tenham o grau de acesso aos dados apropriado aos seus tipos e níveis de autorização.

Os *frameworks* Django e React Native utilizados na construção do sistema por padrão já possuem diversos mecanismos de segurança de diferentes formas de ataque como *Cross-site scripting* e *SQL injection*, dentre outros modos populares de *hacking*.

6.7 CAPACIDADE DE MANUTENÇÃO

Capacidade de manutenção representa o grau de eficácia e eficiência com que um produto ou sistema pode ser modificado para melhorá-lo, corrigi-lo ou adaptá-lo às mudanças do ambiente e dos requisitos.

A maior parte de todos os *frameworks* utilizados nesse projetos já são bem consolidados na empresa parceira. Outros softwares desenvolvidos pela empresa também utilizam esses *frameworks*, permitindo reuso de certas funções. A testabilidade através de testes unitários também permite uma maior confiança na implementação ou manutenção de funcionalidades, já que garante a contínua integridade do sistema.

6.8 PORTABILIDADE

Portabilidade é grau de eficácia e eficiência com que um sistema, produto ou componente pode ser transferido de um hardware, software ou outro ambiente operacional ou de uso para outro.

O *framework* React-Native utilizado, permite criar versões do aplicativo tanto para *smartphones* Android e iOS, a partir da mesma base de código, ou seja, o aplicativo é suportado nos dois sistemas operacionais. Suas eventuais atualizações também serão bastante facilitadas devido as integrações com a Play Store e Apple Store, plataformas onde o aplicativo é distribuído.

7 CONCLUSÃO

Pelo projeto apresentado ao longo do desenvolvimento deste trabalho e a análise realizada no Capítulo 6, conclui-se que os objetivos geral e específicos deste projeto foram atingidos. A prova disso é a obtenção de um novo software, ainda não existente no mercado, que permite alertar e guiar populações de zonas a jusante de barragens em situação de emergências.

Frente aos resultados obtidos, porém, alguns pontos de melhoria puderam ser levantados para que sejam desenvolvidos no futuro pela empresa parceira. Esses pontos não foram incluídos no escopo deste trabalho para que não fossem alterados os prazos relativos à entrega do projeto como um produto para a empresa. Assim, a Seção 7.1 a seguir apresenta tais pontos passíveis de melhoria futura.

7.1 PRÓXIMOS PASSOS

Devido à pandemia da COVID-19, os dados topológicos e civis da região piloto deste projeto ainda não foram levantados. Ou seja, o software não foi testado com dados reais. A partir desses teste é possível que seja necessário fazer ajustes no código e no algoritmo de roteamento para obter os melhor desempenho.

Ainda devido ao tempo limitado de execução deste PFC, algumas funcionalidades não puderam ser incluídas no escopo do projeto. Uma delas é o envio automático de SMSs juntamente com as notificações para todas as pessoas registradas no sistema. Deste modo, seria possível notificá-las mesmo que não tenham o acesso à internet.

7.2 CONSIDERAÇÕES FINAIS

Ao longo desse projeto foram utilizadas diversas ferramentas tecnológicas e conceitos aprendidos durante o curso de Engenharia de Controle e Automação da Universidade Federal de Santa Catarina.

A experiencia sem dúvidas foi bastante enriquecedora profissionalmente. Apesar do estado pandêmico atual, o projeto conseguiu ser muito bem administrado graças ao time de profissionais muito competentes da empresa parceira. Como autor deste projeto, espero que ele possa ser utilizado no futuro de modo a contribuir para a segurança geral de diversas famílias e pessoas moradoras das ZAS.

REFERÊNCIAS

ADRIAN HOLOVATY, Simon Willison. **Django**. Jul. 2005. Disponível em: https://www.djangoproject.com/. Acesso em: 29 out. 2020.

ANDREW HUNT, David Thomas. **The Pragmatic Programmer**. Harlow, Reino Unido: Addison Wesley, 1999. ISBN 9780201616224.

BELLMAN, Richard. On a routing problem. **Quarterly of Applied Mathematics**, American Mathematical Society, n. 5, p. 345, 1958.

DAN ABRAMOV, Andrew Clark. **Redux**. Jun. 2015. Disponível em: https://redux.js.org/. Acesso em: 28 out. 2020.

DIJKSTRA, Edsger W. A note on two problems in connexion with graphs. **Numerische mathematik**, Springer, n. 1, p. 269–271, 1959.

FLUX. Facebook. 2014. Disponível em: https://facebook.github.io/flux/. Acesso em: 28 out. 2020.

REACT. Facebook. Mai. 2013. Disponível em: https://reactjs.org/. Acesso em: 28 out. 2020.

REACT Hooks. Facebook. Fev. 2019. Disponível em: https://reactjs.org/docs/hooks-intro.html. Acesso em: 28 out. 2020.

REACT Native. Facebook. Fev. 2015. Disponível em: https://reactnative.dev/. Acesso em: 28 out. 2020.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado) – UNIVERSITY OF CALIFORNIA.

FLOYD, Robert W. Algorithm 97: Shortest Path. **Communications of the ACM**, American Mathematical Society, n. 16, p. 87–90, 1962.

G1. Brumadinho: mais duas vítimas do rompimento da barragem da Vale são identificadas. **Portal G1**, 28 dez. 2019. Disponível em: https://g1.globo.com/mg/minas-gerais/noticia/2019/12/28/brumadinho-mais-

duas-vitimas-do-rompimento-da-barragem-da-vale-sao-identificadas.ghtml. Acesso em: 14 out. 2020.

H. BUTLER S. GILLIES, T. Schaub. **The GeoJSON Format**. , ago. 2016. Disponível em: https://tools.ietf.org/html/rfc7946t.

HILLEBRAND, Friedhelm. **Short Message Service (SMS): The Creation of Personal Global Text Messaging**. [*S.l.*]: Wiley, 2010. ISBN 9780470688656.

SYSTEMS and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva, CH, mar. 2011.

KRUG, Steve. **Don't Make Me Think: A Common Sense Approach to Web Usability**. [*S.l.*]: New Riders, 2005. ISBN 9780321344755.

NEVES, Luiz Paniago. **Legislação Federal Brasileira em Segurança de Barragens**. : 2020. Disponível em: https://www.gov.br/anm/pt-br/assuntos/barragens/e-book-livre-legislacao-federal-brasileira-em-seguranca-de-barragens-autor-luiz-paniago-neves. Acesso em: 24 out. 2020.

OLAN, Michael. Unit testing: test eartly, test often. **Journal of Computing Sciences in Colleges**, v. 19, 2003.

REESE, George. **Database Programming with JDBC Javas**. Sebastopol, Estados Unidos da América: O'Reilly Media, Inc, 2000. cap. 8. ISBN 9781565926165.

SNISB - Sistema Nacional de Informações sobre Segurança de Barragens. 2020. Disponível em: http://www.snisb.gov.br/. Acesso em: 3 out. 2020.

SUTHERLAND, Jeff. **Scrum: The Art of Doing Twice the Work in Half the Time**. [*S.l.*]: Random House Business, 2015. ISBN 9781847941107.

SWARTOUT, Paul. **Continuous Delivery and DevOps - A Quickstart Guide**. [*S.l.*]: Packt, 2018. ISBN 9781788995474.

TORVALDS, Linus. **Git SCM**. 2005. Disponível em: https://git-scm.com/. Acesso em: 29 out. 2020.

WIEGERS, Karl E. **Peer Reviews in Software: A Practical Guide**. [*S.l.*]: Addison-Wesley Professional, 2001. ISBN 9780201734850.

APÊNDICE A – DETALHAMENTO DOS CASOS DE USO

Nesse capitulo, o conteúdo do diagrama de Casos de Uso (Figura 6) é explicado com maiores detalhes.

A.1 ATORES

Um ator é uma pessoa, organização ou sistema externo que desempenha um papel em uma ou mais interações com o sistema. Existem três atores:

- **Morador ZAS:** Morador de uma Zona de Auto-Salvamento.
- **Registro de dados da Barragem:** Sistema de registros coordenado pela empresa responsável pela barragem.
- **Gerenciador do sistema:** Funcionário da barragem, responsável por criar alertas manuais no sistema.

A.2 CASOS DE USO

Os casos de uso são um conjunto de ações, serviços e funções que o sistema precisa executar. Existem quatro casos de uso apontados:

- **Acessar dados hídricos da barragem**: Acesso aos dados dos sensores de nível e fluxo da barragem através de uma API;
- **Gerar rotas de fuga**: Geração de rotas de fuga de acordo com o nível da cota da barragem;
- **Analisar edifícios e população afetada**: Análise quantitativa de edifícios e pessoas afetadas em cada nível de cota da barragem;
- **Notificar as ZAS**: Notificação dos moradores sobre eventuais situações de risco através de notificações no sistema.

APÊNDICE B – TELAS DO APLICATIVO

Figura 11 – Tela do mapa no aplicativo com o modo manual ativo.

Fonte – Autor.

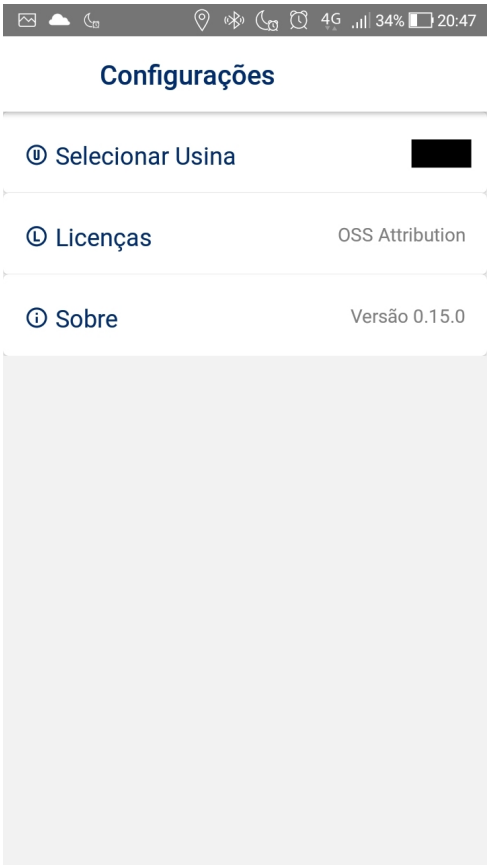
Figura 12 – Tela mostrando as construções afetadas.

Fonte – Autor.

Figura 13 – Tela de exibição dos dados hídricos da barragem no aplicativo.

Fonte – Autor.

Figura 14 – Tela de configurações do aplicativo.



Fonte – Autor.