IKARO MONTANARI

ACADEMIC CONTROL

VILHENA
2021

IKARO MONTANARI

Academic Control

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia, *campus* Vilhena, como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Gilberto Pereira da Silva

VILHENA
2021

IKARO MONTANARI

ACADEMIC CONTROL

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia, *campus* Vilhena, como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Aprovado em dezembro de 2021.

BANCA EXAMINADORA

gov.br Documento assinado digitalmente
GILBERTO PEREIRA DA SILVA
Data: 06/05/2022 11:49:14-0300
Verifique em https://verificador.iti.br

Prof. Esp. Gilberto Pereira da Silva – Orientador, IFRO

gov.br Documento assinado digitalmente
Juliano Fischer Naves
Data: 18/05/2022 16:49:18-0300
Verifique em https://verificador.iti.br

Prof. Dr. Juliano Fischer Naves – IFRO

gov.br Documento assinado digitalmente
ROBERTO SIMPLICIO GUIMARAES
Data: 23/05/2022 09:50:11-0300
Verifique em https://verificador.iti.br

Prof. Me. Roberto Simplício Guimarães – IFRO

VILHENA
2021

FICHA CATALOGRÁFICA

Biblioteca IFRO – Campus Vilhena

M264d

MONTANARI, Ikaro Bruno da Mata
Academic control /Ikaro Bruno da Mata Montanari – Vilhena, Rondônia, 2021.

54f. ; il.
Orientador Prof. Esp. Gilberto Pereira da Silva

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

1. Academic control 2. API 3.Front End 4. Associação I. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO II. Título

005.133

Bibliotecária responsável Rosilene Maria do Couto Marques CRB 11/321

Dedicatória

In memoriam a Jefferson Bruno Souza

Agradecimentos

Grato ao Senhor Jesus por me manter firme, pois sempre esteve comigo mediante a tantos problemas que tive durante os três anos de decorrência do curso.

Agradeço ao meu Professor Marco Antônio Augusto de Andrade que não somente me ensinou a amar a minha profissão, mas colaborou para trilhar o caminho do conhecimento e em momentos que queria abrir mão, o mesmo se mostrou importante não somente com seu cago, mas com seus alunos e me deu a força que faltava para prosseguir.

Agradeço ao meu professor orientador Gilberto Pereira da Silva que me apoiou em todos os sentidos desse projeto, desde a idealização e durante o processo de produção não mediu esforços para me atender e me explicar alguns conceitos importantes.

Agradeço a todos meus professores que me apoiaram nessa caminhada, com certeza tive o melhor corpo docente que podia esperar para minha formação, foram todos meus colegas de classe e hoje muitos são meus amigos. Sempre me ajudaram a permanecer no curso e a compartilhar seus conhecimentos quando foi necessário, aqui deixo gravado os nomes de Vitor, Evaldo, Wilian, Gabriel e Alisson. Quando achei que não fosse conseguir, eles me apoiaram para terminar cada semestre e se estou concluindo este curso, devo muito a eles.

Agradeço aos ensinamentos de meu pai e toda sua determinação em me ensinar o caráter da vida, e a minha doce mãe que me ensinou todo o respeito e educação que tenho, sem a base que eles me proporcionaram eu não seria ninguém.

Agradeço meu amigo Marcos Roberto, que através de suas palavras me impulsionou para a execução deste trabalho e na persistência para prosseguir.

Por último, sou grato a todos que de maneiras diferentes contribuíram para o profissional que me tornei hoje.

Resumo

O *Academic Control* é um sistema que visa trazer um aprimoramento na gestão de contratos e emissão de boletos da associação de acadêmicos de Chupinguaia através de um sistema que registra solicitações de novos entrantes, assim como aprova a entrada dos mesmos.

Em um breve resumo a associação foi fundada em 2014 com o intuito de transportar estudantes que residiam em Chupinguaia até a cidade de Vilhena onde ficavam suas respectivas universidades. Ao longo dos anos foi se tornando cada vez mais comum estudantes que prefeririam residir em sua cidade, porém não abriam mão de ter um ensino superior em algumas das faculdades em Vilhena.

Entretanto, levava dias para registrar toda documentação necessária e gerar um contrato para assinatura dos associados, bem como a emissão de títulos bancários para eles, porém a proposta do *Academic Control* é que toda parte contratual e emissão de boletos possa ser feita de forma remota sem a necessidade da presença física do associado.

Com a aplicação desse conceito surge o desafio de gerir toda essa estrutura, e como já mencionado, sem a necessidade da presença de nenhuma das partes. Tal adversidade não exigiria menos das habilidades de um profissional capacitado para cumpri-la.

No desenvolvimento do software utilizou-se as metodologias de *Srum* e *Kaban* para a melhor execução dos processos e através dessa gestão mais aprimorada foi possível desenvolver as duas partes do projeto, Api e *front-end*. O ciclo de vida foi baseado em três etapas: iniciação, execução e finalização, esse escopo foi seguido desde os levantamentos dos requisitos junto à associação até o processo de finalização.

O projeto irá contribuir para uma melhor gestão organizacional da associação e das regras de negócio da mesma, além de explorar um bom nicho de mercado.

Palavras-chave: *Academic Control,* Api, *Front-end*, Associação

Abstract

The academic control is a system that aims to bring a different experience to students of the then Chupinguaia academic association, through a system that registers requests for new entrants as well as approves their entry, having a contract with the institution has never been so simple.

The association was founded in 2014 with the aim of transporting students who lived in Chupinguaia to the city of Vilhena where their respective universities were located, and over the years it became increasingly common to see students who would prefer to reside in their city, but did not give up on having a higher education in one of the renowned faculties in Vilhena.

In turn, where it took days to register all the necessary documentation and generate a contract for the members to sign as well as the issuance of bank certificates for them, everything can now be done remotely without the need for the physical presence of the associate. He will have access to your page and your bills that can be printed or merely paid through the typed line in bank apps.

Applying this concept arises the challenge of managing this entire structure and as already mentioned without the need for the presence of any of the parties, such a challenge would require no less than the skills of a qualified professional to fulfill it.

In the development of the software, the Srum and Kaban methodologies were used for the best execution of the processes that, through this improved management, it was possible to develop the two parts of this project, Api and Front-end. The lifecycle was based on three stages: initiation, execution and completion, this scope was followed from the requirements gathering with the association to the completion process.

The project will contribute to a better management of academic transport and the association's business rules, in addition to exploring an underexplored market niche.

Keywords: Academic Control, Api, Front-end, Association.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API: *Application Programming Interface*

JSON: J*avaScript Object Notation*

NPM: *Node Package Manager*

FRONT-END: *Front-end* projeta e constroem os elementos da experiência do usuário na página *web* ou aplicativo, incluindo botões, menus, páginas, links, gráficos e muito mais.

Sumário

1.Introdução

1.1 Contexto e problema

O *Academic Control* (Controle de acadêmicos) é um *software* que permite que cidadãos interessados em se tornarem associados da associação de acadêmicos de Chupinguaia possam fazer isso online, usando um computador ou celular com acesso a internet.

A associação de acadêmicos de Chupinguaia foi criada em meados de 2014 com o objetivo de transportar estudantes para as faculdades na cidade de Vilhena - Rondônia em virtude da grande maioria não poder mudar sua residência para a cidade a fim de cursar um ensino superior. Com o passar dos anos, mais pessoas aderiram à ideia de não ter que sair definitivamente da cidade de Chupinguaia para cursar a graduação, o que resultou em um aumento de pessoas interessadas em fazer uso do serviço de transporte pela associação de acadêmicos.

Porém, a forma de ingresso é manual, custosa e em determinados momentos, demorada. Todo o trabalho realizado até então tornou-se moroso, devido ao processo de documentação, desde o envio até a emissão dos boletos do associado, ser feito de forma manual.

Em virtude desse incremento da demanda, os processos tomam muito tempo da administração e do novo associado. Se tal situação já não fosse o suficiente, há entre os associados, estudantes residentes de distritos próximos à cidade de Chupinguaia, o que resulta no problema do processo documental manual e presencial, pois obriga que esses acadêmicos dirijam-se à cidade de Chupinguaia para realizarem seus credenciamentos.

Aqui fica a pergunta: Existe uma forma de automatizar tudo isso sem a necessidade de sair de casa ou de ter pessoas exclusivas para isso? — A estratégia por parte desse projeto é responder esse questionamento e levar a tecnologia e sua facilidade a todos os envolvidos, desde a administração da associação até os associados.

O software é escrito em linguagem Javascript e desenvolvido com dois *frameworks* Node.js na parte da Api e Vue.js no *front-end*, e utiliza na parte do servidor a arquitetura de banco de dados relacional SQL. No quesito segurança usa-se a autenticação de criptografia de senhas com o objetivo de garantir maior acerto na proteção de dados.

O sistema já está disponível na internet e consegue realizar todas as funções dispostas, tais como o cadastro do associado até a emissão de boletos. Possui uma utilização simples para facilitar a usabilidade do usuário final e esta monografia vai discorrer sobre a problemática: - Como criar o *Academic Control*?

1.2 Objetivos

1.2.1 Objetivo geral

Criar um sistema de gerenciamento para uma associação de acadêmicos que fornece transporte intermunicipal como forma de mantimento.

1.2.2 Objetivos específicos

- Compreender mecanismos de implementação adequado;
- Definir requisitos para a boa elaboração dos artefatos e execução da codificação;
- Implementar API e o *FRONT-END* da aplicação;

1.3 Justificativa

A adoção de processos mais eficientes para realizar tarefas está instaurada desde os primórdios da humanidade, não aceitar a evolução ou talvez ter aversão a ela talvez possa ser um fator definitivo para a continuidade de um negócio. Segundo Jeff Sutherland (2014), ficar preso em seu antigo modo de se realizar as tarefas, e enrijecer a forma de controlar os processos, resultará em fracasso.

Fugir desse pensamento pode ser considerado regressão e desperdício de tempo, pois eliminar o desperdício deve ser um dos focos principais quando nos referimos a negócios e projetos de acordo com Taiichi Ohno (1978). Pensando nisso, propomos uma forma mais fácil de se contratar o serviço de transporte prestado pela associação de acadêmicos e até mesmo colaboramos para uma simplificação dos processos.

2. Fundamentação teórica

Neste capítulo iremos dissertar sobre construção da Api, armazenamento de imagens em nuvem, criptografia e autenticação, estado da aplicação e componentização, tais conceitos são fundamentais para o entendimento do projeto.

2.0.1 Construção da API

O *Academic Control* usa uma base para a API *REST,* utilizando o *framework* Node.js que nos permite aprimoramento e utilização de outras tecnologias fundamentais para o projeto. Fielding (2000) foi o primeiro a citar *Representational State Transfer* (REST) em sua tese de doutorado, com a apresentação dos desafios e problemas de escalabilidade na infraestrutura da *Web* Fielding, propôs arquiteturas de *software* para criação de interfaces com maior escalabilidade.

Graças a ferramenta Node.js a comunicação com o banco de dados relacional que foi usado se tornou mais aprimorada. Para o banco de dados da aplicação foi usado o *PostgreSQL* que é gratuito e tem todas as características de um banco SQL (*Structured Query Language*) e como consequência dessa escolha nos foi permitido utilizar a ferramenta *Sequelize* para realizar as nossas criações de tabelas e respectivas consultas.

Falando ainda sobre Node.js já compreendemos que sua utilização é uma vasta na comunidade de desenvolvedores e a escolha da tecnologia se dá muito a esses fatores, segundo uma pesquisa do site especializado Stack Overflow (2019) os resultados mostraram que cerca de 49% dos usuários entrevistados utilizam Node.js para construção de APIs contra 37% da .NET e 23% da .NET Core. Em paralelo a essa pesquisa 39% usam PostgreSQL para banco de dados ficando atrás somente do MySQL com 54%, com base nisso temos a dimensão do quanto estas tecnologias são utilizadas em vários projetos..

Além do Node.js como base e PostgreSQL, usamos a ferramenta chamada *Express,* que permitiu a utilização das funções de requisição e resposta via HTTP (*Hyper Text Transfer Protocol*).

Na construção da Api foram adotadas boas práticas que envolvem validação e criptografia de informações, funções fundamentais em um sistema robusto e protegido de falhas. Para a linguagem Javascript existem muitas bibliotecas que permitem automatizar parte dessas ações. Para validação de informações e tipos utilizamos da biblioteca chamada *Yup* que funciona como uma espécie de camada executada antes das demais funções de cada

módulo que exigem validação de tipos e quantidade, impedindo assim que o usuário coloque um dado errado em algum local em tempo de execução.

No processo de criação da Api foi notado que as funções básicas CRUD (*Create, Read, Update, Delete*) foram bem-criadas e validadas, porém foi notado que algumas funções eram extremamente importantes quando começamos a desenvolver o *front-end*, como a criação de uma listagem por contratos pendentes de aprovação por exemplo, essa função em específico exigiu que tivéssemos buscas em duas tabelas no banco de dados onde é retornado o contrato e o usuário pertencente aquele contrato, essa busca só foi capaz graças às chaves estrangeiras contidas dentro da tabela contrato. Portanto com esse conjunto de operações na Api com o serviço *REST*, permitidos graças ao protocolo HTTP, o acesso pode ser feito por qualquer serviço online de consumo, como descrito por Massé (2012).

2.0.2 Armazenamento de imagens em nuvem.

Porque armazenar em nuvem? — A primeira questão é conceituar o que é armazenamento em nuvem que, segundo Veras (2012), prolonga-se em um conjunto de ferramentas virtuais utilizados de maneira fácil e acessíveis, tais como *hardware*, *software*, plataformas de desenvolvimento e serviços. Ok, agora que conceituamos podemos responder a pergunta de maneira simples, armazenar em nuvem traz mais desempenho e facilidade para a aplicação pois retiramos de nossa API toda a responsabilidade de gestão e armazenamento.

Onde conseguimos armazenar tais imagens? — Utilizamos o Amazon S3 ou *Amazon Simple Storage Service* onde podemos ter um *Buffer* (área de armazenamento temporário de dados durante sua transferência entre dispositivos de diferentes taxas de transferência) para hospedar nossas imagens. Essa ação apoia na inclusão, busca e consumo em toda a aplicação de forma fácil, prática e simples.

2.0.3 Criptografia e Autenticação.

Em qualquer sistema de informação surge a necessidade de segurança das informações, principalmente quando a própria informação é um ativo para a organização ou negócios. Segundo Nakamura e Geus (2002), a informação na condição de ativo, necessita ser protegida. Nesse ínterim, pensar nos requisitos de segurança das informações, confiabilidade, integridade e autenticidade é fundamental para entendermos a utilização de criptografia e autenticação. Por esse motivo, temos a obrigação de garantir que as senhas do nosso usuário não sejam vazadas, e se forem, que não possam ser decifradas, pois segundo Oliveira (2001),

a segurança da informação é o processo de proteção tanto de informações quanto de ativos digitais, sejam eles armazenados em computadores ou em redes de processamento de dados.

Para o projeto utilizamos uma biblioteca chamada *bcrypt* que nos permite criptografar nossas senhas na quantidade de *bits* desejada, sendo assim a criptografia torna-se assertiva. É importante destacar que o processo criptográfico ocorre antes da inclusão da senha cadastrada no banco de dados, fazendo com que a senha salva esteja criptografada, o que por sua vez nos faz comparar as senhas já criptografadas durante o processo de *login*, por exemplo, criptografando a senha informada pelo usuário e posteriormente comparando as duas senhas, garantindo que em nenhum momento da aplicação a "senha real" por assim dizer seja informada.

Além de promover a criptografia das senhas, precisamos autenticar. Existem várias maneiras de se fazer isso, porém, escolhemos o JWT (*Json Web Token*), que segundo JWT.IO (2020), é um padrão aberto que define de uma maneira comprimida e independente um *hash* para transmitir informações com segurança entre as partes como um objeto *JSON*, e nos permite determinar o tempo de expiração, extrair informações de acesso como ID do usuário logado, o que torna a aplicação mais segura já que esse ID pode ser usado em outras funções de consulta de informações.

Por último, para executarmos as funções destinadas aos boletos precisamos integrar nossa aplicação com a API de algum banco, o escolhido para esse projeto foi o Sicoob que em sua autenticação utiliza a mesma tecnologia usada pelo Google, Facebook, entre outras empresas, nos referimos da autenticação *oAuth2.0*, que segundo sua documentação oficial OAUTH 2.0 (2021) se trata de um protocolo que fornece fluxos específicos para aplicações *web* e tem como parâmetro a utilização de *clientID* e *clientSECRET* para a emissão de *Token* para acesso em suas funções, fora os escopos que deverão ser informados. Ou seja, tudo aquilo que desejamos usar da Api do Sicoob devemos informar quando formos obter o *token* de acesso. Uma das características interessantes desse tipo de autenticação é que ela nos permite ter o que chamamos de *Refresh-TOKEN*, essa opção pode ser executada a partir do *token* atual, que ainda é válido, e a partir disso ter um novo token com todas as funções já dispostas inicialmente. Essa vantagem permite que o usuário administrador efetue o *login* com seus dados bancários somente uma vez, visto que, existe a possibilidade de criar uma rotina de tarefas que pode solicitar o *Refresh-TOKEN* a cada momento desejado sem a influência ou comando do usuário, tal função resultará em sempre manter o sistema funcionando e com seus devidos acessos.

2.0.4 Componentização.

Desenvolver aplicações *web* também requer que nosso usuário consiga visualizar tudo aquilo que foi desenvolvido na api, segundo Otto e Thornton (2017) deve ser considerado sua adaptabilidade. Para isso, considera-se a capacidade de adaptação da interface ao usuário e aos diferentes dispositivos.

Nesta perspectiva, temos tudo que o nosso usuário final visualiza que por sua vez adota pelo menos três ferramentas para constituir toda essa arquitetura, dentro das camadas de um *front-end* temos a linguagem de marcação, o esqueleto, nosso *HTML* (*HyperText Markup Language*), segundo MDN WEB DOCS (2021) ele que fica responsável por toda estrutura da nossa página na *web*, porém somente ele pode não ser suficiente para deixar a renderização agradável para a utilização do usuário, por isso temos o *CSS* (*Cascading Style Sheets*) que nada mais é do que segundo o MDN WEB DOCS (2021) uma linguagem de estilo (*en-US*) usada para descrever a apresentação de um documento escrito em *HTML*, é o *CSS* quem coloca as cores, os espaços e todas as composições visuais que agradam o usuário final. Por último, mas não menos importante, temos a nossa linguagem de programação que manipula os dados e efetua as funções como *login* por exemplo..

Além de usarmos *Javascript* na API, usamos também no *front-end* e para alcançar melhores resultados dentro da proposta do sistema, agregamos o *framework* VUE, que segundo sua documentação VUE.3.0 (2021) é uma estrutura progressiva para construção de interfaces voltadas para o usuário, e ao contrário de outras estruturas monolíticas, foi projetado desde o início para ser adotado de forma incremental. Com isso entramos em um tópico importante, a componentização.

O *framework* citado acima foi criado em 2014 e sua arquitetura é baseada em que cada compoente.vue se comporte de maneira isolada, de acordo com VUE.3.0 (2021), esse comportamento nos permite renderizar componentes de várias maneiras diferentes, um bom exemplo seria renderizar na tela o componente com os dados do usuário e caso precise excluir algum dado, chamar outro componente e renderizá-lo de maneira separada, somente isso já traz para o projeto possibilidades variadas, como melhora na performance de roteamento entre componentes e também reutilização de estilos *CSS*. Além disso, temos as diretivas da ferramenta que permite a interceptação de dados em tempo de execução, por exemplo a reatividade que se trata segundo VUE.3.0 REACTIVITY (2021) de um paradigma que permite ajustar mudanças em variáveis de forma declarativa.

2.0.5 Estado da aplicação.

Muitas vezes quando temos uma aplicação *web* é comum termos variáveis para armazenar dados, porém, ao componentizar a aplicação *front-end* não seria interessante criar uma variável em cada componente e a todo momento fazer consultas na API para coletar dados como informações do usuário, nesse cenário temos a opção de usar os estados da nossa aplicação. Os *States,* segundo a documentação VUEX (2021), é um gerenciador de estados e bibliotecas que permite armazenar de forma centralizada informações para serem usadas em todos os componentes.

Essa função nos permite controlar variáveis como *token* ou mesmo dados do usuário logado em tempo de execução da aplicação, isso nos permite ter um dinamismo no momento de executar ações como consultas ao banco de dados e chamadas de funções sem a intervenção do usuário.

Usar o *States* do VUE nos permite manter a sessão com o *token* ativo, pois todos os componentes irão consultar o *State* para validação do *token*. Essa é uma das aplicações dos estados do VUE que além de nos fornecer uma melhor performance para aplicação devido a retirada da responsabilidade do componente temos uma espécie de variável global que pode ser consultada em todos os componentes da aplicação.

2.1 Trabalhos similares

Os seguintes *softwares* proprietários, para o gerenciamento de associação de acadêmicos que foram encontrados:

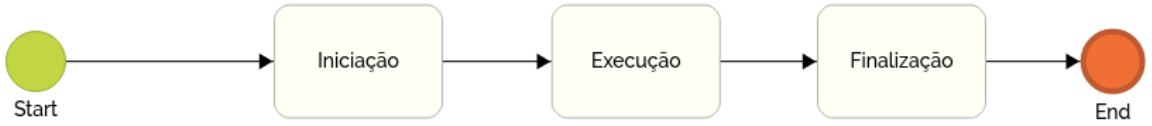
- IScholar – Sistema ischolar: Nele a dinâmica é mais voltada para gestão da universidade em si, como controle de matrículas e pagamentos, aulas, horários de professores e outros fatores.
- Sistema Quality - Quality: Assim como seu anterior a premissa aqui é a mesma, onde existe o controle de aulas, matrículas e mensalidades. Inclusive, uma das empresas que utilizam esse *software* é a Fisk e a faculdade Fael.
- Fatura Simples - ERAMO SOFTWARE LTDA: esse sistema já é voltado para controle de associados, que por sua vez controla tanto a parte de cadastro, quanto de cobrança dos associados em débito. A empresa responsável por ele tem projetos em piloto para novos entrantes e se mostra uma ótima opção quando o quesito é suporte e implantação.

3. Materiais e métodos

3.1. Ciclo de vida e processos.

Iniciar um projeto já codificado pode ser uma tarefa simples, porém iniciar da maneira correta requer estudo e análise por parte do autor do plano, pois cada parte do *software* deve ser pensada com o objetivo de garantir uma boa execução das etapas. Para o projeto em questão, utilizamos as etapas de iniciação, execução, e finalização.

Figura 1 - Ciclo de vida e processos



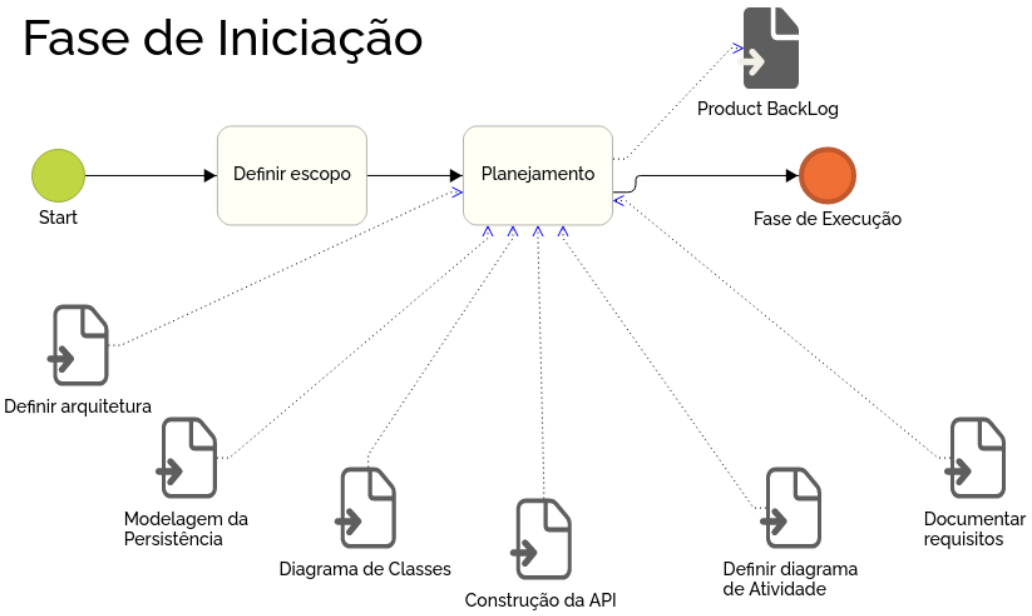
Fonte: Elaborado pelo autor

Em cada etapa foram desenvolvidos artefatos como levantamento de requisitos, diagramação *UML* e documentação da API, que corresponderam na melhor execução do projeto final. Sem as partes necessárias o resultado poderia até ser adquirido, mas aconteceria sem os devidos cuidados com as boas práticas de desenvolvimento e muito menos em tempo hábil.

Etapa de iniciação representado pela Figura 2, as energias foram encontradas na parte de idealização do *software*, nela imaginamos como nosso sistema irá se comportar, com o intuito de garantir um bom escopo para a codificação.

O escopo do *Academic Control* foi desenhado com o objetivo de ser eficiente com a proposta inicial, que é cadastro e gestão de associados. Nesse escopo arquitetamos os seguintes pontos: a arquitetura do *software,* modelo de persistência, documento da API, modelagem *UML* (Diagrama de Classe e Atividade).

Figura 2 - Fase de Iniciação



Fonte: Elaborado pelo autor

Fase de execução, na etapa da execução do *Academic Control* a metodologia usada para percorrer todo esse trajeto foi o *Scrum*, por ser uma metodologia ágil, permitiu ter mais organização no tempo de execução das tarefas, separando cada etapa da codificação em *sprints* com intervalos de duas semanas, que por sua vez se mostraram eficientes no cumprimento dos prazos

Para entendimento da metodologia *Scrum,* foi elaborado um breve conceito sobre seu funcionamento de acordo com Carvalho et al (2012):

- *Sprint:* Se trata do ciclo de desenvolvimento, geralmente pode possuir a duração de 2 a 4 semanas.
- *Product Backlog:* São os requisitos do sistema que são debatidos entre os membros da equipe a fim de chegarem a um divisor comum, são atualizados a cada *Sprint*, sem mensurar que essa parte deve estar acessível a todos os membros participantes.
- *Scrum Team*: Trata-se da equipe responsável por idealizar o *Product backlog* e por sua vez efetuar a codificação e implementação das funções em suas devidas *sprints*.

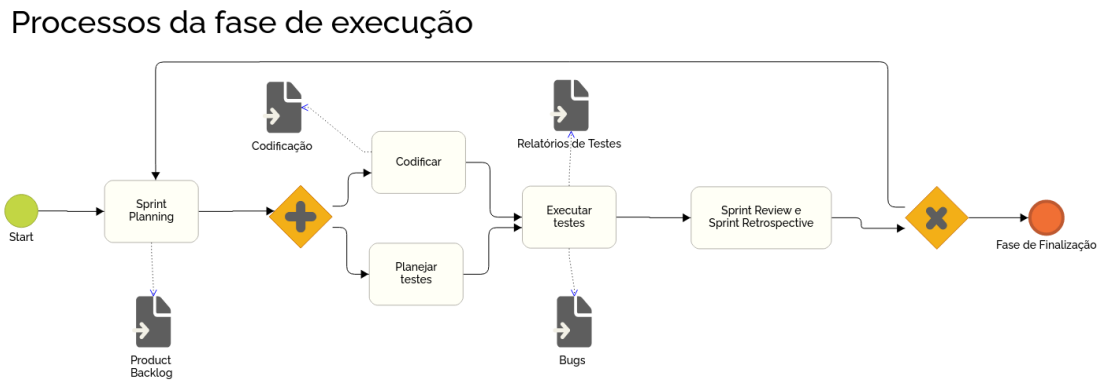
- *Sprint Planning*: Nessa etapa está concentrada a reunião para debater os assuntos referentes ao *Product Backlog* e quais requisitos serão implementados em um determinado ciclo do projeto.
- *Scrum Master:* Esse participante é o que trabalha ao lado da equipe, e se revela quando solicitado, aqui sua prioridade máxima é apoiar na resolução dos problemas que possam impedir o resultado de ser adquirido.
- *Product Owner:* Esse integrante é responsável por apoiar a equipe nas regras de negócio do sistema, ele fica responsável pela interação para com o cliente, e compreende as necessidades do cliente para definir quais são suas prioridades para com o sistema elaborado.
- *Daily Scrum*: Se trata de uma reunião diária com objetivo de realizar alinhamento com os membros do *Scrum Team*, nela são discutidas as etapas que foram concluídas no dia anterior, agrega mais valor nas atividades da equipe, e minimiza erros que podem levar a falhas na entrega dos prazos.
- *Sprint Review e Sprint Retrospective*: Aqui requer que todos os membros da equipe participem para poder revisar tudo que foi feito no final de uma *Sprint* e avaliar possíveis melhorias para com as funções implementadas.
- *Story Points:* Pontuação que quantifica a complexidade de uma tarefa ou requisito apresentado.

Scrum de acordo com Schwaber e Sutherland (2013) é uma ferramenta que nos permite resolver problemas complexos de forma adaptativa. O *scrum* atua de maneira leve e simples de se compreender em seu contexto geral, pois em sua aplicação requererá mudanças de comportamento. Segundo Jeff Sutherland (2014) com menos pessoas e em menos tempo, consegue apresentar resultados melhores e com maior qualidade que metodologias como a de cascata, que nessa visão de metodologia ágil não se irá simplesmente codificar todo o *software* em uma noite de maneira desordenada.

Conforme a Figura 3, a fase de execução desenvolve-se da seguinte maneira:

O ciclo de codificação começa com a *Sprint Planning* que é uma reunião entre os membros do projeto nos quais o *Product Backlog* é debatido e nessa ocasião parte dele é colocado em desenvolvimento na próxima *Sprint*. Para quantificar a viabilidade de uma *Sprint* é levado em consideração os *Story Points,* que foram levantados na etapa de definição do *Product Backlog.*

Figura 3 - Fase de execução

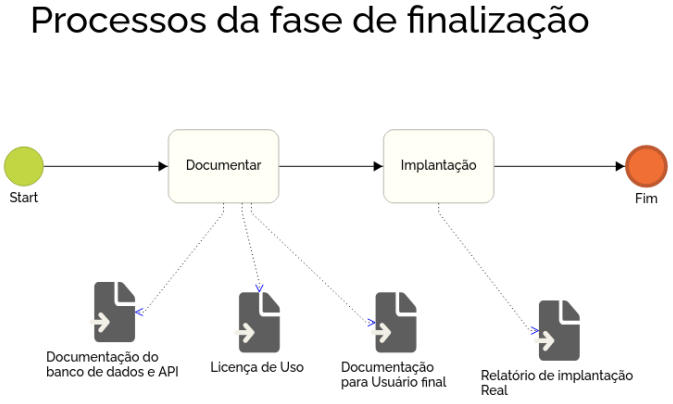


Fonte: Elaborado pelo autor

Realizado o *Sprint Planning*, a etapa de codificação é iniciada em paralelo a execução de testes, e com a emissão dos relatórios é possível identificar *bugs*, partindo disso devem ser criadas *tasks* durante a *Sprint* para resolução do problema. No final da *Sprint* uma reunião é realizada, sendo feito uma retrospectiva e o alcance dos objetivos são novamente discutidos.

Etapa de finalização, Figura 4, os requisitos desenvolvidos na etapa de execução, por fim são documentados. Uma licença é constituída pelo produto e um relatório de implantação real é gerado, após isso o projeto é concluído. Nessa etapa é fundamental a entrega do produto, sem ela não é possível quantificar o resultado real e demonstrar a qualidade do sistema.

Figura 4 - Etapa de Finalização



Fonte: Elaborado pelo autor

Em análise geral o *Academic Control* dividiu seus esforços tanto na aplicação da metodologia utilizada quanto na própria execução das *Sprints* criadas ao decorrer do projeto. Sendo assim conforme definição acima descrita foram realizadas várias reuniões juntos aos professores orientadores que vez desempenharam seu papel mostrando como poderia ser realizado as etapas, utilização meios como Google Meet e a ferramenta do WhatsApp para comunicação remota foram fundamentais, visto que tivemos um período pandêmico em virtude da *covid-19*. A plataforma Taiga que foi eficiente no quesito organizacional o que contribuiu para o planejamento e codificação do sistema.

3.2 - Ferramentas e tecnologias utilizadas

Neste subcapítulo serão descritos as tecnologias utilizadas no projeto.

3.2.1 Node Js

O Node[1] é um *framework* desenvolvido em *Javascript* em meados de 2009, ele foi desenvolvido baseado no interpretador V8 do google que permite a execução do já mencionado *Javascript* fora de um navegador da *web* (NODEJS, 2020). Nele temos o icônico gerenciador de pacotes *NPM* (*node package manager*) onde tem o propósito de executar o código armazenado num *package node.js* instalado na base global do software ou localmente se julgado necessário.

Uma de suas características principais é a execução das requisições/eventos https em *single-thread*, o que permite que apenas uma chamada *Event Loop* seja responsável por executar o código *Javascript* sem a necessidade da criação de uma nova *thread* que por sua vez utilizaria mais recursos computacionais.

3.2.2 PostgreSQL

PostgreSQL[2] Trata-se de um SGBD (sistema de gerenciamento de banco de dados) desenvolvido em código aberto que segundo a documentação POSTGRESQL (2021) possui recursos avançados como consultas complexas, integridade transacional, controle de ocorrências multi-versão, entre outras.

Uma de suas grandes vantagens é sua integração com a ferramenta *Sequelize* que permite a criação de comandos SQL na linguagem *Javascript*, facilitando dessa forma a criação de tablas e a execução de consultas e inserções no banco de dados

[1] *Disponível em https://nodejs.org/en/*
[2] *Disponível em https://www.postgresql.org/*

3.2.3 Vue

Trata-se de um *framework* também em *Javascript* que atua na parte do cliente (*front-end*) que de acordo com a documentação VUE.3.0 (2021) é uma estrutura gradual para construção de interfaces de usuário que foi projetado desde o início para ser adotado de forma ampliada. Diferente do *Node.js* que atua na parte da Api, aqui a função do *vue* é a componentização do nosso *front-end* e reatividade de tais componentes.

Atualmente, o Vue.js[3] conta com mais de 15% da utilização de estruturas *web* a frente de tecnologias como Laravel[4] (10%) e Django[5] (12%) de acordo com resultados da pesquisa do desenvolvedor do site Stack Overflow (2019). Uma das suas principais características é a facilidade no aprendizado pois como descrito em sua documentação VUE.3.0 (2021) o *vue* atua com uma espécie de encapsulamento dos elementos *html* para reutilização, em alto nível os componentes *vue* são elementos personalizados pelos quais o compilador associa determinados comportamentos.

3.2.4 Heroku

O Heroku[6] É uma plataforma como serviço em nuvem baseada em contêiner. Os desenvolvedores usam a ferramenta para implantar, gerenciar e dimensionar aplicações, em determinadas ações segundo o site oficial HEROKU (2021) de maneira gratuita.

A plataforma está em desenvolvimento desde junho de 2007 quando era apenas compatível com a linguagem *Ruby*, hoje suporta tecnologias como *Node JS, Java, Scala*, *Python*.

3.2.5 JavaScript (JS)

O *Javascript* é uma linguagem de programação estruturada, segundo MDN (2021) em alto nível com uma tipagem dinâmica, fraca e multi paradigma que suporta orientação a objetos sejam imperativos ou declarativos.

No cenário atual é a principal linguagem para programação de *client-side* em navegadores *web* tomando 67% da parcela do mercado mundial com base no resultados da pesquisa do desenvolvedor do site Stack Overflow (2019), a frente de tecnologias como Python (41%) e PHP (26%), sem mensurar que é umas das três tecnologias mais usadas para desenvolvimento voltados para a *world wide web*.

---

[3] *Disponível em https://vuejs.org/*
[4] *Disponível em https:https://laravel.com/*
[5] *Disponível em https://www.djangoproject.com/*
[6] *Disponível em https://dashboard.heroku.com/*

3.2.6 Insomnia

Aplicações *back-end* tem por padrão apresentar informação ao cliente, pensando nisso temos o *insomnia*[7] que por sua vez está descrito em sua página oficial INSOMNIA (2021) como um console que permite interação e projeção de API's com requisições tipo HTTP (*Hypertext Transfer Protocol*) com uma interface fácil e agradável.

Aqui a possibilidade de teste e diversas API 's é feita de maneira simples de prática, sendo totalmente direto ao ponto, uma de suas vantagens é a opção de exportação de documentação das rotas inseridas.

3.2.7 Visual Studio Code

Visual Studio Code[8] É uma ferramenta de código aberto para criação e edição de códigos, desenvolvida pela Microsoft, ela se encontra disponível para *Windows*, *Linux* e *MacOS*.

Por sua vez a proposta é literalmente codificar, nela além da edição ocorrer de maneira simples e minimalista conta também com vários *plugins* que ajudam no trabalho durante a escrita do código.

Além disso, possui segundo sua página oficial MICROSOFT (2021) suporte às mais variadas linguagens como *Python* e *JavaScript*, possui também uma comunidade ativa e uma loja de extensões/*plugins* bem consistentes. Um bom exemplo de *plugins* é o *TabNine* que atua com um autocomplete que aprende suas formas e padrões de escrita, ou até mesmo o *ESlint* que permite configurar uma forma de escrita e acusa erros quando a escrita foge de tal padrão definido.

3.2.8 Astah

O *Astah*[9] É uma empresa que desenvolve editores de diagramas *UML* que permitem a visualização do design de *software* (ASTAH, 2021). Sua proposta é fornecer uma ferramenta fácil, prática e colaborativa, infelizmente sua versão *community* foi descontinuada em 2018.

3.2.9 Swagger.io

O *Swagger*[10] de caracteriza como ferramenta para descrição de API (*Application Programming Interface*) rest express utilizando *JSON* como base. O *Swagger* por sua vez

---

[7] *Disponível em https://insomnia.rest/*
[8] *Disponível em https://code.visualstudio.com/*
[9] *Disponível em https://astah.net/*
[10] *Disponível em https://swagger.io/*

inclui documentação automatizada, geração de código e geração de casos de testes quando necessário (SWAGGER, 2021).

A ferramenta foi criada em 2011 por Tony Tam, tudo isso mediante a uma necessidade de automação da documentação da Api quando desenvolvia produtos do *Wordnik*. Com isso Tam projetou uma documentação simples a base de *JSON* que poderia ser facilmente manipulada e flexível.

3.2.10 UML

A modelagem *UML* se trata de uma linguagem-padrão para elaboração de diagramas que por sua vez representam as estruturas de um *software*. A diagramação *UML* pode ser empregada para a visualização, a especificação, a construção e até mesmo na documentação de artefatos que são usados pelo sistema.

A utilização desse recurso permite enxergar como o sistema irá ou deve se comportar, aqui é interessante entender que a diagramação *UML* não é uma metodologia, ou seja ela não lhe diz o que deve ser realizado primeiro, e sim organizar os artefatos a fim de que a sintaxe do projeto fique clara para codificação. Nessa etapa uma boa diagramação e entendimento dos artefatos faz com que as etapas como execução (desenvolvimento) sejam realizadas de maneira mais assertiva.

Ao se deparar com os diagramas e aspectos fornecidos pela *UML*, é comum pensar que isso estaria aumentando os esforços no desenvolvimento de um sistema, entretanto se cada ferramenta for utilizado acertadamente com objetivos claros, bem possível que resultará em um valor invisível na qualidade do *software.* (VIEIRA, 2003, p. 10)

No projeto do *Academic Control*, utilizamos os diagramas de classe para a representatividade das classes que compõem o sistema e o diagrama de sequência que por sua vez representa o comportamento do *software* e tempo de execução de suas funções.

3.2.11 Taiga

A ferramenta *Taiga*[11] foi utilizada para o gerenciamento de todo o projeto, ela por sua vez organiza as estruturas das etapas a fim de garantir que a execução não somente da codificação ocorra de maneira bem-sucedida. Aqui a missão mais primordial é converter todas nossas literaturas em formas bem organizadas de desenvolvimento que sejam simples de compreensão e possam agilizar no quesito prazos (TAIGA, 2021).

---

[11] *Disponível em https://projetos.fslab.dev/projects/*

3.2.12 - Git e GitHub

Ao realizar a primeira linha de código uma nova demanda surge, a necessidade de versionamento de código, pois assim como uma empresa tem *backup* de seus dados um bom desenvolvedor deve guardar o código e também separar suas funções com o objetivo de nunca comprometer seu projeto principal. Ao trabalhar em equipe podem surgir conflitos entre métodos ou classes implementados, nesse cenário o git possibilita formas de testes e apoio na resolução de conflitos. (GAMA, 2012)

O GitHub[12], é um dos principais sites de hospedagem de código-fonte, lá é qualquer usuário que se cadastre pode contribuir com códigos públicos e privados de maneira ilimitada, sua usabilidade remota permite que ele seja usado principalmente para divulgação de trabalhos de vários programadores ao redor do mundo. (GITHUB, 2021).

3.2.13 - Ferramentas de comunicação/reunião.

Por decorrência a pandemia do *covid-19,* encontros presenciais para alinhamento junto aos professores coordenadores e orientadores, sendo assim procuramos meios alternativos de comunicação:

- Rocket Chat[13] É uma aplicação para mensagens instantâneas de código aberto que foi por sua vez administrada e fornecida pelo Instituto Federal de Rondônia - IFRO, auxiliou na execução da *Daily Scrum*, compartilhamento de arquivos e mensagens de texto.
- Google Meet[14] Ferramenta do Google para reuniões em formato de Web Conferência, foi utilizada nas execuções das *Sprint Planning, Sprint Review e Sprint Retrospective* e em reuniões formais para alinhamento da equipe.
- WhatsApp[15] A aplicação mantida pelo facebook forneceu contato rápido, prático e seguro de maneira informal.

3.3 - Requisitos

Em um projeto dessa magnitude é indispensável a fase de levantamento de requisitos. Basicamente esse ponto é o que dá o *start* inicial em qualquer projeto, o simples fato de analisar o caso e abstrair o que seria implantado através da elicitação dos requisitos torna a complexibilidade do problema ligeiramente menor, pois com os requisitos dispostos e

[12] *Disponível em https://github.com/*
[13] *Disponível em https://pt-br.rocket.chat/*
[14] *Disponível em https://meet.google.com/*
[15] *Disponível em https://www.whatsapp.com/*

montados temos a dimensão do que será realizado e a partir desse ponto conseguimos sintetizar o restante dos artefatos, para Pompilho (1995) algumas das razões para o baixo índice de satisfação dos usuários está na fase de levantamento de requisitos.

- Etapa 1: O cliente solicitou que a equipe de desenvolvimento se reunisse para pensar em requisitos para um sistema de gerenciamento de sua associação de acadêmicos.
- Etapa 2: Pensamos e discutimos sobre requisitos do *software*.
- Etapa 3: Com o cliente foram definidos os requisitos, e formulado o escopo de todo o projeto.

Neste contexto os requisitos podem ser divididos em funcionais e não funcionais, os requisitos funcionais especificam como o sistema interage com o contexto a sua volta, e os requisitos não funcionais remetem a atributos de qualidade do sistema proposto (MACHADO, 2012).

3.3.1 - Requisitos Funcionais

- Realizar Cadastro De Futuros Associados Para Aprovação,
- Gerar boletos referentes ao contrato do associado de forma mensal
- Login e senha para usuários
- Login e senha para administradores
- Na área do associado ele poderá imprimir seus boletos, ver seus dados cadastrais.
- Na área do administrador, onde ele irá efetuar a aprovação do novo associado e validando os dados necessários para isso.
- O administrador irá emitir os boletos aos associados.
- O administrador poderá efetuar o cancelamento dos contratos dos associados.

3.3.2 - Requisitos não Funcionais

- A área do usuário deverá ser simples e minimalista.
- Possuir uma interface com design responsivo
- O usuário poderá ter uma foto em sua tela inicial.
- O usuário deverá fazer login.
- O sistema deverá está hospedado para esta disponível em tempo real
- A senha do associado deverá ter no mínimo seis caracteres.

3.3.3 - Regras de negócios

- O associado terá de escolher um período de contrato de seis ou doze meses..

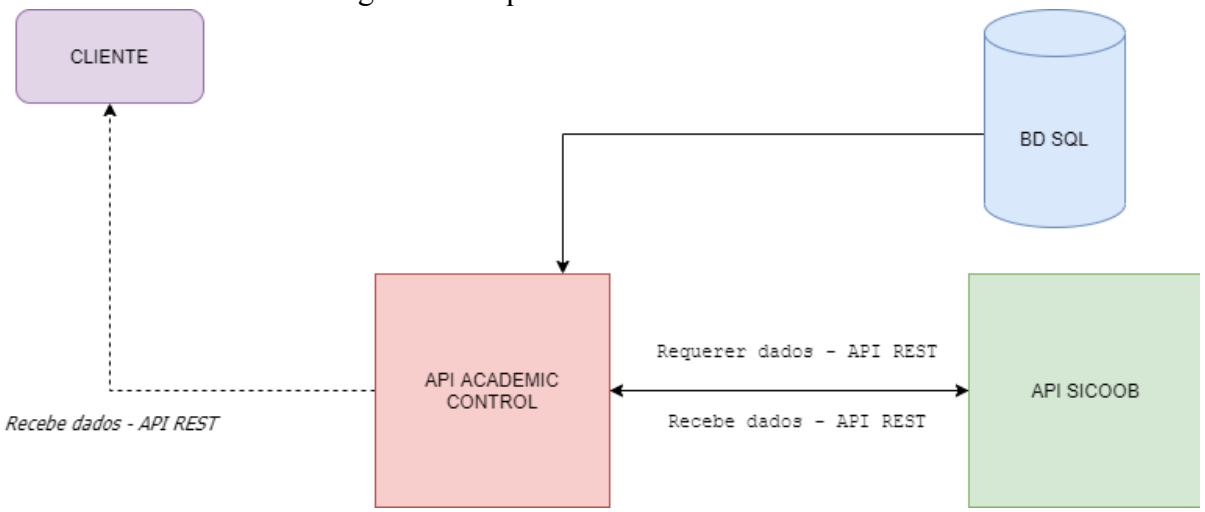
- Um associado só poderá finalizar o contrato depois de quitar os boletos..

3.4 - Arquitetura de software

Definir arquitetura de *software* é uma tarefa complicada principalmente quando o entendimento sobre o assunto é vago. Porém o entendimento desses artefatos é crucial para o bom desempenho tanto das metodologias usadas quanto do próprio sistema, Silva (2007) definiu que a ideia de um mapeamento simbiônico é a possibilidade de considerar que requisitos e arquitetura são artefatos que beneficiam um ao outro.

No geral a arquitetura foi desenvolvida usando o padrão de requisição e resposta onde o *front-end* solicita resposta a API que retorna os dados, um fato principal aqui é que quem solicita as informações a API SICOOB não é o *front-end* e sim a API *ACADEMIC CONTROL*, ela que faz o trabalho de autorização de validação dos dados como mostra na Figura 5.

Figura 5 - Arquitetura Academic Control



Fonte: Elaborado pelo autor

Na representação acima, uma aplicação *NodeJS* ao receber uma requisição HTTP, solicita dados do *Postgres* no qual os envia. A API *Academic Control* quando necessário incluir um boleto, ou listar os boletos do pagador logado ou até mesmo emitir segunda via do título, consulta a API SICOOB que retorna seus dados e assim o fluxo se encerra fornecendo informações ao *front-end.*

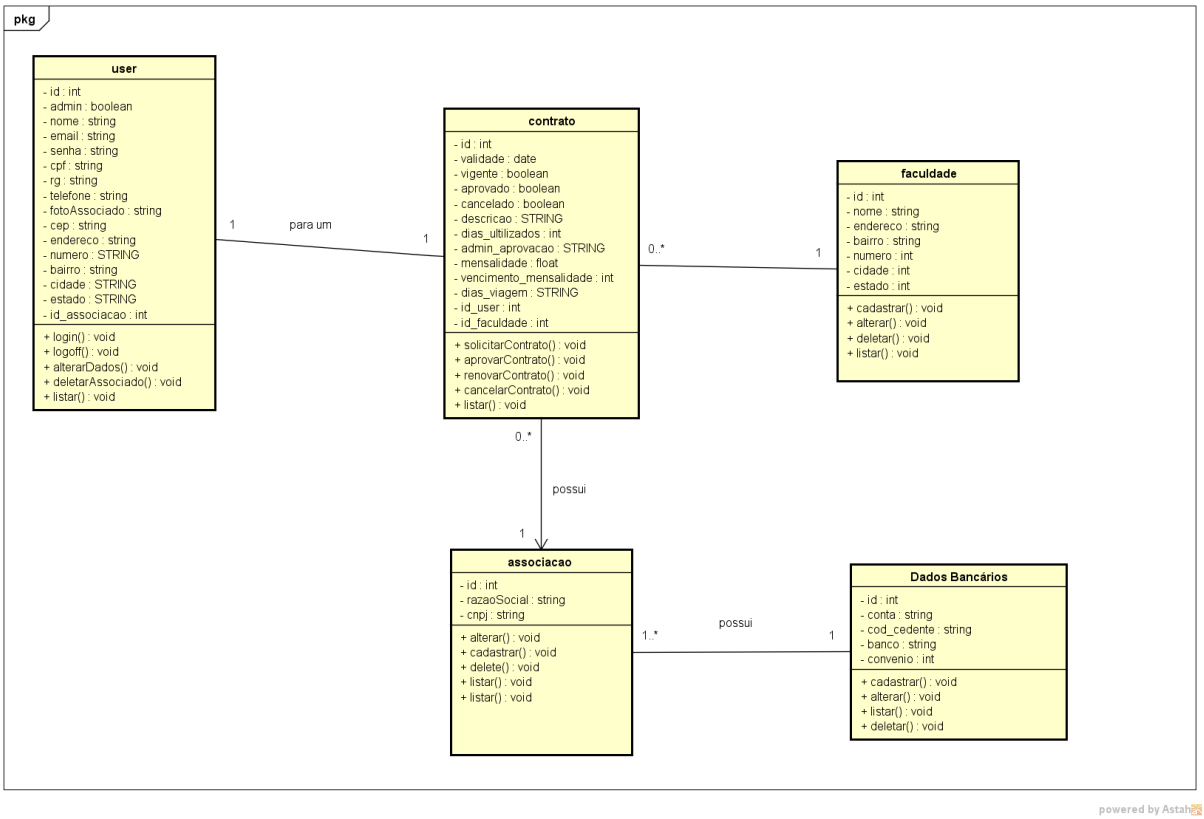
3.5 Modelagem

A modelagem do sistema, foi feito a partir dos paradigmas da *UML*. Nos subcapítulos a seguir serão apresentados os diagramas de Classe e Atividade.

3.5.1 - Diagrama de Classes

O diagrama de classes expresso na figura 6 foi concebido a partir do levantamento de requisitos, passando por um processo de validação e revisão dos requisitos.

Figura 6 - Diagrama de classes

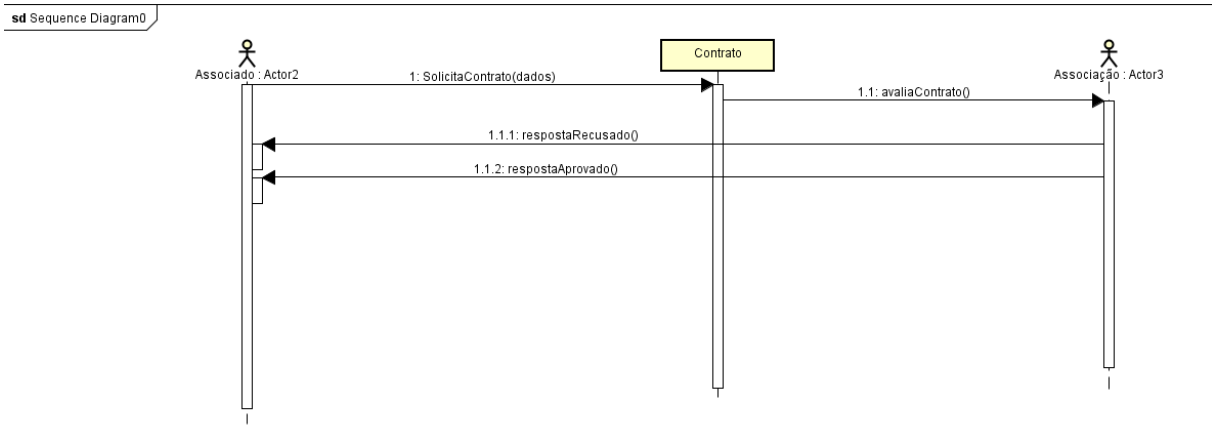


Fonte: Elaborado pelo autor

3.5.2 - Diagrama de Sequência

O diagrama de sequência arquiteta a ordem de eventos que ocorre em determinado processo, como GUEDES (2014) aponta, que o diagrama de sequência é é processo fundamental para compreensão do fluxo de acontecimentos. Nisso o Diagrama de Atividade foi elaborado desmembrado apenas algumas das funções implementadas como a de cadastro de associados conforme a Figura 7.

Figura 7 - Diagrama de sequência



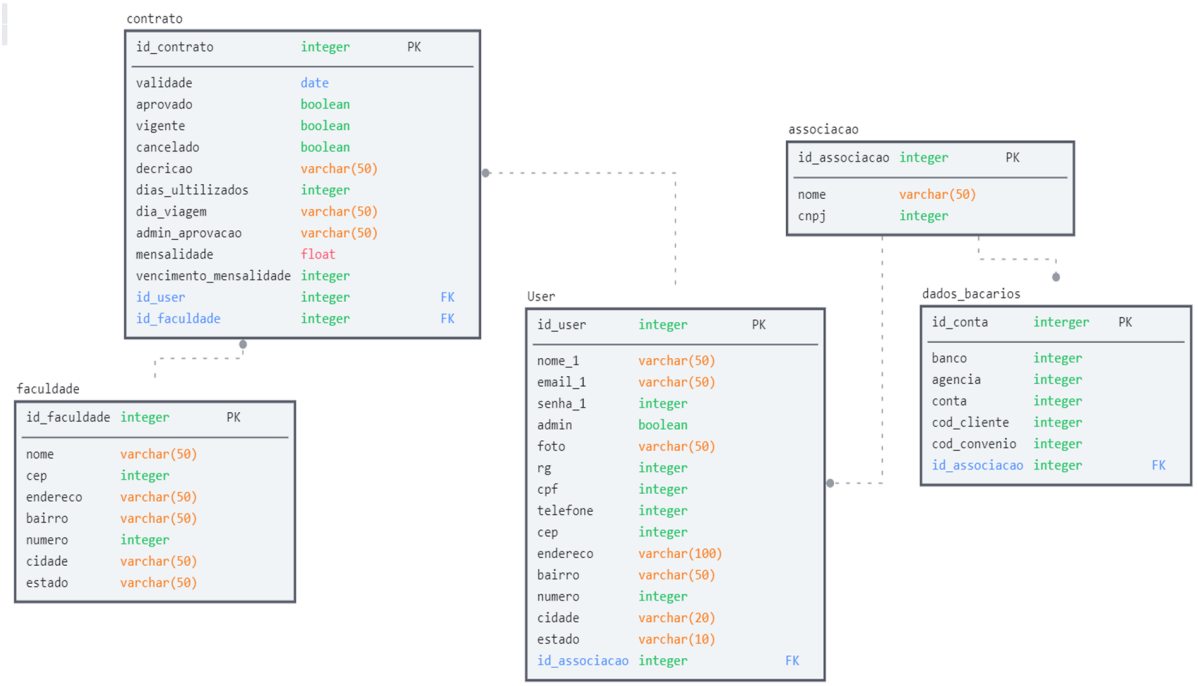
Fonte: Elaborado pelo autor

3.6 - Persistência

A persistência em sua primeira versão foi falha pois trazia consigo informações em excesso, na nessa versão de persistência por exemplo nos referimos ao boleto como uma tabela, contendo todos os campos que existem em um boleto como data de vencimento, valor e demais, porém tal informação é desnecessária quando temos a API de boletos do Sicoob que nos fornece todos os dados pertinentes aos boletos.

Pensando nisso realizamos a nova modelagem da persistência apresentada na figura 8, nela limpamos e alteramos algumas informações das tabelas da versão anterior como inserção de dados do endereço na tabela de usuário e como alteração dos campos na tabela de contrato e também a exclusão da tabela de cobrança. Outro ponto foi a exclusão da tabela de cobrança que tinha como função registrar as cobranças feitas a cada associado, porém analisando mais concisamente os requisitos informados pelo cliente, não existe a necessidade de uma tabela no banco de dados para gravar tais informações.

Figura 8 – **Diagrama de entidade e relacionamento**



Fonte: Elaborado pelo autor

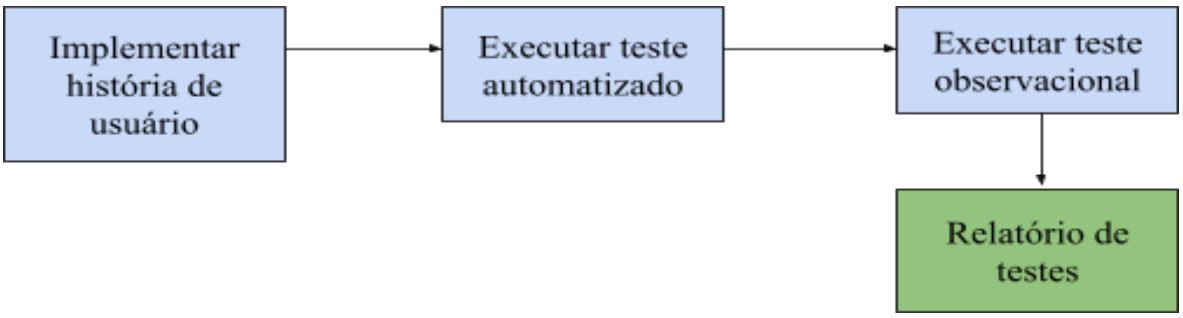
3.7 - Plano de testes

A elaboração de testes de *software* de acordo com o Bernardo(2008) é técnica voltada para a melhoria da qualidade dos sistemas. Foi elaborado para estruturar a qualidade das respostas da API, nesse cenário os testes respondem questões simples e até mesmo complexas, no nosso caso iremos responder os seguintes aspectos:

- Quais atributos serão testados?
  - *Status*, *Content-Type*, e JSON.
- Quem realizará os testes?
  - Os testes serão realizados pelo programador da API *ACADEMIC CONTROL*
- Qual recurso será utilizado?
  - A aplicação *Postman* será o recurso para realização de testes.
- Como o processo e qualidade do sistema serão acompanhados?

- Os testes serão automatizados com a ferramenta *postman,* assim como testagens observacionais serão realizadas. O acompanhamento dos testes dar-se-ão através de relatórios de testagens.

Figura 9 - Fluxo de testagem



Fonte: Elaborado pelo autor

3.8 - Licença de uso

A licença de uso aplicada, baseou-se nos termos da licença criada pelo Instituto de Tecnologia do *Massachusetts* (MIT). Essa escolha se caracterizou pelo simples motivo de que a informação pode ser livre e aberta desde que a façamos, nesta oportunidade deixei de maneira aberta a toda a comunidade de desenvolvedores o código a fim de agregar ainda mais para com o conhecimento de toda a comunidade.

4. Resultados e discussões

Neste capítulo serão apresentados resultados e discussões referentes à aplicação proposta.

4.1 - Gerenciamento de configuração e mudanças.

O gerenciamento de mudanças e de versões foi realizado junto a plataforma do GitHub, que permitiu que pudéssemos codificar e hospedar nosso código fonte de maneira eficaz e segura. Uma das peculiaridades durante o processo foi o entendimento a respeito do uso de branches que até o momento eram vagos, porém como um projeto mais robusto, cada história de usuário que fora criado na aplicação *Taiga* foi convertido em uma ou mais funções que tiveram suas respectivas branches.

4.2 - Processos de desenvolvimento

O processo de desenvolvimento da aplicação foi guiado pela metodologia *Scrum*, sendo fundamental para a organização de todo o projeto, outra ferramenta utilizada foi a plataforma *Taiga*, apoiando a dinâmica da metodologia na prática. No total foram 10 *sprints* ao todo que levaram cerca de 4 meses para serem totalmente concluídos. Cada história de usuário foi fragmentada em uma ou mais tarefas de acordo com cada *sprint.* No momento da codificação outras demandas por outras histórias de usuário foram surgindo, tais como melhores consultas no banco de dados, um exemplo é a listagem de usuários por contratos pendentes, todo esse processo foi documentado nos comentários de cada *sprint*.

As *Sprints* foram realizadas nos determinados períodos:

Tabela 1 - *Sprint* 1

***Sprint* 1**	
Período	03-agosto à 05-agosto
História de usuário prevista	Concluído?
Crud básico e configuração da aplicação	Sim
Observações	
Essa história foi basicamente o *start*, para a configuração inicial da aplicação e a comunicação com o banco de dados *PostgreSQL*	

Fonte: Elaborado pelo autor

Tabela 2 - *Sprint* 2

***Sprint* 2**	
Período	06-agosto à 14-agosto
História de usuário prevista	Concluído?
Preparação dos *models*	Sim
Observações	
O foco nessa *sprint* era codificar toda a parte não sintetizada do usuário, ou seja, as funções mais simples e necessárias para o *front-end* consumir, tal como primeiras consultas, as duas primeiras *sprint* foram criadas a fim de registrar os momentos totalmente iniciais da aplicação, sem contar que nessa sprint que realizamos o *deploy* de nossa Api para o *Heroku.*	

Fonte: Elaborado pelo autor

Tabela 3 - *Sprint* 3

***Sprint* 3**	
Período	26-agosto à 09-setembro
História de usuário prevista	Concluído?
Associado e Administrador podem fazer *login* e *logOff*	Sim
Observações	
Com as primeiras funções da Api totalmente desenvolvidas e por sua vez com nossa aplicação *back-end* já hospedada, foi a vez do Vue.js fazer seu papel, ao nos depararmos com a realização de login e controle de sessão via JWT utilizamos o VueX para poder realizar o comportamento de login e por sua vez salvar em algum parte do navegador para que sempre que necessário pode-se ser consultado, obviamente um comportamento padrão que que requereu atenção especial para o desenvolvimento.	

Fonte: Elaborado pelo autor

Tabela 4 - *Sprint* 4

***Sprint* 4**	
Período	31-agosto à 14-setembro
História de usuário prevista	Concluído?
Usuário cria sua conta e solicita associação	Sim
Observações	

A Sprint 4 trouxe por sua vez o escopo da parte cadastral do usuário, essa *sprint* complementa a anterior, sendo desenvolvido nesta *sprint* todas as partes de cadastro de usuário e contrato que seriam utilizados nas *sprints* futuras.

Fonte: Elaborado pelo autor

Tabela 5 - *Sprint* 5

***Sprint* 5**	
Período	25-setembro a 09-outubro
História de usuário prevista	Concluído?
Administrador avalia solicitação do associado	Sim
Administrador pode cancelar contrato de associação	Sim
Observações	
A Sprint 5 fechou o desenvolvimento das operações básicas do administrador do sistema, pelas quais o mesmo pode realizar aprovações de novos associados e efetuar cancelamentos de contratos de determinados associados.	

Fonte: Elaborado pelo autor

Tabela 6 - *Sprint* 6

***Sprint* 6**	
Período	17-outubro à 20-novembro
Api Sicoob	Concluído?
Associado pode listar sua situação financeira	Sim
Administrador emitir relatório financeiro	Sim
Observações	
Esta *Sprint* foi adiada durante o processo desenvolvimento, por necessitar de maior esforço para obter êxito no ato de integração com a Api de boletos do Sicoob. Por fim, essas atividades geraram a necessidade da implementação das funções de acesso ao token e *Refresh token*, bem como da criação de uma rotina de tarefas para que fosse realizada a ação de renovação de *token* a cada 30 minutos, mantendo a aplicação funcionando e autorizada.	

Fonte: Elaborado pelo autor

Tabela 7 - *Sprint* 7

***Sprint* 7**	
Período	23-novembro à 27-novembro
Emissão de boletos	Concluído?
Administrador pode emitir boleto ao associado	Sim
Observações	
Nesta *sprint* foi finalizada a integração da API *Academic Control* e a API SICOOB para que a emissão de boletos ocorresse de maneira correta. Uma ressalva, todos os testes da API SICOOB foram no ambiente de homologação, por ainda não ter credenciais para o ambiente de produção.	

Fonte: Elaborado pelo autor

Tabela 8 - *Sprint* 8

***Sprint* 8**	
Período	29 novembro 2021- 03 dezembro 2021
Alteração de Dados	Concluído?
Associado pode alterar seus dados	Sim
Observações	
Esta função foi implementada na api, mas só nesta *sprint* foi feita a integração com o *front-end.*	

Fonte: Elaborado pelo autor

Tabela 9 - *Sprint* 9

***Sprint* 9**	
Período	03 dezembro 2021 - 07 dezembro 2021
Renovar e Cancelar contrato	Concluído?
Associado pode solicitar renovação do contrato	Não
Associado pode solicitar cancelamento do contrato	Não
Observações	
Esta função serve para renovar e cancelar o contrato solicitado pelo associado.	

Fonte: Elaborado pelo autor

Tabela 10 - *Sprint* 10

***Sprint* 10**	
Período	07 dezembro 2021-08 dezembro 2021
Api Sicoob	Concluído?
Administrador pode realizar cobranças	Não
Observações	
Essa tem como objetivo permitir ao administrador disparar *email* de cobrança ao associado.	

Fonte: Elaborado pelo autor

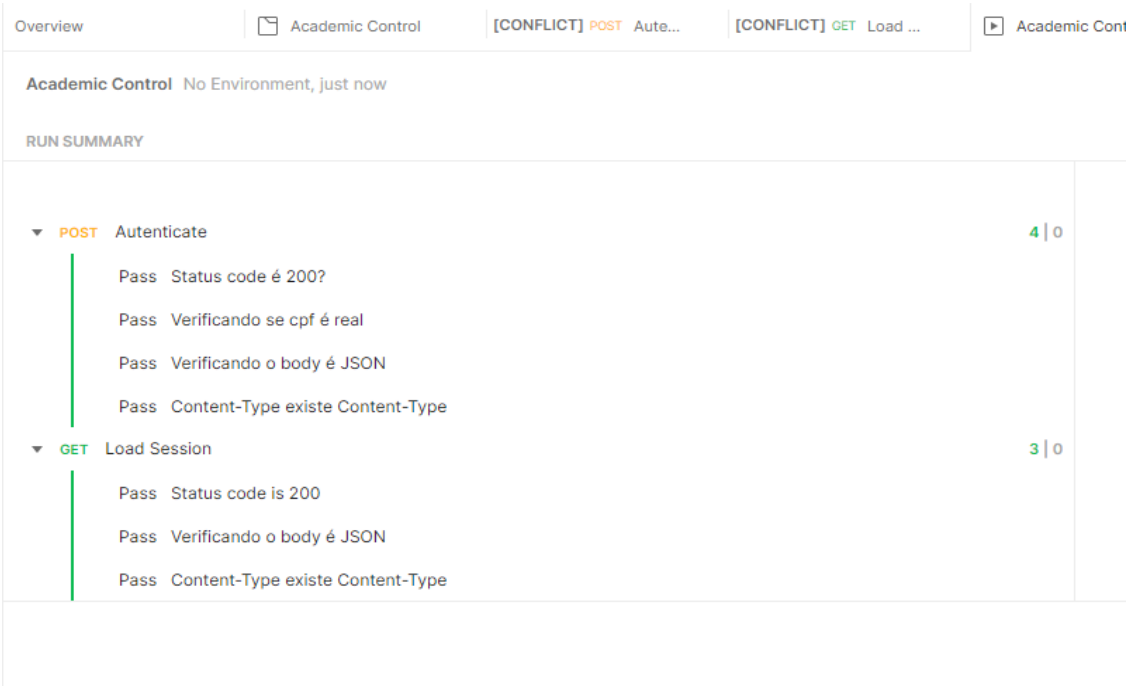
4.3 - Relatório de testes

Durante o processo foram executados os testes observacionais e automatizados. Após a implantação de cada história de usuário, foram gerados os seguintes relatórios.

- Teste automatizado:

Obter resultados de sucesso *status* (200) nas requisições da API é fundamental para se certificar a disponibilidade dos serviços. A figura abaixo representa o resultado dos *status* das requisições, assim foi possível observar que as rotas respondem adequadamente às requisições.

Figura 10 - Relatório de teste automatizado



Fonte: Elaborado pelo autor

- Relatório observacional:

O relatório observacional foi executado com o auxílio da ferramenta *Postman,* por ela o usuário pode realizar testes flexíveis e manuais identificando problemas que um algoritmo talvez não fosse capaz, como por exemplo, uma regra de negócio programada erroneamente.

Tabela 11 - Teste A

Teste - *Login* e Autenticação	
Sprint	3
Criticidade:	Alta
Teste:	Requisições na API
Caso de Teste	Realizar chamada a API para poder obter *token* de acesso
Pré condição:	Ter *e-mail* e senhas cadastrados
Procedimento:	a) Autenticar-se na API b) Verificar se dados informados existem c) Gerar *token* de acesso d) Consumir dados
Resultado esperado:	a) O sistema irá validar as credenciais e retornar dados do usuário seguido do *token* de acesso com expiração em 2 horas..
Bugs encontrados:	Não foram encontrados *bugs* no teste.

Fonte: Elaborado pelo autor

Tabela 12 - Teste B

Teste - *Load Session*	
Sprint	3
Criticidade:	Alta
Teste:	Requisições na API
Caso de Teste	Buscar dados na Api quando o *state* do VueX fosse limpo por uma atualização na página
Pré condição:	Estar autenticado e com token armazenado no *localStorage*
Procedimento:	e) Validar *token* f) Verificar ID do usuário do *token* g) Realizar consulta de dados do usuário

	h) Consumir dados
Resultado esperado:	b) O sistema buscará os mesmos dados que quando solicitado *token*, a diferença será que dessa vez não retornará um novo *token*, somente as informações pertinentes aquele usuário
Bugs encontrados:	Erro ao consumir pois os dados estavam com listagem diferentes. Quando o usuário recarrega a página a requisição *load Session* trazia os dados com uma listagem divergente do primeiro, o que resultava em uma falha ao consumir os dados do usuário logado.

Fonte: Elaborado pelo autor

No relatório de teste da rota *Load Session* foi encontrado erro ao consumir os dados reportados da Api *Academic Control*, que teve que ser tratada, o problema foi corrigido e a aplicação voltou a funcionar normalmente renderizando os dados do usuário de acordo com o token de acesso do mesmo. Vale ressaltar que em quase todas as buscas de usuário é utilizado o ID dentro do JTW, o que retira a necessidade de informar o ID para a requisição, tornando a aplicação mais dinâmica e segura.

4.4 - Documentação

A documentação do sistema é fundamental para saber quais caminhos devem ser seguidos para que os dados possam ser consumidos de maneira eficaz. Abaixo será listado as ferramentas pertinentes a documentação.

4.4.1 - Documentação para desenvolvedores

A documentação para os desenvolvedores pode ser obtida através dos seguintes canais.

- Swagger.io: A descrição das rotas da API está disponibilizada na página inicial do projeto. Nela é possível identificar as rotas de consumo, descrição do método de autenticação, descrição das possíveis respostas às requisições da API, e informações acerca do projeto que se encontra na Figura 11 logo abaixo.
- Read.me:[16] No repositório do GitHub, é possível consultar o arquivo *Readme*, no qual apresenta informações básicas para configuração da aplicação em servidores próprios ou orientações para desenvolvedores.

[16] *Disponível em: https://github.com/montanari2019/academic_apiv1*

Figura 11 - Screenshot da documentação gerada com Swagger.io

Fonte: Elaborado pelo autor

4.5 - Implantação

O projeto *Academic Control* foi dividido em duas etapas API (servidor) e *front-end* (cliente), na Api ainda temos de hospedar o banco de dados, que neste sistema utilizamos a plataforma gratuita ElephantSQL[17] que nos concedeu um servidor com máximo de 10mb para hospedagem do nosso banco de dados. A Api em contrapartida foi hospedada no Heroku assim como o *front-end*..

Para acessar a API segue o endereço:

- https://api-academic-control-v2.herokuapp.com/inicio

[17] *Disponível em: https://www.elephantsql.com/*

Para acessar o *front-end* segue o endereço:

- Indisponível (*Localhost*)

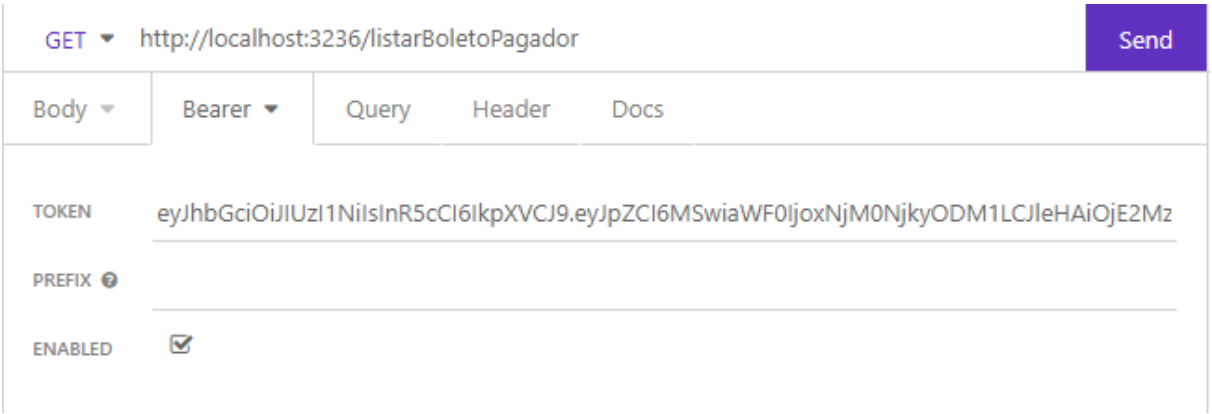
4.6 - Demonstração do software

A principal função do sistema talvez seja a emissão e listagem de boletos que foi alcançada graças a integração com a Api de boletos do Sicoob, porém a sim diversas outras funções como gestão de usuários, listagem de contratos pendentes de aprovação e entre outras. Aqui iremos listar algumas das principais funções.

- Listar boleto por pagador.

Na listagem de boletos por pagador solicitamos a Api do Sicoob os boletos referentes a um pagador específico, passando o número de cpf como parâmetro, em contrapartida em nossa Api unicamente pedimos Api ao mesmo estar autenticado, ressalto que essa função é destinada a tela do usuário onde o mesmo tem acesso a todos os seus boletos.

Figura 12– Listagem de boletos

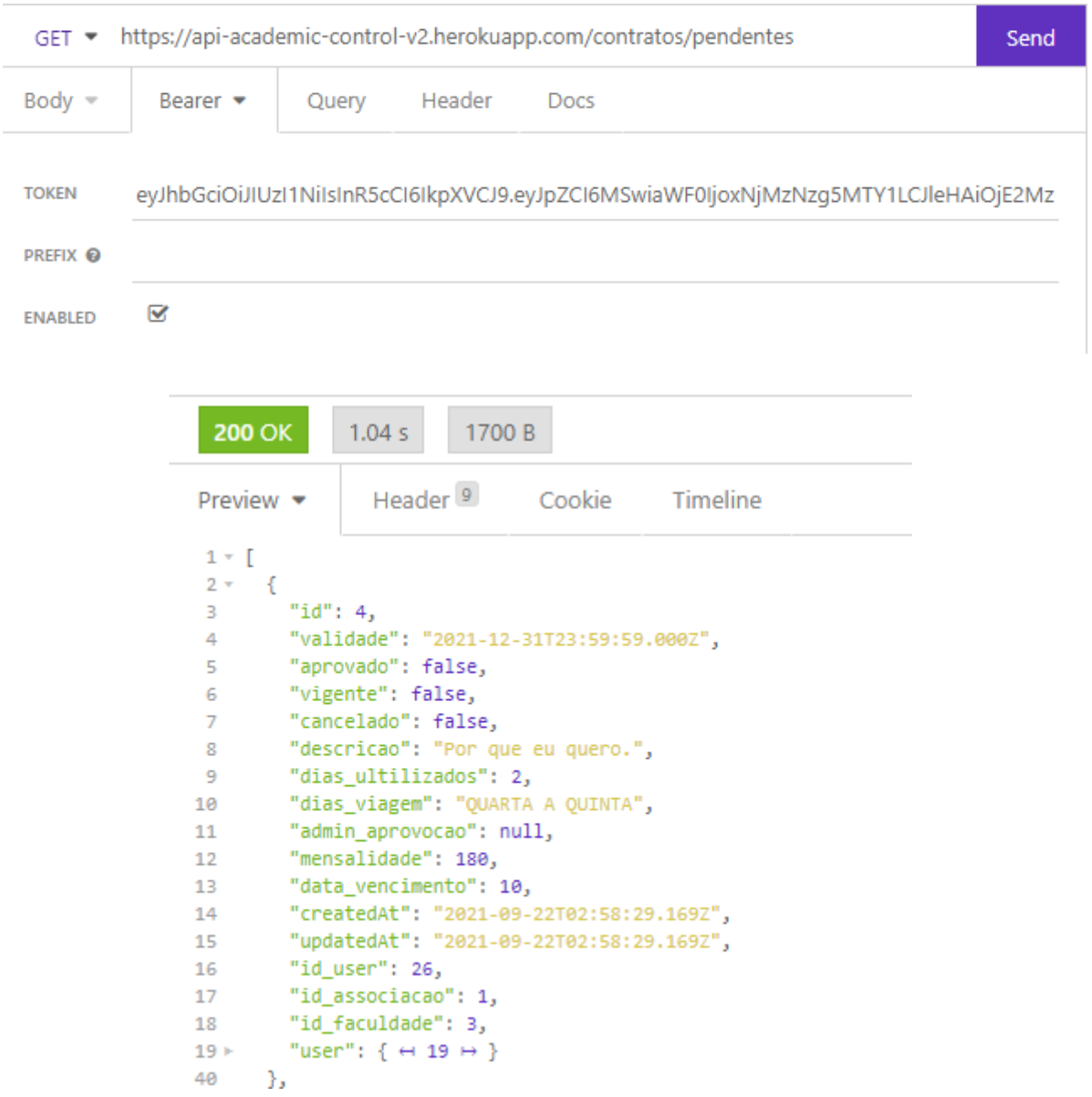


Fonte - Elaborado pelo autor

- Listar contratos pendentes de aprovação.

Neste exemplo, listamos todos os contratos que estão pendentes para o administrador da associação analisar e realizar a aprovação ou recusa deles, aqui não se passa parâmetro pois listamos os contratos pelo id da associação que se encontra no usuário administrador logado.

Figura 13 - Listar contratos pendentes de aprovação

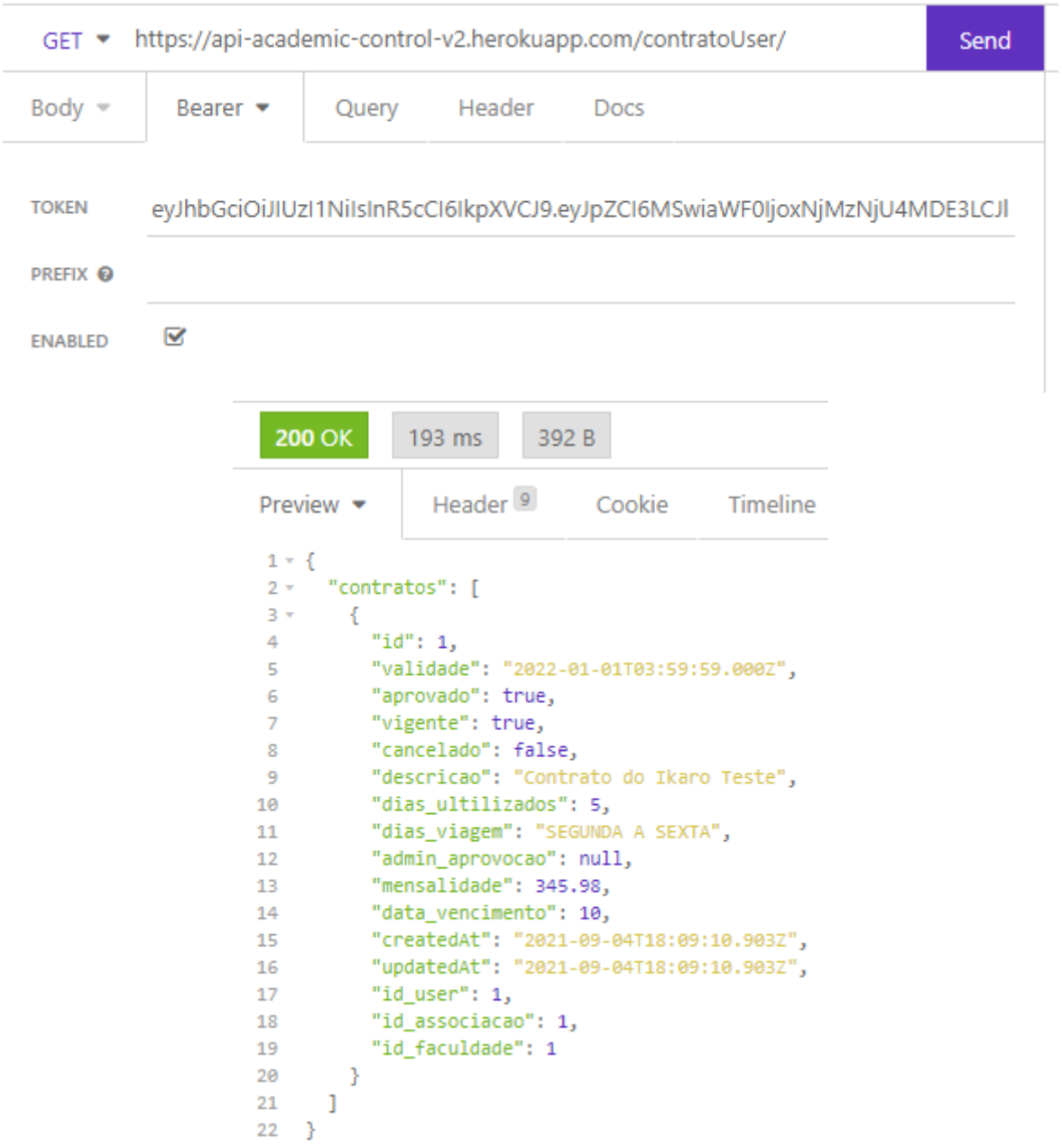


Fonte: Elaborado pelo autor

- Listar contrato pelo usuário logado;

Aqui o sistema novamente irá solicitar o token de acesso para prosseguir com a listagem, e o mesmo irá listar o contrato de acordo com o ID do token do usuário logado.

Figura 14 - Listar contrato pelo usuário logado

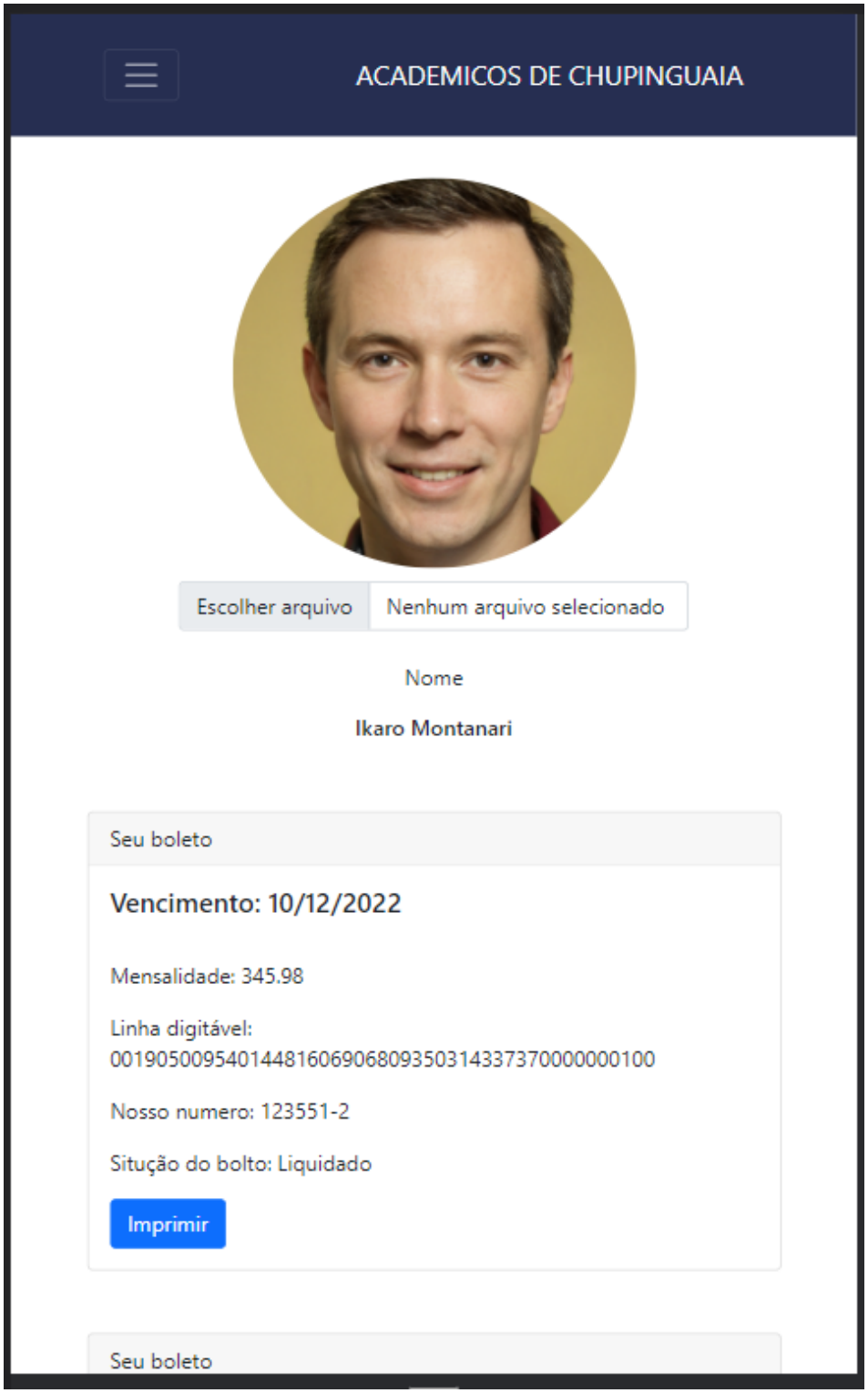


Fonte: Elaborado pelo autor

- Área do usuário

O sistema deve apresentar as informações de uma forma que o usuário final compreenda, aqui optamos por utilizar um designer simples e minimalista a fim de garantir uma boa visualização e também seja agradável.

Figura 15 - Área do usuário



Fonte: Elaborado pelo autor

5. Considerações finais

Uma ferramenta de gestão é fundamental para boa execução de um negócio, ao olharmos ao nosso redor vemos que cada vez se torna implícito no quesito boa gestão, sendo assim a discussão sobre possíveis meios para melhorar a tramitação dos processos da associação culminou nesse projeto.

Na proposta apresentada vemos que em tese o sistema terá capacidade de realizar todas as suas operações e substituir inclusive a necessidade de um local onde as pessoas se dirigiam para realizar tais solicitações, visto que cada vez mais o mundo globalizado se encaminha para a facilitação nas execuções de processos e simplificação dos mesmos.

Pensar em aprimoramento não é uma tarefa simples, e em seu pensar leva anos para se adquirir e adaptar, porém na ideia singular do pensamento humano a evolução sempre será necessária, falamos hoje somente de um sistema que irá facilitar a utilização dos serviços da associação para com os usuários e administradores que agora não terão mais a necessidade de locomoção para realizar suas demandas.

O projeto é simples com artefatos simples. Porém, é seguro afirmar que o *Academic Control* cumpre seu papel proposto, aprimorando processos existentes que funcionavam de forma manual e sem o apoio de um sistema informatizado, melhorando, portanto, a usabilidade dos usuários afetados.

5.1 - Trabalhos futuros

Após o desenvolvimento da solução proposta, foram identificados os futuros trabalhos que podem ser realizados:

a) Desenvolvimento de um ambiente de recuperação de senhas;
b) Integração com bases de órgãos como serasa para verificar inadimplência de novos associados;
c) Integração de outros meios de pagamentos como cartão de crédito para pagamento das mensalidades;
d) Implementação de um mecanismo de buscas mais aprimorados e paginação na listagem de usuários.

Referências

BERNARDO. **A Importância dos Testes Automatizados.** 2008 **.** Disponível em: http://ccsl.ime.usp.br/agilcoop/files/A%20Importancia%20dos%20Testes%20Automatizados.pdf . Acesso em: 01 dez 2021

CARVALHO, BERNARDO VASCONCELOS DE ET AL. **Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica**. [*S. l.*: *s. n.*], 2012.

CODE. **Getting Started**. Disponível em: https://code.visualstudio.com/docs. Acesso em: 27 nov. 2021.

FIELDING,:ROY THOMAS: **Architectural styles and the design of network-based software architectures**, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf Acesso em: 21 nov. 2021

GAMA, ALEXANDRE. **GitHub e Git - Colaboração e Organização**. [*S. l.*], 2021. Disponível em: https://www.devmedia.com.br/github-e-git-colaboracao-e-organizacao/24150. Acesso em: 27 nov. 2021.

GITHUB. **GitHub**, 27 nov. 2021. Disponível em: https://github.com/. Acesso em: 27 nov. 2021.

GUEDES, GILLEANES T. A. **UML 2 – Guia Prático**. 2. ed. [S. l.: s. n.], 2014

HEROKU. **heroku**. Disponível https://www.heroku.com/about . Acesso em: 21 nov. 2021.

INSOMNIA. **Insomnia**. Disponível em https://docs.insomnia.rest/insomnia/get-started Acesso em: 21 nov. 2021.

JEFF SUTHERLAND. **Scrum: a arte de fazer o dobro do trabalho na metade do tempo**. 2014. Disponível em: https://img.travessa.com.br/capitulo/LEYA/SCRUM_A_ARTE_DE_FAZER_O_DOBRO_DO_TRABALHO_NA_METADE_DO_TEMPO-9788544104514.pdf. Acesso em: 20 nov. 2021.

JWT. **Jwt**. Disponível em: https://jwt.io/introduction . Acesso em: 21 nov. 2021.

MACHADO, FELIPE NERY. **Análise e Gestão de Requisitos de Software Onde nascem os sistemas**. 2016.

MASSÉ, MARK: **Rest api design rulebook**. o'reilly media, inc, 2011. Disponível em: https://www.oreilly.com/library/view/rest-api-design/9781449317904/ . Acesso em: 21 nov. 2021

MDN WEB DOCS. **css** . 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS . Acesso em: 21 nov. 2021.

MDN WEB DOCS. **html: linguagem de marcação de hipertexto**. 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML . Acesso em: 21 nov. 2021.

MDN WEB DOCS. **JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript . Acesso em: 21 nov. 2021.

NAKAMURA, EMÍLIO; GEUS, PAULO. **Segurança de redes em ambientes corporativos**. São Paulo: Berkeley Brasil, 2002.

NODEJS. **Nodejs.** Disponível em: https://nodejs.org/en/about/. Acesso em: 21 nov. 2021.

OAUTH 2.0. **Oauth 2.0**. Disponível em: https://oauth.net/2/ . Acesso em: 21 nov. 2021.

OLIVEIRA, D. **Estrutura organizacional: sistemas, organizações e métodos: uma abordagem gerencial**. São Paulo: Atlas, 2001.

OTTO, M. THORNTON, **J. bootstrap is the most popular html, css, and js framework for developing responsive, mobile first projects on the web.** 2017. Disponível em: https://getbootstrap.com/docs/3.3/ . Acesso em: 21 nov. 2021

POMPILHO, S. **Análise Essencial Guia Prático de Análise de Sistemas**. Rio de Janeiro: Ed. Ciência Moderna Ltda, 1995.

POSTGRESQL. **PostgreSQL**. Disponível em: https://www.postgresql.org/docs/ Acesso em: 27 nov. 2021.

SCHWABER, SUTHERLAND . **O Guia definitivo para o Scrum: as regras do jogo.** 2013. Disponível em: https://andrelmgomes.com.br/wp-content/uploads/2020/11/Guia-do-Scrum-2020-PT-BR-EN-US-1.pdf . Acesso em 01 dez 2021

SILVA, L. ET. AL. **On the Symbiosis of Aspect-Oriented Requirements and Architectural Descriptions.** In: Proc. of the EarlyAspcts co-located with AOSD 2007. 2007.

STACK OVERFLOW: **Resultados da pesquisa do desenvolvedor** (2019) Disponível em: https://insights.stackoverflow.com/survey/2019/#technology . Acesso em: 21 nov. 2021

SWAGGER. **Swagger.io**. Disponível em: https://swagger.io/. Acesso em: 21 nov. 2021.

TAIGA. **Taiga**. 27 nov. 2021. Disponível em: https://www.taiga.io/. Acesso em: 27 nov. 2021

TAIICHI OHNO. **Sistema toyota de produção**. 1978. Disponível em: https://www.scielo.br/j/prod/a/bRXLyrMFFK6WZGCvYNxC8sR/?format=pdf&lang=pt. Acesso em: 20 nov. 2021.

VERAS, MANOEL. **Visão Geral: Conceito. *In*: CLOUD Computing**: **Nova arquitetura da TI**.: BRASPORT, 2012.

VUE.3.0 . **o que é vue.js?**. 2021. Disponível em: https://v3.vuejs.org/guide/introduction.html . Acesso em: 21 nov. 2021.

VUE.3.0, **reactivity. reactivity in depth**. 2021. Disponível em: https://v3.vuejs.org/guide/reactivity.html#what-is-reactivity . Acesso em: 21 nov. 2021.

VUEX. **o que é vuex?**. 2021. Disponível em: https://next.vuex.vuejs.org/ . Acesso em: 21 nov. 2021.

Apêndice A – Licença de uso

Apêndice B – Licença de uso

Relatório de história de usuário. (Escopo do projeto)

Tabela 13 - Histórias de usuário

ID	**Descrição**	**Pontos**
1	Criação das funções básicas	40
2	Configurações básicas do banco de dados	20
3	Associado e Admin podem efetuar login e logoff	26
4	Usuário cria conta e solicita associação	34
5	Admin pode cancelar contrato de associação	23
6	Admin vai avaliar a solicitação do associado	26
7	Associado pode listar sua situação financeira	60
8	Administrador pode emitir relatório financeiro	23
9	Admin pode emitir boleto ao associado	16
10	Associado pode alterar seus dados	11
11	Associado pode solicitar renovação de contrato	33
12	Associado pode cancelar contrato	29
13	Admin pode realizar cobranças	18
Total	Total de pontos definidos	359

Fonte: Elaborado pelo autor