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

Eduardo de Souza Mangrich

Projeto e Implementação de Chatbot no Processo de Autorização de Parcelas para Pagamentos na Indústria da Construção

Florianópolis, SC
2021

Eduardo de Souza Mangrich

Projeto e Implementação de Chatbot no Processo de Autorização de Parcelas para Pagamentos na Indústria da Construção

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. Ricardo José Rabelo
Supervisor: Maurício Borges dos Santos

Florianópolis, SC
2021

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Eduardo de Souza Mangrich

Projeto e Implementação de Chatbot no Processo de Autorização de Parcelas para Pagamentos na Indústria da Construção

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, 03 de Dezembro de 2021.

Prof. Hector Bessa Silveira
Coordenador do Curso

Banca Examinadora:

Prof. Ricardo José Rabelo
Orientador
UFSC/CTC/DAS

Maurício Borges dos Santos
Product Manager
Softplan Planejamento e Sistemas

Prof. Carlos Montez, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca

UFSC/CTC/DAS

Este trabalho é dedicado aos meus colegas de classe e
aos meus queridos pais.

AGRADECIMENTOS

Agradeço a Softplan pela oportunidade de trabalhar neste grandioso projeto e por todo o aprendizado transmitido pelos colegas diariamente, em especial o Maurício Borges dos Santos, Vinicius Roggia Gomes, Flávio Bernardino dos Santos Filho e Edilson Carvalho que tornaram possível a elaboração desse trabalho. Agradeço a minha família por sempre me consolar nos momentos difíceis e me motivar a seguir sempre em frente. A minha namorada Bruna por sempre me apoiar nas minhas dificuldades e me estimular a ser uma pessoa cada vez melhor.

RESUMO

O Sienge é uma plataforma que conta com um portfólio de tecnologias que se integram a ela de forma a oferecer a melhor solução para as necessidades das empresas da indústria da construção. Dentro do Sienge, no módulo financeiro, é possível cadastrar despesas no submódulo de contas a pagar e gerenciar autorizações de cada parcela referente a essas despesas antes do pagamento na tela de autorização de parcelas. Entretanto, realizar essas operações em dispositivos móveis diminuem a produtividade do usuário em razão da má usabilidade da tela que é pouco adaptável a telas menores. Isso motivou a elaboração de uma nova interface ao usuário que permitisse melhorar a usabilidade e produtividade não importando o dispositivo que fosse utilizado. Essa interface resultou na elaboração do projeto e implementação de um chatbot, uma API de autorização de parcelas para pagamento e uma API de autenticação de usuários. Para isso, foram feitas pesquisas na literatura sobre metodologias e estratégias para desenvolver o chatbot e essas APIs utilizando as melhores práticas. Com base nesses estudos, foi implementada uma autenticação de dois fatores via API, um chatbot baseado em Telegram e uma API de autorização de parcelas para pagamentos que reutiliza os serviços já existentes na tela. Essa solução trouxe uma melhora expressiva na usabilidade e produtividade garantindo segurança e menor probabilidade de ocorrência de falhas com a implementação de testes automatizados no trabalho.

Palavras-chave: Telegram, Chatbot, Autenticação, Autorização.

ABSTRACT

Sienge is a platform that relys on a portfolio of technologies that integrate with it to offer the best solution to company's needs in bulding industry. Within Sienge, in financial module, it is possible to record expenses in the submodule of bills to pay and manage authorizations of each installment related to this bill before payment in the installment authorization screen. However, performing these operations on mobile devices decrease user's productivity due to the bad usability of screen which is little adaptable to smaller sceen sizes. This fact boosted the creation of a new user interface that could allow the improvement of the usability and productivity no matter the device that was used. This interface resulted in the implementation and design of a chatbot, installment authorization API for payment and an user authentication API. To achieve that, researches were made in literature about metodologies and strategies to develop the chatbot and those apis using the best practices. Based on these studies, it has been implemented a two-factor authentication through API, a Telegram-based chatbot and an installment authorization API for payments that reuses the existing services on screen. This solution has brought a significant improvement in usability and productivity assuring safety and lower probability of ocurrence of bugs thanks to the implementation of automatized tests in the code.

Keywords: Telegram, Chatbot, Authentication, Authorization.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AIML	Artificial Intelligence Mark-up Language
ALICE	Artificial Linguistic Internet Computer Entity
API	Application Programming Interface
CSS	Cascading Style Sheets
DDL	Data Definition Language
DML	Data Manipulation Language
EAS	Enterprise Application Software
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
GPTW	Great Place to Work
HOTP	HMAC-based One-Time Password
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IOC	Inversion of Control
JAR	Java ARchive
JDBC	Java Database Connectivity
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWT	JSON Web Token
MIME	Multipurpose Internet Mail Extensions
NNTP	Network News Transfer Protocol
ORM	Object/Relational Mapping
OTP	One-Time Password
POM	Project Object Model
REST	Representational State Transfer
RMI	Remote Method Invocation
SAJ	Sistema de Automação da Justiça
SEQUEL	Structured English Query Language
SGBD	Sistema Gerenciador de Banco de Dados
Sienge	Sistema Integrado de Engenharia
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
TDD	Test-Driven Development
TOTP	Time-based One-Time Password
UNGP	Unidade da Gestão Pública
UNIC	Unidade da Indústria da Construção

UNJ	Unidade da Justiça
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAIS	Wide Area Information Server
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Vivemos na era da informação na qual os sistemas informacionais estão cada vez mais presentes em nossas vidas. Segundo Joseph Johnson (2021), aproximadamente 4.66 bilhões de pessoas foram usuárias ativas de internet em outubro de 2020, o que corresponde a 59% da população mundial. Entre esses usuários 91% realizaram o acesso por meio de algum dispositivo móvel.

Um dos tipos mais comuns de sistemas informacionais é o Enterprise Application Software (EAS), também conhecido como Enterprise Software, que cuida da eficiência dos processos de negócio nas organizações. É estimado um gasto global de 506 bilhões de dólares em 2021 com esse sistema, um aumento de 8.8% comparado ao ano anterior (LIU, 2021b).

Entre os sub-segmentos de um EAS podemos destacar o Enterprise Resource Planning (ERP), software de gestão de processo de negócio que integra diversos aspectos de uma operação de negócio (vendas, financeiro, engenharia...) em um único banco de dados, uma única aplicação e interface de usuário. Segundo Liu (2021a) o tamanho de mercado mundial estimado para 2021 é de 95 bilhões de dólares com previsão de crescimento até 97 bilhões de dólares em 2024.

De forma a atender as expectativas e requisitos dos consumidores e de parceiros de negócios, empresas estão focando cada vez mais em tecnologias móveis. Isso cria novos desafios para sistemas informacionais de negócio, especialmente os ERPs. Por exemplo, informações e funcionalidades necessárias para representantes de vendas estão provavelmente armazenadas nos ERPs e o acesso a elas através de uma interface móvel é um pré requisito para tomadas de decisão flexíveis. Esse acesso pode ajudar a tornar o ERP mais eficaz em um ambiente dinâmico (DABKOWSKI; JANKOWSKA, 2003).

Segundo uma enquete realizada pela Redshift Research Ltd na qual foram entrevistados 1500 profissionais de negócio que usam diferentes ERPs, 65% veem a mobilidade como importante para o acesso a informação e permissão da comunicação entre trabalhadores virtuais. Além disso, um em cada 2 tem algum tipo de acesso remoto ao ERP, apenas 25% tem acesso ao ERP via smartphone ou tablet, 43% querem acessar o ERP via smartphone e 38% via tablet (OMAR, 2015).

Uma das dificuldades no uso do ERP a partir de dispositivos móveis é o tamanho da tela. Segundo Sontow (2014) adotar sistemas ERP em smartphones ou tablets requer uma nova interface de usuário por causa do espaço limitado na tela e operação via toque na tela. Um outro desafio é com relação a capacidade limitada de processamento e desempenho de dispositivos móveis. Isso impactará diretamente na usabilidade do usuário (OMAR, 2015). Existe também o desafio de lidar com a usabilidade na entrada de dados do usuário. Segundo um estudo recente da Northern Illinois

University, a velocidade de digitação média em um teclado virtual foi de 25 palavras por minuto enquanto que em um teclado convencional ou em um teclado de notebook foi de 63 palavras por minuto (KIM *et al.*, 2014). Segundo outro estudo realizado por Kim *et al.* (2012), há maiores taxas de desconforto em teclados virtuais do que em teclados convencionais. E para Omar (2015), a usabilidade será prejudicada quando se utilizar um teclado virtual para um ERP móvel devido a menor velocidade de entrada de dígitos; a probabilidade de causar erros será maior e o sentimento do usuário final será de insatisfação.

Dificuldades semelhantes também são encontradas em telas do módulo financeiro dentro do Sienge (uma plataforma que possui como espinha dorsal o Sienge ERP desenvolvido pela empresa Softplan) quando essas são acessadas através de algum dispositivo móvel. Entre essas telas está a tela de autorização de pagamentos que está entre as 8 mais acessadas levando em conta os submódulos: contas a pagar, suporte a decisão e caixa e bancos. De forma a atacar essas dificuldades o autor propõe-se a desenvolver seu trabalho de conclusão de curso voltado a implementação de uma nova interface de usuário em dispositivos móveis para essa tela. Ela pertence ao módulo financeiro no qual é feito o cadastro de títulos a pagar. Esses títulos necessitam de autorizações de parcelas para pagamento, as quais podem ser realizadas por algum usuário do sistema.

A tela de autorização de pagamentos, localizada dentro do submódulo de contas a pagar, foi desenvolvida pensando em usuários que a utilizariam via computador ou notebook. Entretanto, com o avanço da tecnologia dos dispositivos móveis que favoreceu o uso de ERPs em smartphones, a usabilidade da tela nesses dispositivos ficou comprometida. Além do espaço limitado pelo tamanho da tela, o qual dificulta a navegação por toque, outro fator limitante é a velocidade de inserção de dados pelo usuário, a qual se tornou mais lenta. Esses problemas criaram a necessidade de uma interface mais amigável ao usuário quando ele desejar realizar operações a partir de um dispositivo móvel.

Existem diversas abordagens possíveis para a implementação dessa interface no processo de autorização de parcelas para pagamento em dispositivos móveis. Entre elas pode-se citar: a criação de uma nova tela de autorização de pagamentos para aplicativos em dispositivos móveis, remodelagem da tela atual no Sienge tornando-a mais responsiva para telas menores ou o uso de chatbots que permitam realizar o processo de autorização através da interação entre o usuário e o sistema. O uso de chatbots é uma abordagem que tem sido amplamente utilizada por diversas empresas, pois ela torna a realização de diversas tarefas algo intuitivo, como pedir alguma comida, ou realizar alguma reserva em algum hotel. Além disso, existem plataformas de mensageria como WhatsApp e Telegram que disponibilizam APIs que ajudam a configurar e desenvolver os diálogos por trás da conversa entre o chatbot e o usuário

o que faz aumentar a alcance dessa solução a várias pessoas.

Assim o autor propõe o desenvolvimento de um *chatbot* integrado a uma API de autorização de parcelas para pagamentos e a uma API de autenticação de usuários. Esse chatbot permitirá consultar, autorizar, desautorizar e reprovar parcelas para pagamento de forma semelhante a tela de autorização de pagamentos dentro do submódulo de contas a pagar, no módulo financeiro do Sienge. A interface do *chatbot* é similar a de serviços de mensageria como Viber, Telegram e WhatsApp os quais são utilizados por grande parte da população mundial e propõe uma interação entre robô e um usuário via troca de mensagens de forma proativa ou reativa.

De forma a implementar essa solução, inicialmente foi feita uma pesquisa bibliográfica sobre as melhores práticas no desenvolvimento de Application Programming Interface (API), técnicas envolvidas na construção de um *chatbot* e metodologias ágeis. Em seguida, iniciou-se o desenvolvimento da API pemitindo consultar informações das parcelas a pagar as quais podem ter 3 status: não autorizadas, autorizadas e reprovadas. Após o desenvolvimento da consulta via API, foi também elaborado o processo de autorização, reprovação e desautorização via API. Esse desenvolvimento deu-se utilizando o framework Spring, usando a metodologia SCRUM e a técnica TDD . Após a conclusão da API de autorização de parcelas para pagamentos, foi desenvolvida uma API para autenticação de usuários usando também o desenvolvimento guiado por testes, porém utilizando o framework Spring Boot. Finalmente, foi realizado o desenvolvimento do *chatbot* usando como linguagem o javascript com o framework Nodejs, usando a API do Telegram para acessar a interface de usuário no serviço de mensageria. A interpretação das mensagens do usuário foi tratada com o uso de expressões regulares que é uma técnica muito eficiente no processamento de texto e foi amplamente usada na interação com o *chatbot*.

1.1 OBJETIVOS

Esse trabalho o terá como objetivo geral o desenvolvimento de um *chatbot* baseado no telegram para autorização de parcelas para pagamento.

Além disso terá os seguintes objetivos específicos:

- Permitir consulta de parcelas para autorização via API;
- Permitir inserir autorização, remover autorização ou reprovar uma parcela para pagamento via API;
- Permitir autenticar o usuário do produto via API;
- Desenvolver diálogos entre o *chatbot* e o usuário permitindo realizar consultas, autorizações e reprovações via *chatbot*;

- Cobrir com testes unitários a API de autorização de pagamentos, a API de autenticação de usuários e o *chatbot* garantindo a menor ocorrência de falhas no processo;

ESTRUTURA DO TRABALHO

No capítulo 1 foi feita uma breve contextualização do problema a ser tratado nesse trabalho, levando-se em conta outros trabalhos desenvolvidos na literatura. Em seguida, o problema foi detalhado no escopo da empresa, apresentada uma possível solução e uma metodologia para resolver esse problema, finalizando a escrita com os objetivos geral e específicos. No capítulo seguinte, foi apresentada uma descrição da empresa e os respectivos processos, problemas existentes e indicadores. No capítulo 3 foi descrita toda a fundamentação teórica do trabalho, abordando teorias, conceitos e técnicas usadas na concepção da solução e a respectiva justificativa. No capítulo 4 foram descritos os requisitos funcionais e não funcionais, abordando a descrição conceitual do que foi feito, por meio de diagramas e fluxos de informação. No capítulo 5 foi detalhado o projeto que foi implementado, a forma como foi implementada a especificação descrita no capítulo anterior, abordando as tecnologias e técnicas que foram utilizadas. No capítulo 6 foi realizada a análise dos resultados levando em conta os prós e contras da solução. Foi descrita a maneira como a solução apresentada resolve o problema ou não, por meio de indicadores. No último capítulo foi apresentado um resumo dos resultados obtidos, demonstrando os avanços e pontuando as limitações apresentadas pela solução desenvolvida, sugerindo possibilidades para trabalhos futuros.

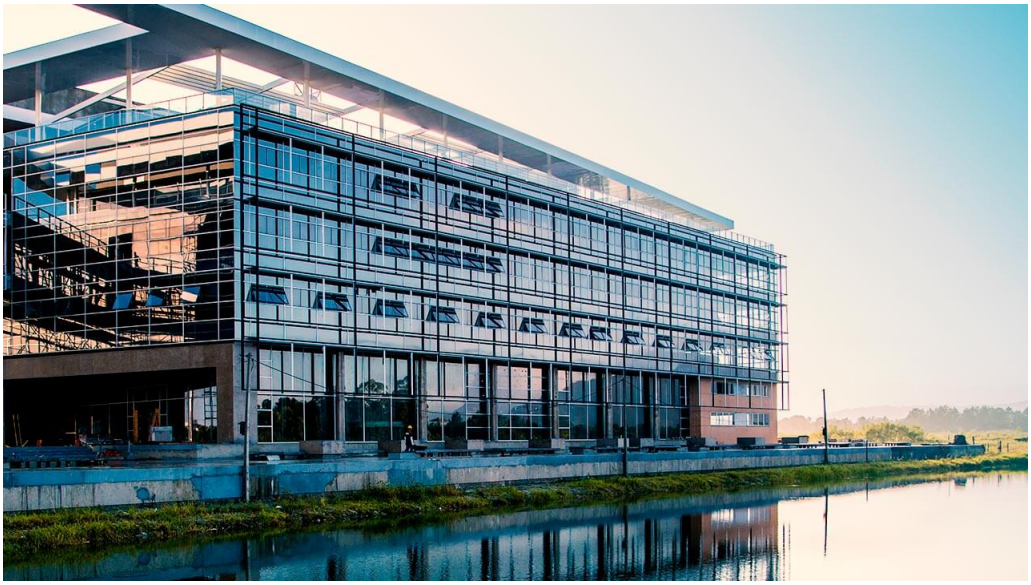
2 SOFTPLAN

Neste capítulo é apresentada a empresa Softplan levando em conta os processos, problemas e indicadores existentes em relação a ela.

2.1 DESCRIÇÃO DA EMPRESA

O projeto desenvolvido pelo autor está ligado a empresa Softplan (ver figura 1), uma empresa de software com quase 2000 colaboradores especialistas em traduzir conhecimento em softwares que simplificam e geram valor nos mercados da Indústria da Construção, Justiça e Gestão Pública. A Softplan atua no mercado de soluções de software há mais de 30 anos e teve como primeiro produto o Sistema Integrado de Engenharia (Sienge), software ERP referência para construção civil. Hoje, a empresa possui mais de 6000 clientes, presença internacional no mercado de software e está na lista da Great Place to Work (GPTW) Brasil entre as melhores empresas para se trabalhar (SOFTPLAN, 2020c).

Figura 1 – Softplan



Fonte: Softplan.

2.2 ESTRUTURA DA EMPRESA

A empresa possui 3 unidades principais: a Unidade da Justiça (UNJ), Unidade da Gestão Pública (UNGP) e Unidade da Indústria da Construção (UNIC) que atendem respectivmente os segmentos da justiça, gestão pública e indústria da construção.

2.3 SEGMENTO DA JUSTIÇA

A Softplan guia organizações na busca por mais eficiência e é pioneira na implantação do processo digital na Justiça brasileira, por meio do Sistema de Automação da Justiça (SAJ)(DIGITAL, 2020a). O SAJ é a solução que gerencia metade de todos os processos da Justiça estadual, promove a integração entre instituições do ecossistema da Justiça e aproxima ainda mais a Justiça de todos os cidadãos. Além de simplificar tarefas diárias e agilizar os procedimentos em advocacia pública, Ministérios Públicos, Tribunais de Justiça, advocacia privada e departamentos jurídicos (DIGITAL, 2020b).

2.4 SEGMENTO DA GESTÃO PÚBLICA

Além do segmento da justiça, a Softplan também assume um papel relevante na modernização dos serviços públicos desenvolvendo soluções que aproximam o governo dos cidadãos, simplificando processos, diminuindo a burocracia e tornando o setor público muito mais eficiente e transparente (SOFTPLAN, 2020d). Entre essas soluções destacam-se: o Solar BPM, que torna os processos digitais, aumentando a eficiência e a agilidade do trabalho, otimizando os recursos e oferecendo serviços online para o cidadão. O SAFF que consiste numa solução para gestão eficiente de projetos com financiamento externo, que permite integrar, compartilhar e padronizar as informações do programa. E o Obras.gov que proporciona uma solução específica para a gestão integrada e eficiente de Obras Públicas (SOFTPLAN, 2020a).

2.5 SEGMENTO DA INDÚSTRIA DA CONSTRUÇÃO

A indústria da construção que é um dos principais e mais antigos setores industriais do país também é um segmento de atuação para a Softplan. As soluções de software propostas pela empresa tornam o gerenciamento de obras, do canteiro ao escritório, muito mais simples e assertivo. Nesse segmento a Softplan possui dois principais produtos, um voltado para micro e pequenas empresas e outro voltado para médias e grandes empresas que são respectivamente o Sienge Go e o Sienge Plataforma (SOFTPLAN, 2020b).

O Sienge Go, produto criado em 2019, leva a cognição e um funcionário digital para micro e pequenas empresas. Por outro lado o Sienge Plataforma, produto estabelecido em 2018, é a única solução de mercado que une em uma plataforma o seu ERP com outros softwares e aplicativos. (SIENGE, 2018).

2.6 SIENGE

O produto Sienge passou por diversas transformações ao longo de sua história até se tornar a plataforma que é hoje. Criado em 1990, a primeira versão foi produzida

na época para o sistema operacional UNIX com interface caracter. A partir de 1993 nasce o Sienge ERP, dedicado a atender as principais áreas de uma construtora de forma integrada. Após melhorias de interface e outras evoluções, em 2007 é lançado o Sienge Web, que foi o primeiro software de gestão de empresas da construção 100% web (SIENGE, 2018).

Hoje o Sienge se posiciona no mercado como uma plataforma, contando com um portfólio de tecnologias que se integram a ela de forma a oferecer a melhor solução para as necessidades das empresas da indústria da construção. Além disso, o Sienge plataforma possui como espinha dorsal o Sienge ERP que atende as mais diversas áreas de uma construtora em um único software (SIENGE, 2018). Dentre os módulos do Sienge ERP que oferecem essas soluções podem-se citar;

- Sienge Suporte a Decisão;
- Sienge Engenharia;
- Sienge Financeiro;
- Sienge Comercial;

2.7 SIENGE FINANCEIRO

O sienge financeiro é o módulo que representa a gestão de compromissos financeiros das empresas clientes do sienge. Nesse módulo é feita a gestão dos pagamentos, dos recebíveis, transferências e verificação dos saldos de contas e do caixa da empresa. Ela é dividida em 3 partes principais: caixa e bancos, contas a receber e contas a pagar (SIENGE, 2012).

2.8 CAIXA E BANCOS

Esse submódulo tem como principais funções: conciliar movimentos bancários a partir de extratos bancários, emitir cheques para pagamentos de compromissos e analisar o extrato de contas correntes cadastradas no sistema (SIENGE, 2012).

2.9 CONTAS A RECEBER

O submódulo de contas a receber possui funções como: integração de arquivos recebíveis de mais de 60 instituições financeiras, controle de clientes inadimplentes, renegociação de títulos a receber de forma fácil e intuitiva, criação de relatórios utilizando APIs de reparcelamento, extrato de cliente entre outras (SIENGE, 2012).

2.10 CONTAS A PAGAR

Já o submódulo de contas a pagar permite cadastrar compromissos de pagamentos futuros, programar todos os pagamentos de alguma empresa cliente do Sienge, enviar e receber arquivos de pagamento de alguma instituição financeira e finalmente gerenciar autorizações de pagamentos (SIENGE, 2012).

No módulo do contas a pagar, todo título (despesa) cadastrado por algum usuário cria uma parcela a pagar com status não autorizado por padrão. Essa parcela deve ser autorizada por outro usuário de acordo com o critério de autorização definido no sistema. Com a autorização realizada, o usuário que criou a parcela poderá realizar a baixa da parcela (pagamento da parcela pelo sistema) e ter controle sobre todas as despesas que foram pagas via relatório de contas pagas disponível no sistema.

2.11 AUTORIZAÇÃO DE PAGAMENTO

Dentro do submódulo do contas a pagar, existe uma tela de autorização de pagamentos. Nessa tela é possível consultar parcelas para autorização inserindo diversas opções de filtros como identificador da empresa do título vinculado a parcela, número do título vinculado a parcela, período de vencimento da parcela, entre outros. Após a consulta, existe a possibilidade de gerenciar autorizações, alterando a autorização de alguma parcela. No Sienge a forma como a parcela é autorizada é parametrizada. Assim, a regra usada para definir se alguma parcela está autorizada ou não dependerá de qual comportamento estiver habilitado pelo parâmetro. Existem 5 opções de comportamento nesse parâmetro:

- Opção 'Nenhum': Não utiliza o controle por alçadas. Assim a autorização de apenas um usuário é suficiente para que a parcela fique autorizada para pagamento;
- Opção 'Quantidade': A parcela só fica autorizada quando alcançar o número de autorizações determinado por um outro parâmetro;
- Opção 'Valor': A parcela fica autorizada somente se alcançar o número de autorizações configurados na alçada;
- Opção 'Valor + Hierarquia': A parcela só fica autorizada quando alcançar o número de autorizações configurados na alçada e seguindo a hierarquia de alçadas definida;
- Opção 'Departamento': A parcela só fica autorizada quando alcançar o número de autorizações configurados na alçada do departamento;

Cada usuário pode inserir uma autorização em uma determinada parcela, ou remover a autorização caso ela já tenha sido dada pelo mesmo usuário. A parcela

ficará ou não autorizada, se a condição para que fique autorizada seja atendida, de acordo com o comportamento do parâmetro.

Além disso, na opção 'Valor + Hierarquia' existe a possibilidade de reprovar a parcela. Quando isso ocorre a parcela não pode mais ser autorizada, ou desautorizada por nenhum usuário.

2.12 PROBLEMA

Hoje, a tela de autorização de pagamentos dentro do Sienge deixa a desejar com relação a usabilidade em dispositivos móveis. A inserção de filtros de busca e navegação na tela é dificultada quando o tamanho de tela é limitado o que diminui a produtividade e acessibilidade da tela. Além disso, o processo de autorização de pagamentos só é acessível por meio do produto Sienge, obrigando o cliente a realizar o processo de autenticação e navegação até a tela de autorização toda vez que desejar realizar alguma autorização ou reprovação.

Levando em conta esse cenário, o autor desse trabalho propõe o desenvolvimento de um *chatbot* baseado em Telegram que irá consumir uma API de autorização de parcelas para pagamentos e uma API de autenticação de usuários, ambos também desenvolvidos pelo autor desse trabalho, o que permitirá a consulta de parcelas do contas a pagar e autorizar ou reprovar parcelas para pagamento.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 CHATBOT

Chatbots são programas de computadores que interagem com usuários usando linguagens naturais. Essa tecnologia começou nos anos 60, o objetivo era ver se sistemas de *chatbots* conseguiam enganar usuários passando-se por verdadeiros humanos (SHAWAR; ATWELL, 2007). Segundo o Lexico (2021), é um programa de computador projetado para simular conversação com usuários humanos, especialmente pela internet.

A demanda por agentes conversacionais se tornou aguda com o difundido uso de máquinas pessoais, com o desejo de se comunicar e com o desejo dos criadores de fornecer interfaces de linguagem natural. Assim como as pessoas usam a linguagem para comunicação humana, pessoas querem usar a linguagem delas para comunicação com computadores. Zadrozny *et al.* (2000) concordou que a melhor forma de facilitar a interação computador-humano é permitir aos usuários expressar os interesses, desejos ou consultas deles diretamente ou naturalmente pela fala, escrita e indicação. Esse foi o motivador por trás do desenvolvimento dos *chatbots*. Diferentes termos tem sido usados para um *chatbot* como: sistema de conversação de máquina, agente virtual, sistema de diálogo e chatterbot. O propósito de um sistema de *chatbot* é simular uma conversa humana. A arquitetura do *chatbot* integra o modelo de linguagem e algoritmos computacionais para emular comunicação em chat informal entre um usuário humano e um computador usando linguagem natural (SHAWAR; ATWELL, 2007).

Inicialmente desenvolvedores construíam e usavam *chatbots* por diversão. Além disso usavam técnicas de combinação de palavras-chave para achar uma correspondência da entrada do usuário como ELIZA (WEIZENBAUM, 1966). Nos anos 70 e 80, antes da chegada das interfaces gráficas para usuários, houve um rápido crescimento na pesquisa de texto e interfaces de linguagem natural. Desde aquela época, uma variedade de novas arquiteturas de *chatbots* foram desenvolvidas como: MegaHAL (HUTCHENS; ALDER, 1998), CONVERSE (BATACHARIA *et al.*, 1999), ELIZABETH (SHAWAR; ATWELL, 2002) e Artificial Linguistic Internet Computer Entity (ALICE) (WALLACE, 2004) que ganhou o prêmio Loebner em 2000, 2001, e 2004. Com as melhorias da mineração de dados e técnicas de aprendizado de máquina, melhores capacidades de tomada de decisão, anotações linguisticas robustas / padrões de ferramentas de processamento como eXtensible Markup Language (XML) e as respectivas aplicações, os *chatbots* se tornaram mais práticos com muitas aplicações comerciais (GLOTZ; BRAUN, 2013) (SHAWAR; ATWELL, 2007).

ALICE foi primeiramente implementada por Wallace em 1995. O conhecimento de Alice sobre padrões de conversação em inglês foi armazenado em arquivos Artificial

Intelligence Mark-up Language (AIML) que são derivados do XML. Foi desenvolvido por Wallace e a comunidade de código aberto Alicebot desde 1995 para permitir a pessoas inserir conhecimento de padrões de diálogo baseado na tecnologia de código aberto ALICE. AIML consiste de objetos de dados chamados de objetos AIML, que são compostos de unidades chamadas de tópicos e categorias. O tópico é um elemento de alto nível opcional que tem um atributo de nome e um conjunto de categorias relacionados a aquele tópico. Categorias são a unidade básica de conhecimento em AIML. Cada categoria é uma regra de correspondência para uma entrada e conversão a uma saída, e consiste em um padrão que corresponde a entrada do usuário e um modelo, que é usado na geração da resposta do chatbot ALICE (SHAWAR; ATWELL, 2007). O formato do AIML é da seguinte forma:

```
<aiml version="1.0">
    <topic name="the topic">
        <category>
            <pattern>PATTERN</pattern>
            <that>THAT</that>
            <template>Template</template>
        </category>
         ..
         ..
    </topic>
</aiml>
```

A tag <that> é opcional e significa que o padrão atual depende de uma saída anterior do chatbot. O padrão AIML é simples consistindo apenas de palavras, espaços e símbolos de asterisco. As palavras podem consistir em letras, numerais, mas nenhum outro caractere. Palavras são separadas por um único espaço e caracteres genéricos funcionam como palavras. A linguagem de padrão não diferencia letras minúsculas e maiúsculasl. A ideia da técnica de correspondência por padrão é baseada em achar a melhor, mais longa correspondência de padrão (SHAWAR; ATWELL, 2007).

Existem 3 tipos de categorias: categorias atômicas, categorias padrão e categorias recursivas.

- Categorias atômicas: são aquelas com padrões que não tem símbolos genéricos como _ e $*$;

```
<category>
    <pattern>10 Dollars</pattern>
    <template>Wow, that is cheap. </template>
</category>
```

Na categoria acima, se o usuário inserir como entrada ‘10 dollars’, então a ALICE responde ‘WOW, that is cheap’.

- Categorias padrão: são aquelas com padrões que tem símbolos genéricos como _ e $*$. Os símbolos genéricos correspondem a qualquer entrada mas eles diferem na ordem alfabética deles. Assumindo que a entrada anterior é ’10 Dollars’, se o robô não achar a categortia anterior com um padrão atômico, então ele vai tentar achar uma categoria com o padrão seguinte (SHAWAR; ATWELL, 2007):

```
<category>
    <pattern>10 *</pattern>
    <template>It is ten.</template>
</category>
```

Então a Alice responde com ’It is ten.’

- Categorias recursivas. são aquelas com modelos tendo tags <srai> e <sr> que referem-se a regras de redução recursiva. Categorias recursivas tem muitas aplicações: redução simbólica que reduz formas gramaticais complexas para mais simples, dividir e conquistar que separa uma entrada em duas ou mais subpartes e combina as respostas uma com a outra e lidar com sinônimos mapeando diferentes formas de dizer a mesma coisa para a mesma resposta (SHAWAR; ATWELL, 2007).

- Redução simbólica;

```
<category>
    <pattern>DO YOU KNOW WHAT THE * IS</pattern>
    <template>
        <srai>What is <star/></srai>
    </template>
</category>
```

Nesse exemplo <srai> é usado para reduzir a entrada a uma forma mais simples ’what is’

- Dividir e conquistar

```
<category>
    <pattern>YES*</pattern>
    <template>
        <srai>YES</srai>
        <sr/>
    <template>
```

```
</category>
```

A entrada é dividida em duas partes, 'yes' e a segunda parte '*' é correspondida com a tag <sr/>.

- Sinônimos;

```
<category>
    <pattern>HALO</pattern>
    <template>
        <srai>Hello</srai>
    </template>
</category>
```

A entrada é mapeada em uma outra forma, que tem o mesmo significado.

Antes de o processo de correspondência começar, o processo de normalização é aplicado para cada entrada, para remover toda a pontuação a entrada é dividida em duas ou mais frases se apropriado e convertida para letra maiúscula. Por exemplo, se a entrada é "I do not know. Do you, or will you, have a robots.txt file?" então após a normalização será "DO YOU OR WILL YOU HAVE A ROBOTS DOT TXT FILE" (SHAWAR; ATWELL, 2007).

Existem mais de 50000 categorias no 'cérebro' da ALICE no atual domínio público que foi lentamente construido por muitos anos pelo Richard Wallace, o pesquisador que manteve e editou o banco de dados da ALICE original. Entretanto, essas categorias são todas codificadas a mão, o que é demorado e restringe a adaptação a novos domínios de discurso e novas linguagens (SHAWAR; ATWELL, 2007).

Desde as tentativas iniciais com ELIZA e ALICE , até a ambição de desenvolver agentes inteligentes capazes de conversar com humanos completamente usando linguagem natural, bots foram crescentemente empregados em diversos campos com sucesso discreto, mas com limitada disseminação. Eles foram aplicados em e-commerce, como Nicole, um assistente virtual com tarefas de serviço ao consumidor, ou como Anna pela IKEA na educação, como Charlie, um bot que permite estudantes comunicarem com a plataforma de aprendizado online INES ou TQ-bot dedicação a tutoria de estudantes e avaliação. A maioria desses bots são baseados em AIML e ALICE (KLOPFENSTEIN *et al.*, 2017).

Após o sucesso de chats no navegador no início dos anos 90 como IRC(Internet Relay Chat), o fim dessa década viu a disseminação de serviços de mensagem instantânea como AIM (AOL Instant Messaging), Yahoo! Messenger, e MSN Messenger. Muitas dessas plataformas permitiram a usuários adicionar bots para a lista de contato deles como se fossem pessoas reais. Dessa forma eles poderiam trocar mensagens de texto simples e receber diferentes tipos de informação (KLOPFENSTEIN *et al.*, 2017).

Há apenas poucos anos que, assistentes inteligentes começaram a chamar atenção do público maior. Segundo McTear et al., diferentes razões influenciaram esse novo sucesso de interfaces conversacionais: o progresso em tecnologias assistivas de inteligência artificial, como reconhecimento de fala e imagem, emergência da web semântica, disponibilidade crescente da conectividade, melhorias no hardware dos dispositivos e o interesse renovado de grandes participantes da tecnologia. O Siri da Apple (considerado o primeiro assistente privado virtual público e habilitado para voz), Cortana da Microsoft, Google Now, Google Assistant, Alexa da Amazon e Samsung S Voice são os principais atores dos últimos 5 anos no campo das interfaces conversacionais (KLOPFENSTEIN *et al.*, 2017).

Também é válido mencionar o agente de respostas estatístico da IBM chamado de 'Watson' que entrou no centro das atenções em 2010 por participar do programa de televisão 'Jeopardy'. Naquela ocasião Watson venceu os dois competidores humanos dele confiando apenas em um banco de dados desconectado de conteúdos não estruturados. Em anos recentes, a IBM desenvolveu Watson com mais profundidade, transformando-o num poderoso assistente de inteligência artificial empregado em cenários de assistência médica e cuidado personalizado. O Watson também pode ser explorado como um serviço cognitivo baseado na nuvem de blocos de construção compostos de IA. Desenvolvedores podem treinar a IA do Watson para responder questões feitas em linguagem natural sobre uma intenção particular e usa-la para construir aplicações como chatbots (KLOPFENSTEIN *et al.*, 2017).

Nos últimos anos, uma nova abordagem para interfaces conversacionais vem ganhando destaque: um aparente retorno as interfaces de texto clássicas foi observado, com a principal diferença que esses novos bots ganharam capacidades adicionais e agora 'vivem' na nuvem (KLOPFENSTEIN *et al.*, 2017).

Desde 2014, muitos sistemas de mensagem online (como Kik, Telegram e WeChat) abriram-se para desenvolvedores externos, oferecendo os meios para construir bots e programaticamente trocar mensagens com usuários pela plataforma deles. API para bots expõe serviços de alto nível (mensageria, pagamentos, diretório de bot, etc.) e elementos de interface de usuário (botões, locais, imagens, etc.) permitindo a desenvolvedores implementar serviços inovadores através de uma experiência de usuário conversacional. Bots incluem acesso a outros serviços que tem utilidade no dia a dia como, pedir comida, gerenciar uma compra em um e-commerce, reservar restaurantes e assim por diante. Por exemplo o Nombot, que é um bot de assistência médica que ajuda usuários a rastrear o consumo diário de comida deles no Telegram (KLOPFENSTEIN *et al.*, 2017).

A possibilidade de construir bots mais práticos e convenientes é em boa parte devido a maior disponibilidade de serviços de APIs abertas a terceiros e a ascensão do modelo de negócio plataforma (KLOPFENSTEIN *et al.*, 2017).

Entre a infinidade de novas e velhas interfaces conversacionais pode-se identificar uma categoria de agentes conversacionais que tendo sido projetados para princípios de simplicidade e eficácia podem ser como substitutos funcionais de aplicações móveis. São chamados de aplicações bot (KLOPFENSTEIN *et al.*, 2017).

Uma aplicação bot é um agente que é dotado de uma interface conversacional acessível por uma plataforma de mensageria que fornece acesso a dados, serviços ou permite ao usuário realizar uma tarefa específica. Uma aplicação bot é geralmente caracterizada pelas seguintes funcionalidades: consciência do histórico de mensagens, interface de usuário aprimorada, uso limitado de processamento de linguagem natural e conversa guiada (KLOPFENSTEIN *et al.*, 2017).

Bots oferecem algumas vantagens para usuários como: disponibilidade instantânea, curva de aprendizado suave, notificações, assincronicidade, requisitos de dados limitados (KLOPFENSTEIN *et al.*, 2017).

Segundo Brandtzaeg e Følstad (2017), um questionário online perguntava aos usuários de chatbot (N = 146, idade 16–55 anos) dos Estados Unidos para relatar as razões deles para usar chatbots. O estudo identifica os fatores motivacionais principais que levam ao uso do chatbot. O fator mais frequentemente reportado é a produtividade, chatbots ajudam usuários a obter assistência ou informação eficiente e a tempo (BRANDTZAEG; FØLSTAD, 2017).

Abaixo estão alguns conceitos fundamentais para a tecnologia do chatbot:

- Correspondência de padrões: é baseado em blocos de estímulo-resposta representativos. Uma sentença é inserida e a resposta é criada consistente com a entrada do usuário. Eliza e ALICE foram os primeiros chatbots desenvolvidos usando algoritmos de reconhecimento de padrões. A desvantagem dessa abordagem é que as respostas são inteiramente previsíveis, repetitivas e carecem de um toque humano. Também não há armazenamento de respostas passadas, o que pode levar a conversações em laço (ADAMOPOULOU; MOUSSIADES, 2020).

- Análise semântica latente: pode ser usada em conjunto com AIML para o desenvolvimento de chatbots. Ela é usada para descobrir semelhanças entre palavras como uma representação vetorial. Perguntas baseadas em modelos como saudações e questões gerais pode ser respondidas usando AIML enquanto outras questões não respondidas usam LSA para dar respostas (ADAMOPOULOU; MOUSSIADES, 2020).

- Chatscript: é um sistema inteligente, que consiste em uma linguagem de script de código aberto e o motor que a executa. É composta de regras que estão associadas com tópicos, encontrando o melhor item que corresponde ao texto de consulta de usuário. Chatscript também inclui memória de longo prazo na forma

de variáveis que podem ser usadas para armazenar informações específicas de usuário como o nome ou idade do usuário. Também diferencia letras maiúsculas e minúsculas, ampliando as respostas possíveis que podem ser dadas a mesma entrada do usuário baseada na emoção desejada (ADAMOPOULOU; MOUSSIADES, 2020).

- Processamento de linguagem natural (NLP): é uma área da inteligência artificial que explora a manipulação de texto de linguagem ou discurso por computadores. Conhecimento do entendimento e uso de linguagem humana é reunido para desenvolver técnicas que farão computadores entender e manipular expressões naturais para realizar tarefas desejadas. A maior parte das técnicas em NLP são baseadas em aprendizado de máquina (ADAMOPOULOU; MOUSSIADES, 2020).

- Compreensão de linguagem natural (NLU): está no centro de qualquer tarefa NLP. É uma técnica para implementar interfaces de usuário naturais como o chatbot. NLU busca extrair contexto e significados das entradas de usuário de linguagem natural que podem estar desestruturadas e responder apropriadamente de acordo com a intenção do usuário. Ela identifica a intenção do usuário e extrai entidades específicas do domínio. Mais especificamente, uma intenção representa um mapeamento entre o que o usuário diz e qual ação deve ser tomada pelo chatbot. Ações correspondem a passos que o chatbot irá tomar quando intenções específicas são disparadas por entradas do usuário e pode ter parâmetros para especificar informações detalhadas sobre isso. Detecção de intenção é tipicamente formulada como classificação de sentença no qual uma única ou múltiplas intenções são previstas para cada sentença (ADAMOPOULOU; MOUSSIADES, 2020).

Uma entidade é uma ferramenta para extrair valores de parâmetros de entradas de linguagem natural. Por exemplo, considerando a sentença 'Como está o tempo na Grécia?'. A intenção do usuário é aprender sobre a previsão do tempo. O valor da entidade é Grécia. Logo, o usuário pede pela previsão do tempo na Grécia. Entidades podem ser definidas pelo sistema ou definidas pelo desenvolvedor (ADAMOPOULOU; MOUSSIADES, 2020).

Finalmente, contextos são textos que armazenam o contexto do objeto que o usuário está se referindo ou está falando sobre. Por exemplo, um usuário pode se referir a um objeto anteriormente definido na sentença seguinte dele. Um usuário pode entrar com 'Ligue o ventilador'. Aqui o contexto salvo é o ventilador para que quando o usuário disser, 'Desliga isso' como próxima entrada, a intenção 'desligar' seja invocada sobre o contexto 'ventilador' (ADAMOPOULOU; MOUSSIADES, 2020).

3.2 LINGUAGENS DE PROGRAMAÇÃO

3.2.1 Java

A linguagem de programação Java foi inicialmente chamada de Oak e foi planejada para aplicações em aparelhos eletrônicos embarcados por James Gosling. Após anos de experiências e contribuições de Ed Frank, Patrick Naughton, Jonathan Payne, e Chris Warth a linguagem foi redirecionada para a internet. Ela foi projetada para ter o menor número de dependências de implementação possível e permitir a desenvolvedores escrever o programa uma vez e executá-lo em qualquer lugar na internet. Essa linguagem é de propósito geral, concorrente, baseada em classes e orientada a objetos. Ela é fortemente tipada e essa especificação cria uma distinção entre erros de compilação que podem e devem ser detectados em tempo de compilação e aqueles que ocorrem em tempo de execução. O tempo de compilação consiste na tradução de programas em representação byte code independente da máquina. Enquanto que o tempo de execução abrange o carregamento e enlace de classes necessárias para executar o programa, otimização dinâmica do programa, geração de código de máquina opcional e a execução do programa (GOSLING *et al.*, 2005).

A linguagem Java é relativamente uma linguagem de alto nível na qual detalhes da representação da máquina não estão disponíveis pela linguagem. Ela gere automaticamente o armazenamento. (GOSLING *et al.*, 2005).

3.2.2 JavaScript

JavaScript é uma linguagem de programação Web. Grande parte dos sites e navegadores modernos utiliza JavaScript nos mais diversos dispositivos como computadores de mesa, consoles de jogos, tablets e smartphones tornando essa a linguagem de programação mais onipresente da história. Ela faz parte da tríade de tecnologias que todos os desenvolvedores Web devem conhecer: Hypertext Markup Language (HTML) cujo objetivo é apresentar o conteúdo das páginas, Cascading Style Sheets (CSS) que especifica como a página será apresentada e JavaScript que especifica o comportamento dela. Essa linguagem é de alto nível, dinâmica, interpretada e não tipada. JavaScript foi criada na NetScape (hoje Mozilla) sendo marca registrada e licenciada pela Sun Microsystems (atual Oracle) usada para descrever a implementação da linguagem (FLANAGAN, 2011).

3.3 HTTP

O Hypertext Transfer Protocol (HTTP) é um protocolo da camada de aplicação para sistemas colaborativos, distribuídos e informacionais de hipermídia. Esse proto-

colo tem sido usado pela iniciativa global World-Wide Web desde 1990 (NIELSEN *et al.*, 1999).

A primeira versão desse protocolo conhecida como HTTP/0.9, era um simples protocolo para transferênca de dados não tratados pela internet. A próxima versão, HTTP/1.0, definida pela RFC 1945, melhorou o protocolo permitindo mensagens no formato Multipurpose Internet Mail Extensions (MIME), contendo meta-dados sobre o dado transferido e modificadores na semântica da requisição ou da resposta. Entretanto a versão HTTP/1.0, não levou suficientemente em conta efeitos da hierarquia de proxies, cache, a necessidade de conexões persistentes, ou hosts virtuais. Além disso, o surgimento de aplicações implementadas de forma incompleta se autodenominando HTTP/1.0 tornou necessária uma mudança na versão do protocolo de forma que duas aplicações comunicadoras pudessem determinar as verdadeiras capacidades de cada uma (NIELSEN *et al.*, 1999).

O HTTP também é usado como um protocolo genérico para comunicação entre agentes de usuário, proxies/gateways até sistemas de internet incluindo aqueles suportados pelos protocolos Simple Mail Transfer Protocol (SMTP), Network News Transfer Protocol (NNTP), File Transfer Protocol (FTP), Gopher e Wide Area Information Server (WAIS) (NIELSEN *et al.*, 1999).

3.4 API

API é uma coleção de código existente que outros programadores podem chamar para ajudar a realizar tarefas de programação. Geralmente as APIs estão disponíveis somente na forma compilada, como uma interface. APIs são criadas e usadas por diversas razões. Primeiramente, elas podem poupar tempo aos desenvolvedores ao prover funcionalidade que um desenvolvedor poderia criar, mas que seria mais rápido reutilizar. Além disso, APIs proporcionam ocultação de informação de forma que dentro de uma única aplicação, detalhes de implementação podem ser alterados sem afetar o código que usa a API. Finalmente, muitas APIs fornecem acesso a funcionalidade que não é facilmente acessível sem as APIs (STYLOS; MYERS, B., 2007).

API é usada para descrever interfaces grandes e pequenas. É também usada, as vezes, para descrever interfaces entre dois componentes no mesmo programa. O termo API inclui frameworks, bibliotecas, kit de ferramenta e kit de desenvolvimento. Uma linguagem não é uma API. A linguagem inclui uma sintaxe e compilador ou interpretador enquanto que uma API é sempre construída em uma linguagem. Uma ferramenta não é uma API, pois uma API consiste de binários, arquivos de definição e documentação, mas não aplicações binárias. Um código fonte de uma aplicação não é uma API, se o código que está sendo criado for usado por apenas uma aplicação (STYLOS; MYERS, B., 2007).

Historicamente, APIs existem desde o advento de computadores pessoais. APIs

surgiram principalmente para a troca entre 2 ou mais programas. A emergência das APIs na WEB (o que é mais conhecido como APIs Web) foi, todavia, presenciado por volta do ano 2000 (BOATENG; OFOEDA; EFFAH, 2019).

3.4.1 API Web

APIs web são usadas como um mecanismo chave de interconectividade para acessar serviços de software pela internet. Aplicações interconectadas podem fornecer um serviço melhor a um custo menor para os usuários delas. Por exemplo, um portal de buscas de negócios pode exibir um negócio próximo ao local atual do usuário em um mapa usando a API do Google Maps. A implementação de APIs Web pode seguir vários protocolos como serviços Web SOAP. APIs web são definidas pela interface HTTP delas, incluindo cabeçalhos HTTP assim como formatos de dado de resposta e de requisição (SOHAN; ANSLOW; MAURER, 2015).

3.4.2 API REST

Representational State Transfer (REST) é um estilo arquitetural para sistemas de hipemídia distribuídos que fornece um conjunto de restrições arquiteturais que, quando aplicadas como um todo, enfatiza a escalabilidade de interações de componente, generalidade de interfaces, distribuição independente de componentes, componentes intermediários para reduzir latência da interação, reforça a segurança e encapsula sistemas legados. A funcionalidade central que distingue o estilo arquitetural REST de outros estilos baseados na rede é a ênfase em uma interface uniforme entre componentes (FIELDING, 2000).

As primeiras restrições aplicadas ao estilo são aquelas do estilo arquitetural cliente-servidor. Separação de responsabilidades é o princípio por trás das restrições cliente-servidor. Separando responsabilidades de interface de responsabilidades de armazenamento de dados, melhora-se a portabilidade da interface do usuário por múltiplas plataformas e melhora a escalabilidade simplificando componentes de serviço (FIELDING, 2000).

Em seguida adiciona-se uma restrição a interação cliente-servidor. A comunicação deve ser sem estado por natureza, de tal forma que cada requisição do cliente ao servidor deve conter toda a informação necessária para entender a requisição e não pode tirar vantagem de qualquer contexto armazenado no servidor. O estado da sessão é então mantido inteiramente no cliente (FIELDING, 2000).

Há também restrições de sistemas com camadas. O estilo de sistema com camadas permite a uma arquitetura ser composta de camadas hierárquicas restringindo comportamento do componente de tal forma que cada componente não pode "ver"além da camada imediata com a qual eles estão interagindo. Restringindo o conhecimento

de um sistema a uma única camada, coloca-se uma fronteira na complexidade geral do sistema e promove independência do substrato (FIELDING, 2000).

3.5 SPRING

O framework Spring é uma solução leve e única para construção de aplicações apropriadas para empresas. O Spring é modular permitindo usar apenas aquelas partes que forem necessárias. É possível utilizar o container Inversion of Control (IOC) com struts ou é possível utilizar o código de integração do Hibernate ou a camada de abstração do Java Database Connectivity (JDBC). O framework Spring é compatível com gerenciamento de transação declarativa, acesso remoto a lógica por meio do Remote Method Invocation (RMI) ou serviços Web e várias outras opções para persistência de dados. Spring é projetado para ser não intrusivo, o que significa que a lógica de domínio não tem dependências no próprio framework. Spring é uma plataforma Java que proporciona um suporte de infraestutura abrangente para o desenvolvimento de aplicações Java. O framework lida com a infraestrutura para permitir ao desenvolvedor focar na aplicação (JOHNSON, R. *et al.*, 2008)

3.6 TESTES DE SOFTWARE

Testar é o processo de executar um programa com a intenção de encontrar erros. Já o conceito de teste de software se trata de um processo ou uma série de processos projetada para garantir que o código faça aquilo que foi projetado para fazer e que por outro lado não faça nada não intencional. O software deve ser previsível e consistente sem apresentar surpresas aos usuários. Uma das principais razões de aplicações pobres em testes é a falsa definição do termo teste usada pelos programadores. Seguem-se algumas dessas falsas definições (MYERS, G. J.; SANDLER, 2004):

- Testar é o processo de demonstração que erros não estão presentes;
- O propósito do teste é demonstrar que os programas realizam as funções esperadas corretamente;
- Testar é o processo de estabelecimento de confiança que o programa faz aquilo que deveria fazer.

Quando se deseja testar algum programa, busca-se agregar valor a ele. Agregar valor ao programa através de testes significa aumentar a qualidade ou a confiabilidade do programa. Aumentar a confiabilidade do programa significa encontrar e remover erros (MYERS, G. J.; SANDLER, 2004).

Existem algumas técnicas conhecidas que são utilizadas no projeto de casos de teste. Entre elas destacam-se:

- Cobertura de lógica: Testa todas as saídas de pontos de decisão pelo menos uma vez e garante que todas as declarações e pontos de entrada são executadas pelo menos uma vez;
- Particionamento de equivalência: Define condição ou classes de erro para ajudar a reduzir o número de testes finitos. Assume que um teste de um valor representativo dentro de uma classe testa todos os valores e condições dentro daquela classe;
- Análise de valores-limite: Testa cada condição de borda de uma classe de equivalência. Também considera classes de equivalência de saída assim como classes de entrada;
- Gráficos de causa e efeito: Produz representações gráficas booleanas de casos de teste potenciais para ajudar a selecionar casos de teste completos e eficientes.
- Adivinhação de erros: Produz casos de teste baseado em conhecimento intuitivo e especialista de membros do time de teste para definir potenciais erros de teste com o objetivo de facilitar o projeto de casos de teste eficiente (MYERS, G. J.; SANDLER, 2004).

  Testes de integração determinam se unidades de software independentes funcionam corretamente quando elas estão conectadas uma a outra (FOWLER, 2018). O teste ponta a ponta é um teste que executa a maioria das partes de uma grande aplicação. Esse tipo de teste frequentemente manipula a aplicação pela interface de usuário como no teste de uma aplicação web usando Selenium e Sahi (FOWLER, 2013). (MYERS, G. J.; SANDLER, 2004).

Testes podem ser classificados com relação ao tamanho do escopo de cobertura e propósito. Eles podem ser: testes unitários, testes de integração ou testes ponta a ponta.

3.6.1 Testes unitários

Teste unitário é um processo de teste de programas subindividuais, subrotinas, classes ou procedimentos em um programa. Antes de testar o programa como um todo, o teste é focado em blocos de construção menores de um programa. Existem 3 motivações para isso: a primeira é que testes unitários são uma forma de gerir elementos combinados de teste já que é focado em menores unidades do programa. A segunda é que testes unitários facilitam a tarefa de depuração já que quando erros são

encontrados eles existem em um módulo específico. A última é que testes unitários introduzem paralelismo no processo de teste ao dar a oportunidade de testar múltiplos módulos simultaneamente. O propósito do teste unitário é comparar a função de um módulo a alguma especificação funcional ou interface que defina o módulo. São necessários dois tipos de informação para realizar esse teste: especificação para o módulo e o código fonte do módulo. A especificação do módulo define os parâmetros de entrada e saída e a respectiva função. Testes unitários são amplamente orientados a caixa branca, pois ao testar entidades maiores como programas inteiros, testes de caixa branca se tornam menos viáveis. (MYERS, G. J.; SANDLER, 2004).

3.6.2 TDD

Test-Driven Development (TDD) é um conjunto de técnicas comprovado que encoraja um projeto simples e uma suíte de testes que inspira confiança. No desenvolvimento guiado a testes escrevemos novo código se um teste automatizado falhou e eliminamos duplicação. O ciclo do TDD funciona da seguinte forma: primeiro é necessário escrever o teste. É preciso imaginar como a operação aparecerá no código. Em seguida faça o teste passar. Finalmente, remova a duplicação de código garantindo que o teste continua passando. (BECK, 2002)

3.7 ARQUITETURA DE SOFTWARE

A arquitetura de um sistema de software é o formato dado a aquele sistema por aqueles que o construíram. A configuração desse formato está na divisão desse sistema em componentes, no arranjo desses componentes e na forma pela qual esses componentes se comunicam (MARTIN, 2017).

3.7.1 Arquitetura em camadas

O padrão de arquitetura mais comum é o padrão de arquitetura em camadas, conhecido também como padrão de arquitetura de n-camadas. Este padrão é a norma para grande parte das aplicações Java EE e logo é amplamente conhecido pela maioria dos arquitetos, projetistas e desenvolvedores. A arquitetura em camadas corresponde de forma muito próxima a comunicação tradicional da tecnologia da informação e estruturas organizacionais encontradas na maioria das empresas, tornando-a uma escolha natural para a maioria dos esforços em desenvolvimento de aplicações de negócios (RICHARDS, 2015).

Componentes dentro do padrão de arquitetura em camadas são organizados em camadas horizontais, cada camada realizando um papel específico dentro da aplicação(apresentação da lógica ou lógica de negócio por exemplo). Embora o padrão de arquitetura em camadas não especifique o número e os tipos de camadas que

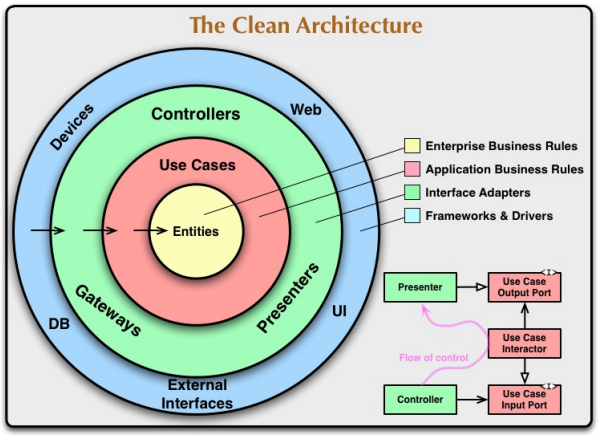
devem existir no padrão, a maioria das arquiteturas em camadas consistem em 4 camadas padrões: apresentação, negócio, persistência e banco de dados. Em alguns casos, a camada de negócio e persistência são combinadas em uma única camada de negócios, particularmente quando a lógica de persistência(SQL ou HSQL por exemplo) é embutida dentro dos componentes da camada de negócio. Logo, aplicações menores poder ter só 3 camadas, enquanto que maiores e mais complexas aplicações de negócio podem conter 5 ou mais camadas (RICHARDS, 2015).

Cada camada do padrão de arquitetura em camadas tem um papel específico e responsabilidade dentro da aplicação. Uma das poderosas características do padrão de arquitetura em camadas é a separação de preocupações entre componentes. Componentes dentro de uma camada específica lidam apenas com a lógica que pertence a aquela camada. Por exemplo, componentes na camada de apresentação lidam apenas com lógica de apresentação, enquanto que componentes localizados na camada de negócio lidam apenas com lógica de negócio (RICHARDS, 2015).

3.7.2 Arquitetura limpa

Arquitetura limpa é um estilo arquitetural que busca a separação de responsabilidades em diversas camadas (representadas por círculos concêntricos) nas quais as dependências do código fonte devem apontar sempre para dentro (ver figura 2). Essa arquitetura garante a independência de frameworks, tornam as regras de negócio testáveis sem interface de usuário, banco de dados, serviço web, ou qualquer elemento externo. Ela favorece a independência das regras de negócio com relação a banco de dados, interface de usuário ou qualquer agente externo (MARTIN, 2017).

Figura 2 – The Clean Architecture



Fonte: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

3.8 PADRÕES DE PROJETO

Padrões de projeto são descrições de objetos e classes que se comunicam e que são customizados para resolver um problema geral de projeto em um contexto particular. Segundo Christopher Alexandre, cada padrão descreve um problema que ocorre de forma repetida em nosso ambiente, e então descreve o centro da solução para aquele problema de tal forma que você pode usar aquela solução um milhão de vezes sem nunca aplicar ela da mesma forma duas vezes. O que ele diz é verdadeiro para padrões de orientação a objetos (GAMMA *et al.*, 1994).

3.9 PERSISTÊNCIA

Persistência é um dos conceitos fundamentais no desenvolvimento de aplicações. Quando tratamos de persistência em Java, falamos sobre armazenar informações em um banco de dados relacional usando Structured Query Language (SQL). Um sistema de gerenciamento de banco de dados relacional não é algo específico de Java e um banco de dados relacional não é algo específico de alguma aplicação particular. A tecnologia relacional proporciona uma forma de compartilhar dados entre diferentes aplicações ou entre diferentes tecnologias que fazem parte da mesma aplicação. A tecnologia relacional é um denominador comum de muitos sistemas distintos e plataformas de tecnologia. Logo o modelo de dados relacional é frequentemente uma representação corporativa comum de entidades de negócio (BAUER; KING, 2004).

3.10 BANCO DE DADOS RELACIONAL

Um sistema de banco de dados é basicamente só um sistema computadorizado de manutenção de registros, ou seja, um sistema computadorizado cujo propósito geral é armazenar informações e permitir a usuários recuperar e atualizar informações sob demanda. O banco de dados por si só pode ser considerado como um tipo de arquivo eletrônico, ou seja, é um repositório ou container para uma coleção de arquivos de dados computadorizados. Usuários desse sistema podem realizar uma variedade de operações envolvendo esses arquivos como:

- Adicionar novos arquivos ao banco de dados.
- Inserir dados a arquivos existentes.
- Recuperar dados de arquivos existentes.
- Deletar dados de arquivos existentes.
- Alterar dados em arquivos existentes.

- Deletar arquivos existentes do banco de dados (DATE, 2003).

Requisições SELECT, INSERT, DELETE e UPDATE (também chamados de comandos, operadores, ou declarações) são todos expressos em uma linguagem chamada de SQL. Originalmente uma linguagem proprietária da IBM, SQL é hoje um padrão internacional suportado por qualquer produto de banco de dados disponível comercialmente. Em SQL arquivos computadorizados são chamados de tabelas, as linhas dessa tabela são chamadas de registros do arquivo e as colunas são chamadas de campos.(DATE, 2003)

Os dados em um banco de dados serão geralmente, pelo menos em um sistema grande, ambos integrados e compartilhados. Ser integrado significa que o banco de dados é uma unificação de arquivos distintos. Ser compartilhado significa que o banco de dados pode ser compartilhado entre diferentes usuários no sentido em que diferentes usuários podem acessar o mesmo dado possivelmente até ao mesmo tempo. Entre o banco de dados físico, ou seja, o dado armazenado fisicamente, e os usuários do sistema está uma camada de software conhecida de forma variada como gerenciador de banco de dados ou servidor de banco de dados ou de forma mais comum o Sistema Gerenciador de Banco de Dados (SGBD). Todas as requisições para acesso ao banco de dados são operadas pelo SGBD. Uma função geral do SGBD é a blindagem de usuários de banco de dados de detalhes do nível de hardware. Além disso, o SGBD suporta operações de usuário como operações SQL (DATE, 2003).

Um banco de dados é uma coleção organizada de informações estruturadas, ou dados, tipicamente armazenados eletronicamente em um sistema de computador. O dado nos tipos mais comuns de banco de dados em operação hoje é tipicamente modelado em filas e colunas em uma série de tabelas para tornar o processamento e consulta de dados eficiente. O dado pode então ser facilmente acessado, gerenciado, modificado, atualizado, controlado e organizado (ORACLE, 2021).

Diversos modelos foram desenvolvidos como o hierárquico, de rede, relacional e orientado a objetos. Esses incluem principalmente a Data Definition Language (DDL) para especificar aspectos estruturais do dado e Data Manipulation Language (DML) para acessar e atualizar o dado. A separação entre a definição lógica do dado da implementação física é central no campo de bancos de dados. Essa separação é frequentemente chamada de princípio da independência dos dados. Essa é seguramente a distinção mais importante entre sistemas de arquivos e sistemas de banco de dados (ABITEBOUL; HULL; VIANU, 1995).

Um modelo relacional proporciona meios para especificar estrutura de dados particulares para restringir conjuntos de dados associados a essas estruturas e para manipulação de dados. O termo modelo relacional refere-se a ampla classe de modelos de banco de dados que tem relações como a estrutura de dados e que incorpora algumas ou todas as capacidades de consultas, capacidades de atualização e restrições

de integridade. Relações são estruturas de dados simples, cada tabela é chamada de relação e possui um nome. Colunas também possuem nomes chamados de atributos, cada linha é chamada de registro ou tupla. As entradas das tuplas vem de um conjunto de constantes, chamado de domínio que inclui por exemplo, conjunto de inteiros, strings e valores booleanos (ABITEBOUL; HULL; VIANU, 1995).

3.11 HIBERNATE

Hibernate é uma implementação aberta de Object/Relational Mapping (ORM). É um projeto ambicioso que busca ser uma solução completa ao problema de gerir a persistência de dados em Java. O Hibernate media a interação da aplicação com um banco de dados relacional deixando o desenvolvedor livre para focar no problema de negócio. É uma solução não intrusiva, assim não é obrigatório seguir muitas regras específicas do Hibernate ou padrões de projeto ao escrever regras de negócio e classes persistentes (BAUER; KING, 2004).

3.12 SQL

SQL é a linguagem padrão para sistemas de gerenciamento de bancos de dados relacionais. Quando ela se originou nos anos 70, a linguagem específica de domínio foi feita para cumprir a necessidade de conduzir consultas de banco de dados que pudessem navegar por uma rede de ponteiros para encontrar o local desejado. Quando SQL foi estabelecido, era chamado de Structured English Query Language (SEQUEL). Entretanto, devido a uma questão de direitos autorais, foi alterado para SQL (KELECHAVA, 2018).

3.13 AUTENTICAÇÃO

Por definição, é o uso de um ou mais mecanismos para provar que você é quem você declara ser. Existem hoje 3 fatores de autenticação universalmente conhecidos: o que você tem, o que você é e o que você sabe. A autenticação por dois fatores é um mecanismo que implementa dois dos fatores mencionados acima e portanto é considerado mais forte e mais seguro que o sistema tradicional de autenticação de 1 fator. Senhas são conhecidas por serem um dos alvos mais fáceis de hackers. Logo a maioria das organizações está procurando por métodos mais seguros para proteger os clientes e funcionários delas (ALOUL; ZAHIDI; EL-HAJJ, 2009).

Um token de segurança é um dispositivo físico que um usuário autorizado de serviços de computador recebe para auxiliar na autenticação. Também é conhecido como token de autenticação ou token criptográfico. Tokens podem vir em dois formatos: hardware ou software. Tokens de hardware são pequenos dispositivos que podem

ser convenientemente carregados. Alguns deles armazenam senhas criptográficas ou dados biométricos enquanto outros exibem um PIN que muda com o tempo. Nesses casos quando um usuário desejar logar ele deve inserir o PIN exibido naquele momento além da senha da conta comum. Já tokens de software são programas que executam em computadores e fornecem um PIN que muda com o tempo. Esses programas implementam o algoritmo One-Time Password (OTP). Algoritmos OTP são críticos para sistemas que os utilizam já que usuários não autorizados não devem ser capazes de adivinhar a próxima senha da sequência. A sequência deve ser a mais aleatória possível, imprevisível e irreversível. Fatores que podem ser usados na geração OTP incluem nomes, tempo, semente (ALOUL; ZAHIDI; EL-HAJJ, 2009).

3.14 ALGORITMO TOTP

Basicamente definimos Time-based One-Time Password (TOTP) como TOTP = HOTP(K, T), onde T é um inteiro e representa o número de passos no tempo entre o contador inicial de tempo T0 e o tempo atual Unix. No algoritmo HMAC-based One-Time Password (HOTP) é aplicado um valor de contador crescente representando a mensagem na computação HMAC. O HOTP é definido como: HOTP(K,C) = Truncate(HMAC-SHA-1(K,C)) no qual K é a chave pública e C o valor contador. TOTP é a variante baseada no tempo do algoritmo HOTP na qual um valor T derivado de uma referência de tempo e um passo no tempo substitui o contador C. Implementações TOTP podem usar funções HMAC-SHA-256 ou HMAC-SHA-512, baseados nas funções hash SHA-256 ou SHA-512 [SHA2](VIEW *et al.*, 2011).

3.15 JWT (JSON WEB TOKEN)

JSON Web Token (JWT) é um meio compacto e seguro para Uniform Resource Locator (URL) de representar declarações a serem transferidas entre dois interessados. As declarações no JWT são codificadas como um objeto Javascript Object Notation (JSON) que é usado como carga de uma estrutura JSON Web Signature (JWS) ou como um texto simples de uma estrutura JSON Web Encryption (JWE) permitindo as declarações serem assinadas digitalmente e/ou criptografado. O JWT é feito para ambientes com restrições de espaço como cabeçalhos de autorização HTTP ou parâmetros de consulta Uniform Resource Identifier (URI). Uma declaração é um pedaço de informação afirmado sobre um assunto. Ela é representada como um nome ou par de valores consistindo de uma declaração de nome e declaração de valor (JONES; BRADLEY; SAKIMURA, 2015).

Um cabeçalho JWT é um objeto JSON que descreve as operações criptográficas aplicadas ao JWT. O token é composto pelo cabeçalho JWT codificado em base 64 seguido de um ponto, seguido do conjunto de declarações codificado em base

64, seguido de um ponto, seguido da assinatura do token JWT (JONES; BRADLEY; SAKIMURA, 2015).

3.16 EXPRESSÕES REGULARES

Expressões regulares são a chave para o processamento de texto eficiente, flexível e poderoso. Com uma notação geral de padrão quase como uma mini linguagem de programação, permite descrever e analisar um texto. Com um suporte adicional fornecido por uma ferramenta particular sendo usada, expressões regulares podem adicionar, remover, isolar e geralmente dobrar todos os tipos de texto e dados (FRIEDL., 2007).

Por exemplo, a expressão "[123456]" corresponde a qualquer um dos dígitos listados de 1 a 6. Com o meta caracter "- " é possível definir uma faixa de caracteres. Logo "[0-9]" e"[a-z]" são formas abreviadas comuns para classes que correspondem a dígitos e letra minúscula da língua inglesa. Um outro exemplo é a expressão "^cat$" que significa literalmente: corresponde se a linha possui um início de linha (o que, claro, todas as linhas possuem) seguido imediatamente por um c, a, t e então seguido imediatamente pelo fim da linha (FRIEDL., 2007).

3.17 TELEGRAM

O Telegram é um aplicativo de mensagens desktop e móvel baseado na nuvem com foco em segurança e velocidade. Ele oferece dois tipos de APIs para desenvolvedores livres de custo. O Bot API que permite facilmente criar programas que usem mensagens do Telegram para uma interface e o Telegram API e TDLib que permitem construir clientes Telegram customizados. O Bot API permite conectar bots ao sistema Telegram. Bots Telegram são contas especiais que não requerem um número adicional para configuração. Essas contas servem como interface para código que executa em algum lugar no servidor do cliente (TELEGRAM, 2021b).

Bots são aplicações de terceiros que executam dentro do Telegram. Usuários podem interagir com bots enviando mensagem a eles, comandos e requisições em linha. Os bots são controlados através de requisições Hypertext Transfer Protocol Secure (HTTPS) para o Bot API. Para criar um bot basta conversar com o BotFather(um outro bot), seguir alguns passos e receber o token de autenticação (TELEGRAM, 2021a)

3.18 GIT

Uma ferramenta que gerencia e rastreia diferentes versões de software ou outro conteúdo é referida como um sistema de controle de versão, gerenciador de código

fonte, sistema de controle de revisão entre outras. O Git é uma ferramenta de controle de versão de baixo custo, particularmente poderosa, flexível, que torna prazeroso o desenvolvimento colaborativo. Foi inventado por Linus Torvalds para apoiar no desenvolvimento do kernel do Linux, mas se provou valioso para uma grande variedade de projetos (LOELIGER, 2009).

3.18.1 GitLab

GitLab começou como um projeto de código aberto para contribuir na colaboração dos times no desenvolvimento de software. A missão do GitLab é fornecer um lugar onde todos possam contribuir. Cada membro do time usa o produto internamente e diretamente impacta nos planos da empresa. Gitlab é uma única aplicação que abrange o ciclo de vida inteiro do desenvolvimento de software (GITLAB, 2021b).

É uma plataforma de devops que empodera organizações a maximizar o retorno total em desenvolvimento de software, entregando software mais rápido, de forma mais eficiente, fortalecendo segurança e conformidade (GITLAB, 2021a).

3.19 APACHE MAVEN

Apache Maven é um software para gerenciamento de projetos e ferramenta de compreensão. Baseado no conceito Project Object Model (POM), Maven pode gerenciar a construção de um projeto com documentação a partir de um pedaço central de informação (MAVEN, 2021a).

Maven, uma palavra iídiche(língua de origem indo-europeia), que significa acumulador de conhecimento, começou com uma tentativa de simplificar processos de construção no projeto Jakarta Turbine. Era desejável uma forma padrão de construir projetos, uma clara definição do que o projeto era composto, uma forma fácil de publicar informações do projeto, e uma forma de compartilhar Java ARchive (JAR)s através de vários projetos. O maior objetivo do Maven é permitir ao desenvolvedor compreender o completo estado do esforço de desenvolvimento no menor período de tempo. Para atingir esse objetivo o Maven trabalha com algumas preocupações como: tornar o processo de construção fácil, fornecer um sistema de construção uniforme, fornecer informações de projeto com qualidade e encorajar melhores práticas de desenvolvimento (MAVEN, 2021b).

3.20 METODOLOGIAS ÁGEIS

A articulação do manifesto ágil em 2001 trouxe mudanças sem precedentes no campo de engenharia de software. É difícil pensar numa década no século 20 que presenciou a introdução de tantos métodos de software, ferramentas, técnicas e melhores práticas (DINGSØYR *et al.*, 2012). Segundo Beck *et al.* (2001), são mais valorizados:

indivíduos e interações sobre processos e ferramentas, software que funciona sobre documentação abrangente, colaboração do cliente sobre negociação de contratos e resposta a mudanças sobre seguir um plano. Uma série de métodos aderindo em diversos graus aos princípios do manifesto apareceram a vista. Esses incluem eXtreme Programming (XP), Scrum, desenvolvimento de software enxuto e desenvolvimento dirigido por funcionalidades. De forma geral, todos esses métodos tentaram abordar os principais princípios do manifesto. Primeiramente houve um movimento nítido em direção ao desenvolvimento colaborativo com pessoas sendo concedidas com privilégios mais do que os processos que antigamente as restringiam. Em seguida, foi defendida a mentalidade enxuta com o objetivo de minimizar trabalho desnecessário, particulamente com relação a criação de documentação supérflua (DINGSØYR *et al.*, 2012).

3.20.1 Scrum

Scrum é uma metodologia de manutenção, melhoria e gestão para um sistema existente ou protótipo de produção. Ela assume um projeto existente e código que é virtualmente sempre o caso do desenvolvimento orientado a objetos devido a presença de bibliotecas de classe. Scrum vai abordar esforços em sistemas de desenvolvimento legados novos ou, remodelados em uma data posterior. O nome Scrum é baseado numa formação para reposição de bola no jogo de rúgbi (SCHWABER, 1997).

3.21 CÓDIGO LIMPO

Segundo Grady Booch, autor de Object Oriented Analysis and Design with Applications, código limpo é simples e direto, se lê como prosa bem escrita, nunca obscurece a intenção do projetista, mas ao invés disso é cheio de abstração pura e linhas diretas de controle. Segundo Ron Jeffries, autor de Extreme Programming Installed and Extreme Programming Adventures in C#, código limpo é feito com duplicação reduzida de código, alta expressividade e construção antecipada de simples abstrações. Finalmente, segundo Ward Cunningham, inventor de Wiki, Fit, coinventor de eXtreme Programming, sabe-se que se está trabalhando com código limpo quando cada rotina que é lida acaba sendo exatamente aquilo que se esperava, pode-se chamar de código bonito quando o código também faz parecer que aquela linguagem foi feita para o problema (MARTIN; COPLIEN, 2009).

3.22 PRINCÍPIOS SOLID

Os princípios SOLID indicarão como organizar funções e estruturas de dados em classes e como essas classes devem estar interconectadas. O objetivo desses princípios é a criação de estruturas de softwares medianas que: toleram mudança,

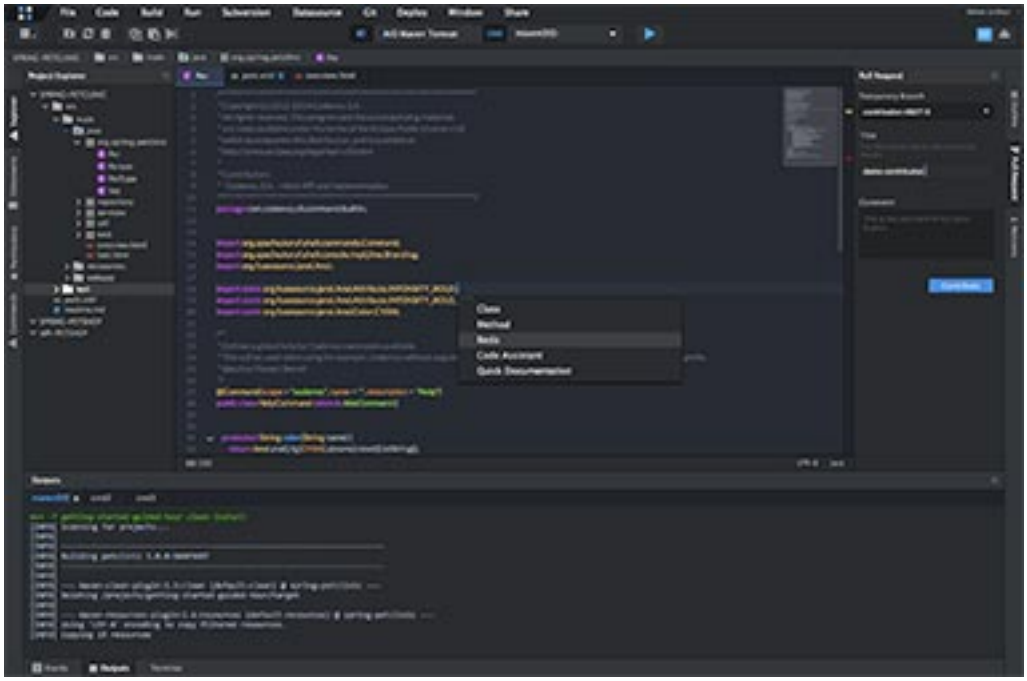
são fáceis de entender e que são a base de componentes que podem ser usados em muitos sistemas de software. O termo mediana se refere ao fato que esses princípios são aplicados por desenvolvedores que trabalham no nível de módulo. Os princípios podem ser resumidos em: o princípio da responsabilidade única, o princípio do aberto e fechado, o princípio de substituição de Liskov, o princípio de segregação de interfaces e o princípio de inversão de dependência (MARTIN, 2017).

- O princípio da responsabilidade única: Um corolário ativo da lei de Conway no qual a melhor estrutura para um sistema de software é altamente influenciada pela estrutura social da organização que usa ela de tal forma que cada módulo de software tem uma e apenas uma razão para mudar;
- O princípio do aberto e fechado: Bertrand Meyer tornou esse princípio famoso nos anos 80. A essência é que para sistemas de software serem fáceis de mudar, eles devem ser projetados para permitir o comportamento desses sistemas mudar pela adição de novo código ao invés de alterar código já existente;
- O princípio de substituição de Liskov: A definição famosa de subtipos de Barbara Liskov de 1988. Em resumo, esse princípio afirma que para construir sistemas de software de partes intercambiáveis, essas partes devem aderir a um contrato que permita a essas partes serem substituídas por uma outra;
- O princípio da segregação de interfaces: Esse princípio aconselha projetistas de software a evitar depender de coisas que eles não usam;
- O princípio da inversão de dependência: O código que implementa políticas de alto nível não deveria depender de código que implementa detalhes de baixo nível. Ao invés disso, detalhes deveriam depender de políticas (MARTIN, 2017);

3.23 IDE

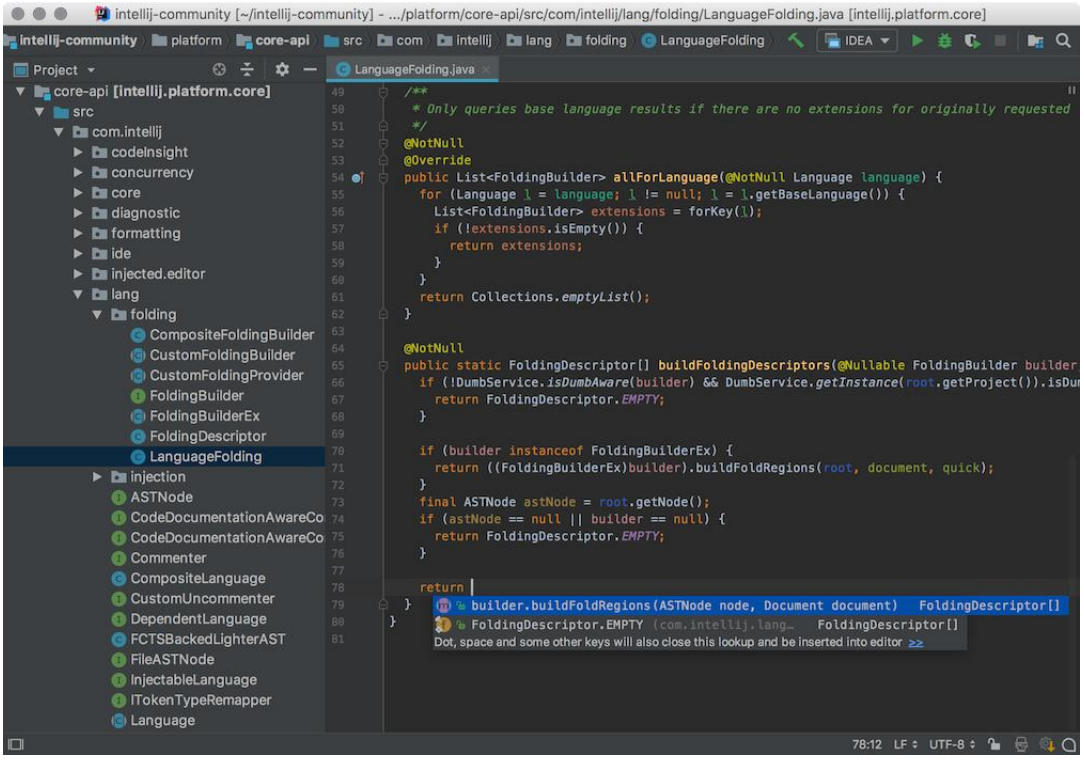
Integrated Development Environment (IDE), é um software para programadores que unifica codificação, compilação, diagnóstico e ferramentas de depuração em uma única interface (DICTIONARY.COM, 2021). Entre as IDEs disponíveis no mercado para Java e Javascript destacam-se: Eclipse IDE, IntelliJ IDEA e Visual Code Studio.

Figura 3 – Eclipse IDE



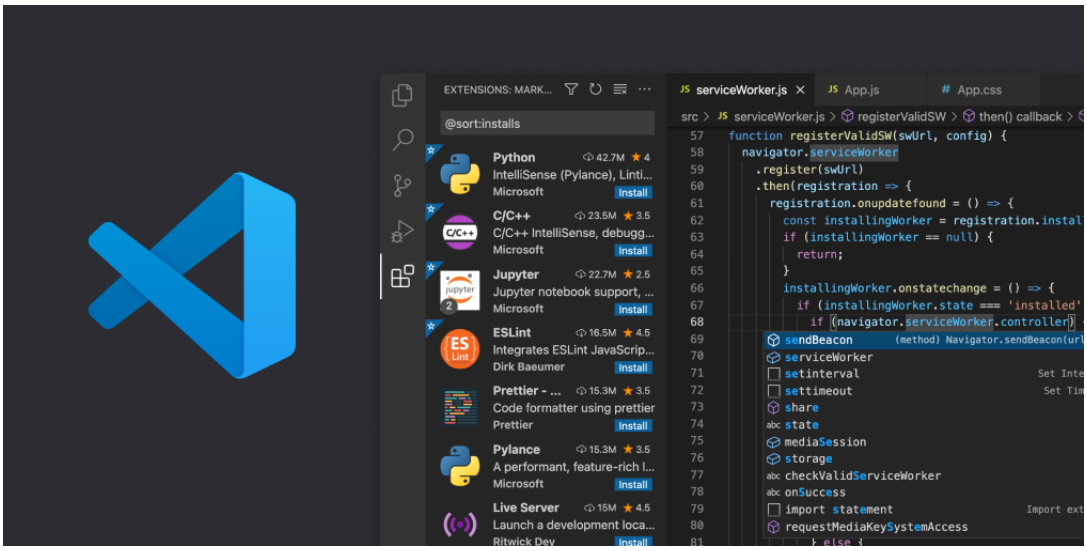
Fonte: https://www.eclipse.org/ide/.

Figura 4 – IntelliJ IDEA



Fonte: https://www.jetbrains.com/pt-br/idea/.

Figura 5 – Visual Code Studio



Fonte: https://code.visualstudio.com/.

4 REQUISITOS E METODOLOGIA

Nesse capítulo serão apresentados todos os requisitos funcionais e não funcionais desse projeto. Em seguida será apresentada a solução conceitual utilizada e por fim a metodologia aplicada para alcançar a proposta final do trabalho.

4.1 REQUISITOS GERAIS

Permitir ao usuário gerenciar a autorização das parcelas para pagamento de qualquer lugar por meio de um *chatbot*.

4.2 REQUISITOS FUNCIONAIS

- Autenticar o usuário.
- Consultar parcelas para pagamento com informação do status de autorização
- Inserir e remover autorização de parcelas para pagamento
- Reprovar parcelas para pagamento.
- Elaborar relatório das parcelas consultadas.

4.3 REQUISITOS NÃO FUNCIONAIS

- Os dados da consulta devem ser obtidos por meio de um método GET de uma API Web.
- A autorização, remoção de autorização e reprovação de parcelas deverão ser realizadas por meio de um método POST de uma API Web.
- A API Web deverá ser feita em Java utilizando o framework Spring.
- As operações de consulta, inserção de autorização, remoção de autorização e reprovação deverão ser realizadas através de um *chatbot*.
- A geração do relatório a partir da consulta das parcelas será feita por meio do *chatbot*.
- A autenticação será realizada por uma API Web utilizando Java e o framework Spring Boot.
- A autenticação será do tipo dois fatores utilizando JWT e TOTP.
- O *chatbot* deverá ser configurado utilizando a plataforma do Telegram e o Bot API.

4.4 LINGUAGENS DE PROGRAMAÇÃO

A linguagem de programação usada nas APIs foi Java. A razão para isso é que todas as classes da camada de serviço (escritas em Java) que são acessadas pelas ações da tela de autorização de pagamentos foram reutilizadas no projeto da API de autorização de parcelas para pagamento. Essa reutilização permitiu o reaproveitamento de código, evitando duplicidade nas definições das regras de negócio.

Além disso, o uso generalizado dessa linguagem pelos times de desenvolvimento e maior familiaridade com a sintaxe da linguagem são aspectos que pesaram a favor na escolha dessa linguagem em comparação a outras.

Com relação a linguagem escolhida para o desenvolvimento do *chatbot*, foi usado Javascript que também é uma linguagem já amplamente utilizada no Sienge e por isso permitiu maior facilidade no uso.

4.5 AMBIENTES DE DESENVOLVIMENTO

A IDE usada para esse projeto foi o Eclipse IDE for Enterprise Java and Web Developers, pois ela oferece ferramentas de desenvolvimento para Java, possui integração com o Maven (gerenciador de dependências da API de autorização de parcelas e da API de autenticação de usuários), é uma IDE gratuita e amplamente utilizada dentro da empresa.

4.6 FRAMEWORKS

O framework usado para a API de autorização de parcelas para pagamento foi o Spring framework. Esse framework é baseado em Java, amplamente difundido no código legado do Sienge, permitindo criar classes com terminais de conexão do protocolo Http, integrando facilmente com soluções ORM como o Hibernate que permitem o mapeamento das classes em Java para as tabelas do banco de dados.

Já para a solução da API de autenticação foi usado o Spring Boot, um framework baseado em Java, usado para construir aplicações baseadas em Spring usando micro serviços. Como a solução da API da autenticação de usuários independe do Sienge tendo um serviço e comunicação com o banco de dados próprio, foi a solução mais adequada nesse caso.

Finalmente, para a implementação do *chatbot* foi usado o framework Node.js que permite o uso de código em javascript fora do navegador. Esse framework é usado junto a biblioteca do Telegraf para configuração dos Bots e desenvolvimento da conversa entre usuário e o *chatbot*.

4.7 VERSIONAMENTO DE CÓDIGO

Para esse projeto foi utilizada a ferramenta Git, para gerenciar o versionamento de código ao longo do desenvolvimento das APIs e do *chatbot*. Além disso, foi utilizado o GitLab para ser o hospedeiro do repositório remoto das aplicações desenvolvidas nesse trabalho. A escolha do GitLab foi baseada no fato que esse serviço já é amplamente usado pelo time de desenvolvimento em outras demandas.

4.8 CAMADA DE PERSISTÊNCIA DE DADOS

A persistência de dados na API de autorização de pagamentos é realizada em 3 tipos de bancos de dados relacionais: Postgres, Firebird e SQL Server. O uso desses bancos nesse projeto é devido a necessidade de dar suporte a clientes que utilizam bases de dados em um desses 3 bancos. Na API de autenticação de usuário, é usada apenas a base Postgres.

Com relação a inserção e atualização de dados, tanto a API de autorização de parcelas para pagamento quanto a API de autenticação de usuários utilizam a JPA (Java Persistence API) para realizar essas operações.

4.9 TÉCNICAS DE DESENVOLVIMENTO

Uma das técnicas de desenvolvimento utilizada nas APIs de autorização e de autenticação foi o desenvolvimento guiado por testes (TDD). Isso garante que o código seja escrito da forma mais simples possível e com menor probabilidade de ocorrência de falhas devido ao fato de essa metodologia ter como resultado a realização de testes unitários e uma ampla cobertura de testes.

Além do TDD, foram aplicados testes de regressão em algumas partes do código, principalmente no desenvolvimento do *chatbot*. Esses testes foram realizados devido a não familiaridade com a biblioteca do Telegraf o que tornou necessária a aplicação de testes após a codificação de algum cenário entre o usuário e o *chatbot*.

Na API de autorização de pagamentos, foi necessária uma maior legibilidade na forma como o objeto de filtros para autorização é construído, devido a não utilização de todos os filtros da tela de autorização de pagamentos e haver ainda sim muitos filtros para a construção desse objeto. Para isso foi utilizado o padrão de projeto builder que permite abstrair a escrita de alguns filtros e ainda facilitar a construção do objeto com uma interface fluente.

Foi usada também a técnica de código limpo que tem por objetivo tornar o código mais legível, aplicando técnicas de refatoração como extração de métodos, declaração de variáveis o mais perto possível de onde for utilizar, refatoração de classes muito grandes em classes menores e uso de nomes que correspondam a termos da regra

de negócio para facilitar o entendimento do uso no código.

Além dessas técnicas, houve uma grande preocupação com a separação de responsabilidades entre as classes do projeto. Isso evita que uma classe fique com um alto acoplamento a longo prazo dificultando a reutilização dela em outros componentes. Para alcançar esse objetivo foi aplicado o princípio da responsabilidade única do SOLID.

Na API de autenticação foi usado também o princípio de inversão de dependência para isolar o domínio da aplicação de todas as tecnologias envolvidas na solução como frameworks, bancos de dados específicos entre outras.

4.10 METODOLOGIA DE DESENVOLVIMENTO

A metodologia usada pelo autor desse trabalho foi a metodologia ágil. Dessa forma, buscou-se melhorias e alterações constantes com base no contato com o supervisor local desse trabalho e em revisões de código feitas pelo time de desenvolvimento. De forma complementar, também foi utilizada a metodologia Scrum de tal maneira que o progresso e pendências do projeto sempre eram discutidos em reuniões diárias, sempre refletindo sobre o que foi feito e como pode ser melhorado.

4.11 ARQUITETURA

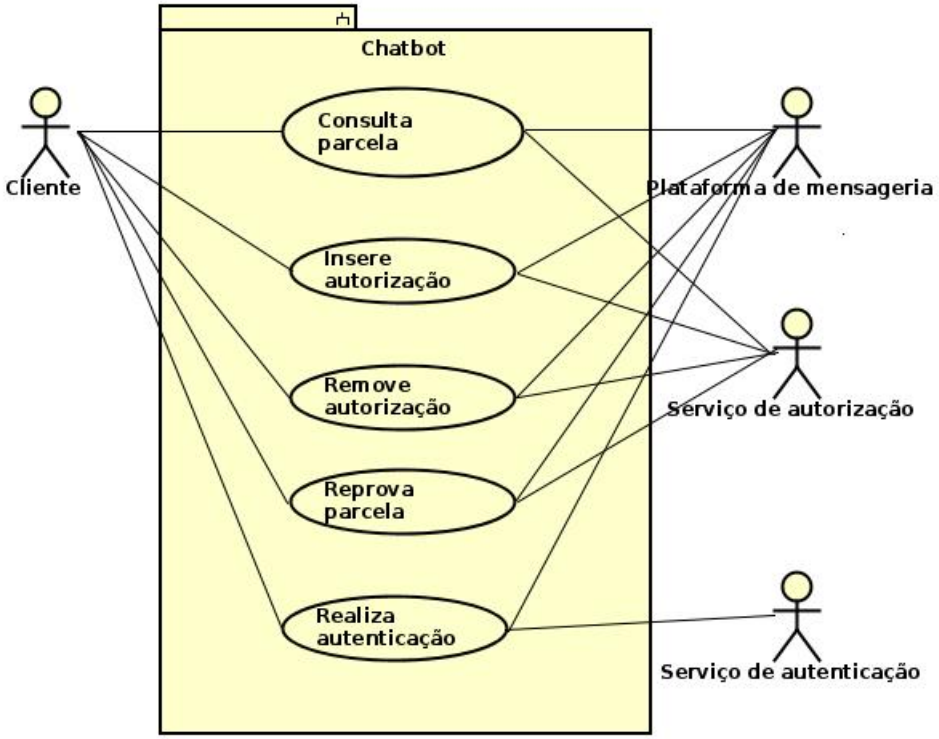
A arquitetura aplicada na API de autorização de parcelas para pagamento e no *chatbot* foi a arquitetura em camadas. Com isso, obtém-se a separação de responsabilidades em cada camada facilitando a testabilidade, e o desenvolvimento da solução.

Por outro lado, na API de autenticação de usuários foi aplicada a arquitetura limpa que consiste no isolamento da camada de domínio dentro da aplicação. Também aplica separação em camadas, porém com esse tipo de arquitetura o domínio da aplicação se torna independente de tecnologias usadas na solução. Isso favorece a reutilização do dominio com diversas outras tecnologias e portanto facilita a evolução tecnológica do projeto a longo prazo.

4.12 CASOS DE USO DE SISTEMA

A figura 6 ilustra os atores envolvidos no processo de gerenciamento de parcelas de pagamento pelo *chatbot*. De forma geral, qualquer ação do cliente envolve o ator plataforma de mensageria pois qualquer informação inserida pelo cliente no *chatbot* será processada pela plataforma. Por outro lado, somente as ações de consulta, inserção de autorização, remoção de autorização e reprovação de parcela envolverão o serviço de autorização, pois será ele que permitirá realizar consulta de parcelas para

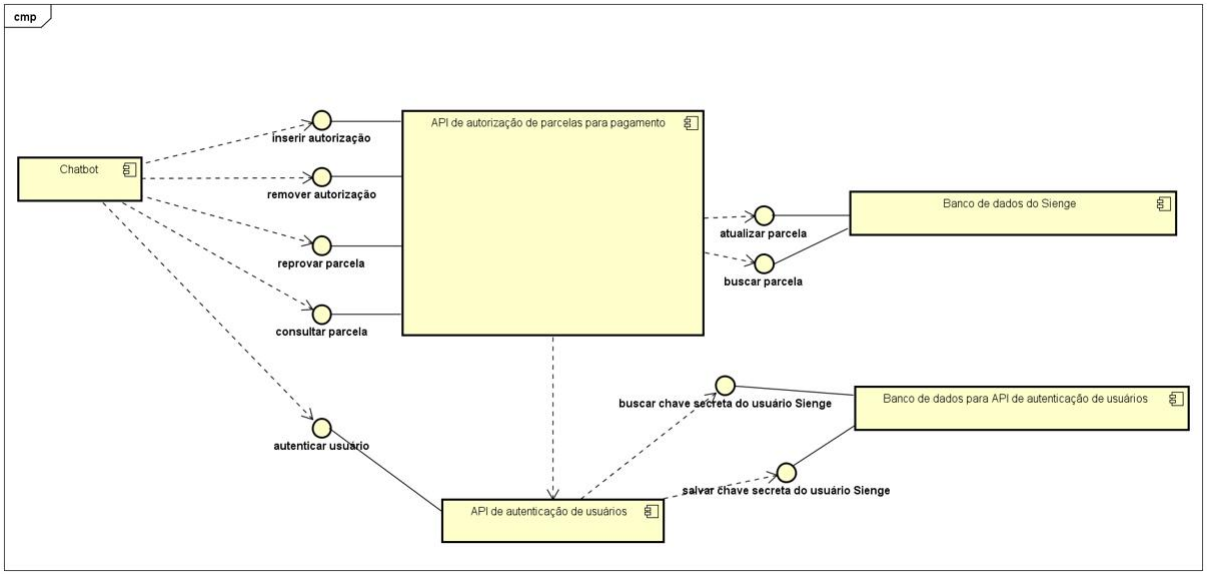
Figura 6 – Casos de uso do projeto



Fonte: Autor.

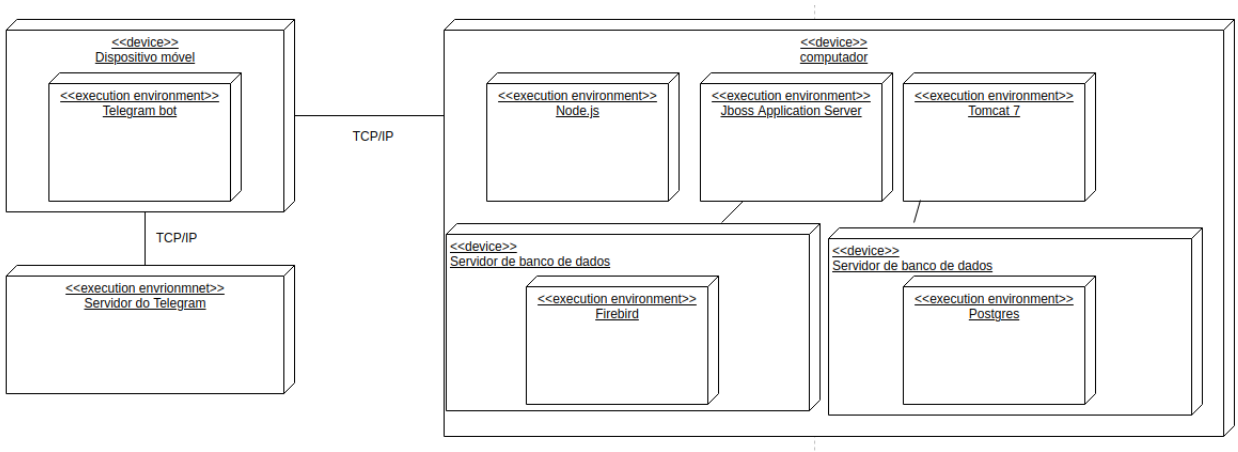
pagamento no Sienge, inserir e remover autorização e reprovar parcela. Esse serviço está localizado dentro do monolito do Sienge. Finalmente, o cliente poderá realizar a autenticação pelo *chatbot* utilizando um serviço de autenticação que será um microsserviço. Os digramas das figuras 7 e 8 explicam com mais detalhes a arquitetura e a representação física da solução.

Figura 7 – Diagrama de componentes



Fonte: Autor.

Figura 8 – Diagrama físico

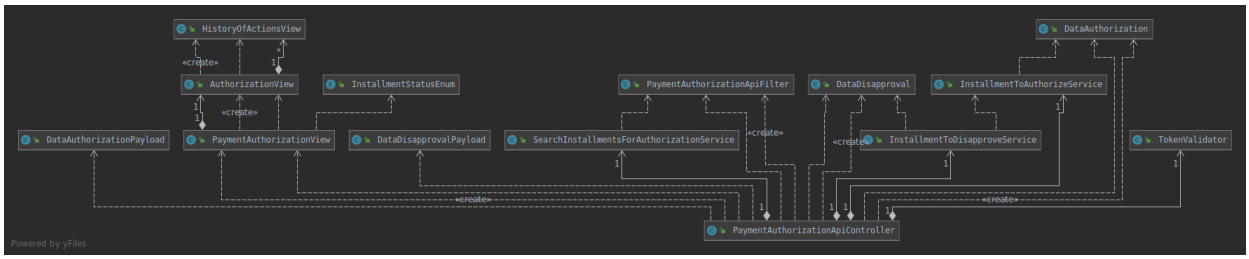


Fonte: Autor.

4.13 API DE AUTORIZAÇÃO DE PARCELAS

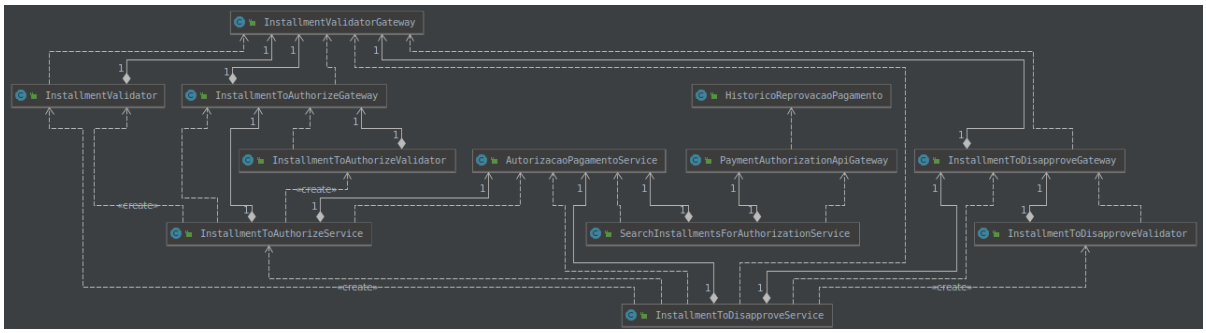
A modelagem das classes envolvidas na reutilização dos serviços de autorização de parcelas para pagamento podem ser visualizadas no diagrama de classes abaixo (ver figuras 9 e 10).

Figura 9 – Diagrama de classes da API de autorização de pagamentos com origem na classe PaymentAuthorizationController



Fonte: Autor.

Figura 10 – Diagrama de classes da API de autorização de pagamentos a partir da camada de serviço

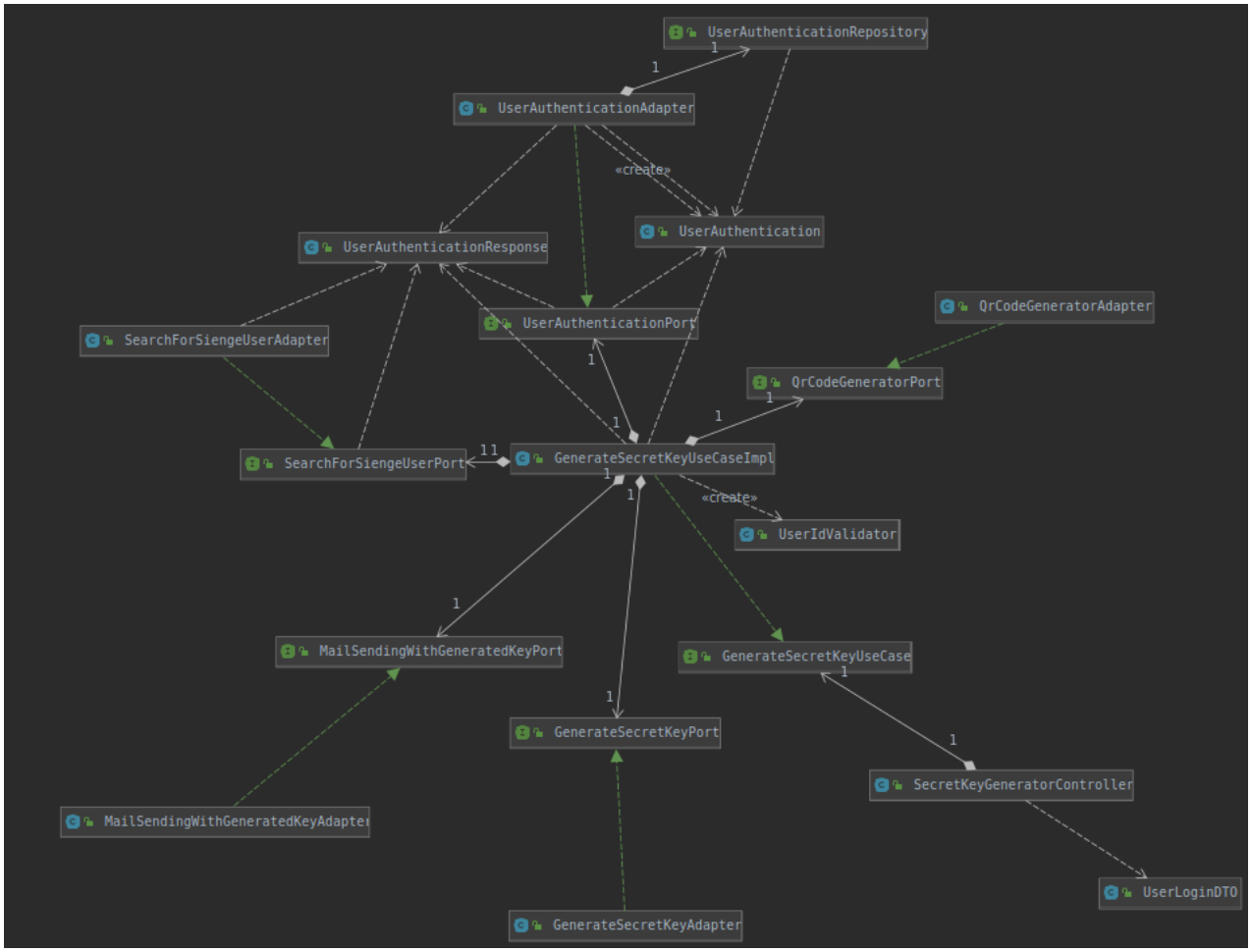


Fonte: Autor.

4.14 API DE AUTENTICAÇÃO DE USUÁRIOS

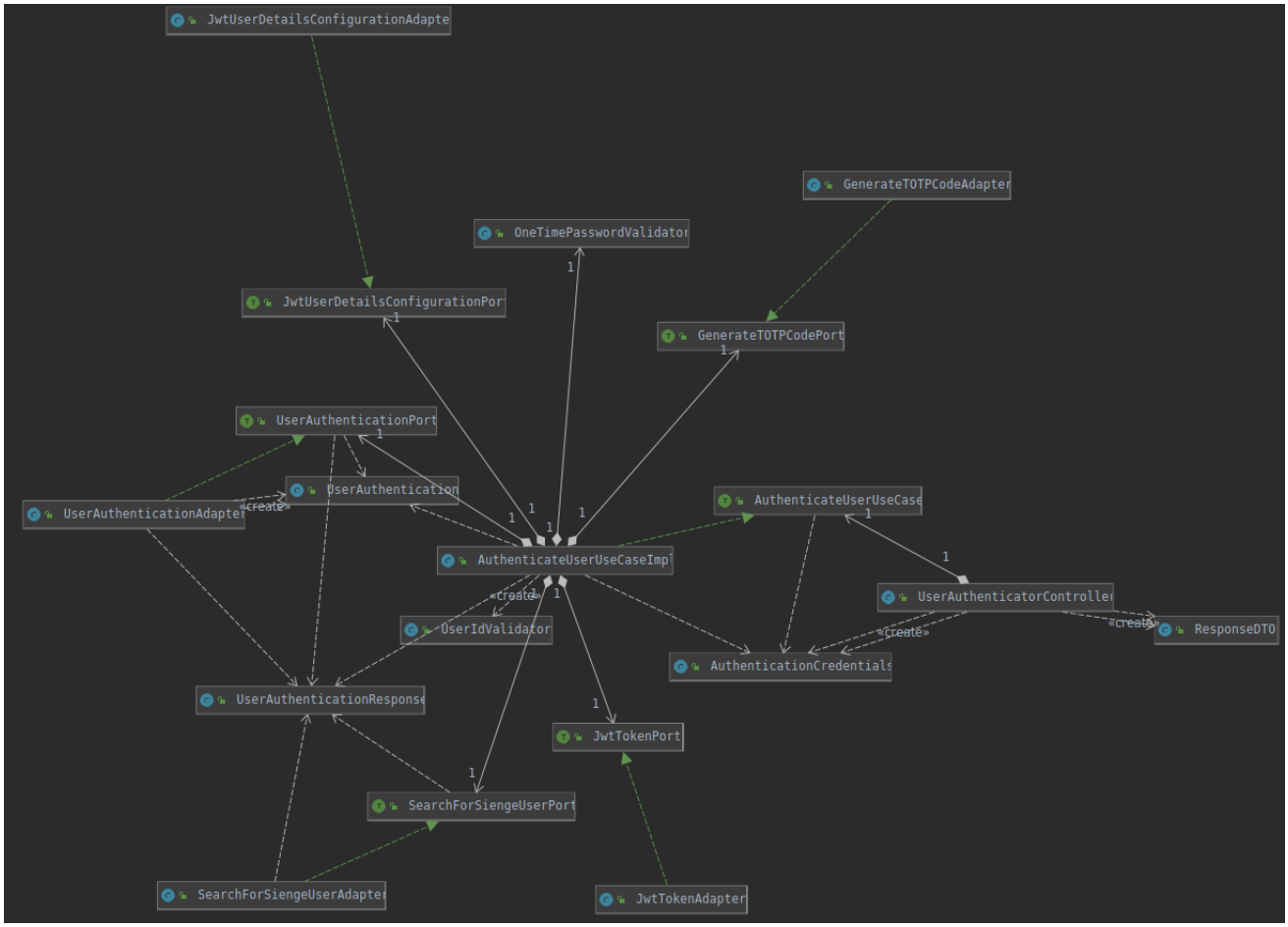
A modelagem das classes envolvidas na implementação da API de autenticação de usuários podem ser visualizados no diagrama de classes abaixo (ver figuras 11 a 13)

Figura 11 – Diagrama de classes da API de autenticação de usuários com origem em SecretKeyGeneratorController



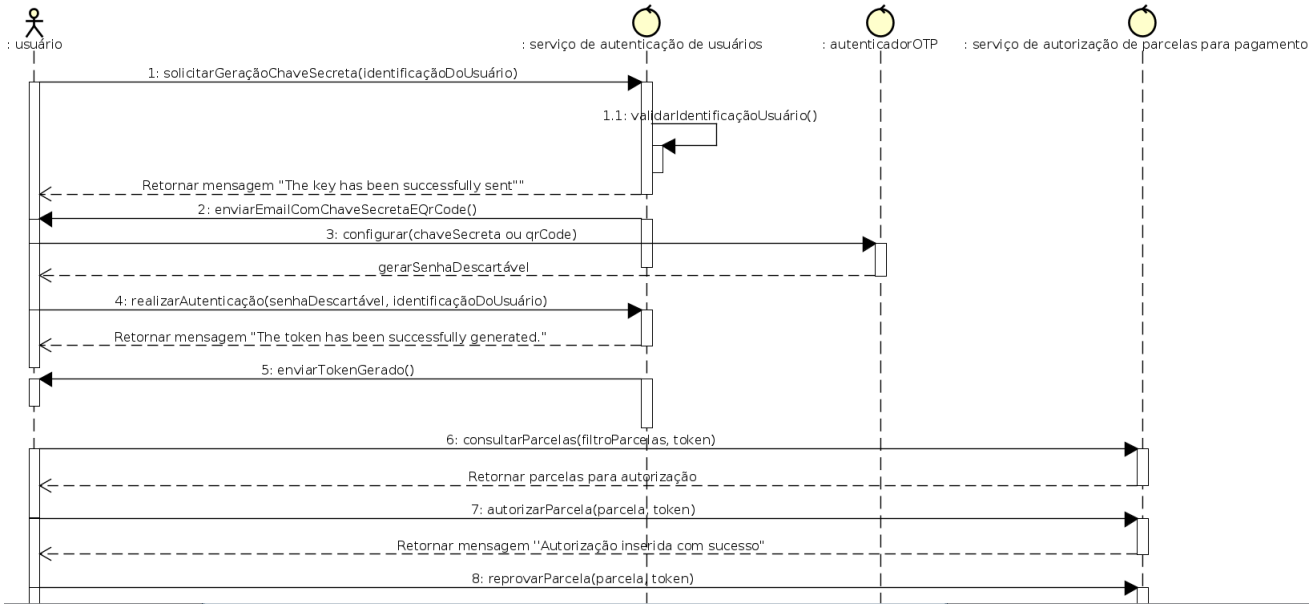
Fonte: Autor.

Figura 12 – Diagrama de classes da API de autenticação de usuários com origem em UserAuthenticatorController



Fonte: Autor.

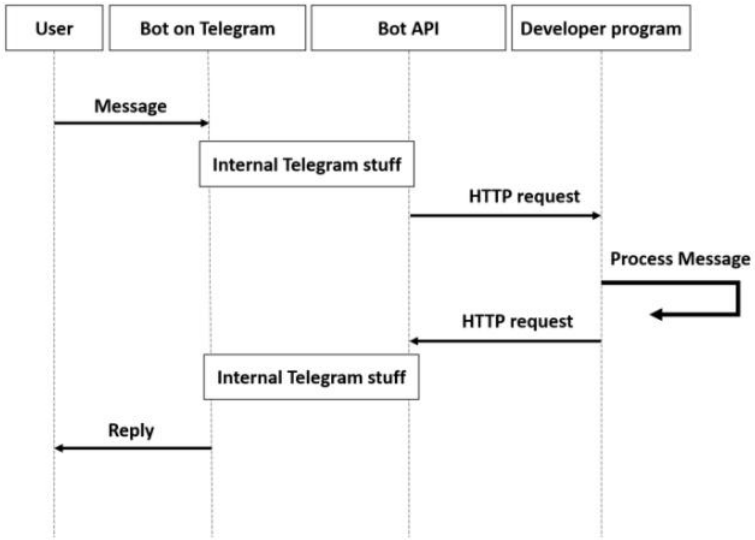
Figura 13 – Diagrama de sequência do processo de autorização de parcelas para pagamento com autenticação



Fonte: Autor.

4.15 FLUXO DE INTEGRAÇÃO ENTRE USUÁRIO E BOTS DO TELEGRAM

Figura 14 – Diagrama de sequência do processo de conversação entre o usuário e o chatbot pela plataforma do Telegram



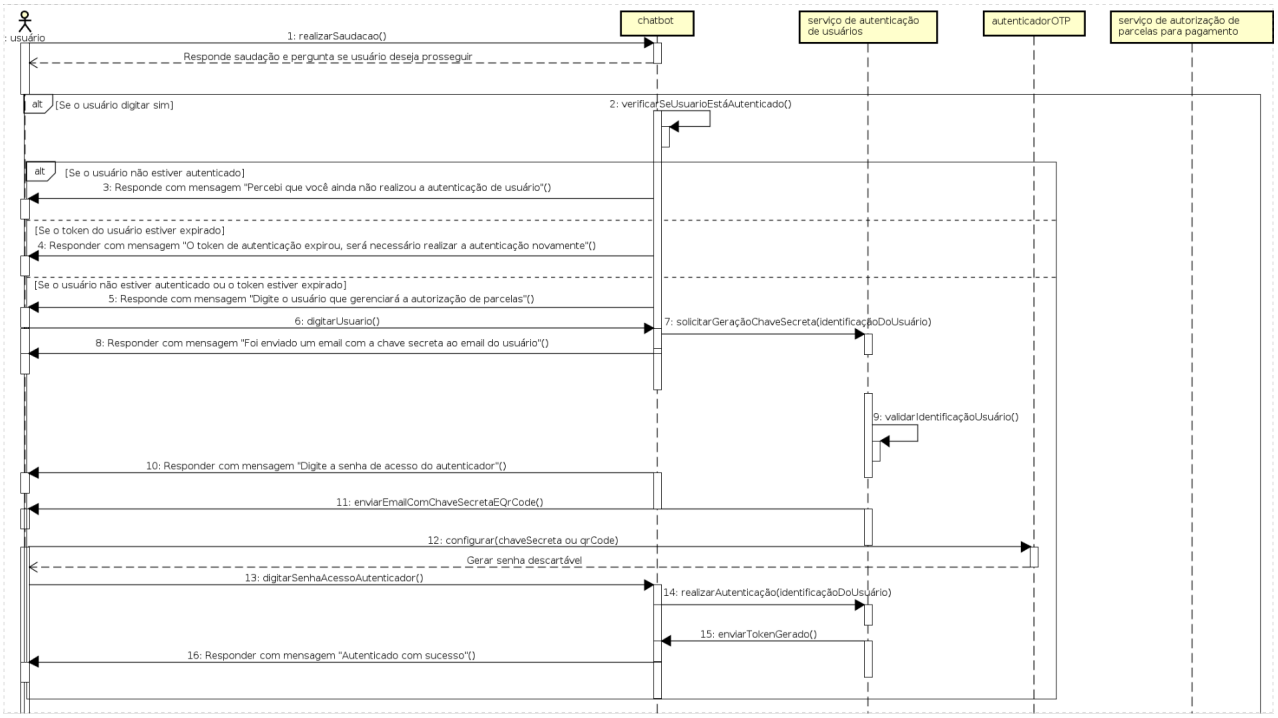
Fonte: https://www.researchgate.net/figure/The-sequence-of-communication-between-users-and-Telegram-bots_fig11_333768779.

4.16 MODELAGEM DO FLUXO DE COMUNICAÇÃO ENTRE O CHATBOT E AS APIS

Para permitir a realização do processo de autenticação de usuário e autorização de parcelas de forma amigável e portátil ao cliente é necessária a utilização de um *chatbot* como um orquestrador de todas as chamadas de APIs que dizem respeito a autorização, autenticação e que permita a interação com o usuário por meio de alguma interface (ver figuras 15 e 16).

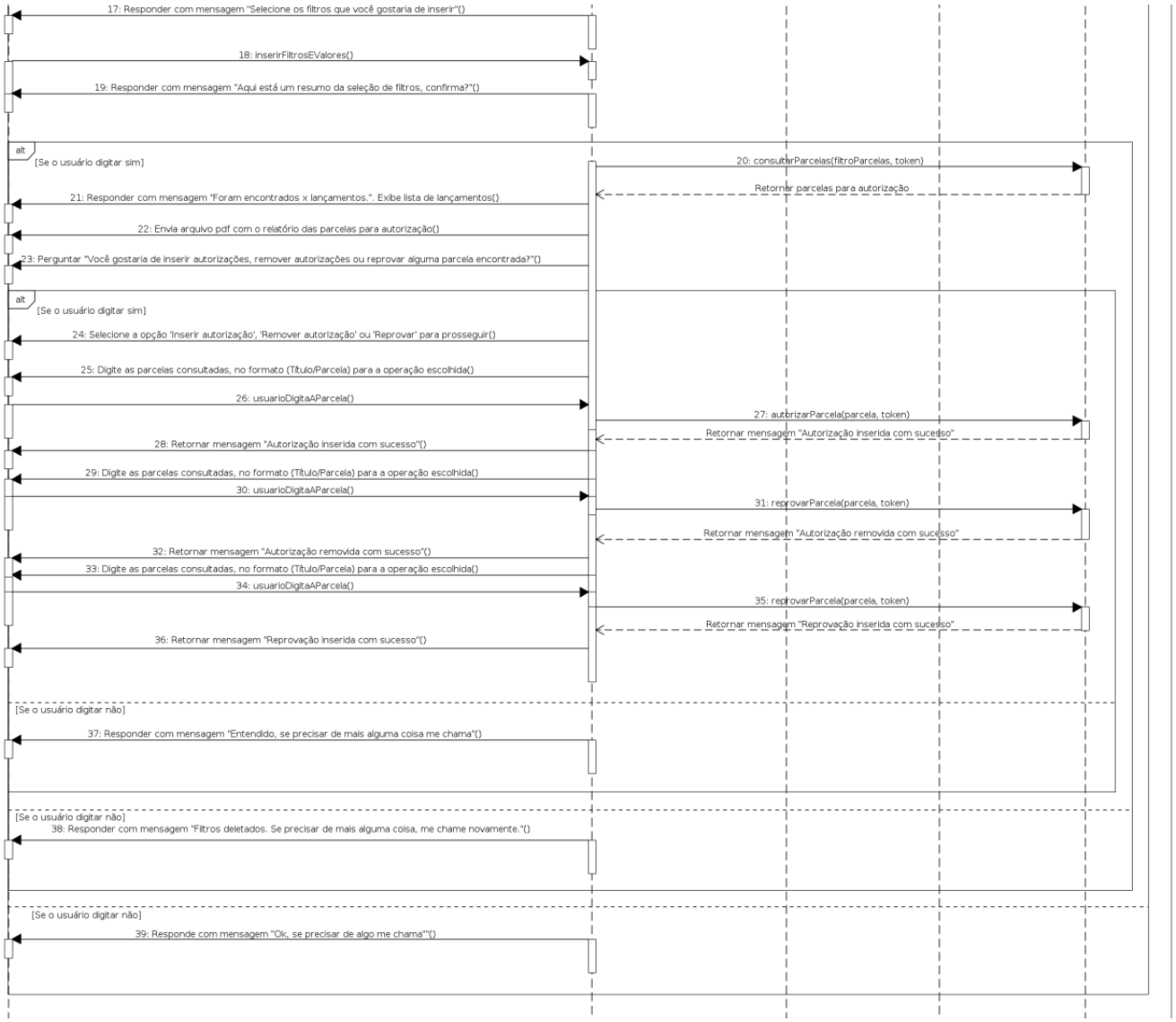
Assim o usuário não precisará realizar requisições diretamente a alguma API, mas fará isso indiretamente por meio da interação com o *chatbot*.

Figura 15 – Diagrama de sequência do processo de autenticação por meio do *chatbot*



Fonte: Autor.

Figura 16 – Diagrama de sequência do processo de autorização por meio do *chatbot* após realização da autenticação



Fonte: Autor.

5 IMPLEMENTAÇÃO DO CHATBOT

Nesse capítulo serão descritos os procedimentos utilizados no projeto do chatbot para autorização de parcelas para pagamento, dentre os quais podemos citar: desenvolvimento da API de autorização de parcelas para pagamento, desenvolvimento de API de autenticação de usuários e finalmente o desenvolvimento e integração do chatbot as APIs.

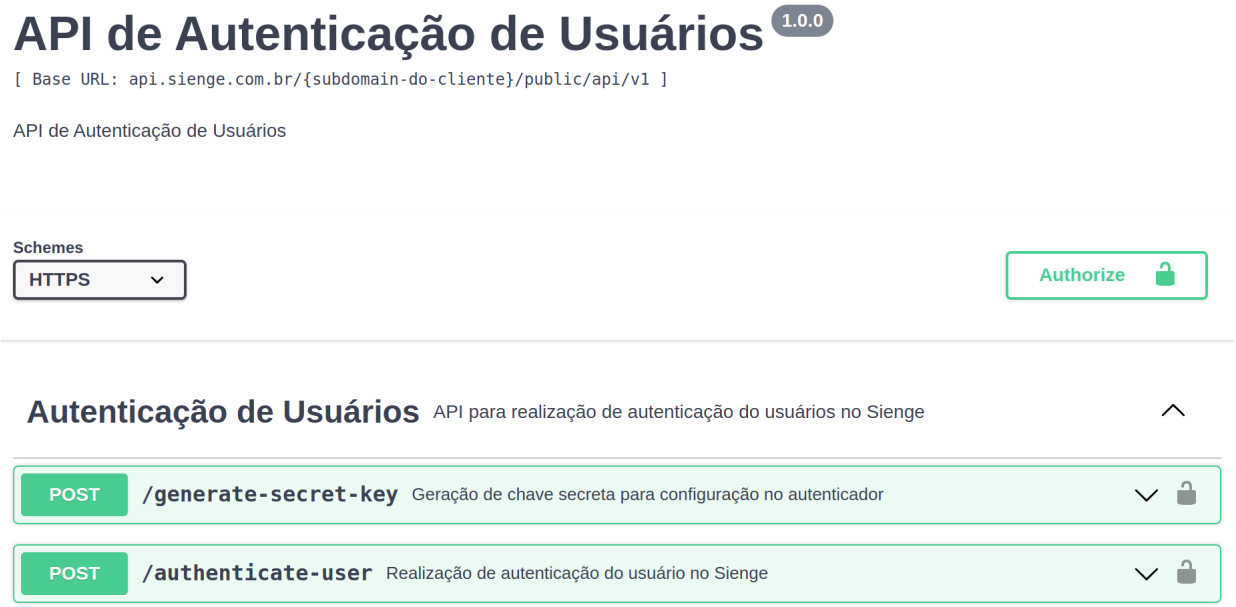
Também foi utilizada a IDE do IntelliJ nesse trabalho com o objetivo de gerar os diagramas de classes das principais classes envolvidas na solução de autenticação e autorização.

5.1 API DE AUTENTICAÇÃO

A API de autorização de parcelas para pagamento depende de um parâmetro de cabeçalho nas requisições para que elas sejam realizadas com êxito. Esse parâmetro é o token resultante do processo de autenticação realizado pela API de autenticação. Essa API tem por objetivo autenticar o usuário do Sienge que realizará a consulta, autorização, remoção da autorização ou reprovação da parcela. Essa API possui 2 terminais de conexão com o cliente (ver figuras 17 a 21):

- /v1/generate-secret-key.
- /v1/authenticate-user.

Figura 17 – Documentação da API de autenticação de usuários

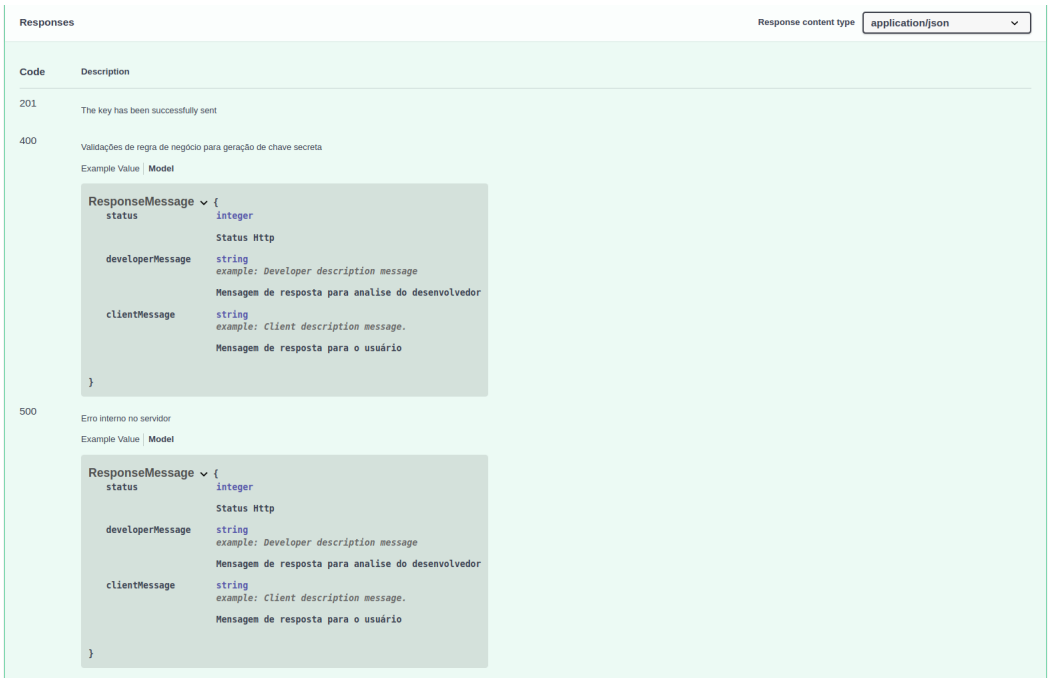


Fonte: Autor.

Figura 18 – Documentação do terminal de conexão /generate-secret-key

Fonte: Autor.

Figura 19 – Documentação do terminal de conexão /authenticate-user

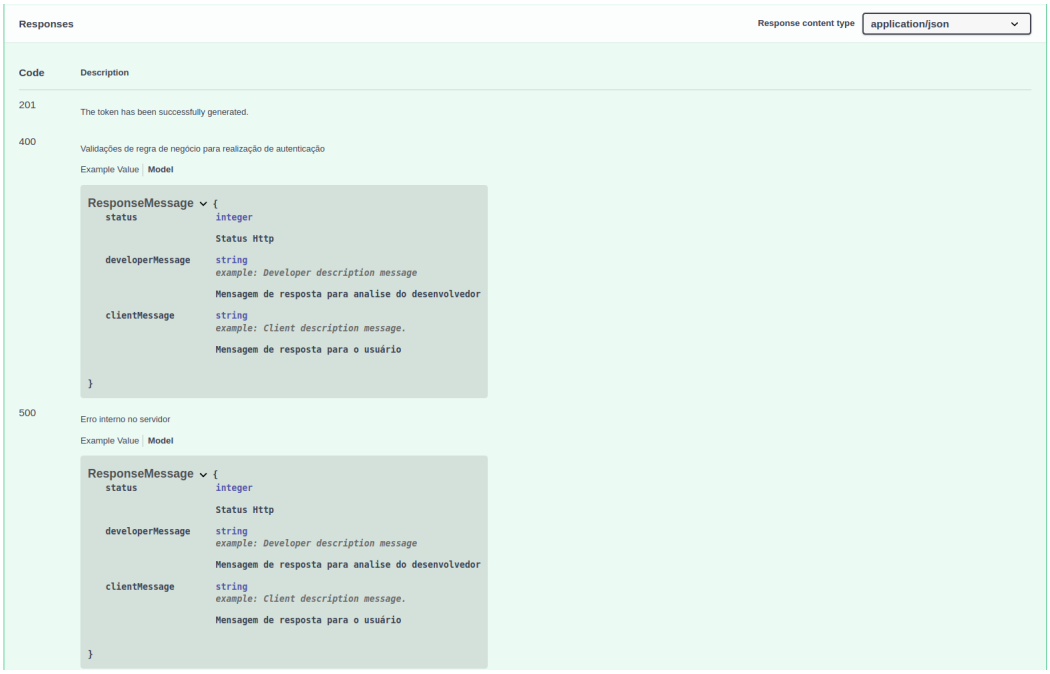


Fonte: Autor.

Figura 20 – Documentação do terminal de conexão /authenticate-user

Fonte: Autor.

Figura 21 – Documentação do terminal de conexão /authenticate-user



Fonte: Autor.

O processo de autenticação de usuário funciona da seguinte forma (ver figura

14): inicialmente um usuário do Sienge realizará uma requisição ao terminal de conexão /v1/generate-secret-key do tipo POST, passando como carga o identificador do usuário no Sienge. Se o usuário existir, o serviço web responderá "The key has been successfully sent". Isso significa que a chave secreta para configuração da geração da senha descartável foi enviada com sucesso ao email do usuário. Ao abrir o email, haverá duas formas de se realizar a configuração: ou por meio da câmera, realizando a leitura do QR Code dentro do autenticador em algum dispositivo, ou inserindo a chave secreta manualmente no autenticador para concluir a configuração. Ao fim desse processo será gerada periodicamente uma senha que expirará depois de um certo tempo. Com a disponibilidade dessa senha e o identificador do usuário do Sienge é possível realizar a autenticação. Isso é feito por meio de uma requisição ao terminal de conexão /v1/authenticate-user. Caso o usuário e a senha estejam corretos, a API retornará "The token has been successfully generated."indicando que o token foi gerado. Dessa forma, qualquer requisição a API de autorização de pagamentos é possível desde que o token seja enviado no cabeçalho da requisição dessa API.

5.2 API DE AUTORIZAÇÃO DE PARCELAS PARA PAGAMENTO

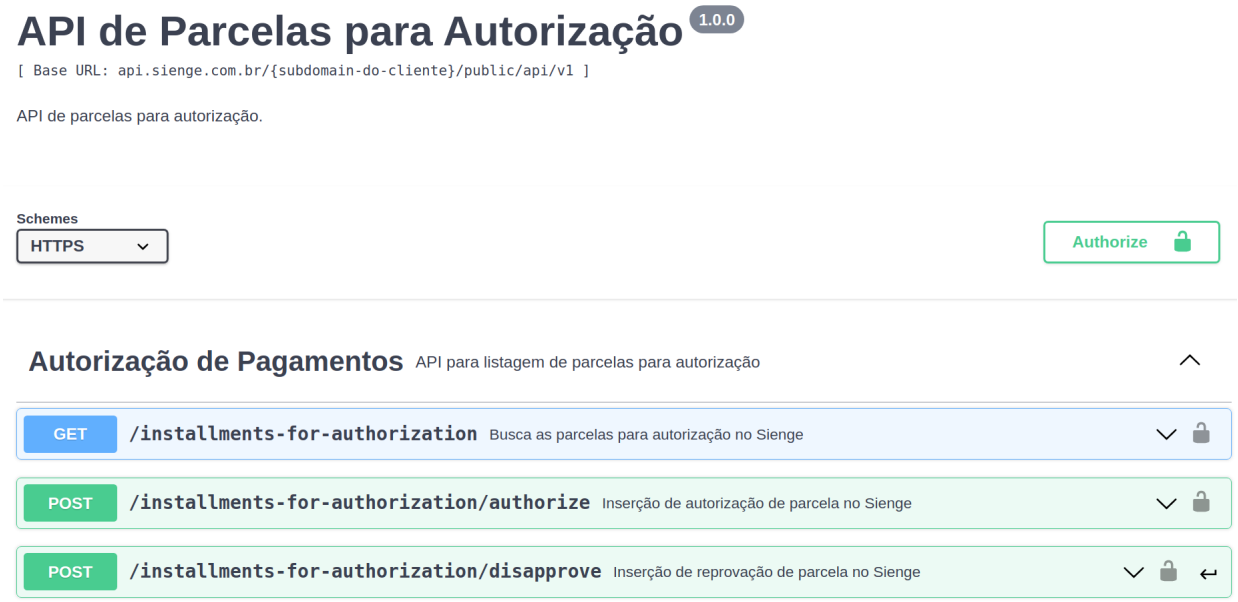
O processo de consulta, autorização e reprovação de parcelas é feito por uma tela dentro do Sienge. Entretanto, os serviços utilizados para consulta e autorização fazem parte de uma arquitetura monolítica na qual a tela e o servidor estão fortemente acoplados. Dessa forma, esses serviços não podem ser acessados por uma aplicação externa. Para que isso ocorra, foi necessário o projeto de uma API Web REST de autorização de parcelas para pagamento que pudesse reutilizar os serviços de consulta, autorização e desautorização.

Essa API possui 3 terminais de conexão com o cliente: (ver figura 23)

- /v1/installments-for-authorization.
- /v1/installments-for-authorization/authorize.
- /v1/installments-for-authorization/disapprove.

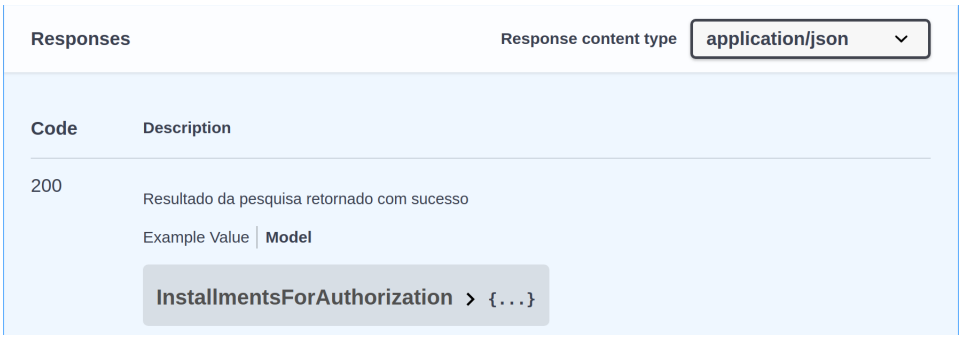
Esses terminais de conexão utilizam uma URL base composta da seguinte forma: { URL do ERP}/ {subdomínio do cliente}/public/api/v1. Abaixo é exibida a documentação completa referente a cada terminal de conexão, com explicação de cada parâmetro e resposta esperada pela API (ver figuras 25 a 36).

Figura 22 – Documentação API de parcelas para autorização



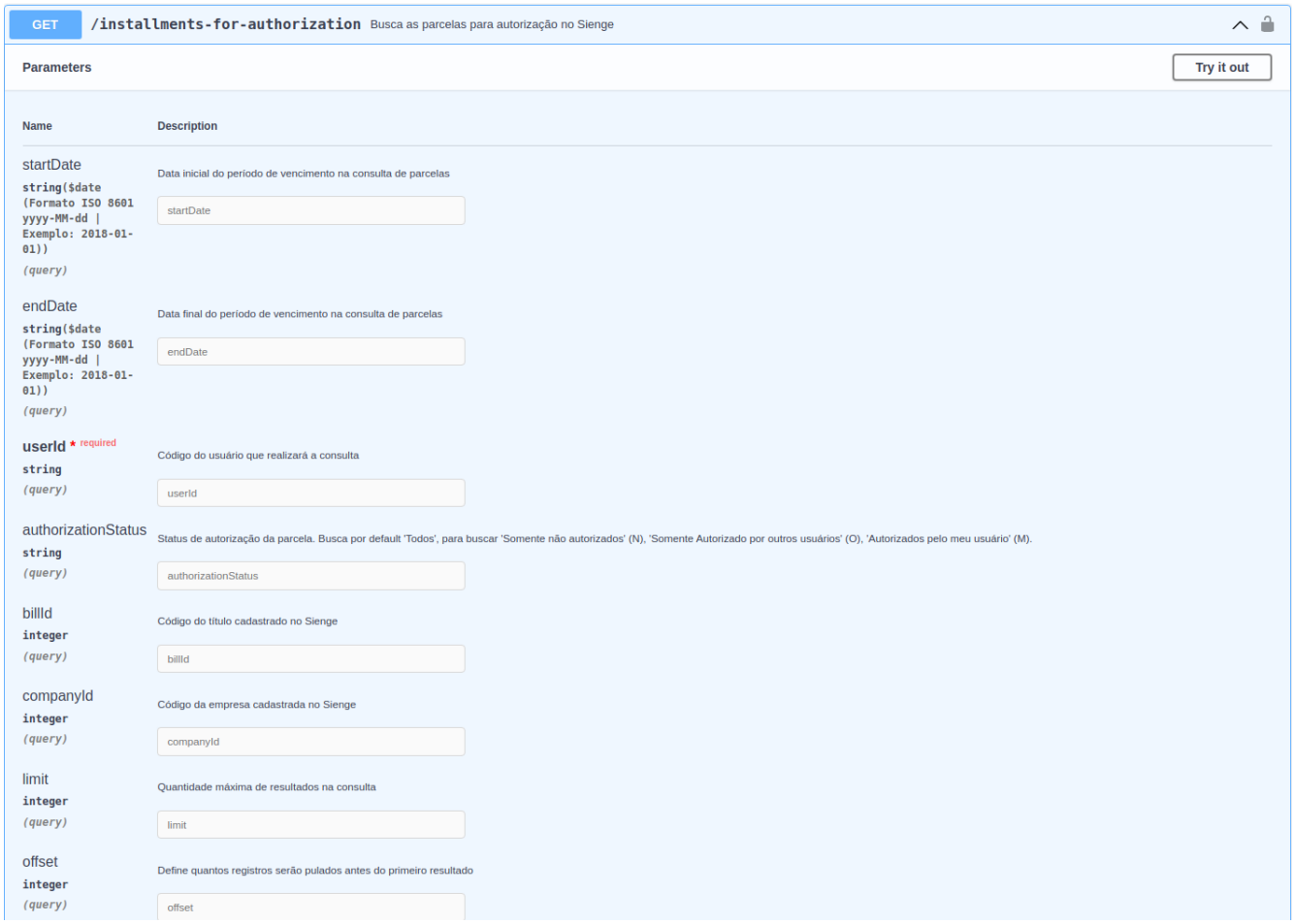
Fonte: Autor.

Figura 24 – Documentação do código de status 200 para o terminal de conexão /v1/installments-for-authorization



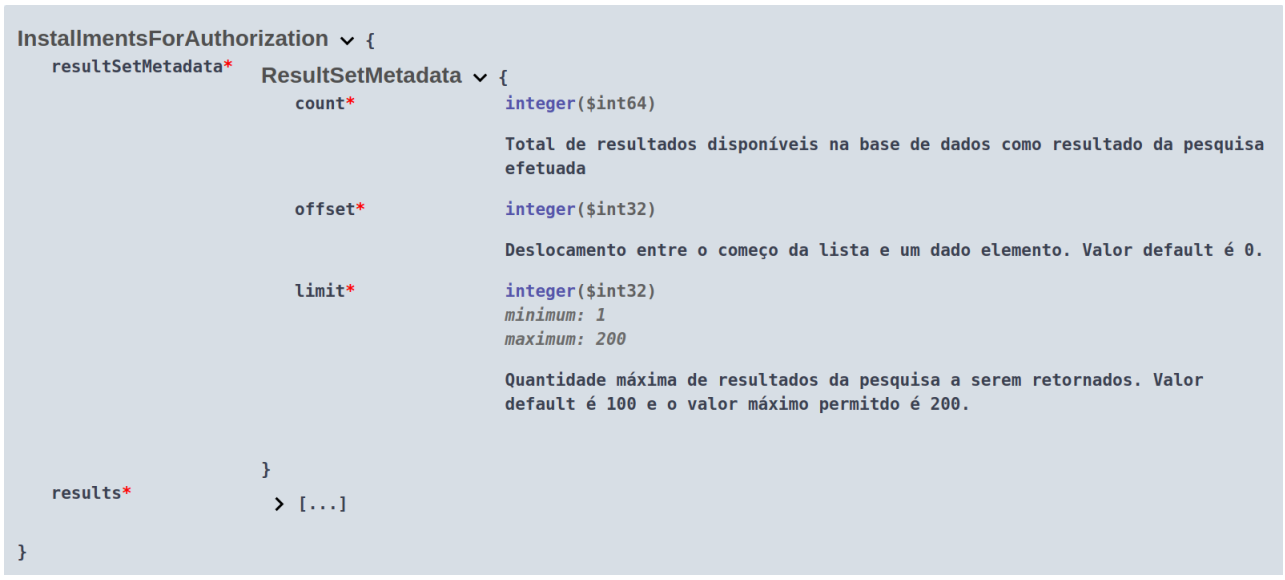
Fonte: Autor.

Figura 23 – Documentação dos parâmetros da requisição e parâmetros do cabeçalho para o terminal de conexão /installments-for-authorization



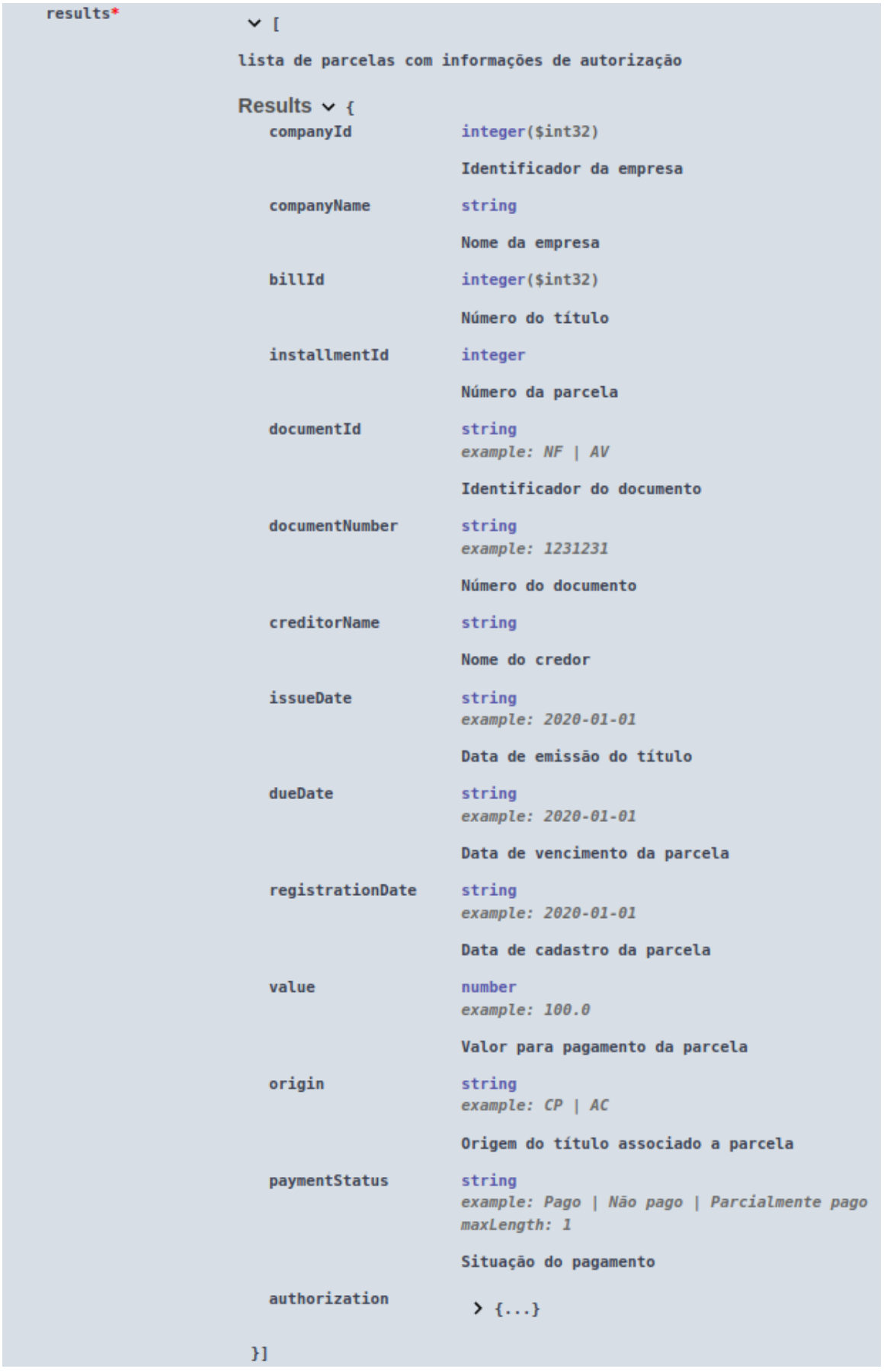
Fonte: Autor.

Figura 25 – Documentação da resposta para /v1/installments-for-authorization



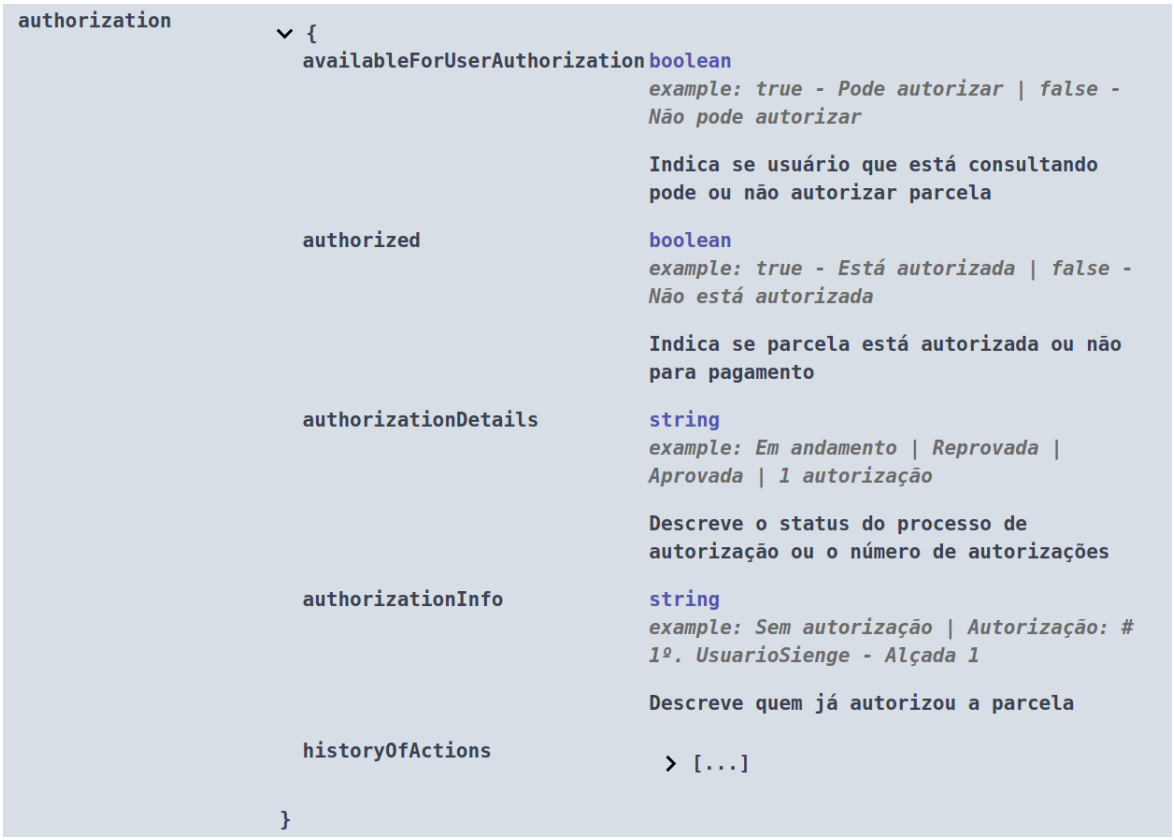
Fonte: Autor.

Figura 26 – Documentação da resposta para /v1/installments-for-authorization



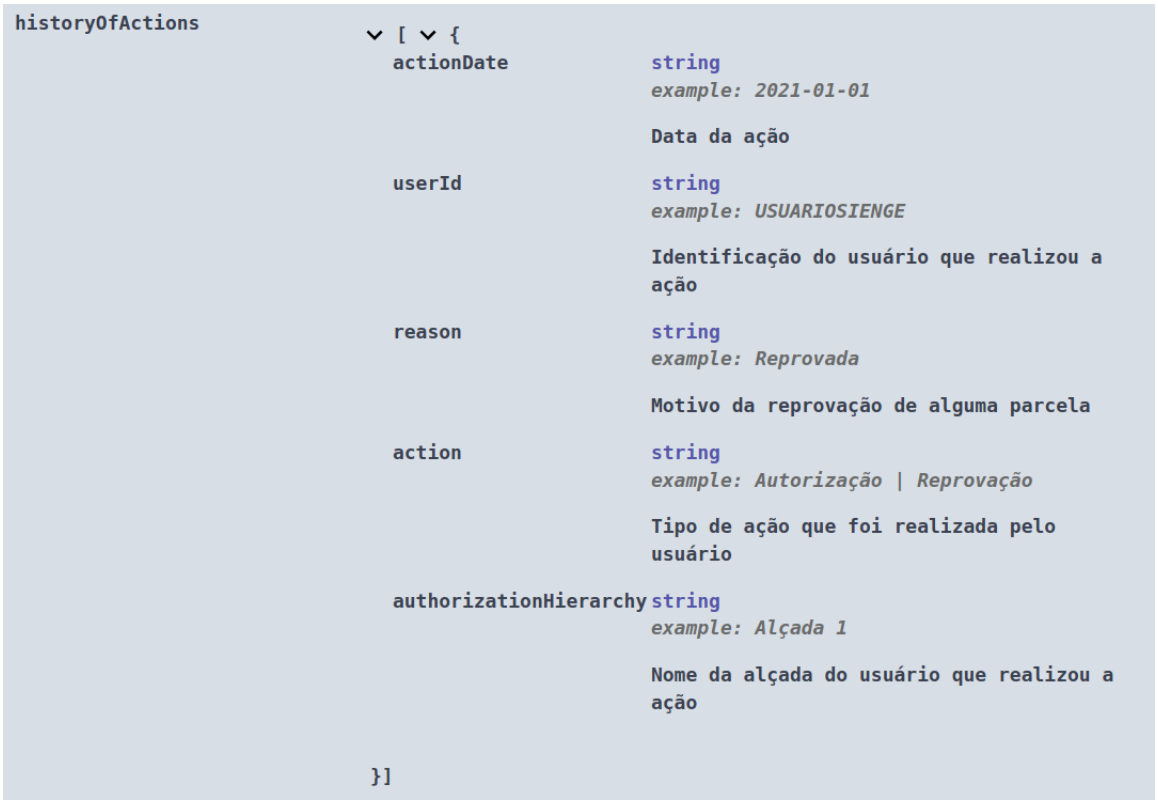
Fonte: Autor.

Figura 27 – Documentação da resposta para /v1/installments-for-authorization



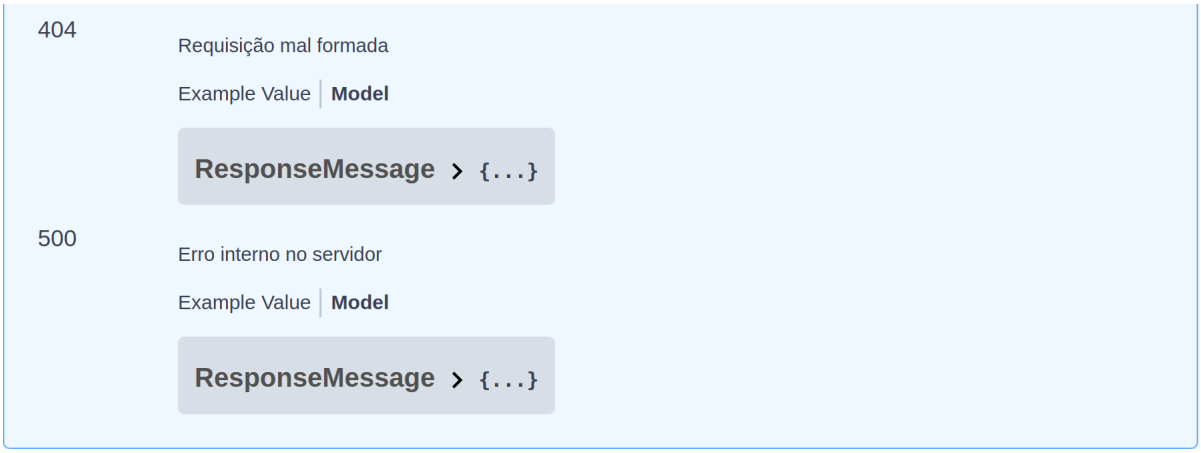
Fonte: Autor.

Figura 28 – Documentação da resposta para /v1/installments-for-authorization



Fonte: Autor.

Figura 29 – Documentação da resposta do terminal de conexão /v1/installments-for-authorization para o status 404 e 500

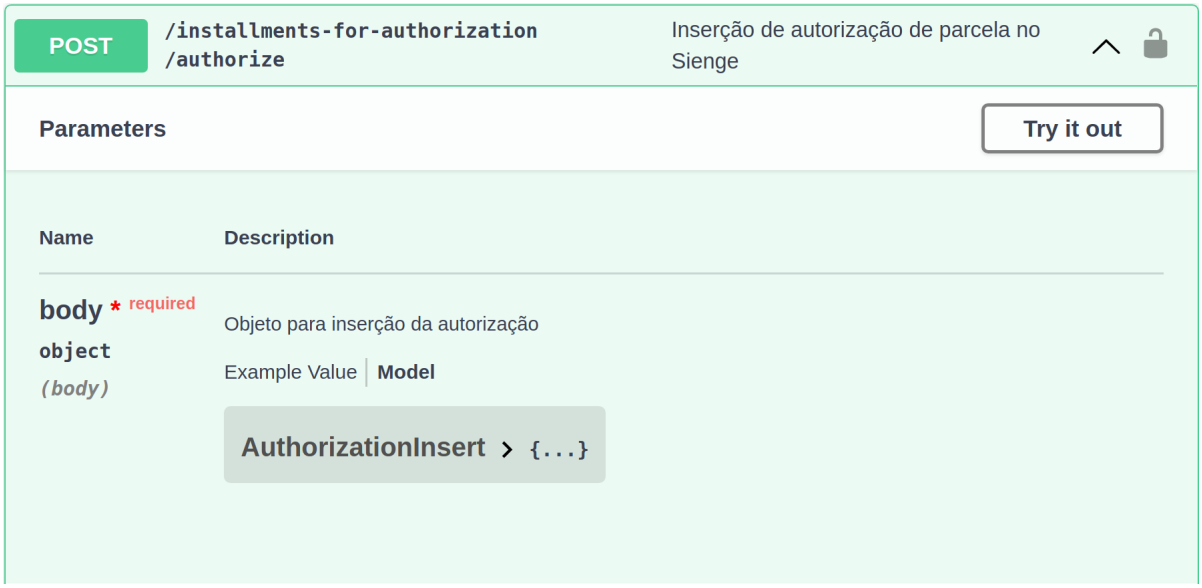


Fonte: Autor.

Figura 30 – Documentação da resposta do terminal de conexão /v1/installments-for-authorization para o status 404 e 500

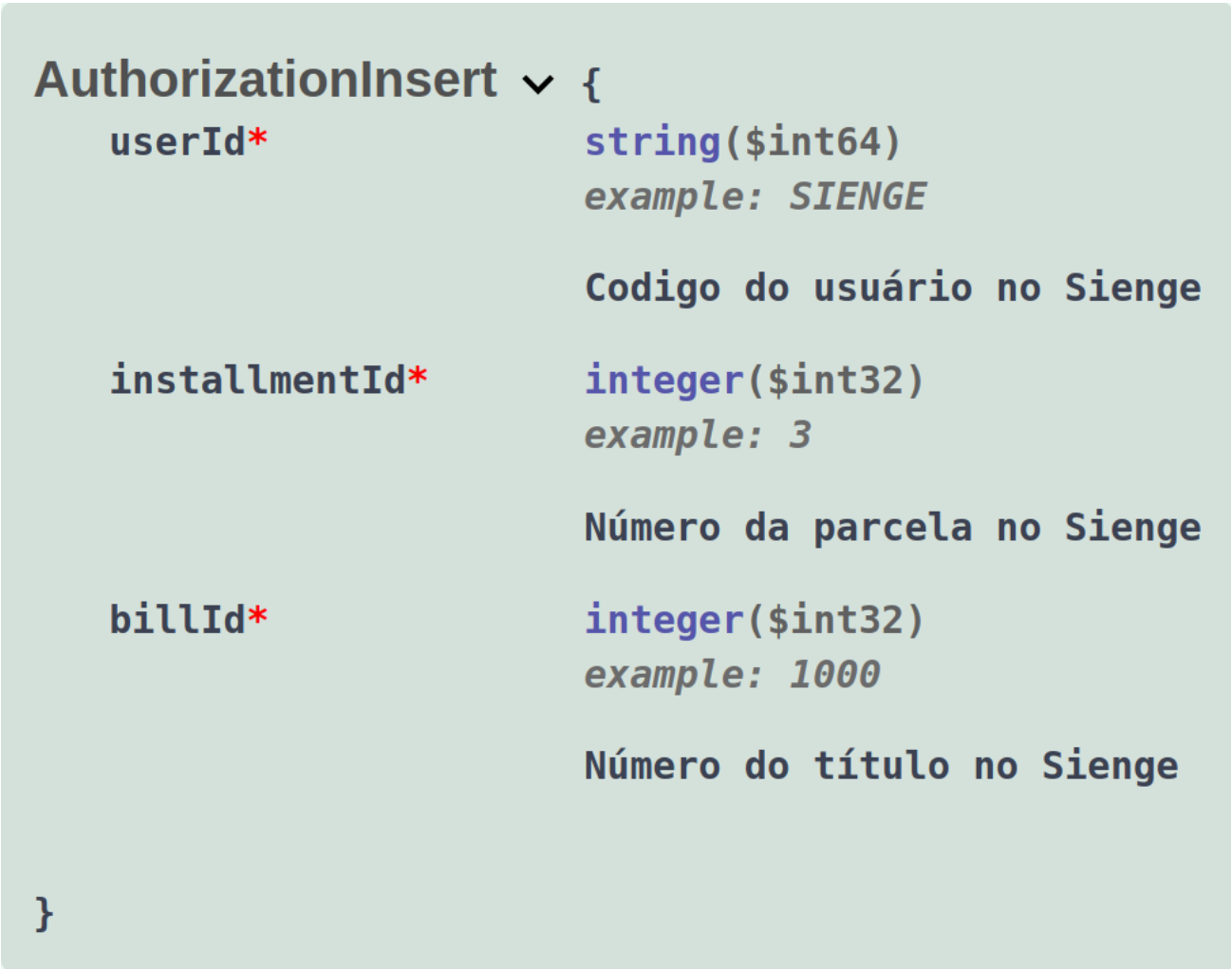
Fonte: Autor.

Figura 31 – Descrição do corpo da requisição para /v1/installments-for-authorization/authorize



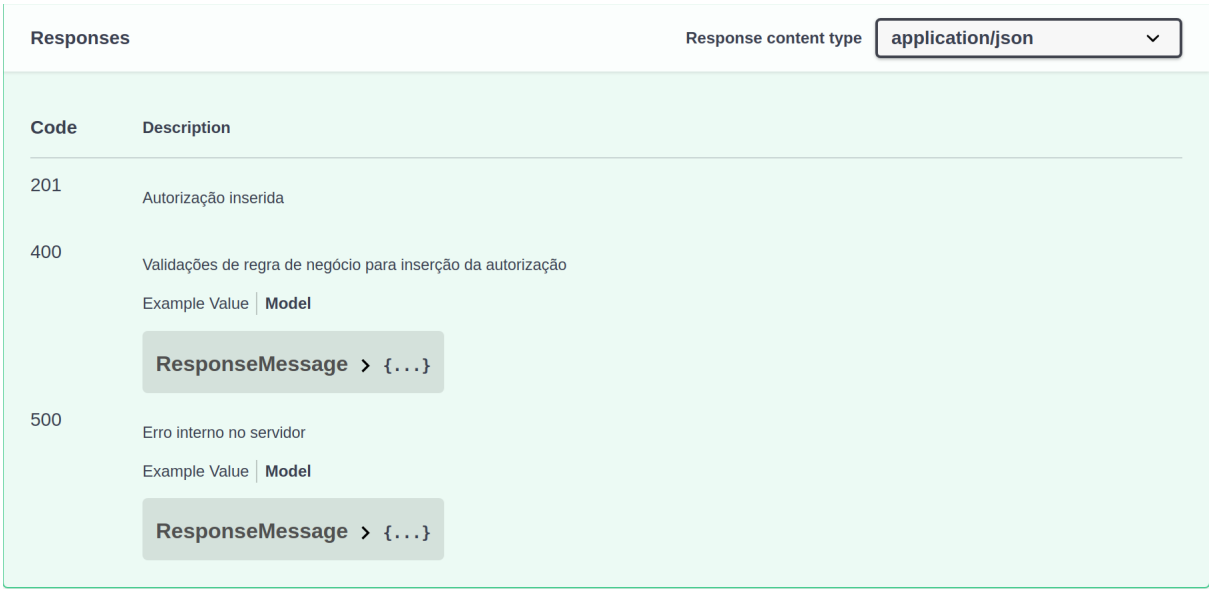
Fonte: Autor.

Figura 32 – Documentação do corpo da requisição para /v1/installments-for-authorization/authorize



Fonte: Autor.

Figura 33 – Documentação da resposta do terminal de conexão /v1/installments-for-authorization/authorize para os status 201, 400 e 500



Fonte: Autor.

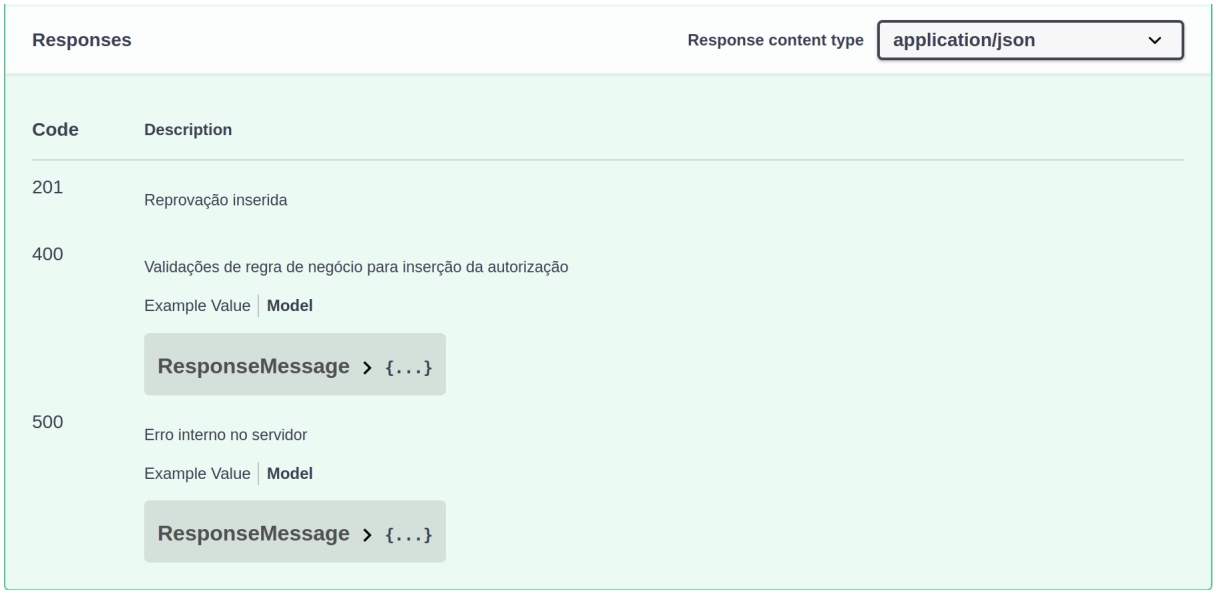
Figura 34 – Documentação do corpo da requisição para /v1/installments-for-authorization/disapprove

Fonte: Autor.

Figura 35 – Documentação do corpo da requisição para /v1/installments-for-authorization/disapprove

Fonte: Autor.

Figura 36 – Documentação da resposta do terminal de conexão /v1/installments-for-authorization/disapprove para os status 201, 400 e 500



Fonte: Autor.

5.3 MÉTRICAS DE CÓDIGO

Uma das métricas usadas para medir a complexidade da solução desenvolvida é o número de linhas de código. Para a API de autenticação de usuários, API de autorização de parcelas para pagamentos e chatbot foram escritos respectivamente, 1377, 3299, 2574 linhas incluindo testes unitários e documentação das APIs.

5.4 ARQUITETURA

As figuras abaixo ilustram como foi feita a separação de pacotes levando em conta a arquitetura por camadas e a arquitetura limpa. Na figura 37 é exibida a camada de apresentação representada pelos pacotes:

- br.com.softplan.unic.paymentauthorization.v1.controller;
- br.com.softplan.unic.paymentauthorization.dtos;

Já nas figuras 38 a 39 é exibida a camada de domínio da API de autorização de parcelas para pagamento desenvolvida nesse trabalho.

Com relação a API de autenticação de usuários os pacotes foram separados seguindo as camadas da arquitetura limpa (ver figura 40). Tudo que diz respeito a configurações do framework ficaram na camada mais externa chamada de 'Frameworks & Drivers' que corresponde aos pacotes:

- br.com.softplan.unic.userAuthentication.infraestructure.factory;

Já na camada de 'Interface Adapters' fica tudo aquilo que diz respeito a apresentação e acesso a outros recursos. Nessa camada ficam os pacotes:

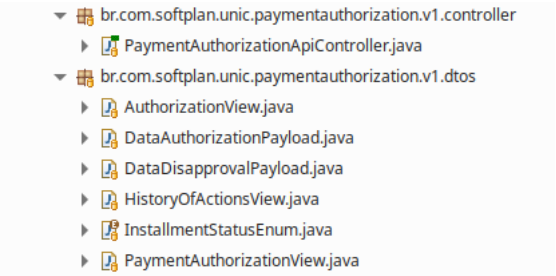
- br.com.softplan.unic.userAuthentication.infraestructure.persistence;
- br.com.softplan.unic.userAuthentication.infrastructure.adapter;
- br.com.softplan.unic.userAuthentication.infrastructure.web.api;

Finalmente na camada de domínio que envolve os casos de uso da aplicação e as entidades há respectivamente dois pacotes:

- br.com.softplan.unic.userAuthentication.domain.usecase;
- br.com.softplan.unic.userAuthentication.domain.model;

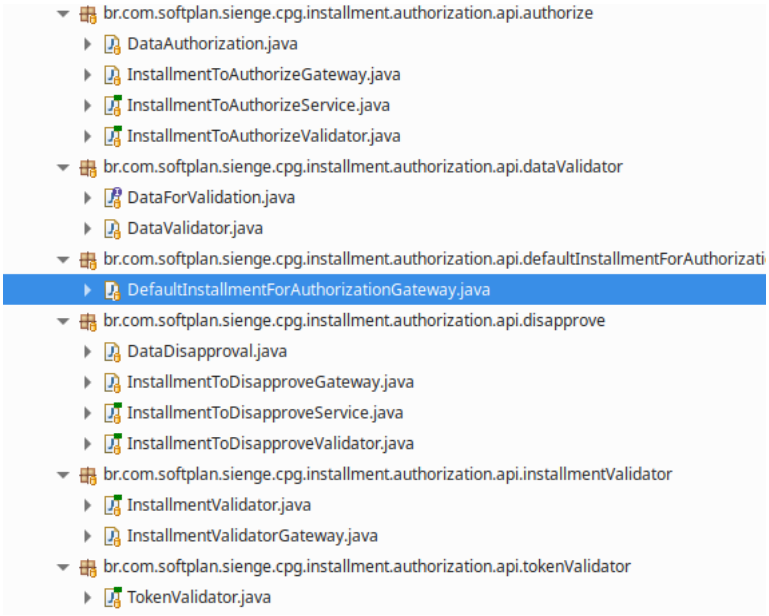
Na estrutura de pastas do chatbot não foi usado um padrão de arquitetura específico. Entretanto, foi aplicado o princípio da responsabilidade única separando responsabilidades entre as cenas do chatbot em diferentes pastas(ver figura 41). Além disso, o programa de inicialização da aplicação, as requisições para APIs externas e funções específicas de geração de tabela e conversão de formato de arquivo também ficaram separadas em pastas para garantir essa separação de responsabilidades.

Figura 37 – Camada de apresentação da API de autorização de parcelas para pagamento para autorização e desautorização de parcelas



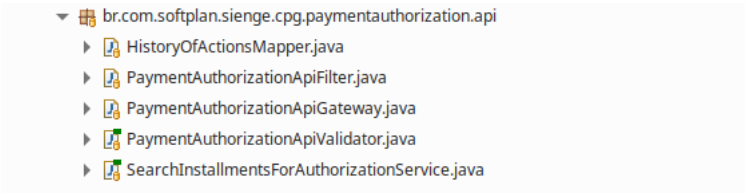
Fonte: Autor.

Figura 38 – Camada de domínio da API de autorização de parcelas para pagamento



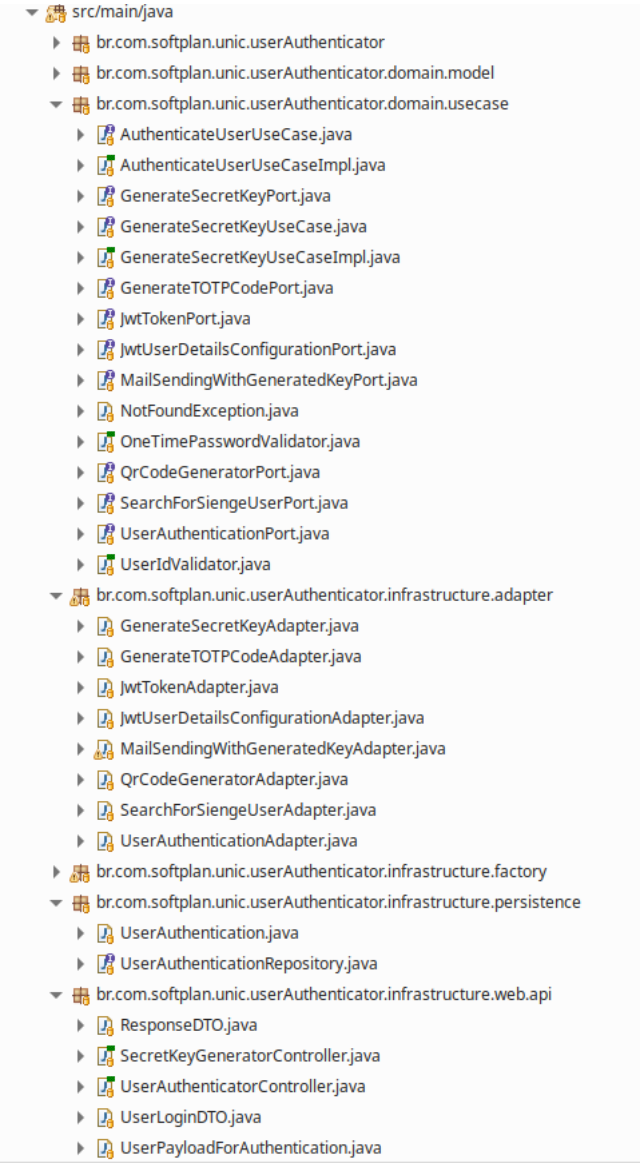
Fonte: Autor.

Figura 39 – Camada de domínio da API de autorização de parcelas para pagamento para consulta de parcelas



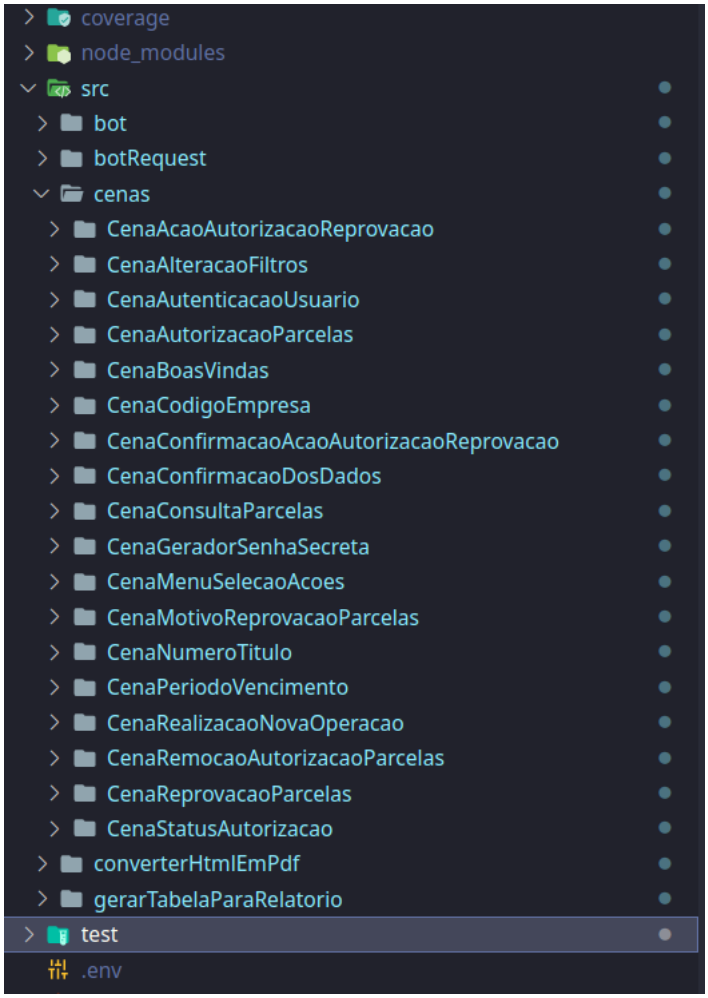
Fonte: Autor.

Figura 40 – Estrutura de pastas da API de autenticação de usuários



Fonte: Autor.

Figura 41 – Estrutura de pastas do *chatbot*



Fonte: Autor.

5.5 AUTENTICAÇÃO DE DOIS FATORES

Ela é baseada no identificador do usuário do Sienge que o usuário sabe e baseada na senha descartável gerada pelo autenticador que é algo que o usuário tem. Essa autenticação se baseou no algoritmo TOTP para gerar a senha descartável e no JWT para geração do token. Foi usado o algoritmo HS512 para assinatura desse token e usada uma chave secreta codificada em SHA-512 para criptografar e descriptografar o token. Somente quem tiver acesso a essa chave secreta que poderá descriptografar ou gerar um novo token o que garante a segurança do uso do *chatbot* e da API de autorização de parcelas para pagamento.

5.6 FLUXO DE CONVERSAS

O desenvolvimento do *chatbot* foi realizado utilizando o BotAPI da plataforma de mensageria do Telegram. O desenvolvimento das cenas entre o *chatbot* e o usuário foi feito utilizando a biblioteca Telegraf dentro do framework Node.js. A técnica utilizada para processar a mensagem do usuário foi expressões regulares, visto que a interação entre o *chatbot* e o usuário é relativamente simples e as mensagens de texto muitas vezes se encaixam em padrões como: datas, lançamentos (título/parcela), número do título e código de empresa. Em casos como no filtro status de autorização em que a resposta deve ser mais específica, o botão dentro do teclado atendeu bem essa necessidade, pois evita que o usuário digite o texto inteiro e possa cometer erros na digitação. Abaixo é exibida uma típica conversa entre o *chatbot* e um usuário que deseja realizar operações de consulta, autorização, remoção de autorização ou reprovação (ver figuras 42 a 55).

Na figura 42 o chatbot realiza uma saudação após o primeiro envio de mensagem do usuário. No início qualquer mensagem do usuário será aceita para começar a conversa. Para esse trecho foi criada uma cena na qual o chatbot aguarda uma resposta positiva ou negativa. Essas respostas são processadas pelas expressões regulares /(Sim|^S$|Yes|^Y$)/i e /(N[ãa]o|^N$|No)/i. Caso a resposta seja ’não’, ’nao’, ’no’, ’N’, ’n’ ou ’Não’ o chatbot responde e a conversa volta pro início. Por outro lado, se a resposta for um texto qualquer o chatbot responde avisando que não entendeu a mensagem escrita e o usuário continua na mesma cena. Esse comportamento é válido para todas as cenas da conversa. Caso o usuário escreva que ’sim’, ’Sim’, ’S’, ’s’, ’yes’, ’Y’ ou y a conversa continua e entra na cena de geração de senha secreta.

Nessa cena o usuário terá que digitar o identificador do usuário no Sienge para prosseguir. O texto digitado é processado pela expressão regular /([A-Z]+)/i a qual aceita pelo menos uma letra no intervalo de letras de A a Z minúsculas ou maiúsculas. Caso seja digitado um número o chatbot responderá que não entendeu e o usuário permanecerá na mesma cena. Caso digite o texto corretamente, o chatbot verificará

se esse usuário existe ou não no Sienge chamando a API de autenticação de usuários (URL base/v1/generate-secret-key) que internamente realizará uma requisição a API de usuários do Sienge (URL base/v1/users). Caso não exista, o chatbot retornará uma mensagem explicando que o usuário não foi encontrado e pedirá que o usuário tente novamente. Nesse caso o usuário permanecerá na cena até que o identificador do usuário do Sienge seja digitado corretamente. Quando o usuário for corretamente digitado, o terminal (URL base/v1/generate-secret-key) da API de autenticação de usuários salvará a chave secreta em uma base de dados Postgres, retornará status 201 e enviará um email ao usuário do Sienge com a chave secreta e o QR Code para configuração em algum autenticador. Isso só ocorrerá se for a primeira vez que o usuário do Sienge estiver utilizando o chatbot. Na próxima vez que esse usuário for realizar a autenticação, a chave secreta já estará salva no banco de dados e portanto, nenhum email será enviado ao usuário.

Após receber o email, o usuário deverá apontar a cãmera do autenticador OTP para o QR Code com o objetivo de configurar a geração da senha descartável ou copiar a chave secreta e realizar a configuração de forma manual no dispositivo móvel. Com isso será gerada uma senha de 6 dígitos a cada 30 segundos, essa senha deve ser inserida no chatbot para prosseguir. Os dígitos são validados pela expressão regular /(^\d{6}$)/. Caso ela não esteja no padrão de 6 dígitos, ou esteja incorreta o chatbot responderá informando se a senha está incorreta, ou se a mensagem não foi entendida. Se a senha estiver correta será gerado um token JWT pelo terminal (URL base/v1/authenticate-user) cujo emissor será o código do usuário do Sienge e cujo assunto será a senha descartável gerada pelo autenticador. Com esse token, o usuário é autenticado com sucesso e qualquer terminal da API de autorização de parcelas para pagamento pode ser chamado passando como parâmetro do cabeçalho o token gerado.

Após o processo de autenticação, entra-se na cena de escolha dos filtros para a consulta de parcelas para autorização. O chatbot pede para escolher os filtros que o usuário gostaria de inserir. Tudo aquilo que for digitado e não estiver dentro dos filtros não será entendido pelo chatbot e portanto, o usuário permanecerá na mesma cena. O texto é processado pela seguinte expressão regular (/Período de vencimento|Status de autorização|Número do título|Código da empresa/). Para evitar erros de digitação, o teclado exibe botões de atalho que facilitam a escolha do filtro de acordo com a regra da expressão regular. Após a escolha dos filtros, eles podem ser removidos clicando naquele que se deseja remover. Ao finalizar a escolha dos filtros, seleciona-se avançar e dessa forma o chatbot começa a percorrer filtro a filtro pedindo para digitar o valor de cada um. Cada filtro do chatbot corresponde a uma cena. Se em uma determinada cena o filtro estiver marcado, o usuário deve digitar o valor do filtro correspondente que será validado por uma expressão

regular. As cenas dos filtros: número do título, código da empresa, período de vencimento e status de autorização possuem as respectivas expressões regulares: /(^\d{1,9}$)/g, (^\d{1,9}$)/g, /(^\d{2}/\d{2}\/(\d{4}|\d{2}).+\d{2}\/{2}\/(\d{4}|\d{2})$)/g e /(Somente não autorizados|Somente autorizados por outros usuários|Autorizados pelo meu usuário|Todos)/g. As duas primeiras só vão corresponder ao texto quando este for um dígito de 0 a 9 digitado de 1 até 9 vezes. A terceira expressão só vai corresponder quando for inserido um período de vencimento do início ao fim da linha no formato dd/MM/yyyy ou dd/MM/yy. Na última expressão só será aceito textos que correspondam exatamente a : 'Somente não autorizados' ou 'Somente autorizados por outros usuários' ou 'Autorizados pelo meu usuário' ou 'Todos'.

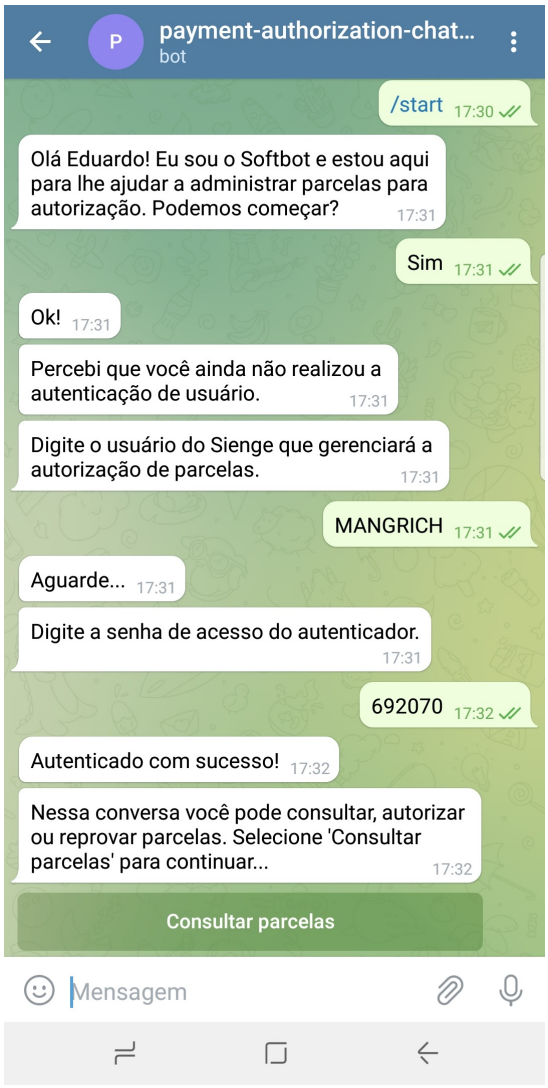
Após a inserção dos valores dos filtros será exibido um resumo de todos os filtros inseridos e o chatbot pedirá para confirmar. Caso o usuário responda 'não', os filtros serão deletados e o usuário voltará para o início da conversa. Se responder 'sim' será feita a consulta no terminal (URL base/v1/installments-for-authorization) das parcelas para autorização com os filtros inseridos. Se houver resultados, o chatbot informará uma lista de Títulos/Parcela e elaborará um relatório com detalhes de cada parcela que retornou. Caso não haja resultados, o chatbot informará que a consulta não trouxe resultados e perguntará se o usuário gostaria de alterar os filtros. Caso o usuário responder de forma positiva, o usuário volta para a cena de seleção de filtros, caso contrário a conversa volta pro início.

No cenário em que há resultados, o usuário poderá optar por inserir autorização, remover autorização ou reprovar parcela. Caso não deseje realizar nenhuma dessas operações ele digita 'não' e a conversa volta para o início. Se optar por remover ou inserir autorização o usuário deverá inserir as parcelas no formato (Titulo/Parcela) que gostaria de inserir ou digitar Todas para autorizar ou desautorizar todas. Após a escolha das parcelas será chamado o terminal (URL base/v1/installments-for-authorization/authorize) ou (URL base/v1/installments-for-authorization/disapprove) para inserir ou remover autorização respectivamente. Caso o proceso de autorização ou desautorização ocorram com sucesso o chatbot responderá que as parcelas foram autorizadas com sucesso ou que as autorizações foram removidas com sucesso. Após isso o chatbot perguntará se o usuário gostaria de realizar alguma outra operação. Se a resposta for positiva, o usuário volta para a cena de escolha da operação (inserção da autorização, remoção da autorização ou reprovação da parcela). Caso contrário a conversa voltará para o início.

Se a operação escolhida for a reprovação de parcelas, o usuário deverá digitar uma única parcela no formato (Título/Parcela) para reprovação. Após isso deverá digitar o motivo da reprovação que deverá ser um texto. Se o processo de autorização estiver configurado para hierarquia + valor e a parcela estiver na alçada do usuário que estiver reprovando, a parcela será reprovada com sucesso e o chatbot responderá que a

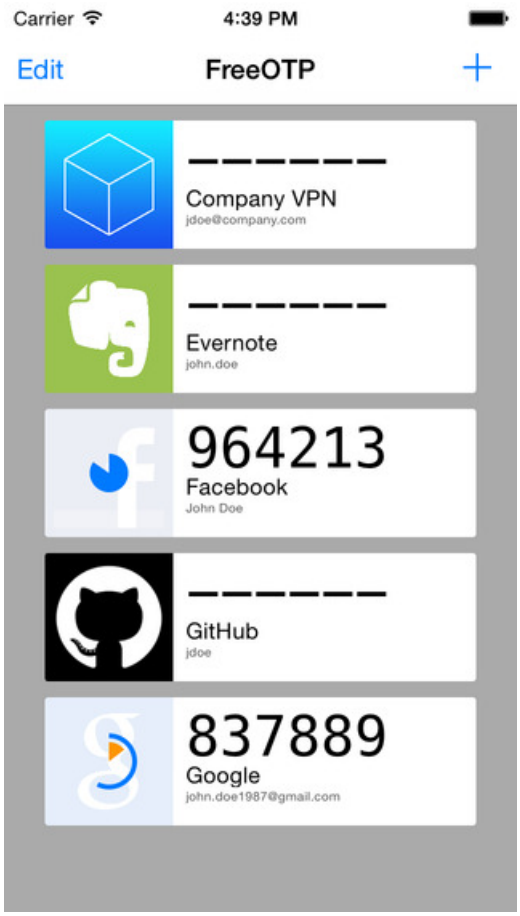
operação teve sucesso. Caso contrário o usuário será avisado que a reprovação só pode ser realizada quando a parametrização estiver configurada para hierarquia+valor.

Figura 42 – Interação de boas vindas e autenticação



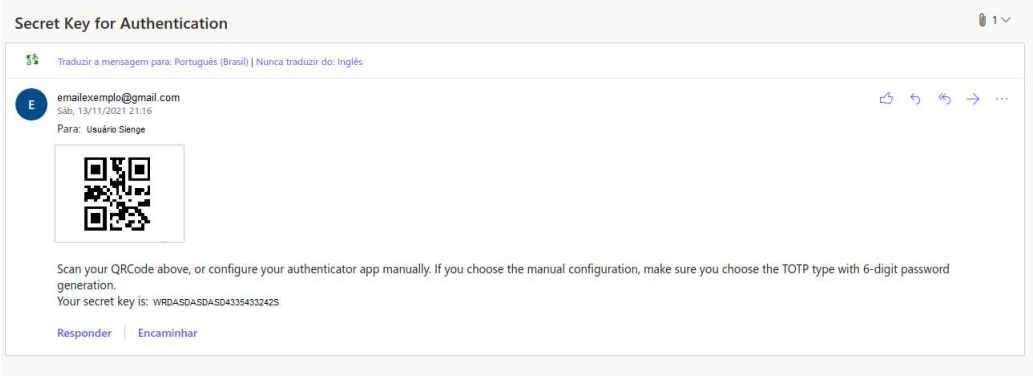
Fonte: Autor.

Figura 43 – Tela do autenticador com a senha descartável



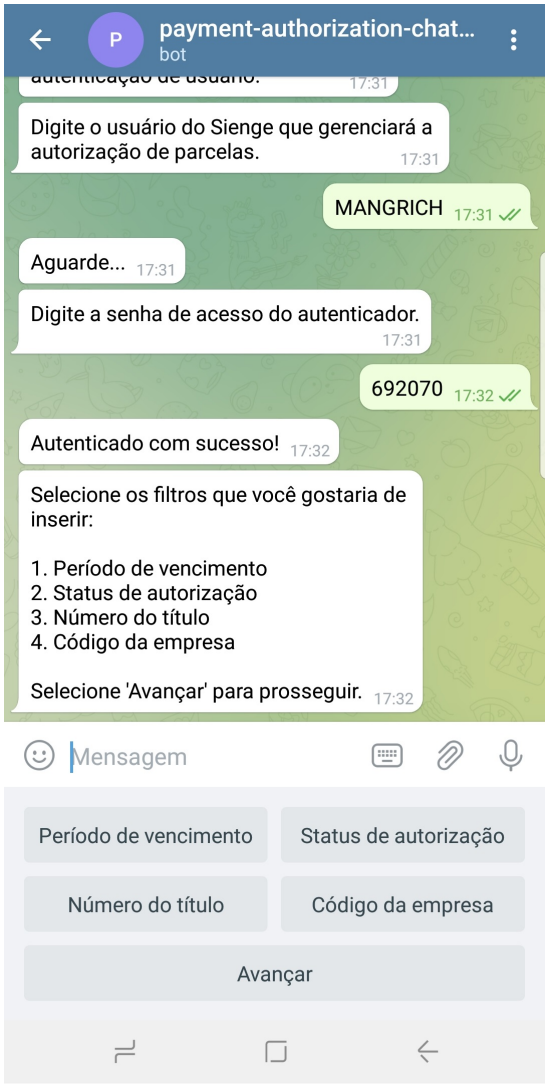
Fonte: https://freeotp.github.io/.

Figura 44 – Email com o QRCode e chave secreta para configuração



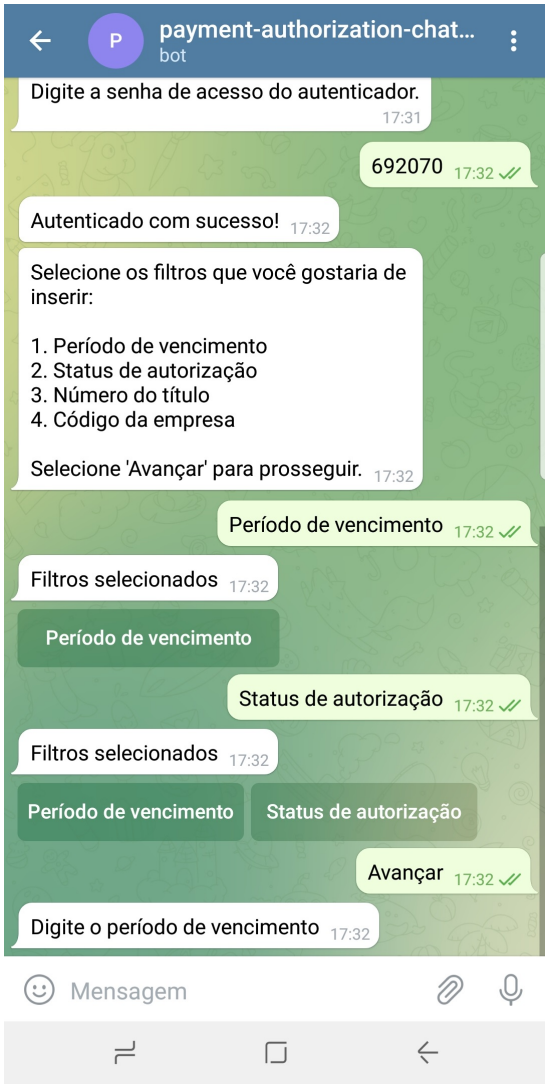
Fonte: Autor.

Figura 45 – Escolha dos filtros



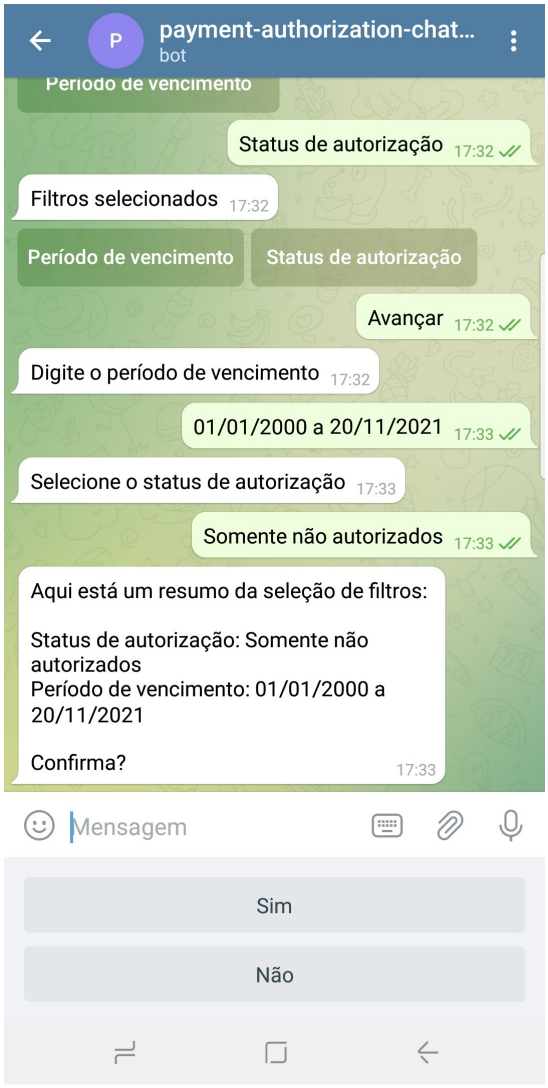
Fonte: Autor.

Figura 46 – Inserção do período de vencimento



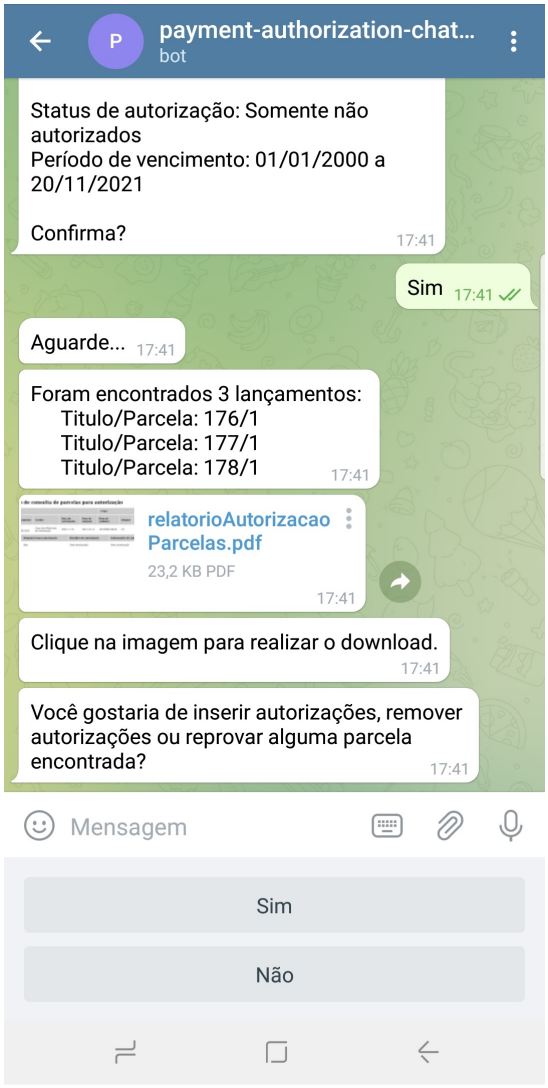
Fonte: Autor.

Figura 47 – Resumo dos filtros



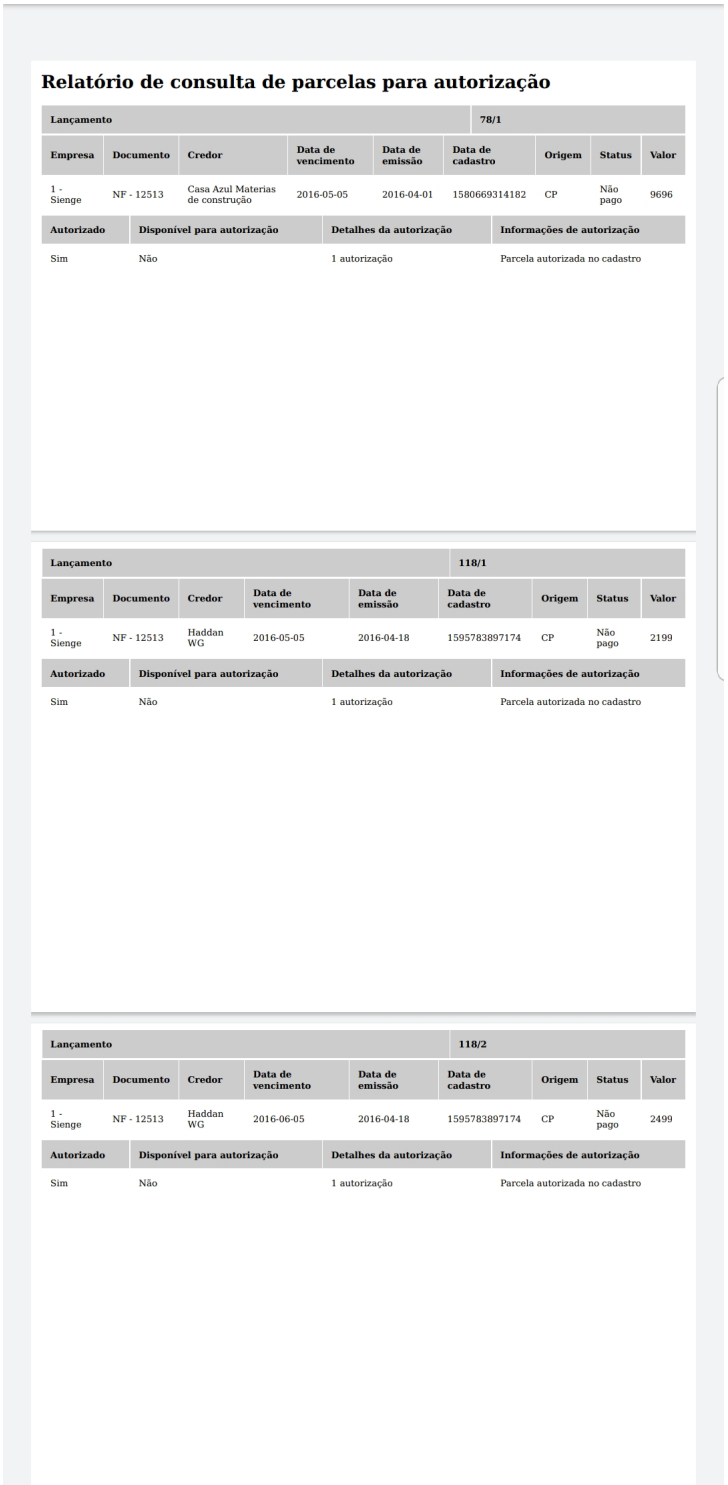
Fonte: Autor.

Figura 48 – Emissão do relatório de parcelas para autorização consultadas



Fonte: Autor.

Figura 49 – Relatório de consulta de parcelas para autorização



Relatório de consulta de parcelas para autorização

Lançamento								
78/1								
Empresa	**Documento**	**Credor**	**Data de vencimento**	**Data de emissão**	**Data de cadastro**	**Origem**	**Status**	**Valor**
1 - Sienge	NF - 12513	Casa Azul Materias de construção	2016-05-05	2016-04-01	1580669314182	CP	Não pago	9696

Autorizado	Disponivel para autorização	Detalhes da autorização	Informações de autorização
Sim	Não	1 autorização	Parcela autorizada no cadastro

Lançamento								
118/1								
Empresa	**Documento**	**Credor**	**Data de vencimento**	**Data de emissão**	**Data de cadastro**	**Origem**	**Status**	**Valor**
1 - Sienge	NF - 12513	Haddan WG	2016-05-05	2016-04-18	1595783897174	CP	Não pago	2199

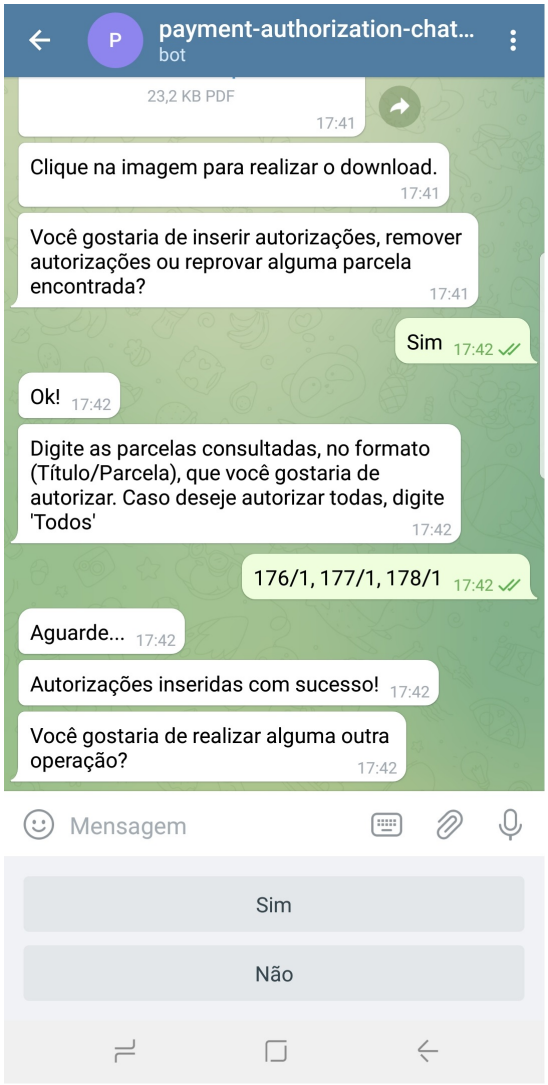
Autorizado	Disponivel para autorização	Detalhes da autorização	Informações de autorização
Sim	Não	1 autorização	Parcela autorizada no cadastro

Lançamento								
118/2								
Empresa	**Documento**	**Credor**	**Data de vencimento**	**Data de emissão**	**Data de cadastro**	**Origem**	**Status**	**Valor**
1 - Sienge	NF - 12513	Haddan WG	2016-06-05	2016-04-18	1595783897174	CP	Não pago	2499

Autorizado	Disponivel para autorização	Detalhes da autorização	Informações de autorização
Sim	Não	1 autorização	Parcela autorizada no cadastro

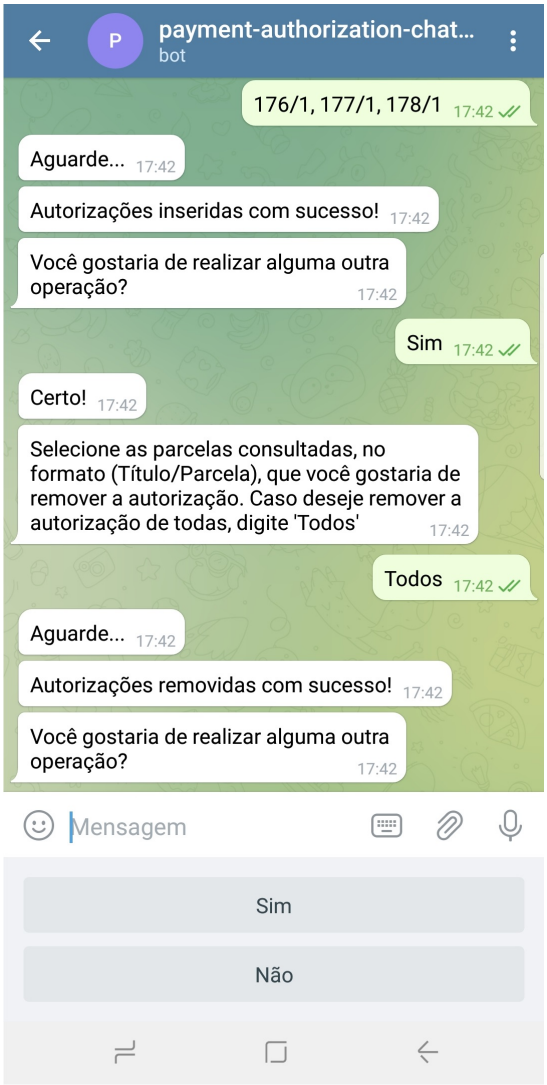
Fonte: Autor.

Figura 50 – Realização de uma nova operação



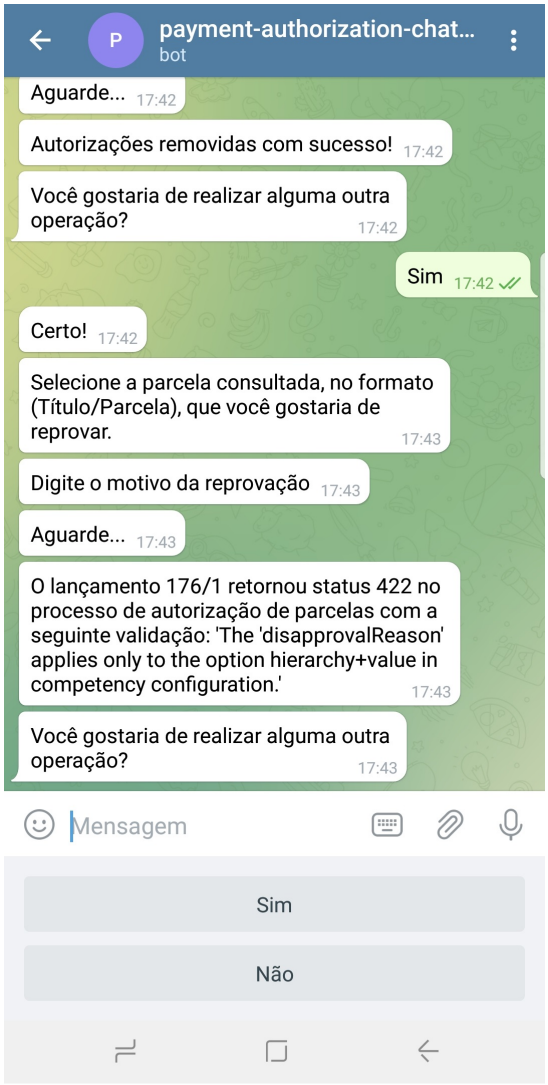
Fonte: Autor.

Figura 51 – Processo de remoção de autorização de parcelas



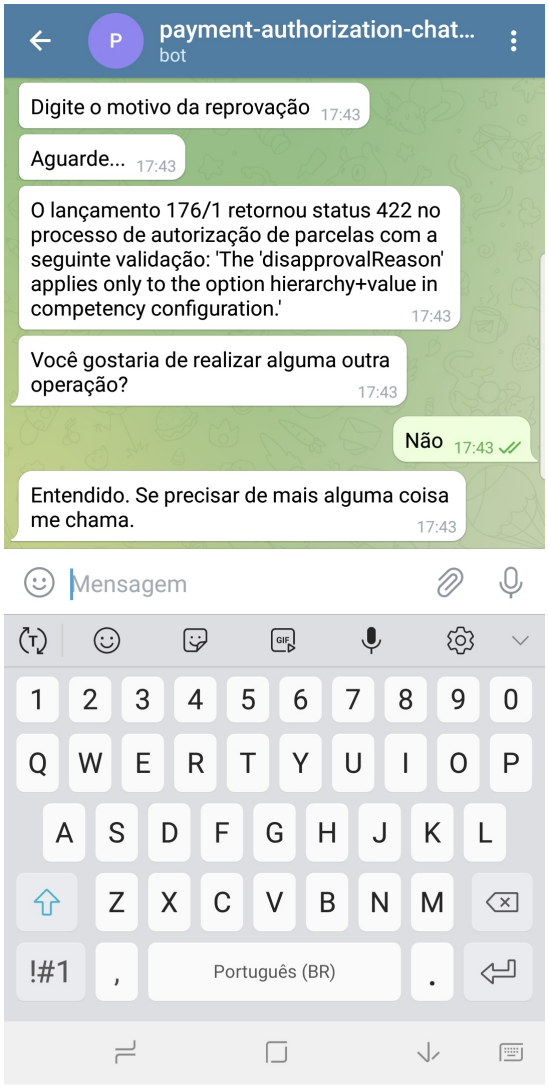
Fonte: Autor.

Figura 52 – Processo de reprovação de parcelas



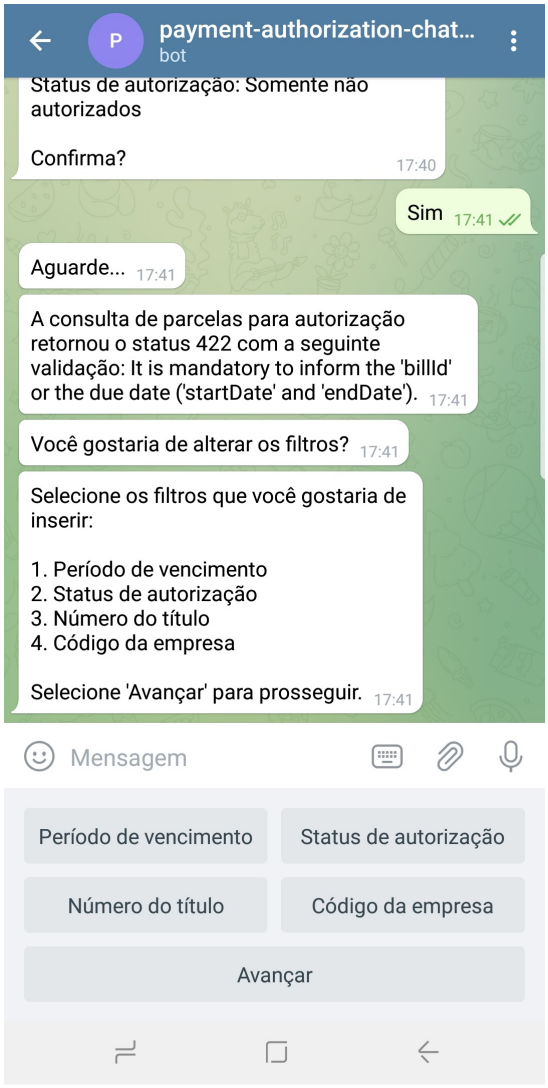
Fonte: Autor.

Figura 53 – Encerramento da conversa



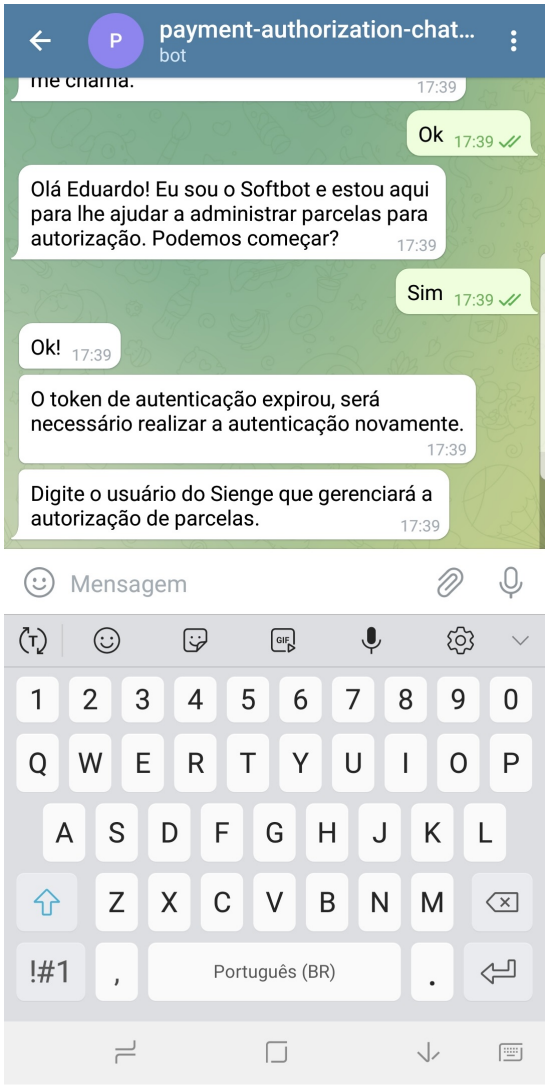
Fonte: Autor.

Figura 54 – Validação de filtros



Fonte: Autor.

Figura 55 – Expiração do token de autenticação



Fonte: Autor.

6 RESULTADOS

Neste capítulo será feita uma análise dos resultados práticos, ressaltando as métricas de código obtidas com a implementação das APIs e do chatbot e também outros resultados qualitativos.

6.1 TESTES

De acordo com a figura 56 foi obtida uma cobertura de 86.44% do total de linhas envolvidas no desenvolvimento do *chatbot*, 68% na cobertura de funções, 79.27% de cobertura nas ramificações e 84.24% de cobertura das instruções. Essa cobertura favorece uma aplicação diminuindo a probabilidade que uma falha venha a ocorrer, porém não elimina totalmente o risco de surgirem falhas.

Não foram realizados testes de integração para o *chatbot*, pois seriam testes muito custosos que demandariam um tempo maior de execução que os testes unitários.

Além disso foram realizados diversos testes ponta a ponta de forma manual que permtiu corrigir diversas falhas e aumentar a confiabilidade do *chatbot* e das APIs envolvidas no processo.

Já com relação a API de autenticação de usuários foi obtida uma cobertura de 66,3% do total de instruções (ver figura 57). Isso também garante que a API funcionará como se espera com menor risco de falhas de forma similar ao *chatbot*.

Finalmente, analisando a cobertura de testes na camada de visualização e aplicação da API de autorização de parcelas para pagamento, percebemos que há 81.4% de cobertura das linhas de código. Já na camada de serviço dentro dos pacotes que dizem respeito a autorização, reprovação, validador de dados, validador de parcela e validador de token há uma cobertura de 83.1% no número de linhas. Esses números reforçam que o processo de autorização de parcelas para pagamento terá um baixo risco de aparecimento de falhas em comparação a uma aplicação sem a execução desses testes (ver figuras 56 e 59).

Figura 56 – Percentual de cobertura de testes no *chatbot*

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	84.16	79.27	68	86.44	
bot/cancelarOperacao	100	100	100	100	
cancelarOperacao.js	100	100	100	100	
botRequest	25	0	0	27.27	
botRequest.js	18.18	0	0	20	7-87
index.js	100	100	100	100	
cenas/CenaAcaoAutorizacaoReprovacao/criarPerguntaAutorizacaoReprovacao	100	100	100	100	
criarPerguntaAutorizacaoReprovacao.js	100	100	100	100	
cenas/CenaAutenticacaoUsuario/autenticarUsuario	100	100	100	100	
autenticarUsuario.js	100	100	100	100	
cenas/CenaAutorizacaoParcelas/autorizarParcelas	100	100	100	100	
autorizarParcelas.js	100	100	100	100	
cenas/CenaBoasVindas/criarMensagemDeBoasVindas	100	100	100	100	
criarMensagemDeBoasVindas.js	100	100	100	100	
index.js	100	100	100	100	
cenas/CenaBoasVindas/verificarToken	100	100	100	100	
index.js	100	100	100	100	
verificarToken.js	100	100	100	100	
cenas/CenaCodigoEmpresa/verificarFiltroCodigoEmpresa	100	100	100	100	
verificarFiltroCodigoEmpresa.js	100	100	100	100	
cenas/CenaConfirmacaoDosDados/criarResumoFiltros	100	100	100	100	
criarResumoFiltros.js	100	100	100	100	
cenas/CenaConfirmacaoDosDados/gerarRelatorioComParcelasParaAutorizacao	100	100	100	100	
gerarRelatorioComParcelasParaAutorizacao.js	100	100	100	100	
cenas/CenaConsultaParcelas/validarSeAlgumFiltroJaFoiInserido	100	100	100	100	
validarSeAlgumFiltroJaFoiInserido.js	100	100	100	100	
cenas/CenaGeradorSenhaSecreta/gerarSenhaSecreta	100	100	100	100	
gerarSenhaSecreta.js	100	100	100	100	
cenas/CenaMotivoReprovacaoParcelas/reprovarParcelas	100	100	100	100	
reprovarParcelas.js	100	100	100	100	
cenas/CenaNumeroTitulo/verificarFiltroNumeroTitulo	100	100	100	100	
verificarFiltroNumeroTitulo.js	100	100	100	100	
cenas/CenaRemocaoAutorizacaoParcelas/removerAutorizacaoParcelas	100	100	100	100	
removerAutorizacaoParcelas.js	100	100	100	100	
cenas/CenaReprovacaoParcelas/verificarParcelaParaReprovacao	100	100	100	100	
verificarParcelaParaReprovacao.js	100	100	100	100	
cenas/CenaStatusAutorizacao/verificarFiltroStatusAutorizacao	100	100	100	100	
verificarFiltroStatusAutorizacao.js	100	100	100	100	
converterHtmlEmPdf	60	100	0	60	
converterHtmlEmPdf.js	50	100	0	50	6-22
index.js	100	100	100	100	
gerarTabelaParaRelatorio	29.73	0	0	35.48	
gerarTabelaParaRelatorio.js	25.71	0	0	31.03	21-35,49-50,58-113,151-175
index.js	100	100	100	100	

Fonte: Autor.

Figura 57 – Percentual de cobertura de testes na API de autenticação

Counter	Coverage	Covered	Missed	Total
Instructions	66,3 %	589	300	889
Branches	83,3 %	15	3	18
Lines	61,2 %	120	76	196
Methods	56,1 %	60	47	107
Types	52,8 %	19	17	36
Complexity	57,8 %	67	49	116

Fonte: Autor.

Figura 58 – Percentual de cobertura na camada de aplicação e visualização da API de autorização de parcelas para pagamento

[all classes]

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	85.7% (6/ 7)	78% (64/ 82)	81.4% (179/ 220)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
br.com.softplan.unic.paymentauthorization.v1.controller	100% (1/ 1)	80% (4/ 5)	78.9% (30/ 38)
br.com.softplan.unic.paymentauthorization.v1.dtos	83.3% (5/ 6)	77.9% (60/ 77)	81.9% (149/ 182)

Fonte: Autor.

Figura 59 – Percentual de cobertura na camada de serviço da API de autorização de parcelas para pagamento

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	71.4% (10/ 14)	71.4% (60/ 84)	83.1% (295/ 355)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
br.com.softplan.sienge.cpg.installment.authorization.api.defaultInstallmentForAuthorization	0% (0/ 1)	0% (0/ 8)	0% (0/ 16)
br.com.softplan.sienge.cpg.installment.authorization.api.installmentValidator	100% (2/ 2)	81.8% (9/ 11)	95.7% (44/ 46)
br.com.softplan.sienge.cpg.installment.authorization.api.tokenValidator	100% (1/ 1)	100% (5/ 5)	100% (19/ 19)
br.com.softplan.sienge.cpg.installment.authorization.api.transactional	0% (0/ 1)	0% (0/ 4)	0% (0/ 18)
br.com.softplan.sienge.cpg.installment.authorization.api.disapprove	75% (3/ 4)	81.2% (26/ 32)	91.9% (137/ 149)
br.com.softplan.sienge.cpg.installment.authorization.api.authorize	75% (3/ 4)	81.8% (18/ 22)	87.6% (85/ 97)
br.com.softplan.sienge.cpg.installment.authorization.api.dataValidator	100% (1/ 1)	100% (2/ 2)	100% (10/ 10)

Fonte: Autor.

6.2 CHATBOT

Foi realizada uma pesquisa com 12 pessoas das áreas de suporte, gerência e desenvolvimento validando a percepção do usuário com relação a usabilidade, produtividade e segurança do processo de autorização de parcelas para pagamento via chatbot em comparação a aquele processo realizado via Sienge em dispositivos móveis. Para obter essa percepção, inicialmente o respondente realizou o processo de consulta de parcelas na tela de autorização de pagamentos do Sienge via dispositivo móvel e autorizou ou desautorizou alguma parcela. Em seguida, o respondente realizou o mesmo processo, porém dessa vez via chatbot por acesso remoto. Ao fim desses dois testes cada respondente preencheu um formulário avaliando qualitativamente a experiência com o chatbot e a experiência com o Sienge no dispositivo móvel.

Entre os entrevistados aproximadamente 58% deles estavam familiarizados com o processo de autorização de parcelas para pagamento realizado no Sienge. Ao avaliar em uma escala de 1 a 10 a usabilidade do processo de autorização de parcelas para pagamento via Sienge em dispositivos móveis, a nota média foi de 5.83, enquanto que a usabilidade do mesmo processo realizado via chatbot teve como nota média a nota 9. Isso indica que a usabilidade do chatbot para o processo de autorização foi aproximadamente 54% melhor avaliada que a usabilidade no Sienge via dispositivo móvel. Já com relação a avaliação da produtividade ao realizar o processo, a tela do Sienge para dispositivos móveis teve como nota média 5.83 enquanto que o chatbot recebeu a nota média de 8.58. Nesse caso houve um aumento de aproximadamente 47% na produtividade do processo por meio do chatbot com relação ao Sienge acessado por dispositivo móvel. Ao avaliar a segurança tanto via chatbot quanto pelo Sienge no dispositivo móvel, o chatbot teve uma nota média de 9 enquanto que o dispositivo móvel teve uma nota média de 8.33. Logo, a percepção de segurança dos usuários foi bem próxima nos dois casos, porém no chatbot ainda foi aproximadamente 8% maior. Com relação a preferência pela realização do processo de autorização via chabot ou Sienge em dispositivos móveis, aproximadamente 92% dos usuários responderam que preferem realizar via chatbot. Finalmente, quando perguntados se recomendariam o uso do chatbot a outras pessoas 100% dos entrevistados responderam que sim.

6.3 ANÁLISE DOS RESULTADOS PRÁTICOS

A implementação do *chatbot* trouxe diversos ganhos em termos qualitativos. A usabilidade foi aperfeiçoada em comparação a tela de autorização de pagamentos em dispositivos móveis, pois o processo de consulta, autorização e reprovação das parcelas se adapta a tela do dispositivo móvel, evitando a necessidade de ampliar ou reduzir o tamanho da imagem para preencher algum campo de formulário. Em vez disso, o *chatbot* guia o usuário em cada etapa do processo, interagindo de forma

intuitiva, sendo necessário ao usuário apenas digitar dados no teclado do celular, clicar em botões que preenchem boa parte da tela e prosseguir até o final da operação. Dessa forma, o usuário também ganha em produtividade, pois consegue realizar o processo de forma mais rápida do que antes.

Além disso, a segurança é garantida pelo processo de autenticação. Nenhuma etapa do processo de autorização pode ser realizada sem a autenticação. Isso permite a realização de qualquer operação da mesma forma que um usuário logado no Sienge consegue fazer na tela de autorização de pagamentos.

Entretanto, não é possível obter a percepção do cliente acerca da usabilidade e ganho de produtividade do *chatbot*, pois não houve implantação em nenhum cliente. A implantação vai além do escopo desse trabalho no qual se propõe o projeto e a implementação do *chatbot* dentro de uma empresa de software. Uma outra limitação desse trabalho é que não foi pensada em uma arquitetura que armazenasse o token de cada chatbot e inicializasse todos os bots na aplicação. Ou seja, somente 1 bot de autorização de pagamentos pode ser inicializado pela aplicação em node o que limita o alcance da solução a outras pessoas.

Além disso, todo o desenvolvimento do trabalho, com exceção da API de autorização de parcelas para pagamento, está hoje na máquina do autor. Para escalar a solução com diversos clientes, o código fonte deve estar no ambiente de produção do Sienge. Dessa forma, a solução proposta pelo autor hoje não é escalável.

Uma outra limitação desse trabalho é que não foi levado em conta a possibilidade de conversas em grupo com mais de 1 usuário. Isso ocorreu devido a restrições no escopo e por não ser uma funcionalidade essencial para que o chatbot funcione dentro do esperado.

Seguindo os princípios da simplicidade e eficácia, esse trabalho não utilizou a técnica de processamento de linguagem natural, pois o contexto é bem limitado (tela de autorização de parcelas para pagamento) e o fluxo sempre passa pela inserção de filtros, consulta e ação de autorizar, desautorizar ou reprovar. Por se tratar de um fluxo sequencial, a conversa foi guiada pelo chatbot tornando a usabilidade mais intuitiva. Entretanto, o uso de processamento de linguagem natural pode tornar a conversa menos artificial e mais humana e assim melhorar a interação com o usuário para entradas de dados que não são previstas pelo chatbot.

Além disso, a inserção de filtros no chatbot foi limitada a 4 filtros: período de vencimento, código da empresa, número do título e status de autorização. Esses são os principais filtros para a consulta de parcelas para autorização. Entretanto, a tela de autorização de pagamentos possui 18 tipos de filtro. Foi decidido limitar a 4 filtros num primeiro momento para diminuir a complexidade da API de autorização de parcelas para pagamento. Esse grupo de filtros pode ser expandido em trabalhos futuros de acordo com as necessidades do cliente.

7 CONCLUSÃO

A pobre usabilidade da tela de autorização de pagamentos em dispositivos móveis junto a pouca produtividade na realização de processos ligados a autorização de parcelas motivou a elaboração de uma solução que fosse mais amigável ao cliente em termos de usabilidade, com maior produtividade e segurança no processo de autorização.

Essa solução incluiu o projeto de uma API de autorização de parcelas para pagamento, externalizando os serviços de consulta, autorização e reprovação para aplicações externas. Incluiu também o projeto de uma API de autenticação de usuários que identificasse o usuário do Sienge antes de prosseguir com a autorização, garantindo que nenhum usuário externo ao Sienge possa realizar autorizações em nome de algum usuário do Sienge. Finalmente, a solução incluiu o projeto de um *chatbot* que interage com o usuário, guiando-o passo a passo para realizar o processo de autorização com autenticação, garantindo boa usabilidade, produtividade e segurança.

Além dessas melhorias, existem outras que podem ser realizadas em trabalhos futuros e que não tiveram foco nesse projeto como:

- Uso de técnicas de processamento de linguagem natural para processar intenções do usuário e tornar a conversa mais humana e menos repetitiva no *chatbot*;
- Customização do relatório de parcelas para autorização, permitindo inserir ou remover filtros de visualização;
- Criação de serviço de notificação via *chatbot* que informa ao usuário quando a parcela que ele cadastrou for autorizada, reprovada, desautorizada por outro usuário;
- Reconhecimento de comandos de voz, expandindo a usabilidade do chatbot que hoje se limita a comandos de toque na tela;

REFERÊNCIAS

ABITEBOUL, Serge; HULL, Richard; VIANU, Victor. **Foundations of Databases**. [*S.l.*]: Addison-Wesley, 1995. ISBN 0-201-53771-0. Acesso em: 8 out. 2021.

ADAMOPOULOU, Eleni; MOUSSIADES, Lefteris. An Overview of Chatbot Technology. *In*: MAGLOGIANNIS, Ilias; ILIADIS, Lazaros; PIMENIDIS, Elias (Ed.). **Artificial Intelligence Applications and Innovations**. Cham: Springer International Publishing, 2020. P. 373–383.

ALOUL, Fadi; ZAHIDI, Syed; EL-HAJJ, Wassim. **Two factor authentication using mobile phones**. [*S.l.*], 2009. P. 641–644. DOI: 10.1109/AICCSA.2009.5069395. Acesso em: 8 out. 2021.

BATACHARIA, B.; LEVY, D.; CATIZONE, R.; KROTOV, A.; WILKS, Yorick. CONVERSE: A conversational companion. **Machine Conversations**, jan. 1999. DOI: 10.1007/978-1-4757-5687-6_17.

BAUER, Christian; KING, Gavin. **Hibernate in Action (In Action Series)**. USA: Manning Publications Co., 2004. ISBN 193239415X. Acesso em: 8 out. 2021.

BECK, Kent. **Test Driven Development. By Example (Addison-Wesley Signature)**. [*S.l.*]: Addison-Wesley Longman, Amsterdam, 2002.

BECK, Kent *et al.* **Manifesto for Agile Software Development**. [*S.l.*: *s.n.*], 2001. Disponível em: http://www.agilemanifesto.org/. Acesso em: 16 out. 2021.

BOATENG, Richard; OFOEDA, Joshua; EFFAH, John. Application Programming Interface (API) Research: A Review of the Past to Inform the Future. **Int. J. Enterp. Inf. Syst.**, IGI Global, USA, v. 15, n. 3, p. 76–95, jul. 2019. ISSN 1548-1115. DOI: 10.4018/IJEIS.2019070105.

BRANDTZAEG, Petter; FØLSTAD, Asbjørn. Why People Use Chatbots. *In*: DOI: 10.1007/978-3-319-70284-1_30.

DABKOWSKI, Andrzej; JANKOWSKA, Anna Maria. **Comprehensive Framework for Mobile ERP System**. [*S.l.*], 2003. Acesso em: 16 fev. 2021.

DATE, C.J. **An Introduction to Database Systems**. 8. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321197844. Acesso em: 8 out. 2021.

DICTIONARY.COM. **Integrated development environment Definition & Meaning**. [*S.l.*]. Disponível em: https://www.dictionary.com/browse/integrated-development-environment. Acesso em: 14 dez. 2021.

DIGITAL, SAJ. **Sobre a Softplan**. [*S.l.*], 2020a. Disponível em: https://www.sajdigital.com/quem-somos/sobre-a-softplan/. Acesso em: 23 mar. 2021.

DIGITAL, SAJ. **Sobre o SAJ**. [*S.l.*], 2020b. Disponível em: https://www.sajdigital.com/institucional/sobre-saj/. Acesso em: 23 mar. 2021.

DINGSØYR, Torgeir; NERUR, Sridhar; BALIJEPALLY, VenuGopal; MOE, Nils Brede. A decade of agile methodologies: Towards explaining agile software development. **Journal of Systems and Software**, v. 85, n. 6, p. 1213–1221, 2012. Special Issue: Agile Development. ISSN 0164-1212. Acesso em: 16 out. 2021.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. [*S.l.*], 2000. Acesso em: 20 nov. 2021.

FLANAGAN, D. **JavaScript: O Guia Definitivo**. [*S.l.*]: Bookman Editora, 2011. ISBN 9788565837484. Acesso em: 10 out. 2021.

FOWLER, Martin. **BroadStackTest**. [*S.l.*], 2013. Acesso em: 9 out. 2021.

FOWLER, Martin. **IntegrationTest**. [*S.l.*], 2018. Acesso em: 9 out. 2021.

FRIEDL., Jeffrey E. F. **Mastering Regular Expressions**. [*S.l.*], 2007. Acesso em: 13 out. 2021.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John M. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. [*S.l.*]: Addison-Wesley Professional, 1994. ISBN 0201633612. Acesso em: 3 out. 2021.

GITLAB. **DevOps Platform**. [*S.l.*], 2021a. Disponível em: https://about.gitlab.com/solutions/devops-platform/. Acesso em: 6 out. 2021.

GITLAB. **What is GitLab?** [*S.l.*], 2021b. Disponível em: https://about.gitlab.com/what-is-gitlab/. Acesso em: 6 out. 2021.

GLOTZ, P.; BRAUN, A. **Chatbots in der Kundenkommunikation**. [*S.l.*]: Springer Berlin Heidelberg, 2013. (Xpert.press). ISBN 9783642190216. Disponível em: https://books.google.com.br/books?id=GBAdBgAAQBAJ.

GOSLING, James; JOY, Bill; STEELE, Guy; BRACHA, Gilad. The Java Language Specification, Third Edition. *In*: [*S.l.*: *s.n.*], jun. 2005. P. 688. Acesso em: 25 set. 2021.

HUTCHENS, Jason L.; ALDER, Michael D. Introducing MegaHAL. *In*: NEW Methods in Language Processing and Computational Natural Language Learning. [*S.l.*: *s.n.*], 1998.

JOHNSON, Joseph. **Global digital population as of October 2020**: Worldwide digital population as of October 2020. [*S.l.*], 2021. Disponível em: https://www.statista.com/statistics/617136/digital-population-worldwide/. Acesso em: 16 fev. 2021.

JOHNSON, Rod *et al.* **The Spring Framework - Reference Documentation**. [*S.l.*], 2008. Acesso em: 2 nov. 2021.

JONES, Michael; BRADLEY, John; SAKIMURA, Nat. **JSON Web Token (JWT)**. [*S.l.*]: RFC Editor, mai. 2015. RFC 7519. (Request for Comments, 7519). DOI: 10.17487/RFC7519. Disponível em: https://rfc-editor.org/rfc/rfc7519.txt. Acesso em: 21 out. 2021.

KELECHAVA, Brad. **The SQL Standard – ISO/IEC 9075:2016 (ANSI X3.135)**. [*S.l.*], 2018. Disponível em: https://blog.ansi.org/2018/10/sql-standard-iso-iec-9075-2016-ansi-x3-135/?_ga=2.194375592.446844567.1639533541-810145022.1639533541. Acesso em: 14 dez. 2021.

KIM, Jeong Ho *et al.* **Are there differences in muscle activity, subjective discomfort, and typing performance between virtual and conventional keyboards**. [*S.l.*], 2012. Acesso em: 16 fev. 2021.

KIM, Jeong Ho *et al.* **Differences in typing forces, muscle activity, comfort, and typing performance among virtual, notebook, and desktop keyboards**. [*S.l.*], 2014. Acesso em: 17 fev. 2021.

KLOPFENSTEIN, Lorenz; DELPRIORI, Saverio; MALATINI, Silvia; BOGLIOLO, Alessandro. The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms. *In*: p. 555–565. DOI: 10.1145/3064663.3064672.

LEXICO. **chatbot | Definition of chatbot in English by Lexico Dictionaries.** [*S.l.*], 2021. Disponível em: https://www.lexico.com/en/definition/chatbot. Acesso em: 29 nov. 2021.

LIU, Shanhong. **Enterprise resource planning (ERP) software market revenues worldwide from 2019 to 2024**: Worldwide enterprise resource planning software market size 2019-2024. [*S.l.*], 2021a. Disponível em: https://www.statista.com/statistics/605888/worldwide-enterprise-resource-planning-market-forecast/. Acesso em: 16 fev. 2021.

LIU, Shanhong. **Information technology (IT) spending on enterprise software worldwide, from 2009 to 2022**: Enterprise software total worldwide expenditure 2009-2022. [*S.l.*], 2021b. Disponível em: https://www.statista.com/statistics/203428/total-enterprise-software-revenue-forecast/. Acesso em: 16 fev. 2021.

LOELIGER, Jon. **Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development**. 1st. [*S.l.*]: O'Reilly Media, Inc., 2009. ISBN 0596520123. Acesso em: 9 out. 2021.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Boston, MA: Prentice Hall, 2017. (Robert C. Martin Series). ISBN 978-0-13-449416-6. Acesso em: 29 out. 2021.

MARTIN, Robert C.; COPLIEN, James O. **Clean code: a handbook of agile software craftsmanship**. [*S.l.*]: Prentice Hall, 2009. ISBN 9780132350884 0132350882. Acesso em: 29 out. 2021.

MAVEN, Apache. **Welcome to Apache Maven**. [*S.l.*], 2021a. Disponível em: https://maven.apache.org/. Acesso em: 5 out. 2021.

MAVEN, Apache. **What is Maven**. [*S.l.*], 2021b. Disponível em: https://maven.apache.org/what-is-maven.html. Acesso em: 5 out. 2021.

MYERS, Glenford J.; SANDLER, Corey. **The Art of Software Testing**. Hoboken, NJ, USA: John Wiley e Sons, Inc., 2004. ISBN 0471469122. Acesso em: 2 out. 2021.

NIELSEN, Henrik; MOGUL, Jeffrey; MASINTER, Larry M; FIELDING, Roy T.; GETTYS, Jim; LEACH, Paul J.; BERNERS-LEE, Tim. **Hypertext Transfer Protocol – HTTP/1.1**. [*S.l.*]: RFC Editor, jun. 1999. RFC 2616. (Request for Comments, 2616). DOI: 10.17487/RFC2616. Disponível em: https://rfc-editor.org/rfc/rfc2616.txt. Acesso em: 27 jun. 2021.

OMAR, Khalil. **Towards Improving the Usability of Mobile ERP**: A Model for Devising Adaptive Mobile UIs to Improve the Usability of Mobile ERP. [*S.l.*], 2015. Acesso em: 16 fev. 2021.

ORACLE. **What Is a Database?** [*S.l.*], 2021. Disponível em: https://www.oracle.com/database/what-is-database/. Acesso em: 14 dez. 2021.

RICHARDS, Mark. **Software Architecture Patterns**. [*S.l.*]: O'Reilly Media, Inc., 2015. ISBN 9781491925409.

SCHWABER, Ken. SCRUM Development Process. *In*: acesso em: 15 out. 2021.

SHAWAR, Bayan; ATWELL, Eric. A Comparison Between Alice and Elizabeth Chatbot Systems. **University of Leeds, School of Computing research report 2002.19**, fev. 2002.

SHAWAR, Bayan; ATWELL, Eric. Chatbots: Are they Really Useful? **LDV Forum**, v. 22, p. 29–49, jan. 2007. Acesso em: 11 out. 2021.

SIENGE. **Financeiro - Sienge**. [*S.l.*], 2012. Disponível em: https://www.sienge.com.br/financeiro/. Acesso em: 20 jun. 2021.

SIENGE. **O Sienge**. [*S.l.*], 2018. Disponível em: https://www.sienge.com.br/o-sienge/. Acesso em: 19 jun. 2021.

SOFTPLAN. **Gestão Pública**. [*S.l.*], 2020a. Disponível em: https://www.softplan.com.br/solucoes/#solucoes_gestao-publica. Acesso em: 23 mar. 2021.

SOFTPLAN. **Indústria da Construção**. [*S.l.*], 2020b. Disponível em: https://www.softplan.com.br/industria-da-construcao/. Acesso em: 23 mar. 2021.

SOFTPLAN. **Quem somos - Softplan**. [*S.l.*], 2020c. Disponível em: https://www.softplan.com.br/quem-somos/. Acesso em: 23 mar. 2021.

SOFTPLAN. **Transformando a Gestão Pública**. [*S.l.*], 2020d. Disponível em: https://www.softplan.com.br/gestao-publica/. Acesso em: 23 mar. 2021.

SOHAN, S.M.; ANSLOW, Craig; MAURER, Frank. A Case Study of Web API Evolution. *In*: 2015 IEEE World Congress on Services. [*S.l.*: *s.n.*], 2015. P. 245–252. DOI: 10.1109/SERVICES.2015.43.

SONTOW, Dr. Karsten. **ERP in Practice - User Satisfaction,Benefits and Prospects**. [*S.l.*], 2014. Acesso em: 16 fev. 2021.

STYLOS, Jeffrey; MYERS, Brad. Mapping the Space of API Design Decisions. *In*: p. 50–60. DOI: 10.1109/VLHCC.2007.44.

TELEGRAM. **Bots: An introduction for developers**. [*S.l.*], 2021a. Disponível em: https://core.telegram.org/bots#inline-mode. Acesso em: 13 out. 2021.

TELEGRAM. **Telegram a new era of messaging**. [*S.l.*], 2021b. Disponível em: https://telegram.org/. Acesso em: 13 out. 2021.

VIEW, Mountain; RYDELL, Johan; PEI, Mingliang; MACHANI, Salah. **TOTP: Time-Based One-Time Password Algorithm**. [*S.l.*]: RFC Editor, mai. 2011. RFC 6238. (Request for Comments, 6238). DOI: 10.17487/RFC6238. Disponível em: https://rfc-editor.org/rfc/rfc6238.txt. Acesso em: 12 out. 2021.

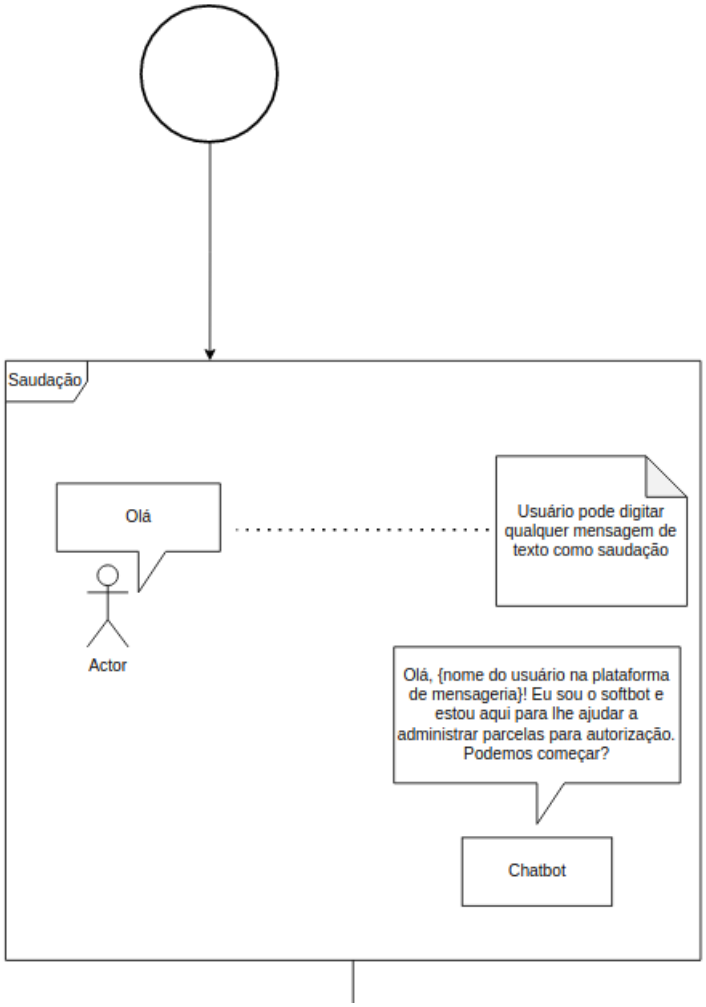
WALLACE, Richard. **The elements of AIML style. ALICE AI Foundation.** [*S.l.*: *s.n.*], 2004.

WEIZENBAUM, Joseph. ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 9, n. 1, p. 36–45, jan. 1966. ISSN 0001-0782. DOI: 10.1145/365153.365168.

ZADROZNY, Wlodek; BUDZIKOWSKA, Malgorzata; CHAI, Joyce; KAMBHATLA, Nanda; LEVESQUE, Sylvie; NICOLOV, Nicolas. Natural Language Dialogue for Personalized Interaction. **Commun. ACM**, v. 43, p. 116–120, ago. 2000.

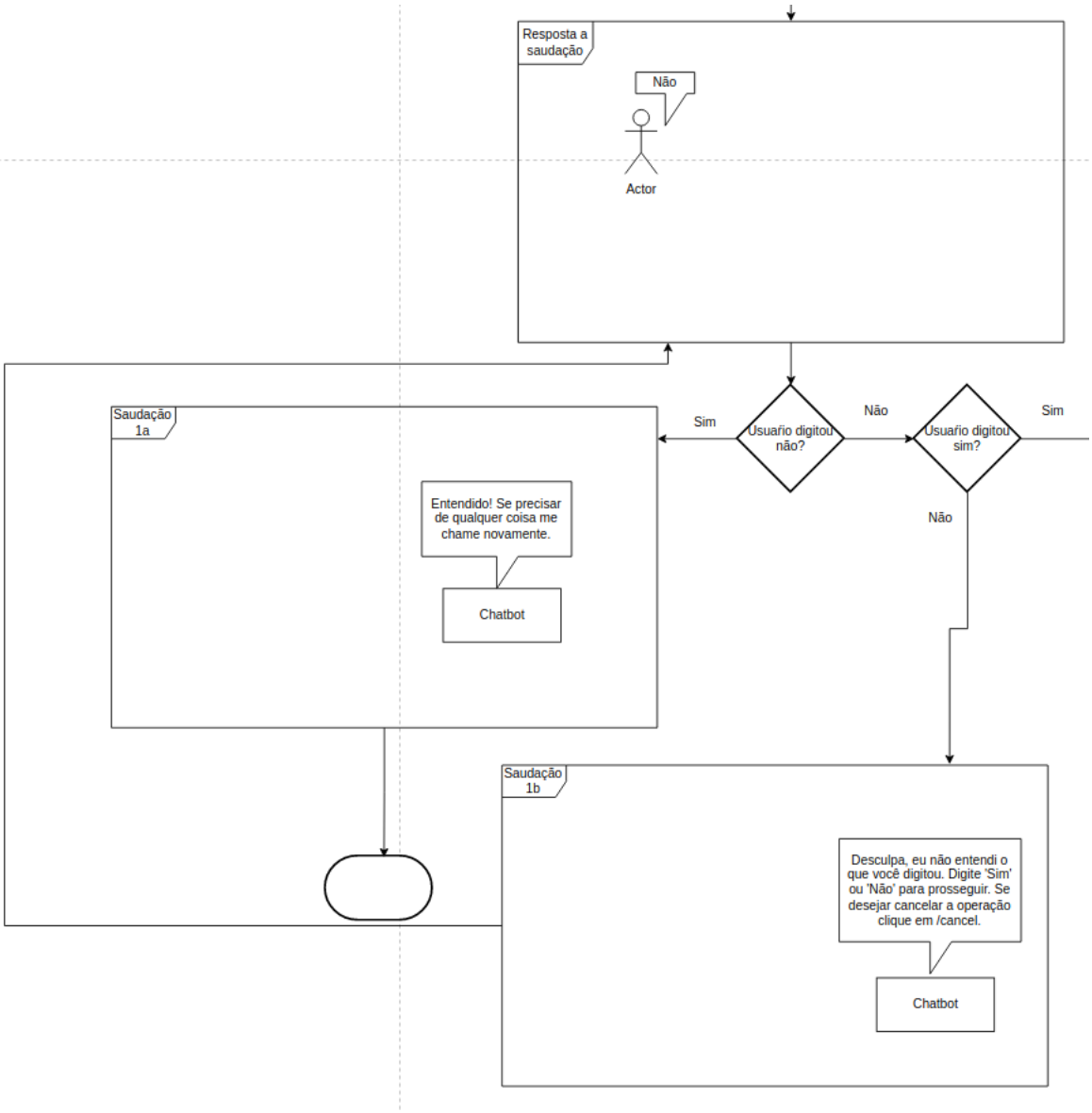
APÊNDICE A – MODELAGEM DA INTERAÇÃO ENTRE USUÁRIO E CHATBOT

Figura 60 – Saudação inicial



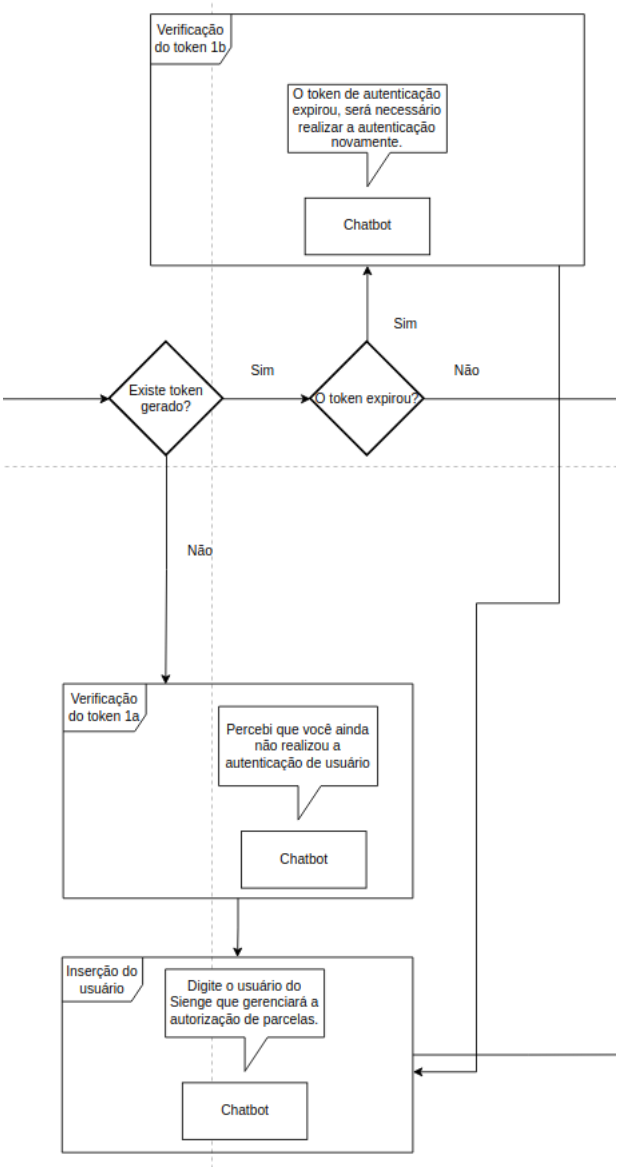
Fonte: Autor.

Figura 61 – Resposta a saudação do chatbot



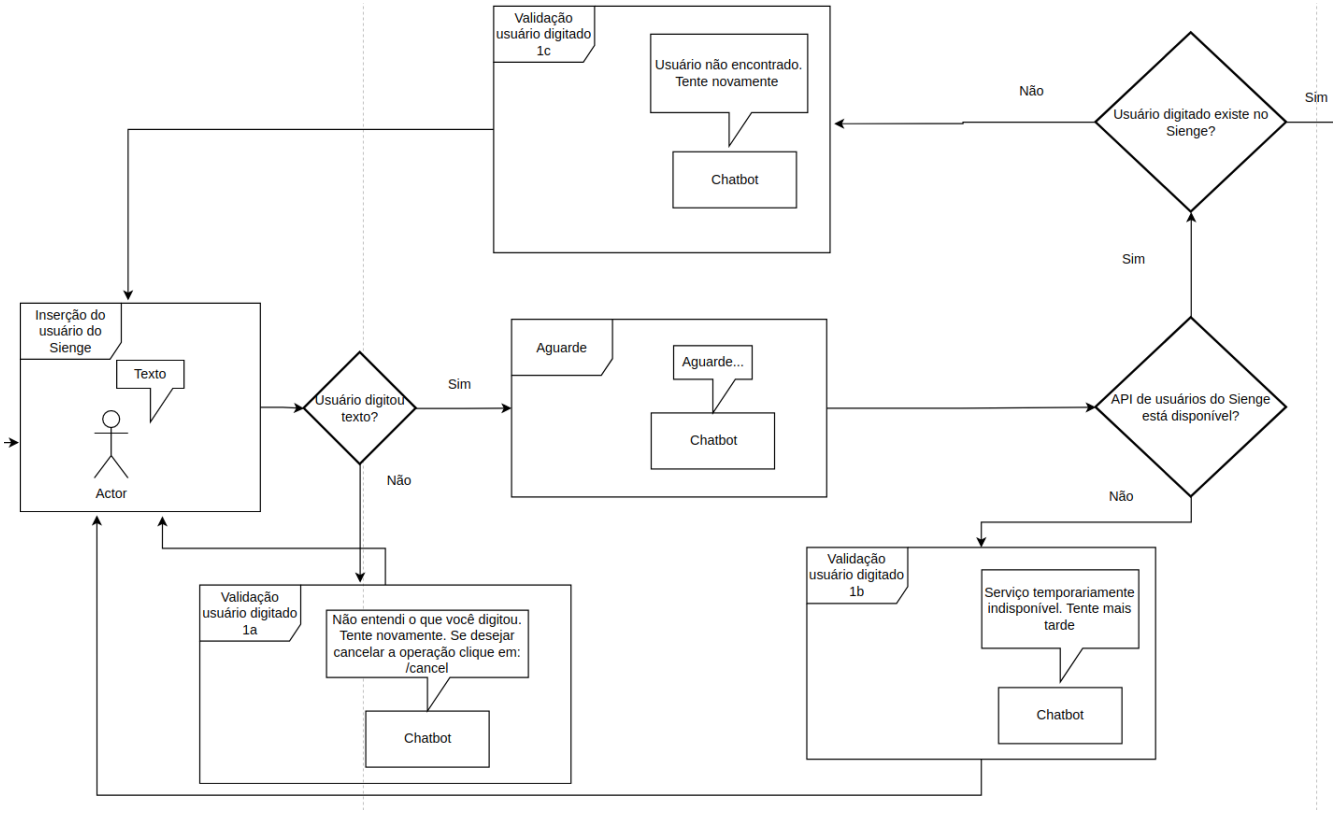
Fonte: Autor.

Figura 62 – Verificação do token JWT



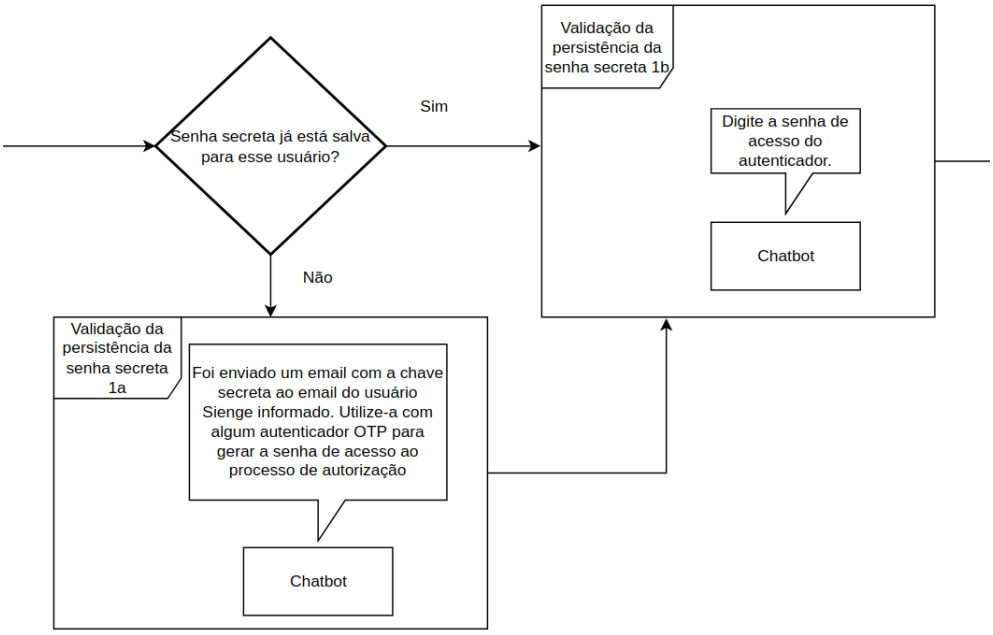
Fonte: Autor.

Figura 63 – Validação do usuário digitado



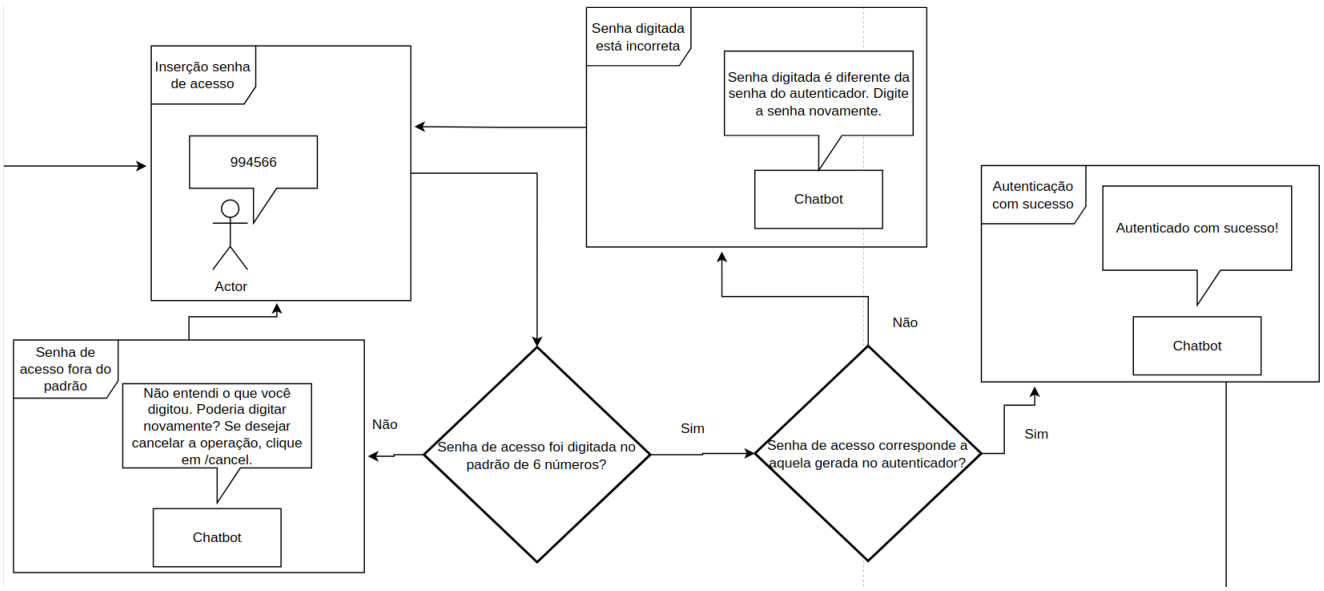
Fonte: Autor.

Figura 64 – Validação da persistência da senha secreta



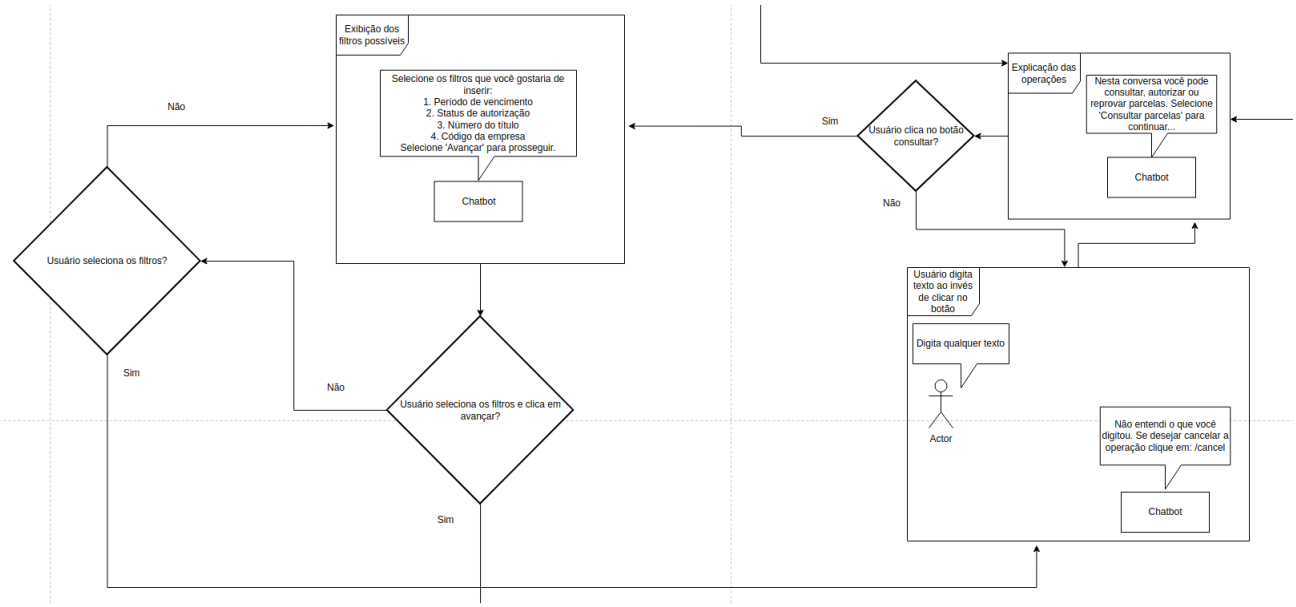
Fonte: Autor.

Figura 65 – Validação da senha de acesso do autenticador



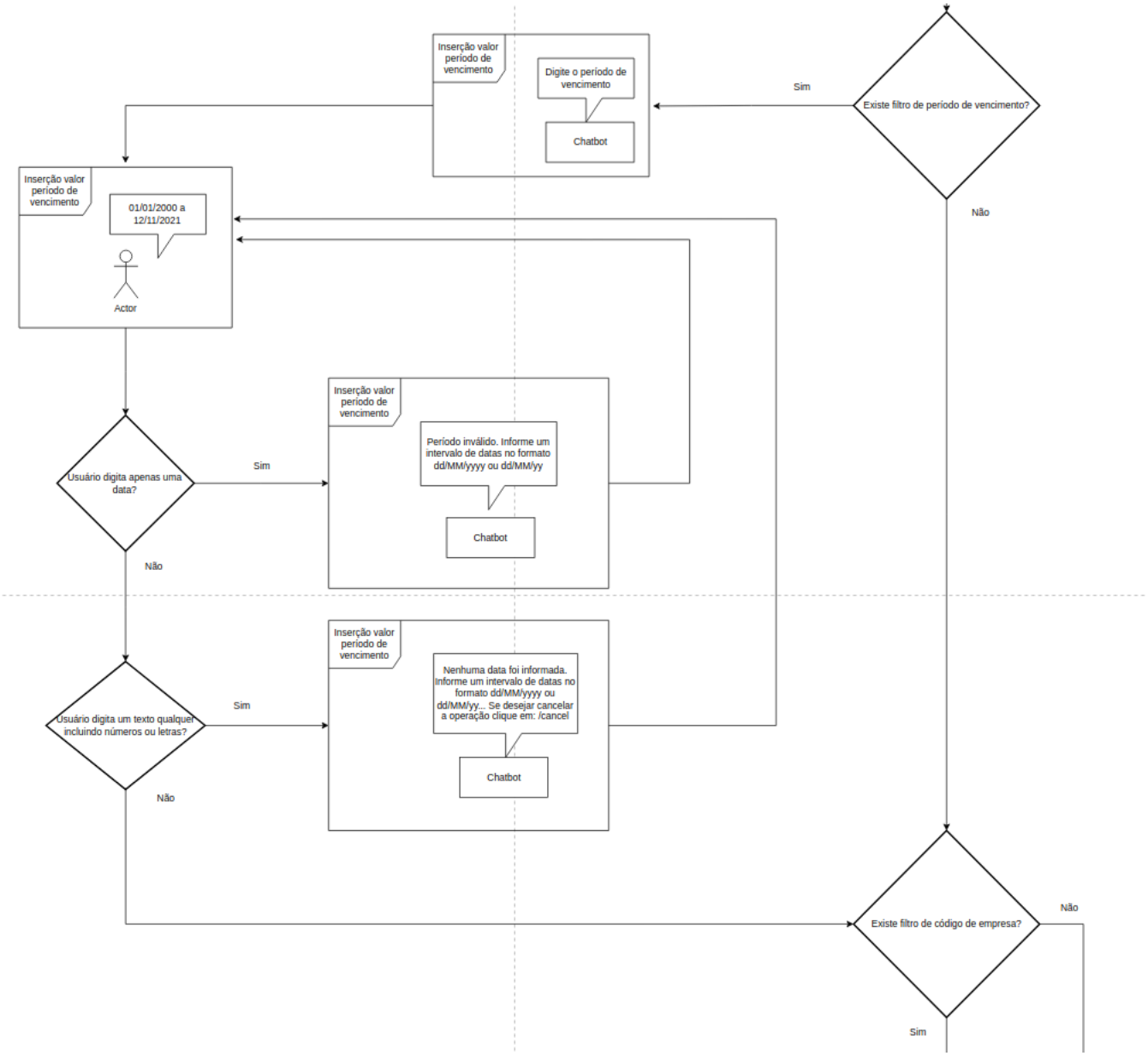
Fonte: Autor.

Figura 66 – Explicação das operações possíveis no chatbot e exibição dos filtros para consulta



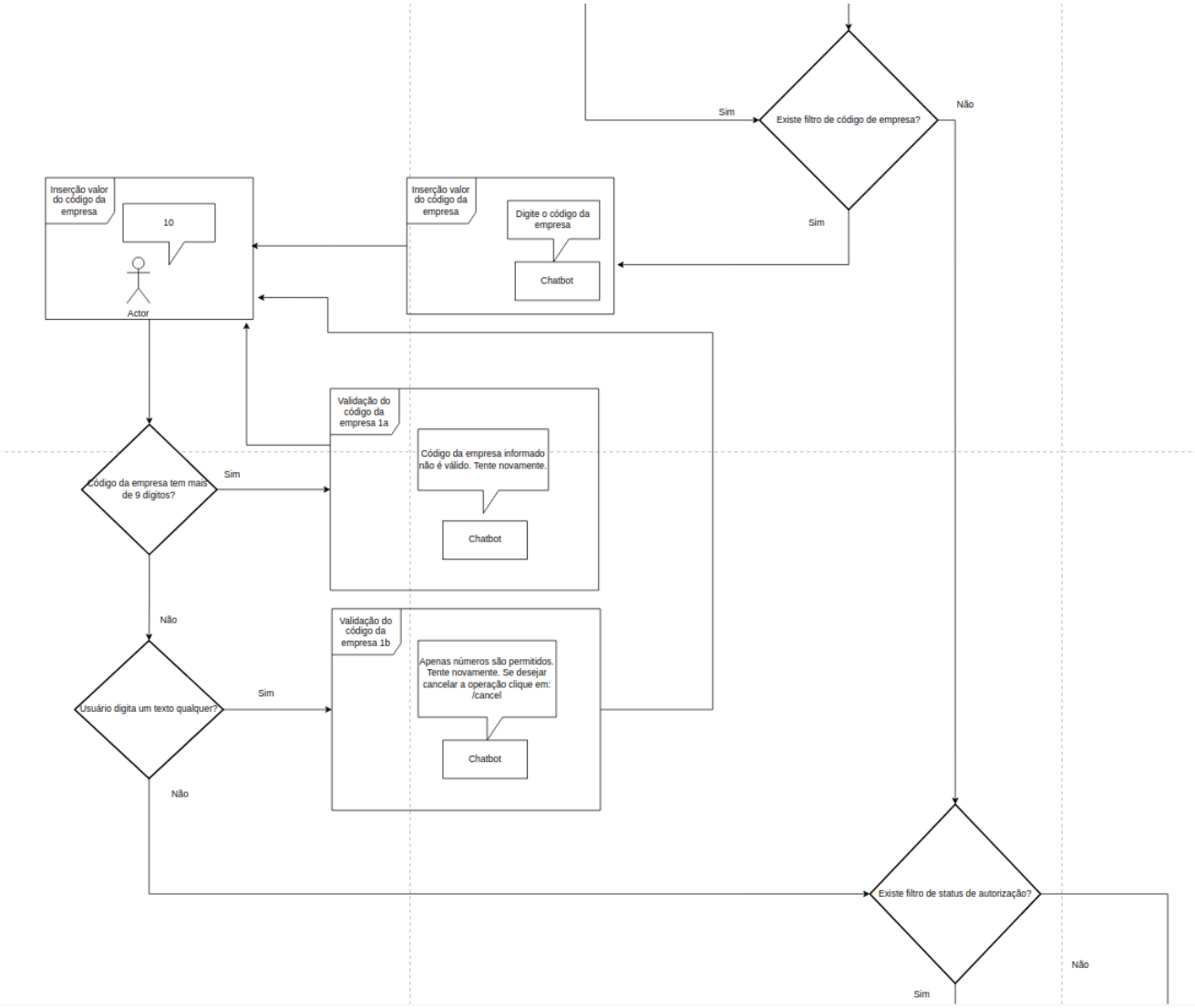
Fonte: Autor.

Figura 67 – Validação de período de vencimento quando este filtro for selecionado



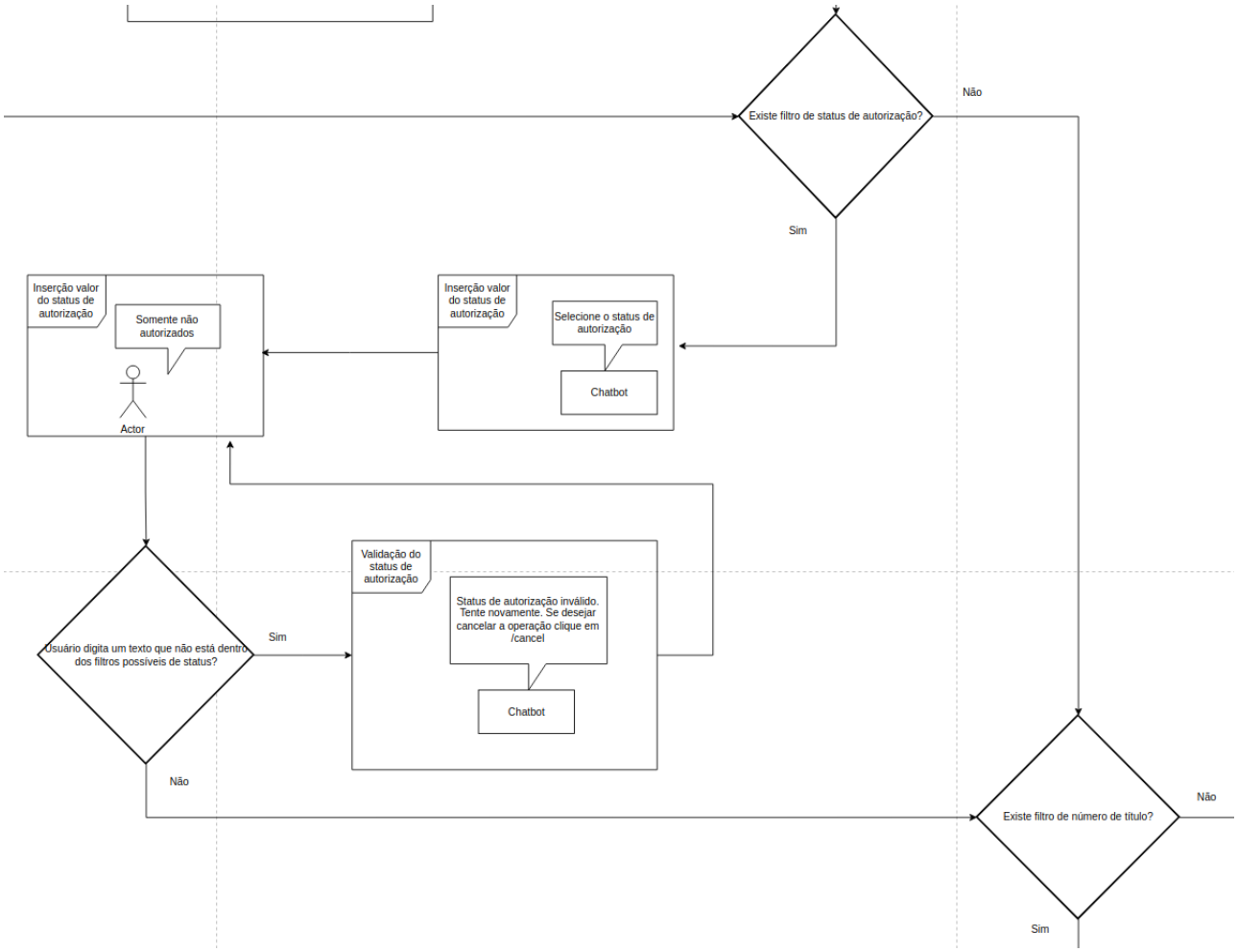
Fonte: Autor.

Figura 68 – Validação do código da empresa quando este filtro for selecionado



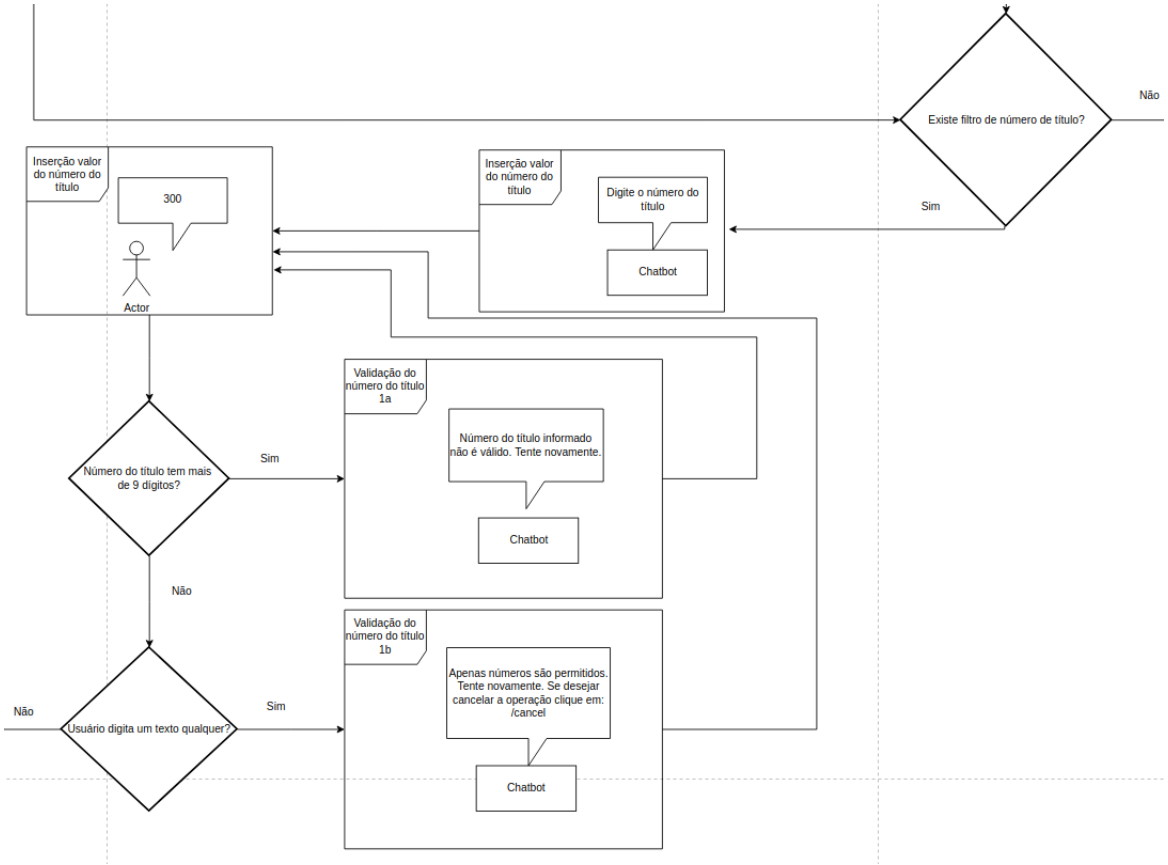
Fonte: Autor.

Figura 69 – Validação do status de autorização quando este filtro for selecionado



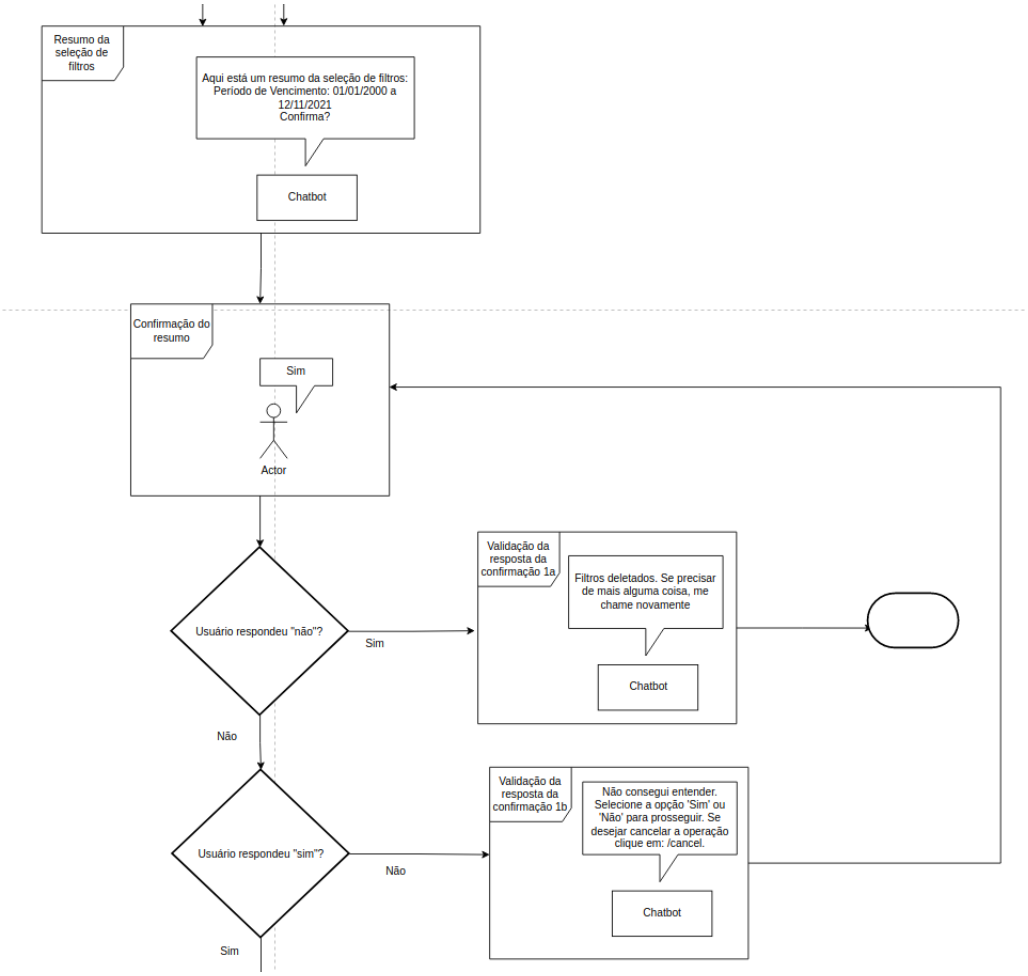
Fonte: Autor.

Figura 70 – Validação do número do título quando este filtro for selecionado



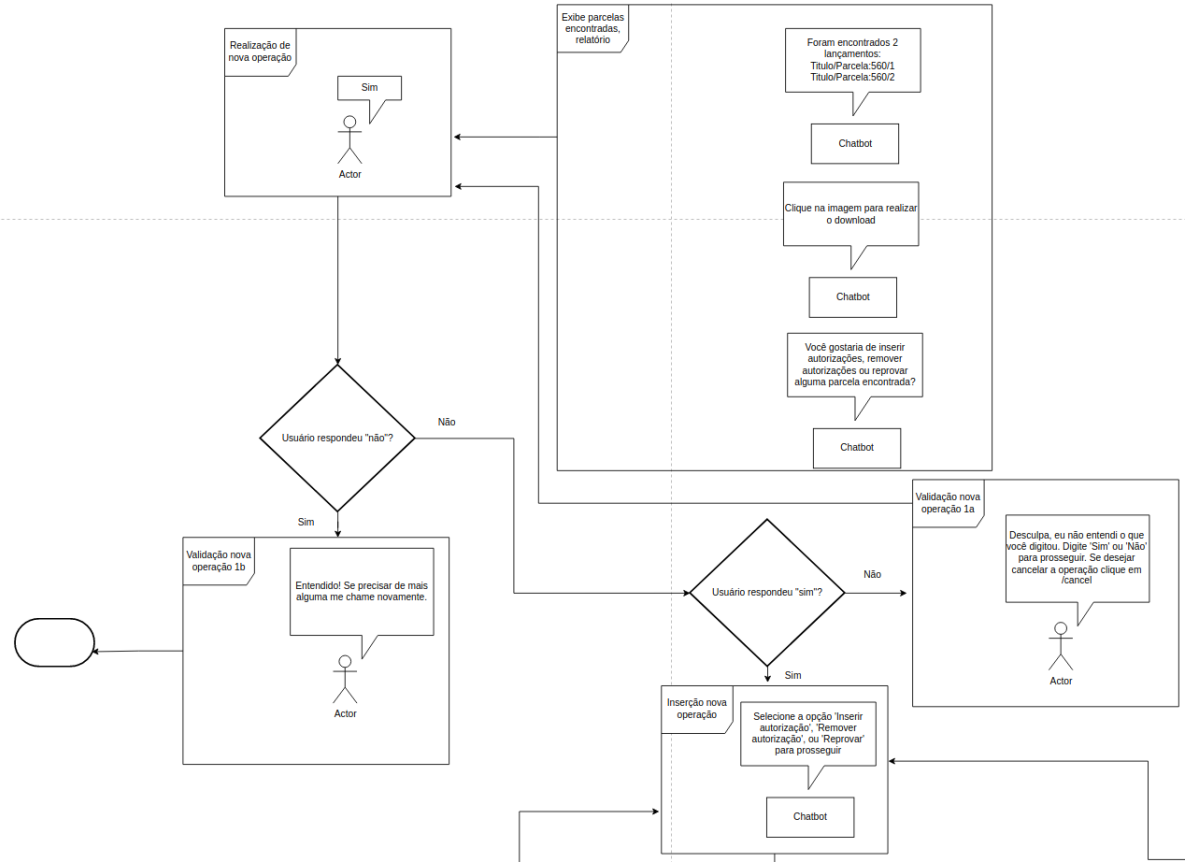
Fonte: Autor.

Figura 71 – Resumo e confirmação da inserção dos filtros.



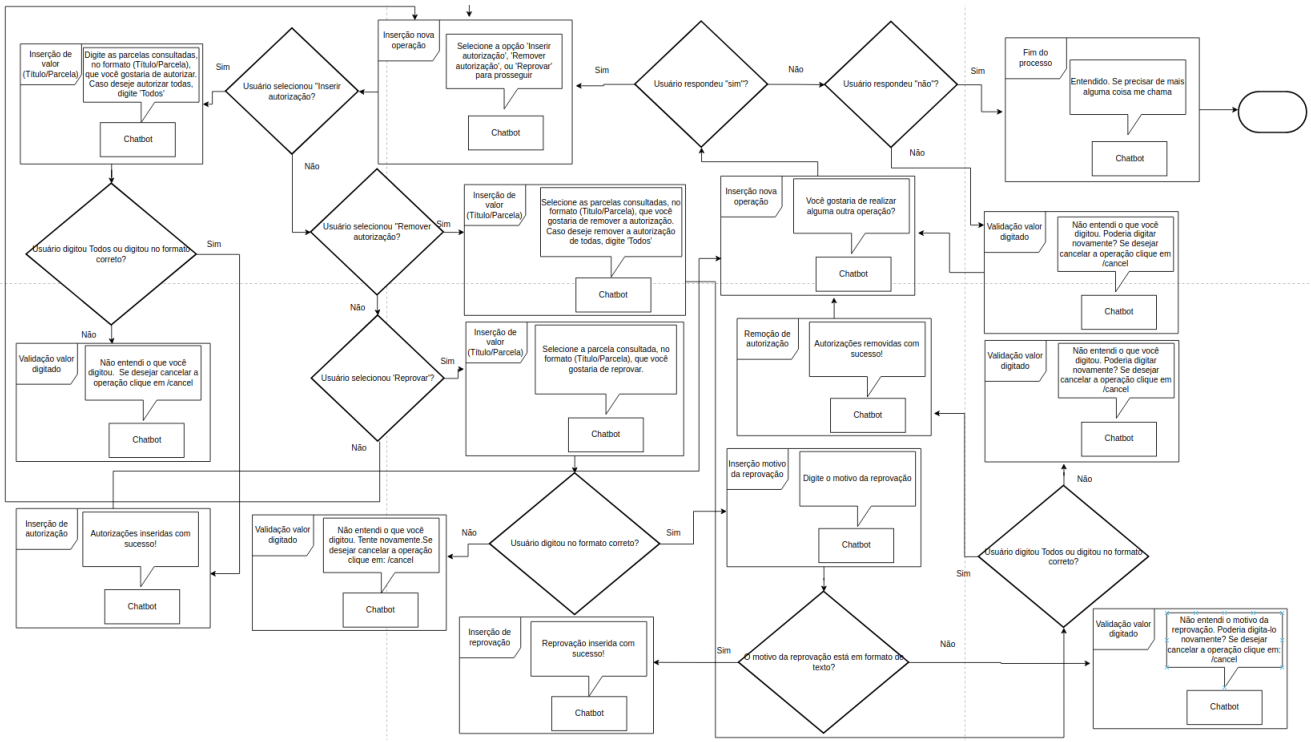
Fonte: Autor.

Figura 72 – Exibição das parcelas encontradas e relatório.



Fonte: Autor.

Figura 73 – Operações de inserção de autorização, remoção de autorização e reprovação de parcela



Fonte: Autor.