UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JEFFERSON MICHAEL DE AZEVEDO JUNIOR

SISTEMA DE GERENCIAMENTO DE CRÉDITOS PARA CONTROLE DO RESTAURANTE UNIVERSITÁRIO

PONTA GROSSA

2022

JEFFERSON MICHAEL DE AZEVEDO JUNIOR

SISTEMA DE GERENCIAMENTO DE CRÉDITOS PARA CONTROLE DO RESTAURANTE UNIVERSITÁRIO

Credit management system to control the University Restaurant

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Mônica Hoeldtke Pietruchinski

PONTA GROSSA

2022

JEFFERSON MICHAEL DE AZEVEDO JUNIOR

SISTEMA DE GERENCIAMENTO DE CRÉDITOS PARA CONTROLE DO RESTAURANTE UNIVERSITÁRIO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 09/novembro/2022

Mônica Hoeldtke Pietruchinski
Doutora
Universidade Tecnológica Federal do Paraná - Campus Ponta Grossa

---

Itamar Iliuk
Doutora
Universidade Tecnológica Federal do Paraná - Campus Ponta Grossa

---

Luiz Rafael Schmitke
Mestre
Universidade Tecnológica Federal do Paraná - Campus Ponta Grossa

PONTA GROSSA

2022

Dedico este trabalho aos meus pais, por
sempre me apoiarem.

AGRADECIMENTOS

Agradeço a minha orientadora Profª. Dra. Mônica Hoeldtke Pietruchinski, pelo suporte e pela sabedoria compartilhada durante o desenvolvimento desse projeto.

Gostaria de agradecer também a toda a minha família e amigos por todo o suporte durante esses difíceis anos de formação. Sem o apoio de vocês eu não estaria aqui hoje.

RESUMO

Com a pandemia do coronavirus finalmente perdendo as forças, a vida no nosso país e no mundo começa, aos poucos, voltar ao normal. No primeiro semestre de 2022, a maioria dos estados brasileiros começaram a liberar o uso de mascara em locais abertos e alguns em locais fechados. Entretanto os cuidados higiênicos aprendidos de maneira, para muitos, abrupta, precisam se manter. Tendo o cenário atual em mente e pensando na realidade do estudante do campus Ponta Grossa, podemos analisar e identificar pontos ondes essas precauções podem ser melhoradas. Um exemplo disso é o restaurante universitário. O ponto analisado foram as fichas, que são utilizadas como auxilio na administração das refeições servidas pelo restaurante. Mesmo com todo cuidado e treinamento que os funcionários recebam para higienizar as fichas, erros podem acontecer. Com esse contexto, esse trabalho realiza o desenvolvimento de um sistema completo de gerenciamento de créditos para o restaurante universitário. Utilizando Python como linguagem principal, foi desenvolvido, com auxilio da *framework web* Flask, uma *API REST* que implementa as funcionalidades administrativas, levantadas em analise, do restaurante, disponibilizando um núcleo de gerencia desses créditos para o usuário, tudo isso sem apoiado em um banco de dados MySQL. Para consumir essa *API*, foi desenvolvido sistema *front-end*, em Flutter, de apoio com dois focos principais, os usuários e os funcionários do restaurante. O sistema possui funcionalidades como: criação de contas, pagamentos online, recargas presenciais, validação de refeições, entre outros.

Palavras-chave: restaurante universitário; python; flask; flutter; mysql.

ABSTRACT

With the coronavirus pandemic finally losing its strength, life in our country and in the world is slowly starting to return to normal. In the first half of 2022, most Brazilian states started to allow the use of masks in open places and some in closed places. However, the hygienic care learned in an abrupt way, for many, needs to be maintained. Keeping the current scenario in mind and thinking about the reality of the Ponta Grossa campus student, we can analyze and identify points where these precautions can be improved. An example of this is the university restaurant. The analyzed point were the cards, which are used as an aid in the administration of the meals served by the restaurant. Even with all the care and training that employees receive to sanitize the chips, mistakes can happen. With this context, this work carries out the development of a complete credit management system for the university restaurant. Using Python as the main language, it was developed, with the help of framework web Flask, a API REST that implements the administrative functionalities, raised in analysis, of the restaurant, providing a management core of these credits to the user, all without being backed up by a MySQL database. To consume this API, a front-end system was developed, in Flutter, to support two main focuses, users and restaurant employees. The system has features such as: account creation, online payments, in-person payments, meal validation, among others.

Keywords: university restaurant; python; flask; flutter; mysql.

LISTA DE FIGURAS

LISTA DE FOTOGRAFIAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ANSI	*American National Standards Institute*
API	*Application Programming Interface*
BDOR	Banco de dados objeto-relacional
CRUD	*Create, Read, Update and Delete*
CWI	*Centrum voor Wiskunde en Informatica*
ER	*Entity Relationship*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ISO	*International Organization for Standardization*
json	*JavaScript Object Notation*
JWT	*JSON Web Token*
ORM	*Object–relational mapping*
PAE	Programa Auxílio Estudantil
PEP	*Python Enhancement Proposals*
RA	Registro Acadêmico
REST	*Representational State Transfer*
RIUT	Repositório Institucional da Universidade Tecnológica Federal do Paraná
RU	Restaurante Universitário
SDK	*Software Development Kit*
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional

SO	Sistema Operacional
SQL	*Structured Query Language*
UI	*User Interface*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
UTFPR	Universidade Tecnológica Federal do Paraná
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O RU tem como objetivo servir refeições para a comunidade acadêmica e possíveis visitantes. Além de oferecer refeições menores que podem servir como café da manhã e lanches da tarde, o principal uso vem no horário de pico, o almoço (OLIVEIRA, 2021). Na Figura 1 está ilustrado os horários do dia com maior movimentação. As refeições principais são fornecidas por um preço acessível aos consumidores e, para os alunos, parte desse custo é abatido por subsídios fornecidos pela universidade (UTFPR, 2022). Além disso, como politica de permanência, por meio do Programa Auxílio Estudantil (PAE) a UTFPR fornece, além de outros auxílios, o auxílio alimentação aos seus estudantes, que fornece aos contemplados auxílio total no valor das refeições principais no RU (UTFPR, 2017).

Figura 1 – Intervalos com maior movimentação, em uma segunda feira, segundo dados coletados pelo Google de seus usuários.

Fonte: Google (2022).

As refeições principais (almoço e janta) são servidas no formato de *self-service*, onde os alimentos ficam dispostos em bandejas e são coletados pelos alunos. Porém antes de alcançar esse ponto, para se alimentar, os alunos precisam passar por um processo repetitivo e que ocorre diariamente. O RU, do campus Ponta Grossa, utiliza um sistema de fichas para gerenciar os alunos que desejam se alimentar, essas fichas são vendidas como forma de comprovação de que o aluno pagou pelas refeições. Na Fotografia 1, podemos observar o local de armazenamento onde as fichas são depositadas pelos alunos após terem seu Registro Acadêmico (RA) identificado através do código de barras presente no crachá.

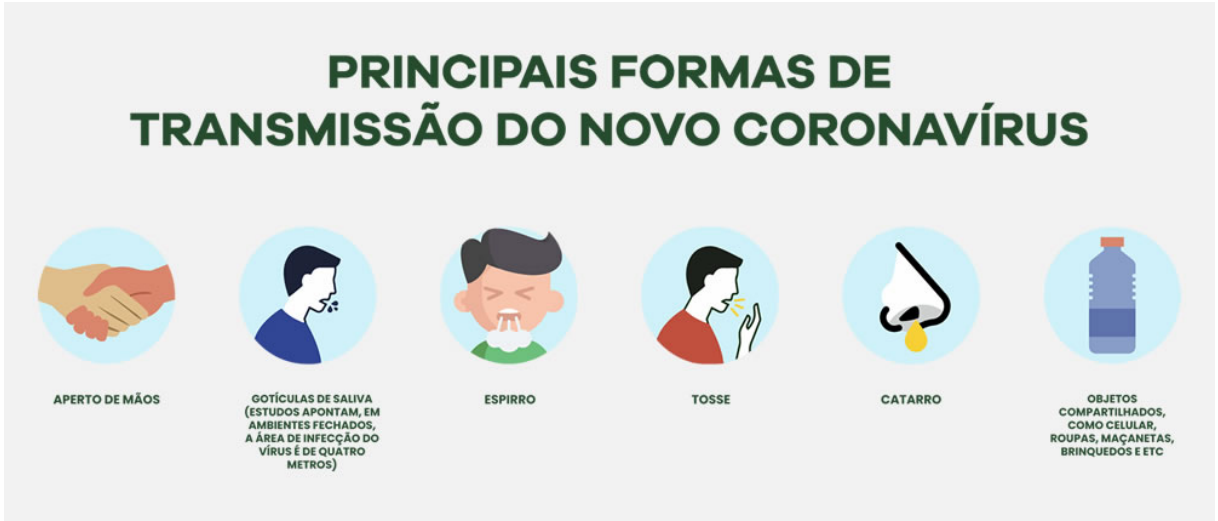
Essas fichas são compradas pelos alunos, após permanecerem na fila do caixa, uma por vez ou varias de uma vez, como estrategia para evitar entrar na fila todos os dias, otimizando tempo. Alguns alunos compram para a semana inteira, outros para o mês inteiro. Então as fichas são guardadas pelo estudantes das mais variadas formas: no bolso, na carteira, junto

Fotografia 1 – Cesta com as fichas utilizadas no do RU

Fonte: Autoria própria (2022).

ao celular, etc. Indo contra algumas recomendações de higiene ilustradas na Figura 2. Além disso, após permanecer em outra fila, são entregues para os funcionários do caixa do RU, na porta de entrada da área onde as refeições ficam dispostas. Em adição a receberem essas fichas, os funcionários precisam lidar com dinheiro e outros produtos comercializados dentro do restaurante. Todo o processo ocorre de forma manual, ou seja, é possível identificar vários possíveis pontos de contaminação, alguns desses pontos vão de acordo com os ilustrado por especialistas no cartaz informativo da Figura 2.

Figura 2 – Principais formas de transmissão do corona vírus



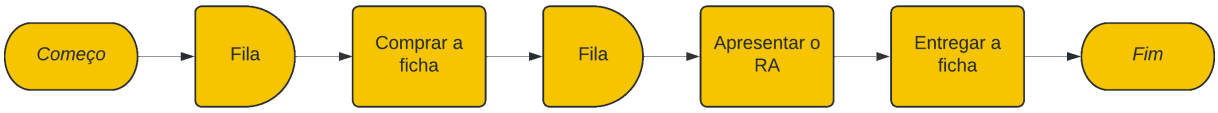
Fonte: UNIFESP (2021).

Além das questões higiênicas, diversas outras dificuldades ocorrem por consequência desse processo manual. Com atrasos sendo a principal delas. Ocorrendo com frequência e derivando diversos outros problemas. Como evasão das filas a favor de opções mais rápidas e menos nutritivas (MONTEIRO *et al.*, 2011; SOUZA *et al.*, 2014). A seguir estão relatadas algumas das causas desses atrasos:

- A interação entre os cliente e funcionários durante a compra e entrega das fichas.
- Verificação do registro acadêmico, que é feita com um leitor de códigos de barras.
- Venda produtos terceiros, como refrigerantes e doces.
- Alunos que, por engano, entram na fila de coleta, sem antes terem comprado as fichas.

Então, como ilustrado no fluxograma na Figura 3, o processo, que inicia na decisão de almoçar ou jantar e vai até a fase final de coleta da comida funciona da seguinte forma: O aluno fica na fila de compra das fichas; quando chega a sua vez, ele realiza a compra de uma ou mais fichas (desconsiderando bolsistas que não precisam comprar fichas); então parte para a outra fila, a de coleta; chegando no caixa ele entrega a sua ficha e apresenta o crachá que contem sua identificação no formato de código de barras; então, por último, realiza a coleta dos alimentos. As duas filas não são geridas por membros da empresa que administra o RU, elas são organizadas de maneira natural pelos alunos. O crachá é concedido ao aluno de forma gratuita no inicio da sua carreira acadêmica e é a forma de identificação do mesmo dentro do campus (BORGES, 2019).

Figura 3 – Processo da fila atual



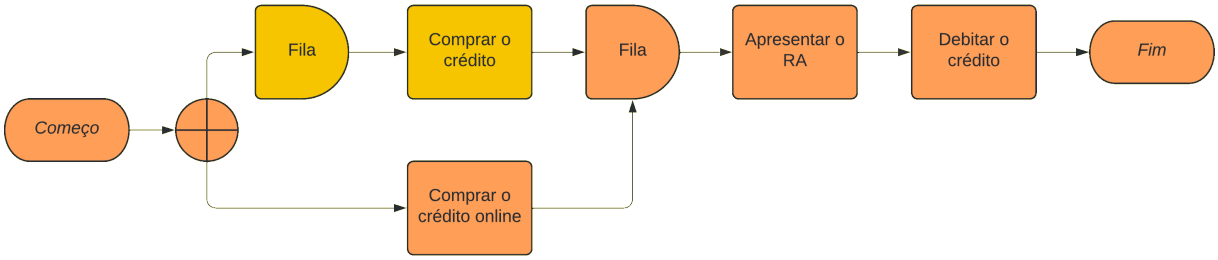
Fonte: Autoria própria (2022).

Uma das soluções apresentadas por esse projeto implementa uma solução em *software* e oferece alternativas mais eficientes para os alunos, como pode ser visto destacado na Figura 4. Onde o processo original de compra presencial é mantido para aqueles que estiverem confortáveis realizando a tarefa como sempre foi feita, mas mesmo assim sem a necessidade de manipulação das fichas, impactando em uma melhora na higiene (GOV.BR, 2021). Contudo, acrescentando a oportunidade de adquirir créditos online, agilizando o processo como um todo e facilitando a vida daqueles dispostos. Além disso, dois processos que antes eram manuais (fichas) se tornariam automatizados pelo sistema (crédito e debito), tornando tudo ainda mais eficiente.

As eventuais aquisições e utilizações dos créditos será feita utilizando o processo de leitura do crachá já existente, onde os funcionários realizam a leitura com o leitor de códigos

de barras no crachá do aluno. Dessa forma, o custo de implantação do sistema em ambiente real seria baixíssimo, pois toda a infraestrutura já existente seria reutilizada. Soluções alternativas existem e já foram implantadas, como cartões com chip de contato, porém, optar por uma solução semelhante, além de desnecessário pois a infraestrutura existente é reutilizável, teriam custos adicionais, como leitores, gravadores e máquinas para fabricação do cartão (USP, 2017; UERJ, 2019).

Figura 4 – Processo da fila proposto



Fonte: Autoria própria (2022).

Mesmo desconsiderando o importante fator de higiene em questão, alguns pontos podem ser melhorados com a implementação de um sistema mais inteligente: maior organização nas filas, resolvendo os transtornos mencionados anteriormente; eliminação dos atrasos gerados quando é preciso pegar e entregar as fichas, proveniente da automatização gerada pelos créditos; problemas durante o período próximo a troca de gerencia, pois os alunos que tiverem um saldo de créditos, não os perderão, já que o sistema continuará em uso mesmo após a troca. Assim tornando a vida dos alunos e servidores que frequentam o RU diariamente muito mais prática e ágil.

1.1 Objetivos

A seguir estão relatados o objetivo principal desse projeto e os objetivos específicos.

1.1.1 Objetivo geral

Este projeto tem como objetivo geral realizar a implementação de um ambiente completo de gerenciamento de crédito para o RU, que permita a aquisição e utilização de créditos, pelos acadêmicos, servidores e possíveis visitantes, de maneira mais prática, higiênica e eficiente. Além disso, disponibilizar para os responsáveis pelo RU um sistema auxiliar de fácil uso e que supra as necessidades do dia a dia.

1.1.2 Objetivos específicos

- Permitir a gerência do RU realizar a recarga e administração de crédito para seus clientes.
- Disponibilizar um sistema que permita aos operadores visualizar os dados dos usuários que realizem as refeições utilizando o serviço.
- Disponibilizar um sistema que distingua entre alunos com e sem auxílio (bolsistas), evitando a geração de complicações ou transtornos para os mesmos.
- Disponibilizar para os alunos uma forma eficiente de aquisição e utilização de créditos.
- Disponibilizar para os alunos maneiras de verificar os saldos existentes em sua conta.
- Incentivar o uso de métodos de recarga sem o contato, porém sem remover o sistema de compras *in loco* já existente.

2 REFERENCIAL TEÓRICO

Neste capitulo são descritas as tecnologias e metodologias utilizadas para dar suporte e base ao desenvolvimento deste projeto. No Quadro 1 é possível observar a relação das tecnologias e ferramentas utilizadas, suas versões, sites para acesso e um resumo das suas aplicações.

Quadro 1 – Ferramentas e tecnologias utilizadas no desenvolvimento do projeto

Nome	Versão	Disponível em	Finalidade
StarUML	5.0.1	https://staruml.io/	Ferramenta de modelagem
GitHub	2022	https://github.com/	Ferramenta de versionamento
MySQL	8.0.29	https://www.mysql.com/	Sistema de gerenciamento de banco de dados
Python	3.10	https://www.python.org/	Linguagem de programação interpretada orientada a objetos
Flask	2.2.2	https://flask.palletsprojects.com/en/2.2.x/	Framework para desenvolvimento web escrita em python
Flutter	3.0.0	https://flutter.dev/	*Software Development Kit* (SDK) para o desenvolvimento multiplataforma

2.1 Metodologias Ágeis

Com a imensa demanda no mundo para aplicativos, web-sites e softwares no geral, metodologias que antes eram muito utilizadas, por serem focadas em documentação e baseadas no contexto e ritmo de suas épocas, se tornam obsoletas(PRESSMAN; MAXIM, 2021). Planejamentos extensos, documentação longa e testes demorados não são mais possíveis com a agilidade e agressividade do mercado e suas constantes mudanças (SOMMERVILLE, 2015).

Isso gerou uma demanda por metodologias que estivessem de acordo com essa nova realidade. A metodologia escolhida para os padrões atuais precisava ser: adaptativa, pois os requisitos podiam mudar no meio do projeto; flexíveis, permitindo todo tipo de alteração necessária durante o projeto; ágil, permitindo que as possíveis mudanças sejam aplicadas de maneira rápida; e que permitisse o rápido lançamento do produto no mercado, pois existem inúmeros competidores que podem também estar desenvolvendo uma solução para o mesmo problema .

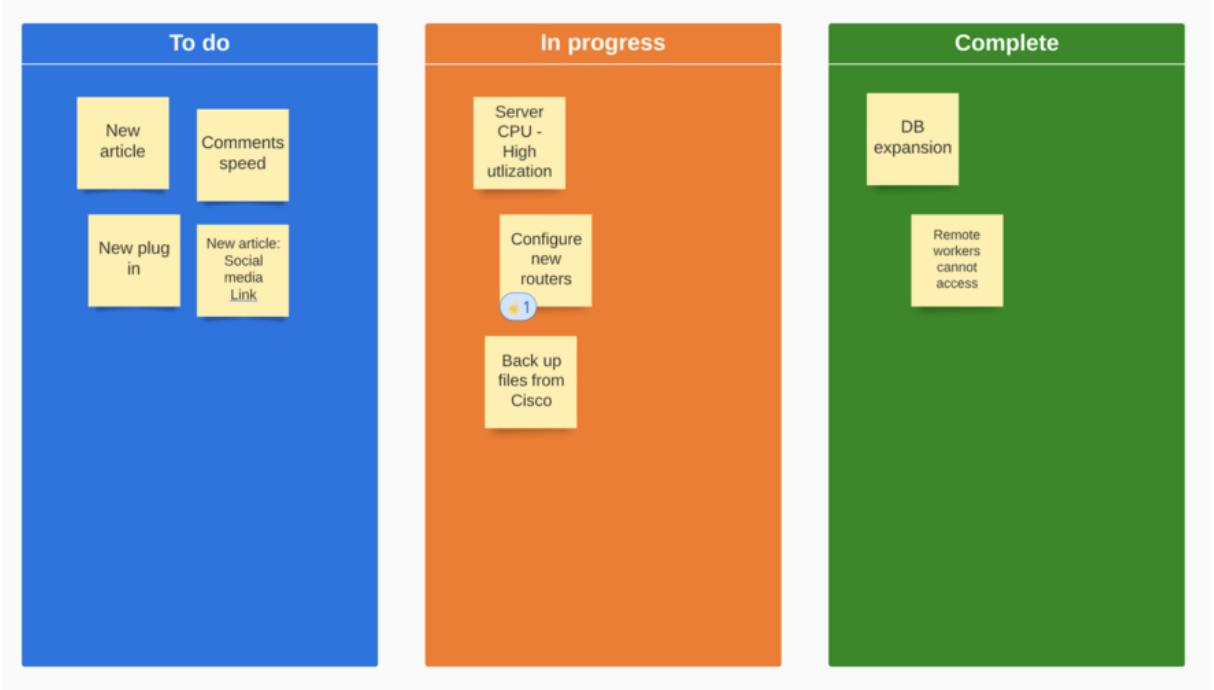
2.1.1 Kanban

Kanban é uma palavra japonesa que significa tabuleta. A metodologia com o mesmo nome teve origem nas fábricas da Toyota e é um subsistema do Sistema de Produção da Toyota (JUNIOR; FILHO, 2010). Taiichi Ohno, engenheiro industrial da Toyota, buscando melhorar a produtividade, desenvolveu o sistema que utiliza notas em papéis para controlar, entre outras coisas, o inventário, componentes e matérias primas da empresa (OHNO; BODEK, 1988). Mais

tarde, essa técnica foi atualizada e trazida para o mundo da tecnologia (POPPENDIECK; CUSUMANO, 2012).

O método faz o uso de cartões que, de maneira visual, representam partes do software que precisam ser finalizadas, esses cartões são organizados em colunas que representam as fases de desenvolvimento. Os cartões vão sendo movidos, da esquerda para direita, de acordo com a fase no qual a tarefa se encontra (LADAS, 2009). Os cartões são chamados de *kanban cards* e o mural, com as colunas, *kanban board*. Um exemplo do quadro e dos cartões, representados em uma ferramenta virtual, pode ser visto na Figura 5.

Figura 5 – Exemplo com quadro e cartões kanban



Fonte: Lucidspark (2022).

O fluxo das tarefas é facilmente visualizado no quadro, ou seja, uma tarefa inicia na coluna mais a esquerda, e, quando elas são concluídas, vão para a próxima coluna (fase), até serem finalizadas. Com isso, as tarefas e o andamento do projeto, são de fáceis visualização para todos os membros envolvidos (ANDERSON, 2010).

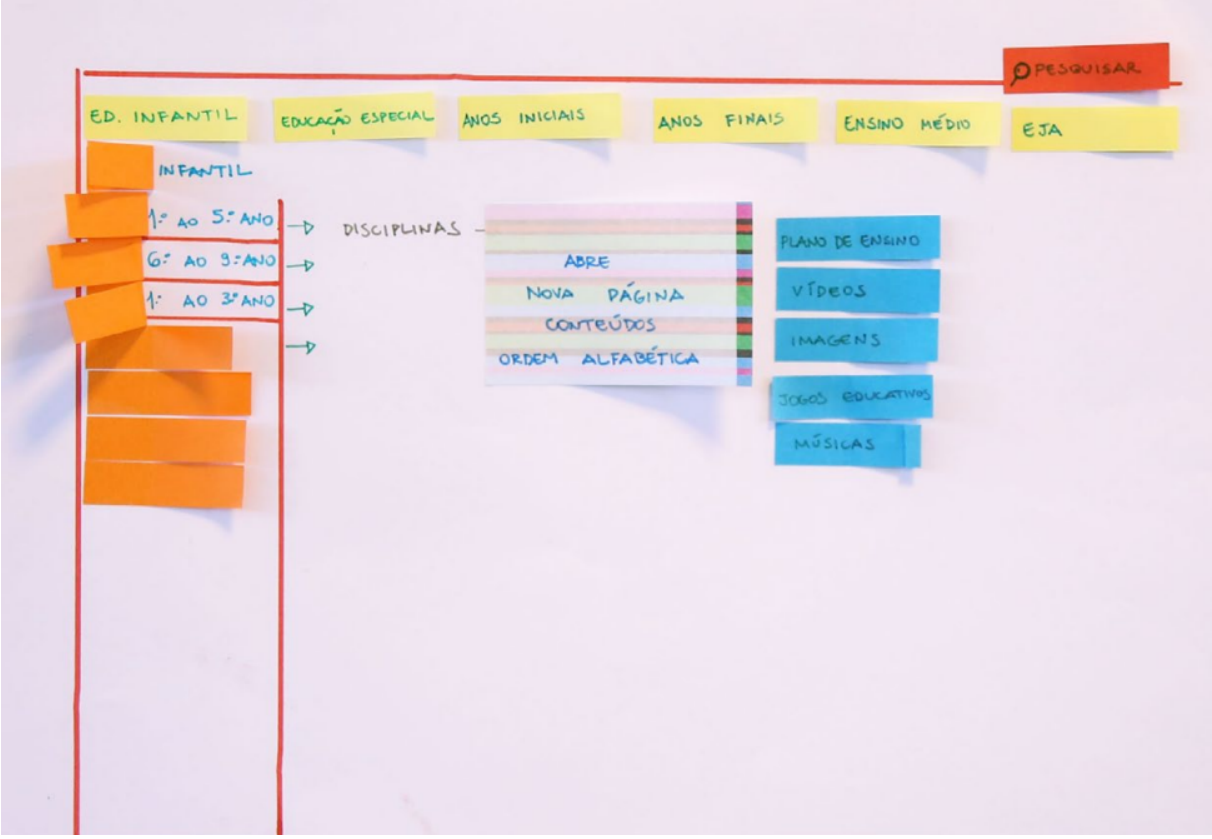
2.2 Ferramentas

Está seção trata das ferramentas utilizadas para o desenvolvimento do projeto, ou seja, ferramentas administrativas, ambientes de desenvolvimento, ferramentas de modelagem e prototipação. Tudo que foi utilizado para dar suporte a evolução do projeto.

2.2.1 Prototipação em papel

Tudo que se é preciso para prototipar em papel são: caneta e papel. Utilizando protótipos desenhados a mão, é possível aplicar os conceitos de *Graphical User Interface* (GUI) sem a necessidade aprender uma ferramenta complexa de prototipagem (BASSO; CHEIRAN; SANTAROSA, 2009). Na Fotografia 2 é possível ver um protótipo em papel feito com a utilização de uma lousa, papéis adesivos e marcadores.

Fotografia 2 – Protótipo em papel feito por professores do ensino fundamental e médio



Fonte: Camata (2018).

Essa técnica tem a finalidade de projetar softwares em suas fases iniciais e, segundo Sefelin, Tscheligi e Giller (2003), gera resultados simples e que não precisam de muito tempo para serem desenhados. Além disso, a prototipagem em papel, permite que diferentes membros do projeto possam colaborar mesmo não tendo um conhecimento avançado na área de design de softwares (SEFELIN; TSCHELIGI; GILLER, 2003; CAMATA, 2018).

2.2.2 UML

A *Unified Modeling Language* (UML) é uma linguagem de notação gráfica, que com a utilização de representações diagramas auxilia no desenvolvimento, planejamento ou estudo de *softwares* (FOWLER, 2004).

Os diagrames UML podem ser divididos em dois tipos principais. O primeiro, os diagramas estruturais, definem características de implementação do sistema, como classes e métodos. O segundo, os diagramas comportamentais, definem características de comportamento do sistema, como interação entre os componentes ou com o usuário (LUCID, 2022; MARTIN, 2003).

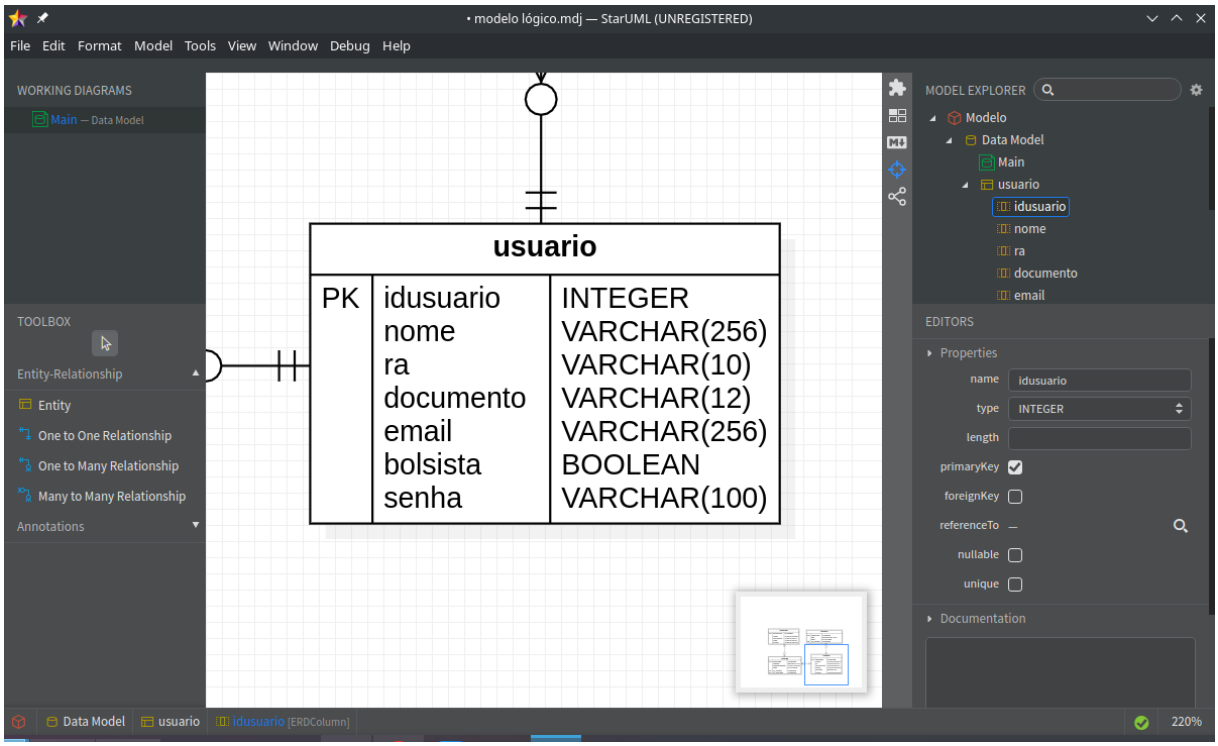
2.2.2.1 StarUML

A modelagem é uma fase importante para o desenvolvimento de um banco de dados futuramente robusto. A partir da modelagem, é possível validar os requisitos e observar, em fases iniciais, o sistema final. E, em caso de erros, o tempo desperdiçado não é significativo.

No caso dos banco de dados relacionais, as relações são definidas e desenhadas. Seus atributos, são rascunhados, para serem melhorados na segunda fase do projeto do banco de dados. Esses modelos servem de auxílio para o desenvolvedor do banco implementar o projeto físico do mesmo.

Software de modelagem com suporte a UML 2.0, diagrama *Entity Relationship* (ER) e fluxogramas. Focado no desenvolvimento ágil de projetos, a ferramenta permite a aplicação de extensões que auxiliam no desenvolvimento StarUML (2022). Na Figura 6 é possível ver a área de trabalho principal da ferramenta.

Figura 6 – Ferramenta de modelagem StarUML



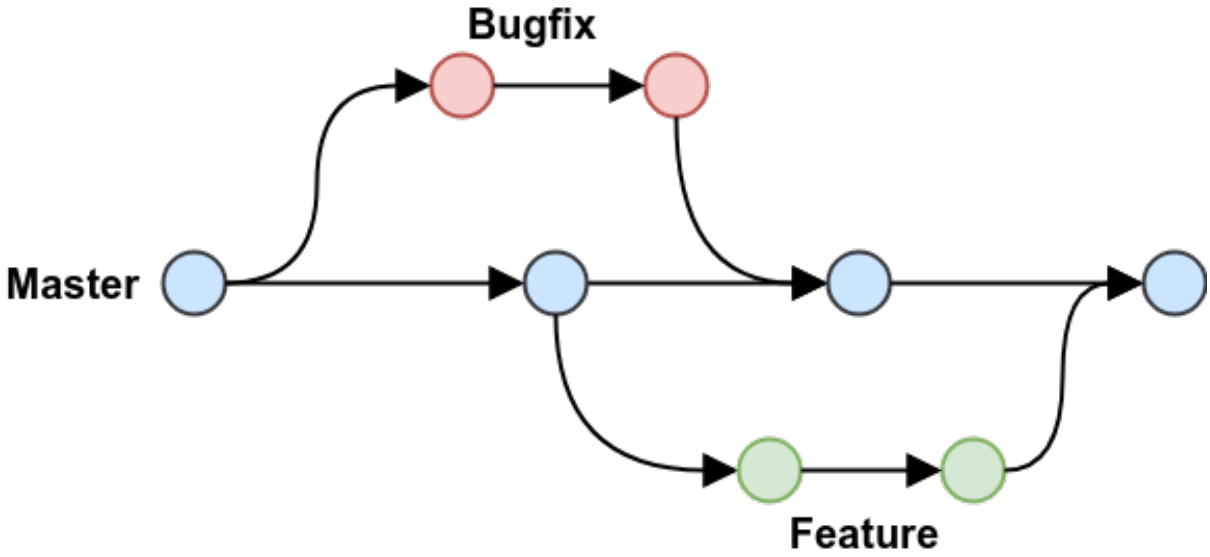
Fonte: Autoria própria (2022).

2.2.3 Controle de versão

Os sistemas de controle de versão, no contexto da computação, são responsáveis por controlar e registrar alterações em arquivos (PRESSMAN; MAXIM, 2021). Os arquivos gerenciados podem ser dos mais variados tipos, como fotos, arquivos de texto simples, trabalhos acadêmicos e, a utilização mais usual, códigos fontes (CHACON; STRAUB, 2014).

Com o versionamento, é possível: Armazenar somente as alterações e não copias inteiras do projeto, economizando espaço; Visualizar versões antigas dos arquivos; facilitar o trabalho de múltiplos profissionais no mesmo projeto; Criar ramos novos, que possibilitam o desenvolvimento de alterações sem afetar o ramo original (ZOLKIFLI; NGAH; DERAMAN, 2018; LOELIGER; MCCULLOUGH, 2012). Uma figura com exemplos das ramificações pode ser vista na Figura 7, onde cada ramo representa uma ação no projeto, e seus respectivos *merges*.

Figura 7 – Ramificação nos projetos Git



Fonte: Dubey (2021).

2.2.3.1 Git

O git, uma ferramenta de controle de versão, surgiu com a necessidade da equipe de desenvolvimento do Linux por um software de controle de versões. Após o termino de uma licença de uso gratuito do software de controle de versão, a equipe desenvolveu o git, tendo seu lançamento oficial em 2005 (CHACON; STRAUB, 2014). Desde então explodiu em popularidade e hoje é o padrão do mercado, tendo suas bases implementadas em diversos plataformas diferentes (MILLER, 2021).

2.2.3.2 Github

Com a popularização do git, diversos serviços utilizaram se da ferramenta para criar suas próprias plataformas de compartilhamento. Como o BitBucket, GitLab, SourceForge e, o mais popular, GitHub (PRAKASH, 2020). O Github é uma plataforma online, que mistura parte rede social, parte controle de versões e parte site para compartilhamento de softwares de código aberto (GITHUB, 2022a).

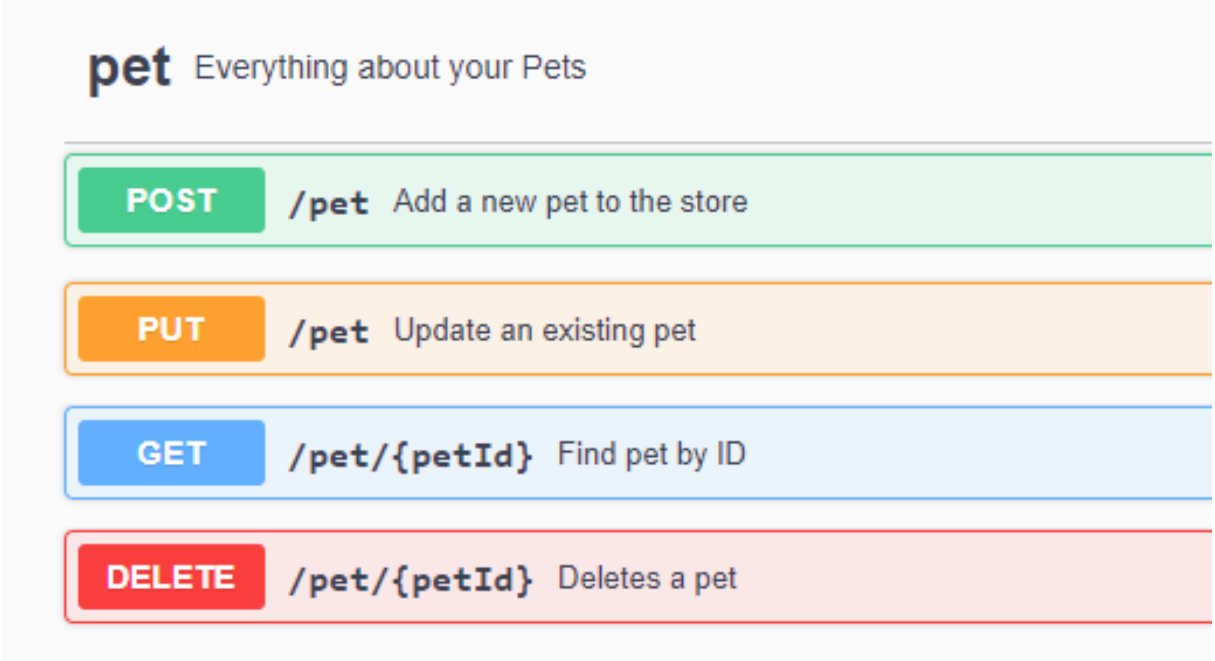
Disponibilizando diversos serviços alternativos, como: organização de projetos (Kanban); Fóruns de discussão; serviços de reportagem de *bugs* e sugestões de funcionalidades novas; possibilidade de apoiar um desenvolvedor, fazendo doações para o mesmo; criação de equipes virtuais, que funcionam como uma conta para equipe compartilhar e utilizar os serviços (GITHUB, 2022a). Uma das maiores redes sociais de desenvolvimento e código aberto da atualidade, possuí diversos repositórios, possibilitando a criação de algorítimos de aprendizado de máquina e *deep learning* e que aprendem e sugerem sugestões em tempo de programação, como o Copilot (GITHUB, 2022b; COSENTINO; IZQUIERDO; CABOT, 2017).

2.2.4 REST

Introduzido em 2000 por Roy Thomas Fielding em sua dissertação de mestrado, na Universidade da Califórnia em Irvine (FIELDING, 2000). *Representational State Transfer* (REST) é um estilo arquitetural de software que define regras para comunicação entre computadores através de requisições *Hypertext Transfer Protocol* (HTTP) (RODRIGUEZ, 2008). Cada recurso é organizado em *Uniform Resource Identifier* (URI) únicos, e através de métodos HTTP como GET, POST, PUT e DELETE, esses recursos são resgatados, criados, alterados ou deletados (RICHARDSON; RUBY, 2008). Um exemplo pode ser visto na Figura 8, onde são ilustrados caminhos, e os respectivos métodos que cada caminho aceita.

Quando um serviço segue as convenções estabelecidas pelo padrão REST é chamado serviço *RESTful* (RICHARDSON; AMUNDSEN; RUBY, 2013). Os recursos consumidos pela APIs podem ser fornecidos, pelo caminho, por cabeçalhos ou pelo corpo da requisição (RODRIGUEZ, 2008), quando no corpo, seguem o formato *JavaScript Object Notation* (json), com chaves e valores. E os desenvolvedores, normalmente fornecem uma documentação em conjunto com a API, quando não desejam revelar suas implementações internas, que funciona tanto como material de aprendizado como manual de instruções (SOHAN *et al.*, 2017).

Figura 8 – Exemplo com caminhos e métodos de uma API para loja de animais de estimação



Fonte: SmartBear (2022).

2.3 Tecnologias

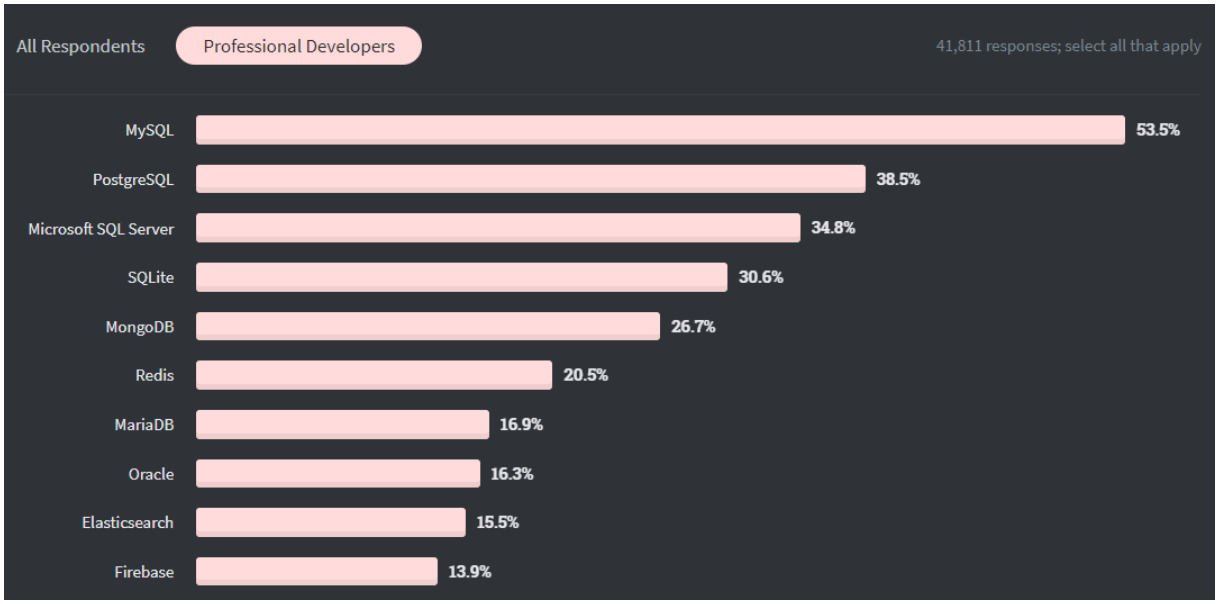
Nesta seção, são apresentadas as tecnologias utilizadas no sistema, ou seja, linguagens de programação, banco de dados, bibliotecas e frameworks utilizadas para elaborar a parte prática do projeto.

2.3.1 MYSQL

Com a criação do modelo relacional em 1970 por Edgar Frank Codd (CODD, 1970), pesquisadores da IBM, criaram a *Structured Query Language* (SQL), como parte do projeto System R e com base no modelo relacional (SILBERSCHATZ *et al.*, 2019; RAMEZ; B *et al.*, 2016). Mais tarde, em 1986, a linguagem foi padronizada pela primeira vez, pela *American National Standards Institute* (ANSI) e, posteriormente, pela *International Organization for Standardization* (ISO) (LIMA, 2007; SILBERSCHATZ *et al.*, 2019). Desde então, a linguagem recebeu diversos outras atualizações e padronizações e, até hoje, continua servindo como base para as tecnologias de banco de dados mais utilizadas do mercado, como ilustrado na Figura 9 em pesquisa realizada pelo Stack Overflow.

Um Sistema de Gerenciamento de Banco de Dados (SGBD) é um sistema utilizado para facilitar a interação entre um programa e um banco de dados (CONNOLLY; BEGG, 2005). O MySQL é um SGBD que utiliza como base o modelo relacional, ou seja, é definido como um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR), e utiliza como interface de interação a sua variação da linguagem SQL (RAMAKRISHNAN; GEHRKE, 2008).

Figura 9 – Tecnologias de bancos de dados mais utilizadas entre profissionais



Fonte: StackOverflow (2020).

Apesar de disponibilizar opções com suporte pago, o SGBD da empresa Oracle, possui código aberto e tem uso liberado sem a necessidade de aquisição de licenças (ORACLE, 2022). Ambas as características incentivam o uso e suporte pela comunidade (WHEELER, 2003), isso pode ser visto na variedade de bibliotecas para *Object–relational mapping* (ORM) disponíveis na literatura (TORRES *et al.*, 2017). Essas bibliotecas facilitam a utilização do SGBD, pois disponibilizam maneiras de se trabalhar com o mesmo sem a utilização de SQL nativo, ao invés disso, o programador trabalha com classes e objetos da linguagem de programação escolhida para o seu projeto (MAKAI, 2022). Além de realizar mapeamento, algumas bibliotecas dão suporte a todas as características básicas do SQL nativo, ou seja, todas as operações de definição, manipulação e *Create, Read, Update and Delete* (CRUD) dos dados, a exemplo temos a biblioteca SQLAlchemy que integra essas características na linguagem de programação Python (BAYER, 2012).

2.3.2 Python

Após obter seu título de mestre na Universidade de Amsterdam, no inicio da década de 80, Guido van Rossum, o criador da linguagem Python, começou a trabalhar no Centro de Pesquisa Nacional em Matemática e Ciência da Computação ou *Centrum voor Wiskunde en Informatica* (CWI) (MICROSOFT, 2021). Sua tarefa inicial era auxiliar o desenvolvimento da linguagem de programação ABC, que tinha como intuito ser acessível para usuários avançadas em computação porém sem histórico prático em programação (Cientistas, Pesquisadores, Engenheiros, etc). O projeto, entretanto, não obteve sucesso esperado e o desenvolvimento da linguagem foi interrompido.

Com o termino do projeto ABC, dá metade pro final da década de 80, o desenvolvedor foi realocado para o projeto Amoeba, um Sistema Operacional (SO) distribuído. O SO ainda estava em fases iniciais, possuindo apenas o *shell* e o compilador (para linguagem C). Sua tarefa, então, era desenvolver um conjunto de aplicações com intuito de preencher o sistema com funcionalidades (VENNERS; SOMMERS, 2003). Como o sistema possuía apenas essa duas opções, para desenvolver uma aplicação, ou trabalhava com os *scripts* em *shell*, ou compilava uma aplicação do zero com o a linguagem C. Segundo ele, ambas as opções não eram agradáveis ou produtivas de se trabalhar.

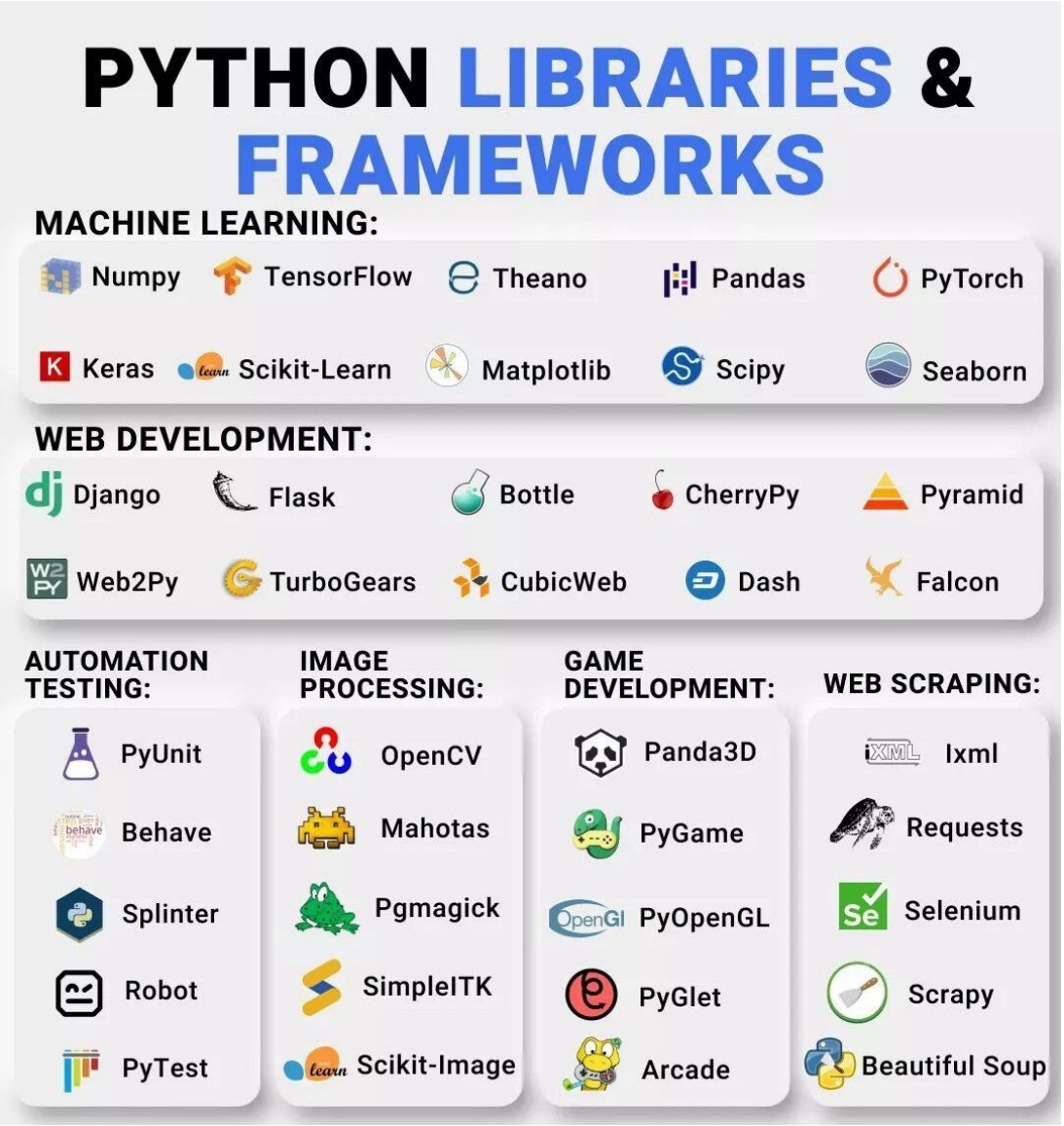
Durante o inicio do seu trabalho no projeto Amoeba, eram recorrentes as lembranças das facilidades e produtividade que a linguagem ABC oferecia. E foi então que o projeto Python começou, com a intenção de ganho em produtividade, apesar de necessário perder um tempo com o seu desenvolvimento, esse tempo seria compensando. Tanto para ele, quanto para todos os membros da equipe (BIANCUZZI *et al.*, 2009). E com a experiencia do projeto ABC em mãos, seus defeitos e qualidades, foi realizado uma tentativa de melhorar o que era ruim e manter o que era bom. O projeto foi um sucesso, a linguagem foi lançada em 1991, e, posteriormente, o seu código fonte foi liberado. O desenvolvedor especula que parte do sucesso da linguagem provém da expansibilidade por parte dos usuários, característica que não era presente na linguagem ABC e que é uma das maiores do Python até hoje, possuindo, atualmente, mais de 400 mil repositórios contendo pacotes de expansão (bibliotecas, módulos, frameworks, etc) (PYTHON, 2022).

Python é uma linguagem de programação interpretada, com suporte a orientação a objetos, programação funcional e procedural, de código aberto (ROSSUM; JR, 2020). Segue propostas de design que recomendam aos programadores exercerem condutas como código legível, bonito, simples e explicito (PETERS, 2004). Teve seu nome inspirado na série de comédia britânica Monty Python. Possui grande aderência da comunidade acadêmica, principalmente nas área de aprendizado de máquina, processamento de dados, entre outros. Sendo muitas vezes a opção de linguagem de programação até para áreas que fogem do escopo direto da computação, como: biologia; astronomia; desenvolvimento de jogos; entre outros (LUTZ, 2013; NAVONE, 2022). Alguns exemplos de extensões da linguagem podem ser visto no cartaz informativo da Figura 10.

2.3.2.1 Flask

Uma micro *framework* web que teve sua origem em uma brincadeira de primeiro de abril (RONACHER, 2011). Devido ao sucesso da piada entre a comunidade, o projeto foi realmente desenvolvido, pelo programador austríaco Armin Ronacher em 2010, e se tornou uma das *frameworks* web mais populares da linguagem Python (JETBRAINS, 2020). A Figura 11 ilustra a popularidade das *frameworks* web Python.

Figura 10 – Exemplos de *Frameworks* e bibliotecas da linguagem Python

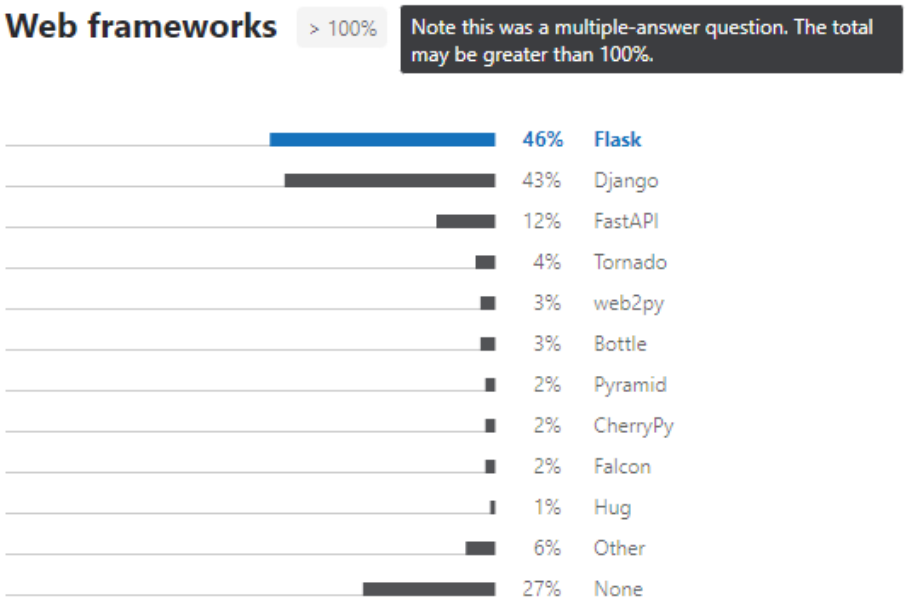


Fonte: Nandan (2022).

A *framework* foi construída em cima de duas bibliotecas base principais (PALLETS, 2010a): A Werkzeug, que implementa as convecções de requisições entre clientes e servidores web, desenvolvidos em Python, da *Python Enhancement Proposals* (PEP) 333 (EBY, 2003; PALLETS, 2007b); E a biblioteca Jinja, que é um motor de modelos web, que auxilia o desenvolvimento de paginas *HyperText Markup Language* (HTML) inserindo suporte a adição de variáveis e chamadas de funções da linguagem Python, dentro do arquivo fonte HTML (PALLETS, 2007a).

Flask é referido como uma ”micro” *framework*, pois não são necessárias, em seu núcleo base, nenhuma ferramentas, bibliotecas ou *softwares* adicionais para se iniciar um projeto (GRINBERG, 2018). A ferramenta não vem, por exemplo, com camada de abstração de banco de dados, ferramentas de validação ou de gerenciamento de sessão, o que é comum em outras *frameworks* web Python, como Django (RELAN, 2019; VINCENT, 2021).

Figura 11 – *Frameworks* web mais populares entre desenvolvedores Python



Fonte: JetBrains (2020).

Tal característica não indica inferioridade, apenas permite ao usuário mais liberdade de escolha, pois a biblioteca é altamente extensível e possui extensões para as mais variadas funcionalidades desejadas, como por exemplo: A extensão Flask-SQLAlchemy, que adiciona suporte a biblioteca SQLAlchemy ao projeto Flask, que por sua vez adiciona suporte a bancos de dados relacionais SQL (PALLETS, 2010b; BAYER, 2012); Ou a extensão Flask-Marshmallow, que adiciona ao projeto Flask, a integração do motor Marshmallow, que é responsável por auxiliar a serialização e desserialização de dados (LORIA, 2022b; LORIA, 2022a).

2.3.3 Flutter

O projeto foi apresentado inicialmente em 2015 sobre o codinome Sky, e no ano de 2017 foi disponibilizado sua versão alfa, recebendo posteriormente o nome de Flutter. No ano seguinte, sua primeira versão estável foi lançada oficialmente,o Flutter versão 1.0 (SNEATH, 2018). Se trata de uma SDK, de código aberto, criada e patrocinada pela Google, contando com um sistema completo de desenvolvimento. Com foco principal no desenvolvimento multiplataforma, a SDK oferece, em suas versões mais atuais, suporte para sistemas como: Android, iOS, macOS, Windows, Linux e Web. Como ilustrado na Figura 12.

Tudo isso compilando nativamente em todas elas e com um único código fonte. A cada ano que passa, a equipe de desenvolvimento, adiciona suporte para mais plataformas, vale ressaltar,que a SDK só foi suportar aplicações Web, com o lançamento do Flutter 2.0 em 2021 (SELLS, 2021). E, no ano seguinte, 2022, as plataformas Windows, macOS e Linux, adicionadas

Figura 12 – Ambientes suportados pela *framework* Flutter

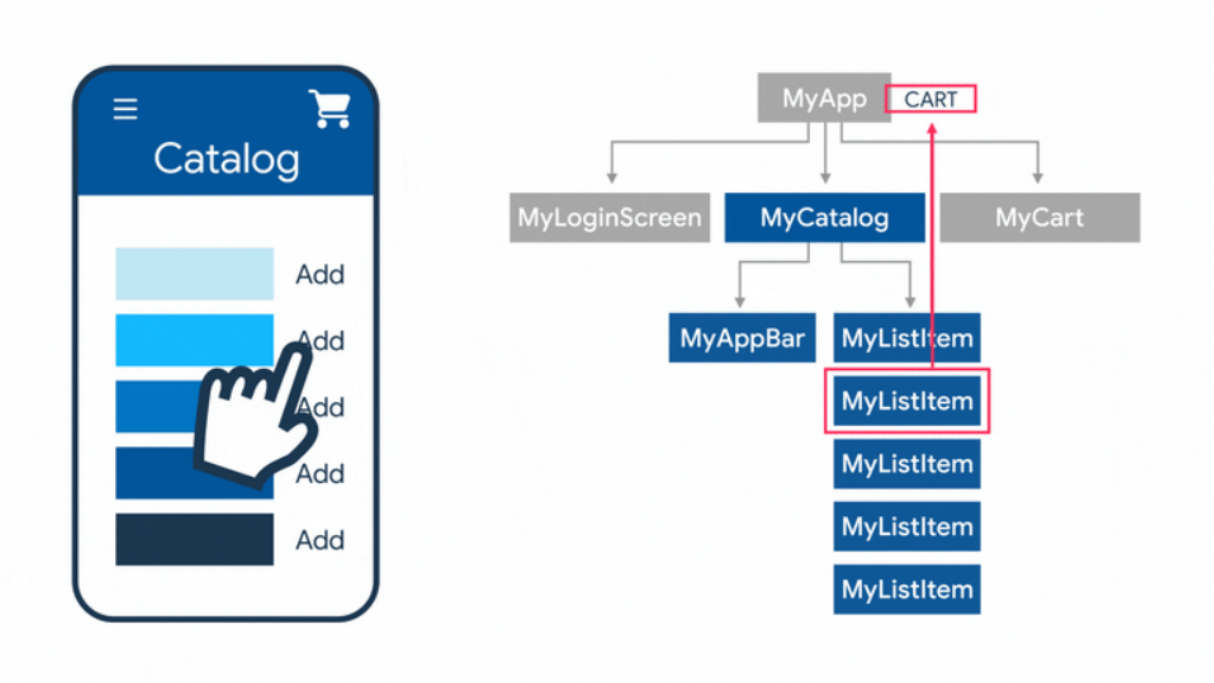
Fonte: Stan (2020).

na versão anterior, foram consideradas estáveis, com a versão 3.0 (FLUTTER, 2022b; SNEATH, 2022b).

Alguns das características principais da ferramenta, além da mencionada anteriormente, são: suporte integrado de ”debugação” com o editor Visual Studio Code; produtividade elevada devido a ferramenta *hot reload* que é responsável por acelerar o processo de desenvolvimento, permitindo a execução e visualização de alterações no código-fonte em segundos; motor de renderização escrito em C++ desenhado para aproveitar o melhor de cada ambiente; a linguagem de programação Dart, que é a linguagem utilizada na SDK; E o conceito de *widgets*; entre outros (SNEATH, 2022a; FLUTTER, 2022a).

Tudo no Flutter é um *widget*, eles definem os mais variados conceitos de *User Interface* (UI). Alguns, por exemplo, são paginas inteiras, outros definem a barra de tarefas ou botões, já outros podem definir o espaçamento entre o *widget* pai e os filhos (WINDMILL, 2020). A organização deles pode ser representada como uma árvore, onde, no começo, está o *widget* raiz, e adiantes, suas ramificações (MELO, 2019). Esse conceito está representado na Figura 13, onde, de maneira simplificada, é possível visualizar a página catálogo, que possui um *app bar* e uma lista de itens, cada item possui uma foto e um botão, quando o botão é clicado o item é adicionado ao carrinho.

Figura 13 – Representação da arvore de *widgets* do Flutter



Fonte: Melo (2019).

3 TRABALHOS RELACIONADOS

O Repositório Institucional da Universidade Tecnológica Federal do Paraná (RIUT) é uma ferramenta voltada para o armazenamento e divulgação de trabalhos acadêmicos produzidos por alunos e servidores da UTFPR (RIUT, 2022). O *website* conta com uma ferramenta de pesquisa, tal funcionalidade pode ser utilizada para pesquisar por trabalhos acadêmicos através de título, autores, palavras-chave, assunto, etc. Realizando uma pesquisa com a palavra-chave ”restaurante universitário”, diversos trabalhos surgem, das mais variadas áreas do conhecimento. Filtrando esses trabalhos por programas e cursos da área de computação, foi possível encontrar três trabalhos que abordam soluções semelhantes.

As publicações selecionadas foram elaboradas por discentes de outros campus da UTFPR. Mesmo com diferentes abordagens, todas apresentam soluções para o mesmo problema: A tentativa de melhorar e prover uma solução mais tecnológica e eficiente no atendimento dos restaurantes universitários, em seus respectivos campus. Todos os trabalhos serviram como inspiração e foram utilizados para dar suporte ao desenvolvimento desse projeto. As ideias principais dos trabalhos são descritas nas seções adiante.

3.1 Sistema web para controle de créditos em um restaurante universitário

O primeiro trabalho, publicado pelo campus de Pato Branco, propôs um sistema web, desenvolvido utilizando: a linguagem de programação Java; o conjunto de *frameworks* de desenvolvimento Spring, que agilizam o desenvolvimento através da injeção de dependências pré configuradas (JOHNSON *et al.*, 2004); o SGBD PostgreSQL, que é um Banco de dados objeto-relacional (BDOR), robusto e de código aberto (DRAKE; WORSLEY, 2002); entre outros.

O autor inicialmente descreve sobre os atrasos que ocorrem diariamente na fila do RU do campus estudado. Comenta que uma das possíveis causas são relacionadas a problemas com trocos, que são causadas pelo uso do papel-moeda (BATISTELLA, 2017). Destacou, que almoço e janta são as refeições mais consumidas em seu campus e como os valores das alimentações são subsidiados pelo governo, auxiliando os alunos. Segundo o autor, no momento da publicação do trabalho, não existia um sistema de fichas em seus campus, os clientes pagavam por suas refeições diariamente após permanecerem na fila. Observa que com a utilização de fichas haveria uma melhora, porém, com um sistema computacional além da melhora nas filas, seria possível facilitar o controle de outros fatores administrativos do restaurante (BATISTELLA, 2017).

Durante a apresentação é demonstrado que o sistema desenvolvido possui cadastro dos usuários, cursos e pratos, realizado por um administrador do sistema e destaca que um usuário comum só pode visualizar seus dados e o prato do dia. Na conclusão do trabalho são relatadas as dificuldades encontradas durante o desenvolvimento, e sugerido que trabalhos

futuros poderiam utilizar o crachá do aluno para melhorar ainda mais a eficiência do sistema (BATISTELLA, 2017).

3.2 RU Digital

Já a segunda publicação, do campus de Guarapuava, com o sistema resultante denominado ”RU Digital” (FILHO, 2018), propôs um sistema web, desenvolvido utilizando: a linguagem Java; o conjunto de *frameworks* Spring; o SGBD MySQL; a biblioteca Thymeleaf que auxilia no gerenciamento de modelos de linguagem de marcação, como o *Extensible Markup Language* (XML) e o HTML (THYMELEAF, 2022); entre outros (FILHO, 2018).

Inicialmente é comentando como as filas afetam negativamente a experiencia dos alunos durante a utilização do RU e comenta como houve uma tentativa de melhorar esse quesito, em seu campus, oferecendo suporte a compras antecipadas com a utilização de planilhas e fichas, salientando que esses métodos ajudam parcialmente, entretanto são suscetíveis a erros. Ademais, demonstra que os maiores atrasos ocorrem em razão dos trocos (FILHO, 2018). O autor enfatiza que, além de resolver o problema de trocos, um sistema permitiria ao usuário uma melhor administração de seus gastos e visualização de seu histórico no restaurante. O acadêmico demonstra, que em pesquisa realizada para verificar o interesse dos alunos em tal sistema, a maioria (91,5%) aprovam a ideia. O gráfico dessa pesquisa pode ser visto no Gráfico 1.

Gráfico 1 – Interesse dos alunos, campus Guarapuava, em um sistema de créditos no RU.

Fonte: Filho (2018).

Durante a apresentação do trabalho, é demonstrado como o sistema foi divido em quatro áreas de acesso. Essas áreas dividem as funcionalidades do sistema em níveis de acesso, ou seja, a área administrativa não pode ser acessada por um gerente, e a área gerencial não

pode ser visualizada por um usuário e assim suscetivamente (FILHO, 2018). Adicionando uma camada de segurança, sendo essa baseada em funções (SANDHU, 1998). A áreas de acesso estão listadas a seguir:

- Administrativa
- Gerencial
- Usuário
- Visitante (usuário sem autenticação)

Durante a conclusão é comentando como o sistema desenvolvido é flexível e não precisa ser empregado apenas em restaurantes, e sim em qualquer estabelecimento que necessite de créditos, como por exemplo terminais rodoviários (FILHO, 2018). Além disso destaca como, tanto os administradores, quanto os clientes podem acompanhar os dados sobre saldos e consumos. E sugere que trabalhos futuros podem implementar crédito compartilhado (diferentes estabelecimentos usando o mesmo saldo) e métodos de pagamentos *online* (FILHO, 2018).

3.3 Ru-Tec

O terceiro, e último, trabalho selecionado, do campus de Campo Mourão, desenvolveu além de um aplicativo móvel intitulado ”RU-TEC” (OLIVEIRA, 2021), um sistema web para administração. Utilizando para seu desenvolvimento: a linguagem de programação Java; a *framework* Spring Boot; React e React Native sendo ambas *frameworks* de JavaScript, porém a primeira com foco em aplicações web e a outra no desenvolvimento de aplicativos (GACKENHEIMER, 2015; BODUCH, 2017); O SGBD PostgreSQL; entre outros.

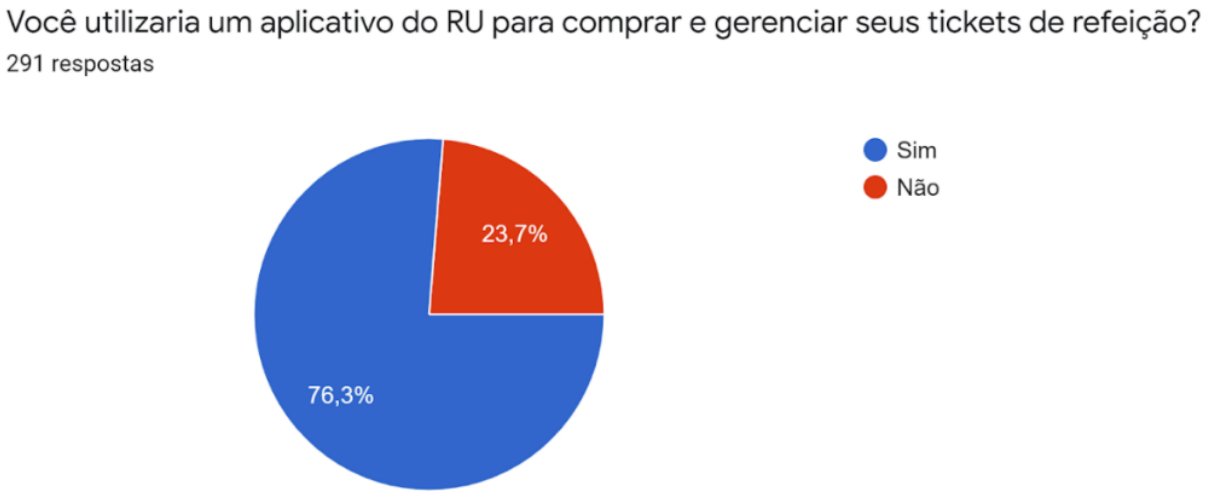
O autor inicia o trabalho comentando como a UTFPR do campus estudado, na data de publicação do trabalho, custeou 56,29% de todas as refeições (almoço e jantar) servidas no restaurante, sendo parte desse auxílio para estudantes bolsistas (total) e estudantes não bolsistas (parcial) (OLIVEIRA, 2021). Além do mais, é destacado a importância cultural do espaço do restaurante, pois as interações que acontecem no mesmo fazem parte do crescimento acadêmico e cultural dos estudantes. Explica como os restaurantes oferecem uma opção alimentar saudável e acessível aos universitários e como o desperdício de alimentos impacta negativamente as expectativas tanto dos clientes como dos administradores, além disso, enfatiza como o número de estudantes universitários cresceu, porém a verba disponibilizada para as universidades não acompanhou esse crescimento, e destaca como é de suma importância uma melhor gestão dado o contexto estabelecido.

O autor segue explicando como o mundo externo à universidade está em constante evolução e como, por tanto, é importante que a universidade se mantenha atualizada com essas mudanças (OLIVEIRA, 2021). E acrescenta que uma parte dessa evolução tecnológica é

proveniente dos aparelhos móveis, que com sua portabilidade, facilidade de manuseio e sua popularização, um desenvolvimento focado nesse contexto seria interessante (OLIVEIRA, 2021).

Durante o desenvolvimento do trabalho, é exposto os resultados de uma pesquisada realizada no campus analisado. Uma das perguntas se refere ao interesse dos alunos em um aplicativo móvel que gerisse as fichas de refeição. Segundo a pesquisa, 76,3% dos alunos demonstraram interesse. O resultado pode ser visto no Gráfico 2

Gráfico 2 – Interesse dos alunos, campus Campo Mourão, em um aplicativo do RU.



Fonte: Oliveira (2021).

E conclui o trabalho comentando como, infelizmente, não foi possível realizar testes da aplicação de maneira presencial, pois a finalização do trabalho coincidiu com o começo da pandemia, e como não foi possível realizar a implantação do sistema, pelo mesmo motivo (OLIVEIRA, 2021).

3.4 Pontos em comum

Com a leitura dos trabalhos comentados previamente, foi possível analisar pontos em comum levantados nos trabalhos. Algumas das pesquisas e observações apresentadas pelos autores podem ser generalizadas para o campus Ponta Grossa, pois o contexto é muito parecido. Os pontos que são semelhantes nos trabalhos avaliados estão listados a seguir:

- Fila: todos os trabalhos avaliados realizaram duras criticas as filas e a demora causada por elas.
- Troco: Foi possível perceber um desagrado geral em relação ao tempo investido na administração do papel-moeda.
- Tecnologia: Sistemas manuais de controle (com fichas ou não) se demonstram ineficientes, e podem ser melhorados.

- Interesse dos alunos: Pesquisas realizadas nos trabalhos demonstraram interesse, em um sistema gerenciador de créditos.

3.5 Funcionalidades ausentes

Nas sugestões de trabalhos futuros apresentadas nas pesquisas descritas neste capítulo, foram escolhidos algumas funcionalidades não implementadas por elas, para serem desenvolvidas por este trabalho. Esses pontos estão listados a seguir:

- Métodos de pagamento *online*: O sistema precisa contar com métodos de pagamento online, além do presencial, disponibilizando flexibilidade para o usuário final.
- Leitura do crachá: Como forma de agilizar o processo e poupar custos com materiais adicionais (cartões) utilizando o crachá do aluno para identificação do mesmo no sistema.
- Validação com a equipe do RU: o sistema deve contar com a avaliação pela equipe de gerência do restaurante, afim de validar os requisitos levantados e as escolhas de *design* realizadas.

4 DESENVOLVIMENTO DA SOLUÇÃO

4.1 Levantamento de requisitos

Durante a fase de levantamento de requisitos, como forma de entender melhor o outro lado do cenário, foram realizados duas atividades com a equipe de gerência e funcionarias do RU atual. A primeira, uma entrevista, argumentando sobre o tema (sistema para o RU), a segunda, devido algumas noções obtidas na entrevista, foi uma apresentação do sistema utilizado atualmente. Vale lembrar que a gerência do RU pode mudar a cada dois anos, mudando também os interesses, com isso é necessário focar nas noções que não são especificas da gerência atual.

Durante a entrevista, foram realizadas perguntas sobre o interesse da equipe em um sistema de que substituísse as fichas e tornasse o processo mais atualizado, no qual foi obtido resposta positiva. Seguido disso, foi questionado sobre o porque das fichas e se essa técnica era alguma determinação da universidade, foi explicado, pela gerente, que as fichas são usadas apenas por sugestão e costume dos alunos em comprar fichas antecipadamente (devido a gerências anteriores), e que a equipe já pensou em realizar a aquisição de um sistema com totens de auto atendimento, porém a ideia não foi finalizada, visto que o custo para implementação e implantação do mesmo seria muito elevado.

Quando questionado sobre quais dados do aluno a empresa precisava guardar, foi respondido que apenas o RA era necessário, o sistema da UTFPR se encarregava do resto. E então o sistema atual da universidade foi apresentado. Através das observações feitas, foi possível averiguar que o mesmo tem a finalidade de informar alunos bolsistas (não precisam entregar nenhuma ficha) e para registrar o consumo dos alunos para futuros repasses de pagamentos da universidade para a empresa terceirizada. Também foi informado que o aparelho de leitura do RA, o computador e o sistema são propriedades da UTFPR, e que a gerência do RU apenas utiliza o sistema e não tem poder nenhum sobre o mesmo.

4.2 Projeto do banco de dados

Seguindo os conceitos de modelagem, apoiados na literatura, estão relatadas as etapas do projeto de banco de dados. Com as fases: conceitual, lógica e física, e seus respectivos artefatos.

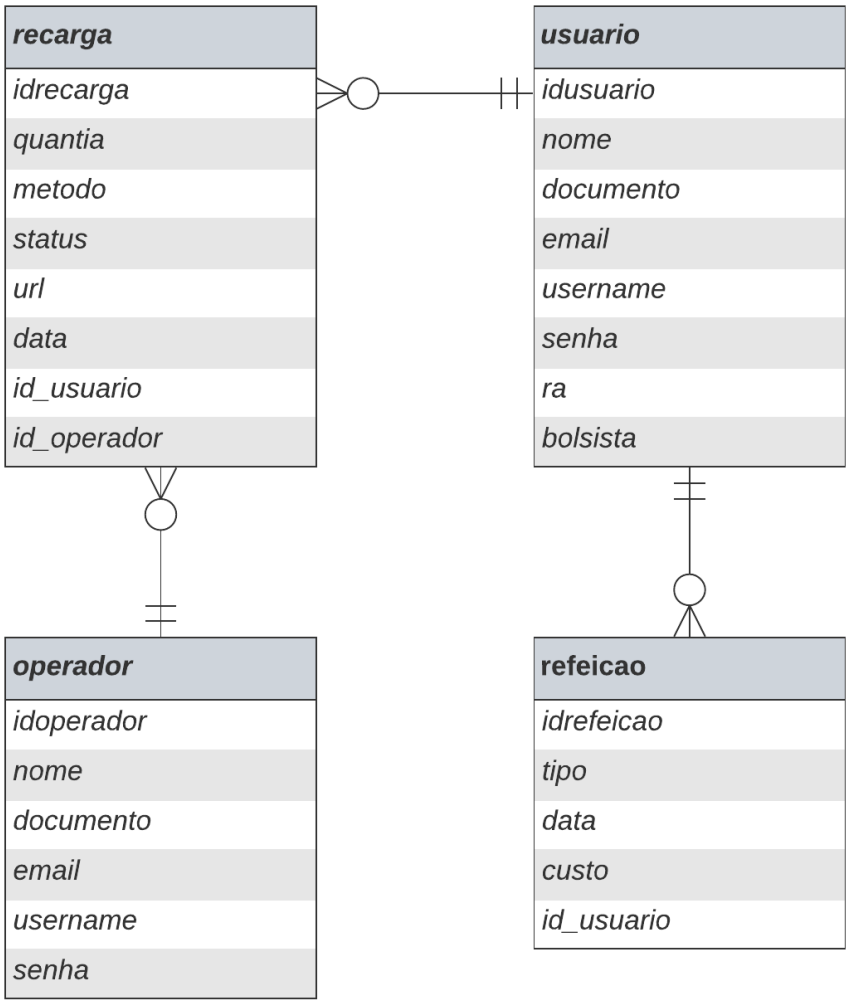
4.2.1 Projeto Conceitual

Através das imersões, reunião e experiência pessoal do RU todos os dias, foi possível identificar o seguinte cenário. Foi identificado dois participantes principais para o projeto:

operadores, que são os responsáveis por vender as fichas e ler o crachá; e os alunos, que são responsáveis por adquirir as fichas e realizar a utilização do restaurante. Constatado esses dois participantes, foram abstraídas mais duas ações, a de recarga e a de alimentação, ambos no processo atual são realizados de forma manual, representados respectivamente através da venda e entrega das fichas.

O modelo conceitual de alto nível que abstraí esse cenário, pode ser visto na Figura 14. Onde, todas as entidades abstraídas, representam essas ações processo atual. Operador, Aluno, Recarga e Refeição.

Figura 14 – Modelo entidade relacionamento do banco de dados

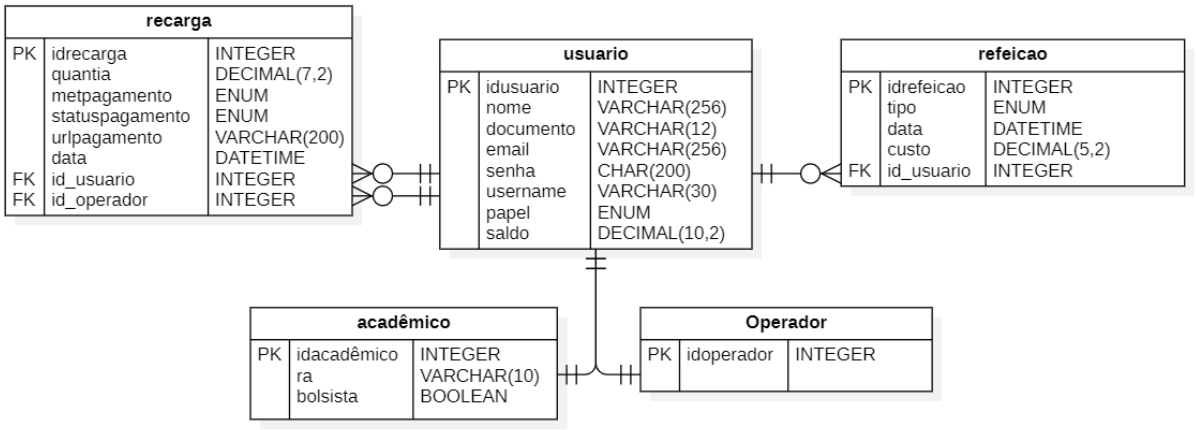


Fonte: Autoria própria (2022).

4.2.2 Projeto Lógico

O modelo relacional, a aprimoração do modelo entidade relacionamento visto na subseção 4.2.1, do banco de dados pode ser visto na Figura 15. Algumas das características em comum das entidades identificadas foram generalizadas em uma relação chamada "usuário", e, atributos que são únicos de cada uma delas, foram deixados em especializações com os mesmos nomes ”Operador” e ”Acadêmico”. Onde o primeiro representa os funcionários que podem realizar a ação de recarga de créditos *in loco* e a leitura do RA do aluno ou professor. E o outro representa de forma principal os alunos, que são os utilizadores principais do RU, deixando a representação de visitantes, servidores ou terceiros para a relação ”usuário” simples, caso desejem ter uma conta no sistema.

Figura 15 – Modelo relacional do banco de dados



Fonte: Autoria própria (2022).

A relação ”recarga” associa a compra de crédito que os usuários fazem e suas contas, através dela e da entidade refeição sera criada uma *view* que representara o saldo disponível na conta de um determinado usuário. A respeito do saldo, existem duas alternativas para se calcular o saldo total do usuário, reservando uma coluna especifica e alterando-a a cada recarga, ou calculando o total através de consultas nas tabelas recarga, que contêm os saldos positivos, e refeição, que contêm os saldos negativos, apesar da segunda opção ainda ser possível, pois o sistema contaram com chamadas que devolvem esses valores, como veremos no seção 4.3, optamos, pela primeira opção, onde um atributo ”saldo” foi registado no campo usuário, e sera alterado, tanto em recargas (adição) quando na realização da alimentação (subtração), conforme necessário.

A relação ”refeição”, além de representar o tipo de refeição que o aluno consumiu, ou seja, almoço ou janta, que são as duas principais refeições servidas no campus até o momento. Guarda também, a informação de valor que será subtraído da conta do usuário. Esse valor, a princípio, podera assumir três valores pré-determinados:

1. Quando o usuário não é acadêmico.

2. Quando o usuário é um acadêmico, ou seja, parte do custo é subsidiado pelo governo.

3. Quando, além de acadêmico, o usuário é bolsista, tendo todo o custo de uma refeição subsidiado pelo governo.

4.2.3 Projeto Físico

Com a utilização da biblioteca ORM SQLAlchemy, foi necessário escrever apenas um comando em MySQL nativo, a criação do banco de dados, apenas informando o nome do mesma. Além desse nenhum outro comando foi necessário, como, por exemplo, criações de tabelas ou definições de restrições de chaves primárias. A biblioteca tomou conta de tudo com as definições dos modelos e a configuração de conexão com o banco. Nos arquivos de configurações do servidor, foi necessário apenas especificar o endereço do banco, sua porta, um usuário e uma senha. Por motivos de segurança, foi criado um usuário com acesso apenas ao banco de dados do projeto, assim, não fornecendo concessões de super usuário para possíveis invasores utilizando possíveis brechas desconhecidas.

A biblioteca, realiza a criação das tabelas a partir da definição das classes do sistema. A seguir, a titulo de exemplificação, na Listagem 1, está listado a classe que define e representa a tabela refeição.

Primeiramente foi definido uma classe para representar o campo do tipo *ENUM*, previamente modelado, que sera utilizado na tabela para indicar o tipo da refeição (almoço ou janta). No exemplo foi utilizado nomes idênticos para os campos e seus valores, porém isso não é obrigatório e foi feito apenas parar simplificar a utilização.

Após isso o modelo refeição é definido. Estendendo uma classe definida na inicialização da biblioteca SQLAlchemy, a classe (Refeição) herda todos os métodos necessários para realizar todos os comandos de CRUD. Herda também os tipos específicos do banco, como tipo *DateTime* ou tipo numérico. Todos esses métodos funcionam de acordo com as definições inseridas nos atributos adiante.

Com isso definimos sua chave primária, ligando uma *flag* especifica e indicando que ela é do tipo *auto increment*. Atribuímos a enumeração previamente definida. O campo data, que consulta a data e hora atual do sistema, no momento da criação do objeto. O custo da refeição, que pode assumir valores diferentes para alunos bolsistas (zero), não bolsistas (valor subsidiado) e visitantes (valor inteiro). E definimos o relacionamento com a tabela usuário, indicando a presença de uma chave estrangeira. O método construtor, define os valores que são recebidos quando um objeto é inicializado, a ausência da chave primária se dá pela característica de incremento automático e, no caso, da data por ser resgatada automaticamente no relógio do SO.

A última classe define o esquema, que é utilizado, com auxílio da biblioteca Marshmallow, para serialização do formato json para objetos, ou a desserialização, no caminho oposto.

Listagem 1 – Definição do modelo refeição

```
...
class Tipo(str, enum.Enum):
    ALM = 'ALM'
    JNT = 'JNT'


class Refeicao(db.Model):
    idrefeicao = db.Column(db.Integer, primary_key=True, autoincrement=True)
    tipo = db.Column(Enum(Tipo), nullable=False)
    data = db.Column(db.DateTime, default=datetime.datetime.now())
    custo = db.Column(db.Numeric(5, 2), nullable=False)
    id_usuario = db.Column(db.Integer, db.ForeignKey('usuario.idusuario'),
    nullable=False)

    def __init__(self, tipo, custo, id_usuario):
        self.tipo = tipo
        self.custo = custo
        self.id_usuario = id_usuario


class RefeicaoSchema(ma.Schema):
    class Meta:
        fields = ('idrefeicao', 'tipo', 'data', 'custo', 'id_usuario')
...
```

Fonte: Autoria própria (2022).

Essa classe é utilizada, por exemplo, para auxiliar na montagem do corpo das requisições do formato REST de maneira automática. Como, no caso da refeição, todos os campos são utilizados pelas requisições, todos estão presentes, mas isso não é obrigatório, já que é possível existir dados que não são enviados ou recebidos mas estão presentes na tabela.

Todo esse processo foi repetido para todas as tabelas identificadas no projeto lógico. Tornando o processo de definição do banco de dados rápido e produtivo. Quando erros aconteciam durante o desenvolvimento do *script*, bastava realizar o *drop* de todas as tabelas e executar o servidor novamente, a biblioteca se encarregava de criar as tabelas com as mudanças de maneira rápida e automática. É possível automatizar esse processo ainda mais, com uma biblioteca auxiliar para a SQLAlchemy, que executa comandos como criação e deleção mais agressivos (*drop database*), porém não foi visto necessidade em usa-la (VESTERINEN, 2022).

4.3 Projeto do *Back-end*

Como a *framework* utilizada para criação do servidor, tem como princípio ser leve e extensível, o projeto, para uma API REST fica bem enxuto, já que não há necessidade de lidar com nenhum modelo HTML e suas respectivas estilizações. Boa parte do trabalho pesado já

é executado pelas bibliotecas. Restando apenas configurar o servidor da maneira desejada, definir os modelos, definir e organizar as regras de negócio.

As configurações base do servidor, seguindo todo o projeto, estão bem concisas. Primeiramente foi gerado uma sequencia de caracteres de maneira randômica, então, uma amostra, definida também de maneira aleatória, é retirada dessa sequencia para ser utilizada como chave de segurança (*SECRET-KEY*), essa chave é gerada toda vez que o servidor é iniciado, sua utilidade é explicada na subseção 4.3.3 adiante. Então é necessário definir o endereço e a porta do servidor. As configurações da biblioteca SQLAlchemy são definidas em seguida, fornecendo para ela: qual banco de dados estamos utilizando (MySQL); o login de usuário e senha de acesso ao SGBD; o endereço e a porta, que foram deixados padrão; e o nome do banco, nesse caso ”rutf”. A ferramente de *logging* é definida durante a inicialização do servidor, auxiliando na ”debugação” do projeto.

4.3.1 Arquitetura

As funcionalidades do servidor foram divididas em quatro módulos principais, cada módulo com suas respectivas rotas (subseção 4.3.2), e cada sub-módulo com suas respectivas funções. Sendo divididos dessa maneira para deixar o projeto mais limpo e organizado (MARTIN, 2009). O módulo *”models”* cuida dos modelos, aqui estão as definições expostas na subseção 4.2.3 referente ao projeto físico, como explicado, de maneira herdada, estão definidas também as funções de acesso ao banco. O módulo *”routes”* cuida das rotas (HTTP), aqui estão definidas as rotas bases, cada rota cuida de funcionalidades especificas, seguindo os princípios REST. É aqui que são definidos quais métodos HTTP serão utilizados em cada rota (*POST*, *GET*, etc), além disso é aplicado também, de maneira individual em cada rota,as restrições de acesso, explicados mais adiante na subseção 4.3.3.

O terceiro e o quarto módulo tem um vinculo diferente, um é auxiliar direto do outro. O terceiro módulo, denominado *”views”* cuida de processar as repostas das requisições. Nele estão definidas as regras de negócio, e como elas devem ser executadas. Porém, foi criado um módulo a mais, o *”utils”* que se trata de rotinas que estariam no módulo anterior, mas, por motivos de organização e limpeza de código, foram ”quebradas” em funções especificas. Isso era feito quando essas funcionalidades especificas cresciam muito. Como por exemplo: encriptações de senhas; formatação das respostas; verificações de custo; etc. Elas eram transferidas, como métodos, para o módulo *”utils”*. Uma exemplificação desses dois módulos pode ser visto na Listagem 3 e na Listagem 2.

No exemplo do módulo *”utils”* (Listagem 2), podemos ver duas funcionalidades especificas referente as refeições. A primeira (linha 2) cuida da validação das informações e montagem da resposta, a segunda (linha 15) cuida do calculo do valor da refeição.

Primeiramente (linha 4) a existência do usuário é verificada. caso usuário não exista, o erro HTTP 404 é levantado imediatamente, indicando que o usuário não foi encontrado. Então

Listagem 2 – Exemplo do módulo ”utils”

```
1  ...
2  def create_refeicao_routine(username, tipo):
3      usuario = UsuarioUtils.get_usuario_pelo_username(username)
4      if not usuario:
5          abort(404, 'Usuario não encontrado')
6
7      custo = __get_custo_refeicao(usuario, tipo)
8
9      if not UsuarioUtils.subtrair_saldo_usuario(usuario, custo):
10         abort(400, 'Saldo insuficiente')
11
12     return Refeicao(tipo, usuario.id, custo)
13
14
15 def __get_custo_refeicao(user, tipo):
16     if user.ra:
17         if user.bolsista == 'AMB' or user.bolsista == tipo:
18             if not __get_refeicoes_pelo_tipo(user.id, tipo):
19                 return Decimal(0.00) #bolsista
20
21         return HelperUtils.get_preco_refeicao_estudante() #estudante
22
23     return HelperUtils.get_preco_refeicao_visita() #visitante
24 ...
```

Fonte: Autoria própria (2022).

o custo da refeição é resgatado (linha 7), o custo da refeição é definido de acordo com o tipo de usuário (aluno, aluno bolsista, visitante) possuindo um valor especifico para cada situação, é o tipo da refeição (almoço ou janta). No caso de estudantes bolsistas, o valor é definido como zero, caso seja sua primeira refeição do mesmo tipo naquele dia de acordo com a seu auxílio (só almoço, só janta, ambos). Então a rotina que subtraí o custo do saldo do usuário é executada, caso usuário não tenha saldo disponível, o erro HTTP 400 é retornado, indicando que o usuário não possui saldo suficiente. O objeto refeição é montado, e retornado para o módulo visão.

O resgate do valor é verificado na segunda funcionalidade da Listagem 2 (linha 15). A primeira condicional é aceita caso o usuário possua RA, ou seja, o usuário é um aluno. Então, é verificado se o mesmo é bolsista (linha 17), e se a bolsa cobre o tipo de refeição atual (almoço ou janta). Só então (linha 18) é verificado se o aluno bolsista já utilizou o seu beneficio naquele dia. Isso é feito pois o aluno bolsista possui auxílio somente para uma refeição dos tipos beneficiados a ele, porém isso não o exclui de se alimentar novamente caso deseje. Caso o aluno não seja bolsista, o valor para alunos é resgatado e retornado. E, caso o usuário não seja nem aluno, o valo para visitantes é retornado.

Ao longo desse exemplo, podemos ver a utilização do módulo *”utils”* referente aos usuários (linhas 3 e 9) e aos auxiliares (linhas 21 e 23). Os contexto de auxiliares sera explicado

mais profundamente na subseção 4.3.2. Indicando que as visões especificas não estão presas as suas respectivas utilidades, e vão ser acessadas por outras visões. Outra função do módulo de utilidades: funções que podem ser usadas por outras visões.

Listagem 3 – Exemplo do módulo ”views”

```
1 ...
2 def post_refeicao():
3     ...
4     refeicao_nova = RefeicaoUtils.create_refeicao_routine(usuario, tipo)
5     ...
6     resultado = refeicao_schema.dump(refeicao_nova)
7     return jsonify({'refeicao': resultado}), 201
8 ...
```

Fonte: Autoria própria (2022).

No exemplo do módulo *”views”* (Listagem 3) podemos ver a visão que trata a requisição de postagem de uma refeição nova. Ela é realizada por um operador para cadastrar no sistema que um usuário deseja se alimentar. Os dados da requisição são lidos e entregues a função de utilidades responsável explicada anteriormente. Caso tudo ocorra bem, a refeição é cadastrada no banco de dados. É possível ver em seguida um exemplo de desserialização (linha 6) do objeto para o formato json, e, em seguida, a montagem e retorno da resposta. Com o código HTTP 201, indicando que o processo foi bem sucedido. Nesse contexto, algumas operação são feitas a mais além disso, porém o exemplo foi resumido para se ater ao exemplo.

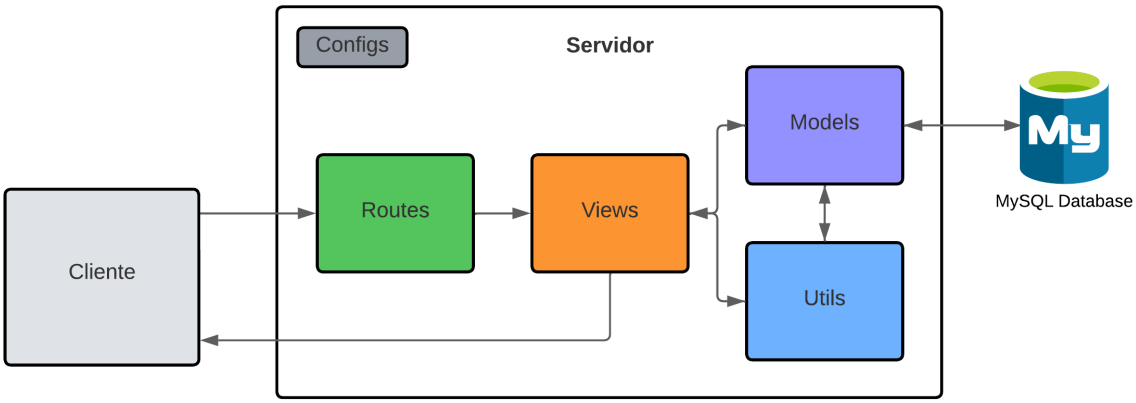
A arquitetura do servidor pode ser vista na Figura 16. Onde são explanados o fluxo dos dados. Recebidos pelo módulo de rotas, vão para as visões, caso algum processamento mais complexo seja necessário, o módulo de utilidades é utilizado. Ao longo disso, consultas no banco, quando necessárias, são realizadas pelos módulos de visão e de utilidades. Não foi representado no diagrama, mas em caso de erros, todos os módulos, com exceção do módulo modelos, podem realizar respostas e abortar os processos.

4.3.2 Rotas

Os recursos principais foram divididos, de acordo com os padrões REST, em funcionalidades especificas. De forma simples, cada entidade principal, aprimoradas na subseção 4.2.2, possui seu próprio recurso (usuários, refeições e recargas). Com suas rotas e seus métodos HTTP, quando necessários, específicos: *POST*, *GET*, *PUT* e *DELETE*.

Alguns métodos de determinados recursos, a princípio, foram deixados de fora intencionalmente. Isso se dá por motivos de segurança, como por exemplo no caso das recargas, que só podem ser canceladas caso ainda estejam em período de ”aguardando pagamento” (recargas online); ou as refeições, que foi determinado um tempo especifico, configurável, em minutos, caso ocorra desistência por parte do aluno ou engano por parte operador.

Figura 16 – Projeto da arquitetura do servidor



Fonte: Autoria própria (2022).

Além dos três recursos principais, outro, auxiliar, foi criado. Esse recurso é utilizado para questões especificas, como: rotas para realizar login; alteração ou consultas no valor das refeições para alunos e visitantes; geração de relatórios, do dia ou gerais; entre outros.

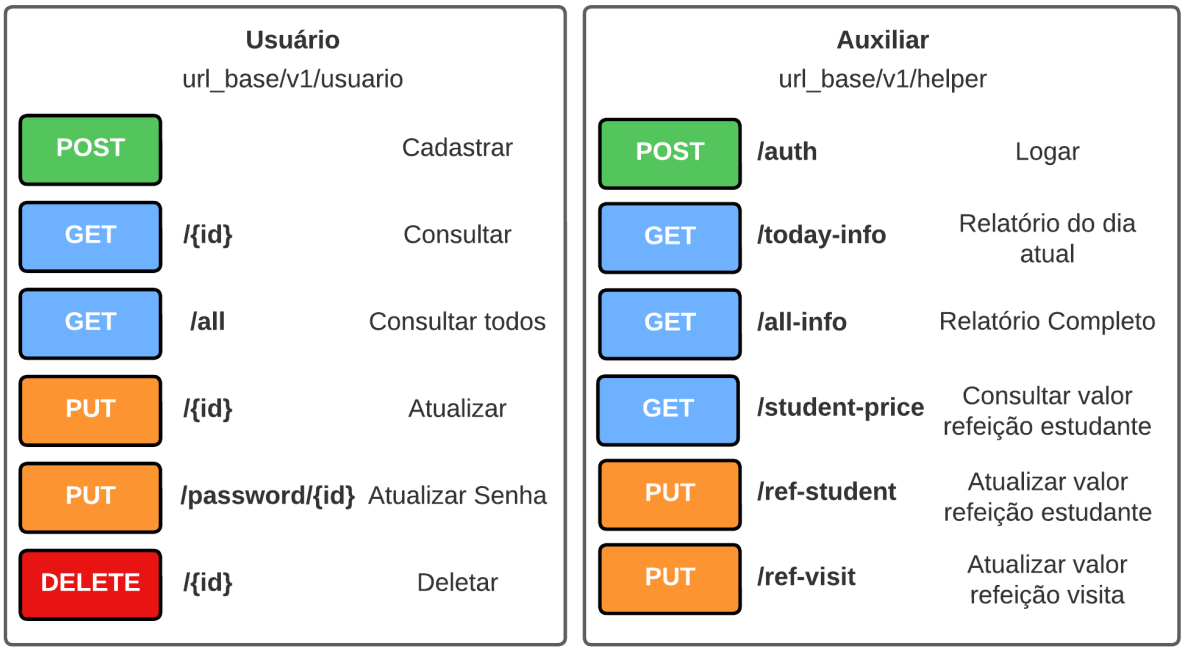
A titulo de exemplo, na Figura 17 estão algumas das rotas disponíveis nos recursos usuário e auxiliar. No caso do usuário, a maioria dos métodos HTTP básicos estão presentes. Mas, na auxiliar, não há a presença, por exemplo, do *DELETE*. Não é necessário utilizar todos os métodos HTTP possíveis caso não haja necessidade. As rotas surgem de acordo com a necessidade, e os métodos de acordo com a padronização. É possível deletar um usuário utilizando qualquer um dos métodos, a ação em si não está associada ao método, mas, por questões de padronização, são utilizadas de acordo com o suas respectivas definições (FIELDING; NOTTINGHAM; RESCHKE, 2022; ARIAWAN, 2019).

Das rotas do usuário, além de todo o CRUD, é possível notar um método *PUT* a mais para atualização da senha, isso é se dá na intenção de adicionar mais uma camada de proteção, evitando, por exemplo, que uma pessoa mal intencionada troque a senha do usuário caso tenha acesso a sua conta, por exemplo através de um aparelho desbloqueado por esquecimento ou qualquer outra situação parecida. A rota que consulta todos os usuários tem informações sensíveis, um usuário comum não pode acessar informações de outros usuários, para isso, foram aplicadas proteções baseadas em *roles* dentro do servidor, explicados detalhadamente na subseção 4.3.3.1 adiante.

4.3.3 Segurança

Por se tratar de questões financeiras, toda proteção possível e viável deve ser aplicada. Diversos passo para proteção foram adotados para garantir a segurança do usuário e evitar problemas futuros. Como se trata de uma aplicação baseada em comunicações com uma API REST, todo método de segurança aplicado deve ser feito no servidor. As limitações inseridas

Figura 17 – Exemplos das rotas usuários e auxiliares



Fonte: Autoria própria (2022).

de forma proposital no *front-end* devem ser apenas para proporcionar a experiência adequada aos tipos de usuários presentes no sistema (operador e usuários).

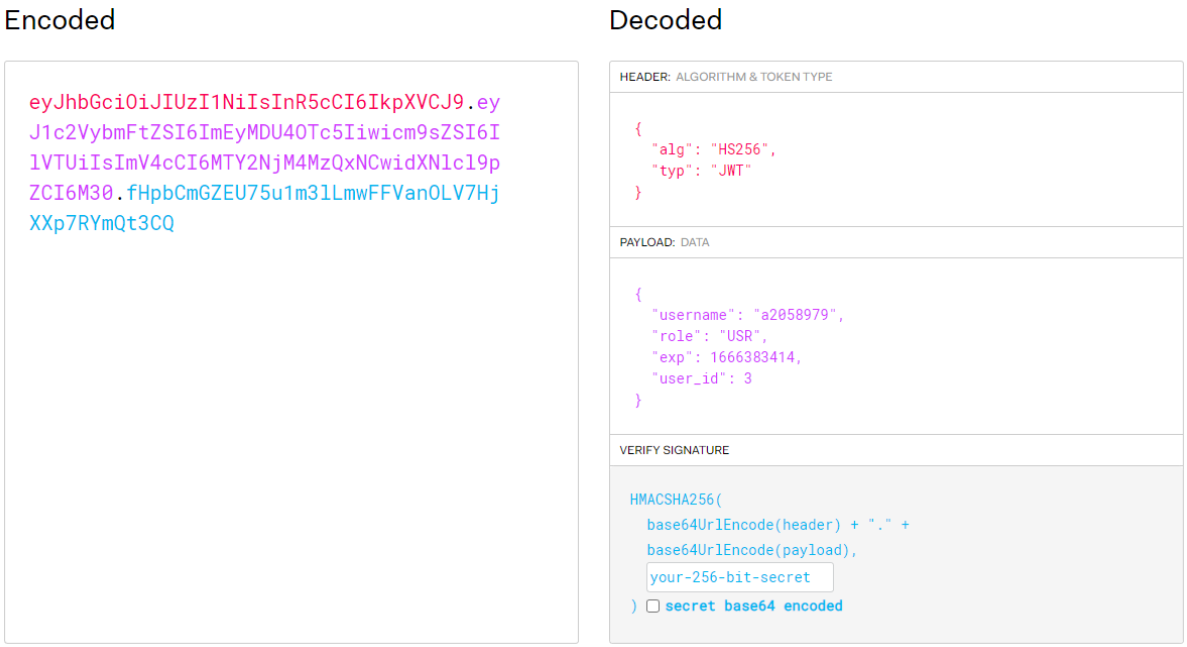
4.3.3.1 Validação nas rotas

Todas as rotas que tratam com informações sensíveis estão protegidas com validação. Essas rotas, estão protegidas de acordo com a função do usuário (operadores ou usuários). Para acessa-las, é necessário um *token* de validação valido. O *token* utilizado é gerado utilizando o padrão JWT, no qual possui três partes principais: o cabeçalho, indicando qual algorítimo de encriptação foi utilizado; a carga, que pode ser informações úteis para aplicação como o id do usuário no banco de dados ou qual sua role no sistema; e o terceiro campo, de validação, que gera uma *hash* codificada utilizando o cabeçalho, a carga e a chave do servidor (*SECRET-KEY*) (PADILLA, 2022; JONES; BRADLEY; SAKIMURA, 2015).

Isso torna a aplicação segura, pois, caso a carga seja modificada, alterando por exemplo a função, o campo de validação não vai ser mais valido, e para gerar um token, é necessário a chave dentro do servidor. Um exemplo de *token* gerado pelo servidor pode ser visto na Figura 18.

Na figura é possível ver, no primeiro bloco do lado esquerdo, o algorítimo de encriptação utilizado, o HS256. A carga útil com as respectivas informações relevantes para o funcionamento do cliente. Vale ressaltar a chave ”exp” que define a duração do *token*, tendo uma validade configurável, adicionando mais uma pequena camada de proteção. E, por ultimo, os dados combinados criptografados.

Figura 18 – Exemplo de *token* JWT gerado pelo sistema



Fonte: Autoria própria (2022).

4.3.3.2 Proteção do banco de dados

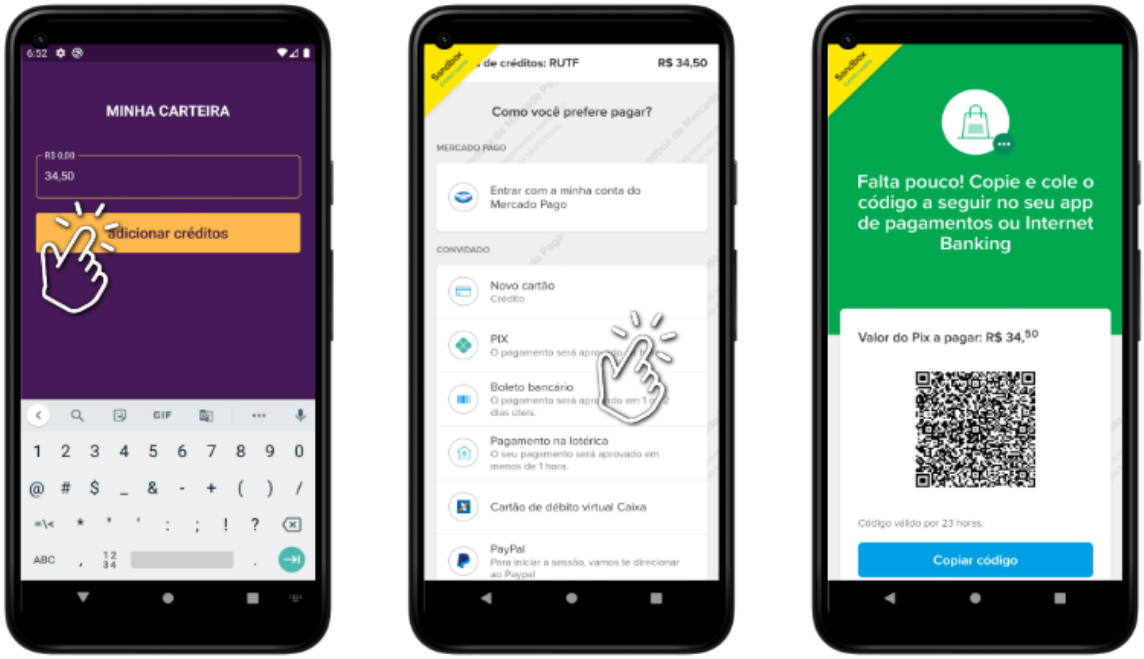
Outra forma de segurança para o banco, além da criação de usuários sem acessos privilegiados mencionadas anteriormente, evitando ataques de *SQL Injection*, que, de maneira simplificada, ocorre quando um usuário mal intencionado digita um comando sql, como ”alter table” ou ”drop database”, dentro de um campo que deveria ser utilizado para digitar, por exemplo, nomes ou campos de e-mail (CLARKE, 2009). Com a utilização da ORM SQLAlchemy, e por recomendação da documentação, nenhum comando puro foi utilizado durante todas as operações de CRUD no sistema. Sendo utilizado apenas as funções disponibilizadas pela biblioteca para o mesmo. Assim protegendo o sistema de ataques, pois a biblioteca trata as informações recebidas, de acordo com seus tipo. Não sendo necessários tratamentos manuais, como analisar todas as strings recebidas de maneira exaustiva, não sendo preciso reinventar a roda. Além disso, ainda tratando do banco de dados, a senha dos usuários é armazenadas no banco utilizando encriptação SHA-256, evitando que possíveis vazamentos liberem as senhas dos usuários.

4.3.3.3 Pagamentos online

Sobre os pagamentos onlines, a abordagem utilizada foi: não guardar nenhuma informação financeira do usuário. O pensamento aqui não foi em transferir a responsabilidade, e sim reconhecer a experiência e cuidados que se deve ter com esse tipo de informação, colocando-as na mão de profissionais mais experientes. A partir do momento que o usuário realiza uma

solicitação de recarga online, ele é redirecionado para a página de pagamento do método escolhido. Cabe ao usuário apenas concordar com as politicas de segurança e utilização do método de pagamento escolhido, lembrando que sempre haverá a opção de pagamento no próprio restaurante, caso essas mesmas politicas não o façam se sentir seguro. Para esse trabalho foi utilizado a API de pagamentos do Mercado Pago, pois a empresa, além de reconhecida, possui uma biblioteca oficial na linguagem Python (MERCADOPAGO, 2021) e fornece um ambiente de *sandbox*, que nada mais é do que um ambiente de testes, com cartões e dinheiros fictícios (MERCADOPAGO, 2022). Um exemplo desse redirecionamento pode ser visto na Figura 19, onde o usuário deseja realizar uma recarga na sua conta e é encaminhado para o serviço de pagamento.

Figura 19 – Recarga e encaminhamento para o serviço de pagamento



Fonte: Autoria própria (2022).

A implementação do método de pagamento dentro do servidor é totalmente flexível, não sendo necessário grandes alterações nas linhas de código entre os métodos de pagamento pesquisados (PAGSEGURO, 2022; ASAAS, 2022). A única informação compartilhada com a API de pagamento é o valor da transação e o id do pagamento criado no banco de dados local. A confirmação do pagamento é feita através de *webhooks* criados nos parâmetros de configuração da API de pagamento. Quando um pagamento é efetuado, o servidor da empresa de pagamento faz uma requisição, na rota previamente configurada, e, através do corpo da requisição, que é diferente em cada API, é resgatado o id previamente inserido, bastando aprovar o pagamento localmente e adicionar o saldo ao usuário.

4.4 Projeto do *Front-end*

Com o *back-end* pronto, nesse contexto, se tratando de uma API REST, basta consumir a API e montar uma experiência agradável e confortável para os usuários utilizarem. Em teoria, qualquer pessoa consegue utilizar os serviços de uma aplicação REST sem o apoio de uma GUI, na prática, isso se torna inviável pois nem todos os usuários tem treinamento prático e técnico para utilizar aplicações complexas, muito menos comandos HTTP. Com esse pensamento em mente e focando no contexto do problema abordado nesse trabalho, foi realizado um projeto *front-end* em Flutter, que, consumindo todas as rotas disponíveis do servidor, entrega ao usuários a experiência adequada.

4.4.1 Plataformas

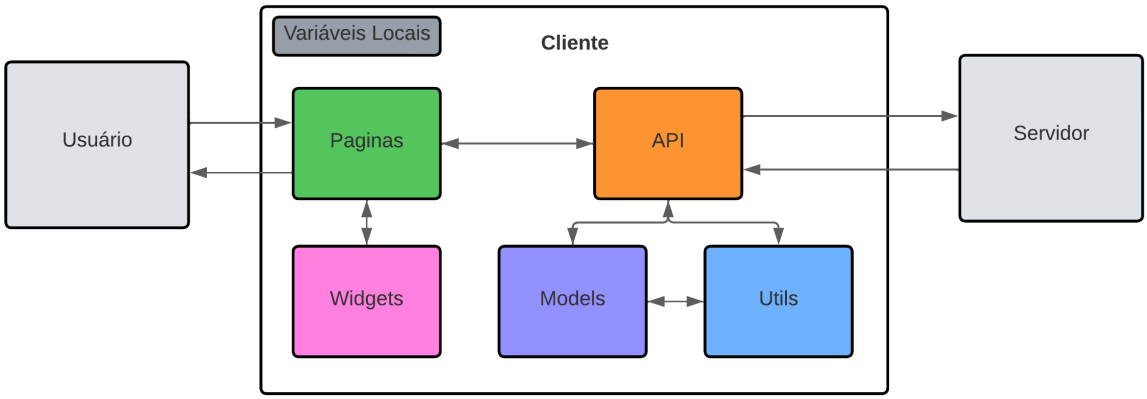
Apesar de produzir um resultado para as diferentes plataformas, como *web*, *desktop* e *mobile*, como relatado no subseção 2.3.3, expectativas, por parte do usuário, a respeito do comportamento e *design* das aplicações são diferentes para cada plataforma (LAL, 2013). Além das expectativas, a forma como usuário interage com a aplicação também muda. Nos aparelhos móveis, a tela tanto é utilizada para entrada quanto para saída de informações, e tem um formato e orientação específicos, o modo retrato. Já nos computadores pessoais, é mais comum a utilização de mouse e teclado para entradas e monitores para saída, que por sua vês utilizam orientação no modo paisagem. As proporções dos elementos também mudam de acordo com as plataformas, mesmo aparelhos moveis já possuindo resoluções de imagem equivalente as dos monitores de computares, as proporções, devido ao tamanho das telas, são diferentes, mudando fatores de responsividade.

Pensando nisso, as escolhas que ditaram o *design* da aplicação fora definidas como: para os clientes do sistema (alunos, servidores e visitantes) o foco foi na plataforma móvel (*smartphones*), pois é a plataforma mais acessível no momento da fila do RU; e para os operadores (funcionários do RU) o foco foi na plataforma *desktop*, devido ao levantamento de requisitos apontar que a infraestrutura disponibilizada pela universidade já cobre esse ambiente.

4.4.2 Arquitetura

O projeto do *front* seguiu os mesmos princípios de organização explanados na subseção 4.3.1. A arquitetura do cliente pode ser vista na Figura 20. Apesar de aparentar maior complexidade lógica em relação ao projeto do servidor, isso não é verdade, pois todas as regras de negócio foram, como mencionados anteriormente, deixadas, por segurança, a cargo do *back-end*.

Figura 20 – Projeto da arquitetura do cliente



Fonte: Autoria própria (2022).

Como antes, no módulo *"Models"* estão todas as representações definidas no servidor, agora espelhadas no cliente, que, como recebe de entrada apenas mensagens no formato json, a serialização não é um processo complexo como antes, bastando apenas ler as mensagens definidas e atribuir os valores nos seus respectivos lugares.

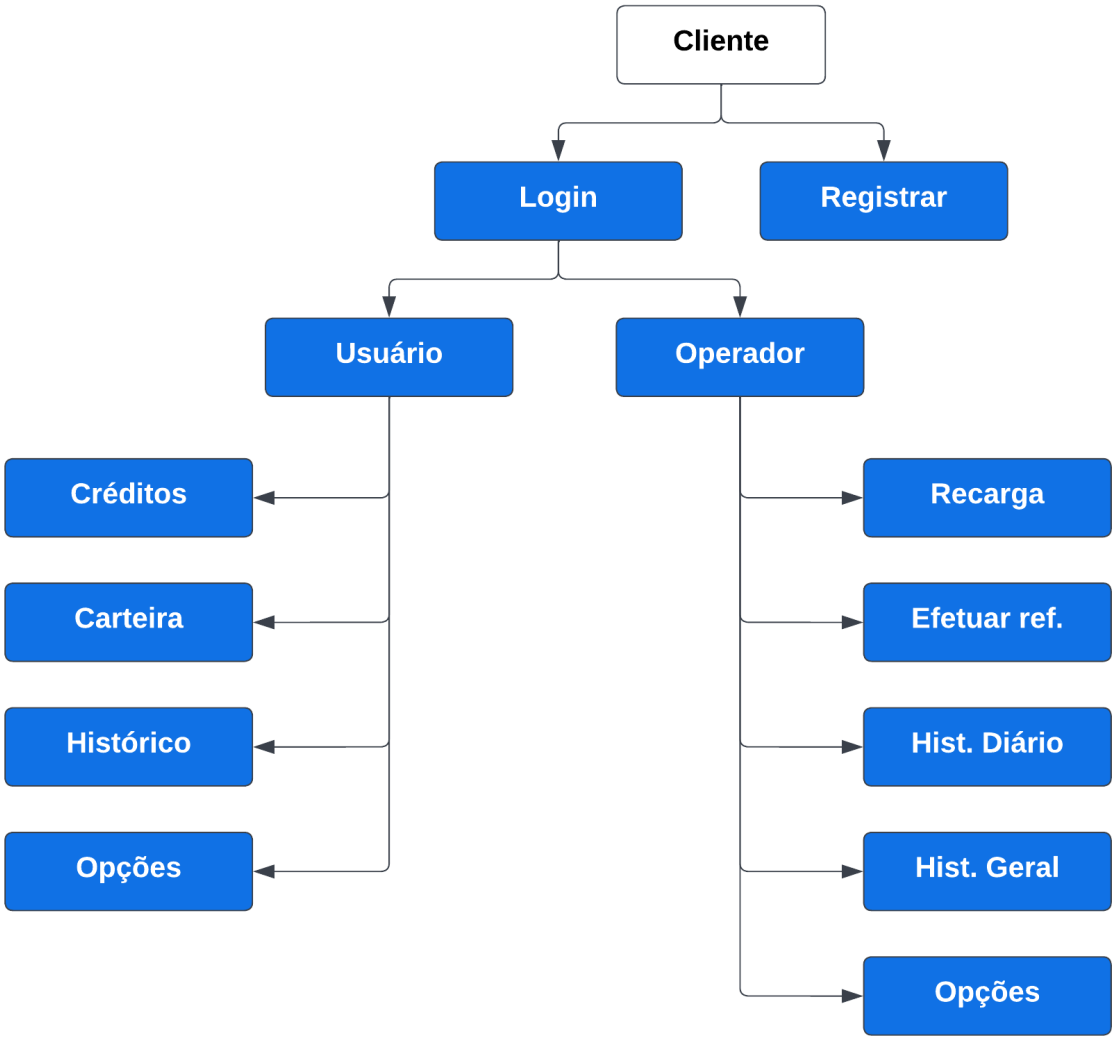
O módulo *"Utils"*, não é nada parecido com o seu semelhante no servidor, ao contrário do que acontecia antes, onde eram definidas todas as regras de negócio mais complexas, nesse, pelo contrário, é responsável apenas por traduzir os formatos de datas utilizados no servidor (*Unix epoch*) e realizar pequenas conversões para deixar os dados mais legíveis.

O módulo "API" é responsável pela comunicação com o servidor. Nele estão definidas as rotas, e como utiliza-las. Seu primeiro uso é na tela de *login*. Após o usuário digitar suas identificações, elas são disparadas ao servidor, e no retorno, com posse do *token* (explicado na subseção 4.3.3.1), guarda as informações nas "Variáveis Locais". Tais informações contêm dados como sua identificação, seu usuário, sua função, entre outras. Além disso, esse repositório do usuário é utilizado para guardar dados a medida que o mesmo utiliza o sistema, como histórico de alimentações ou histórico de recargas, informações que são utilizadas em todos os outros módulos.

No módulo *"Widgets"* estão definidos elementos de GUI customizados. Como botões ou barras de tarefa com cores e formatos de texto específicos ao *design* do sistema. Todo elemento gráfico que precisou ser reutilizado em mais de uma página está disponível nesse módulo.

Já no módulo "Páginas" estão definidas as rotas de acesso principal do sistema. Aqui são definidas como as interações com o usuário devem acontecer. Além disso são aplicados, apenas com a intenção de guiar o uso e não como forma de proteção, limitações nos campos, como campos que aceitam apenas valores numéricos ou de texto. A representação (arvore de *widgets*) que exemplifica o fluxo entre as páginas pode ser visto na Figura 21, onde cada retângulo preenchido representa uma página e cada página tem seu conjunto de *widgets* que definem a sua estrutura visual.

Figura 21 – Representação do fluxo entre as páginas



Fonte: Autoria própria (2022).

4.4.3 Paleta de cores

Buscando providenciar uma experiência mais familiar para os alunos da UTFPR, para a GUI do projeto, foi utilizada uma paleta de cores inspirada nas redes sociais de divulgação oficiais da universidade. Um exemplo dessa inspiração pode ser vista na Figura 22 uma figura, usada como foto de perfil das redes sociais da universidade e sua paleta de cores (simplificada).

Porém os tons das cores originais foram ajustadas, reduzindo o brilho, melhorando a visualização por longos períodos, evitando a fadiga ocular (BALDWIN; COLT, 2018). A paleta de cores utilizada na interface da aplicação pode ser vista na Figura 23.

Figura 22 – Paleta de cores simplificada da foto de perfil das redes sociais oficiais da UTFPR

Fonte: utfpr (2019).

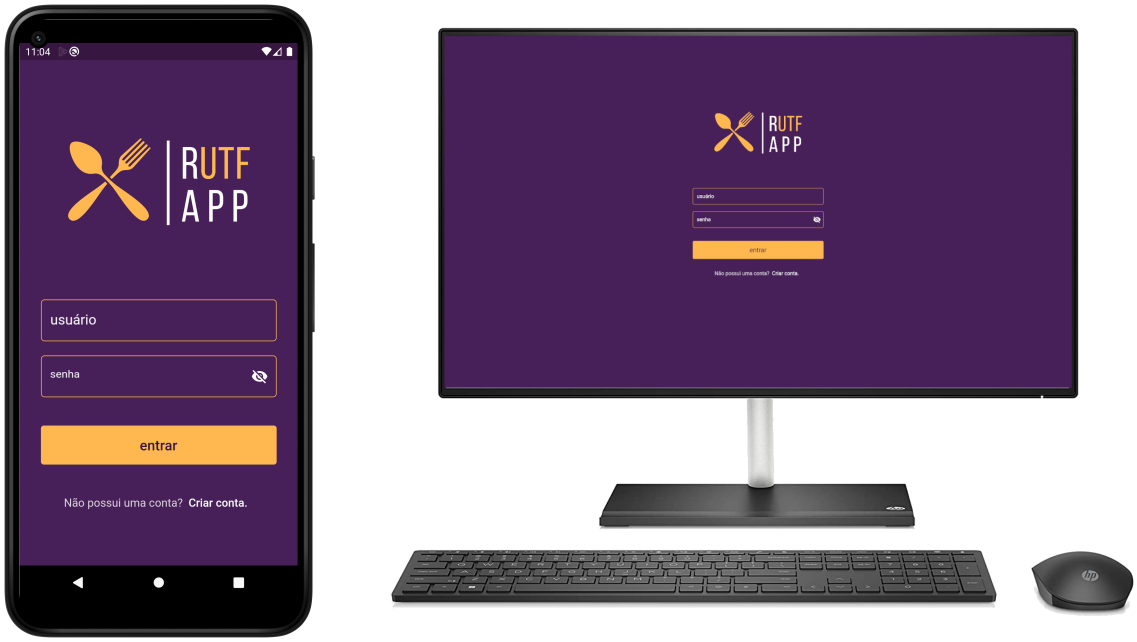
Figura 23 – Paleta de cores do *design* da aplicação

Fonte: Autoria própria (2022).

5 RESULTADOS

Nesse capitulo é apresentado amostras do resultado do trabalho, um sistema para controle de créditos e cadastro de refeições, funcionando em servidor de testes executando em rede local. Sendo divididos em duas partes, o ambiente *mobile* voltado para os usuários (alunos, servidores e visitantes) e o ambiente *desktop* voltado para os operadores (funcionários do restaurante). Na Figura 24 é apresentado a tela inicial do sistema, a tela de login, que é a mesma em ambos os sistemas.

Figura 24 – Tela de login em ambos os ambientes



Fonte: Autoria própria (2022).

5.1 Apresentação do ambiente *mobile*

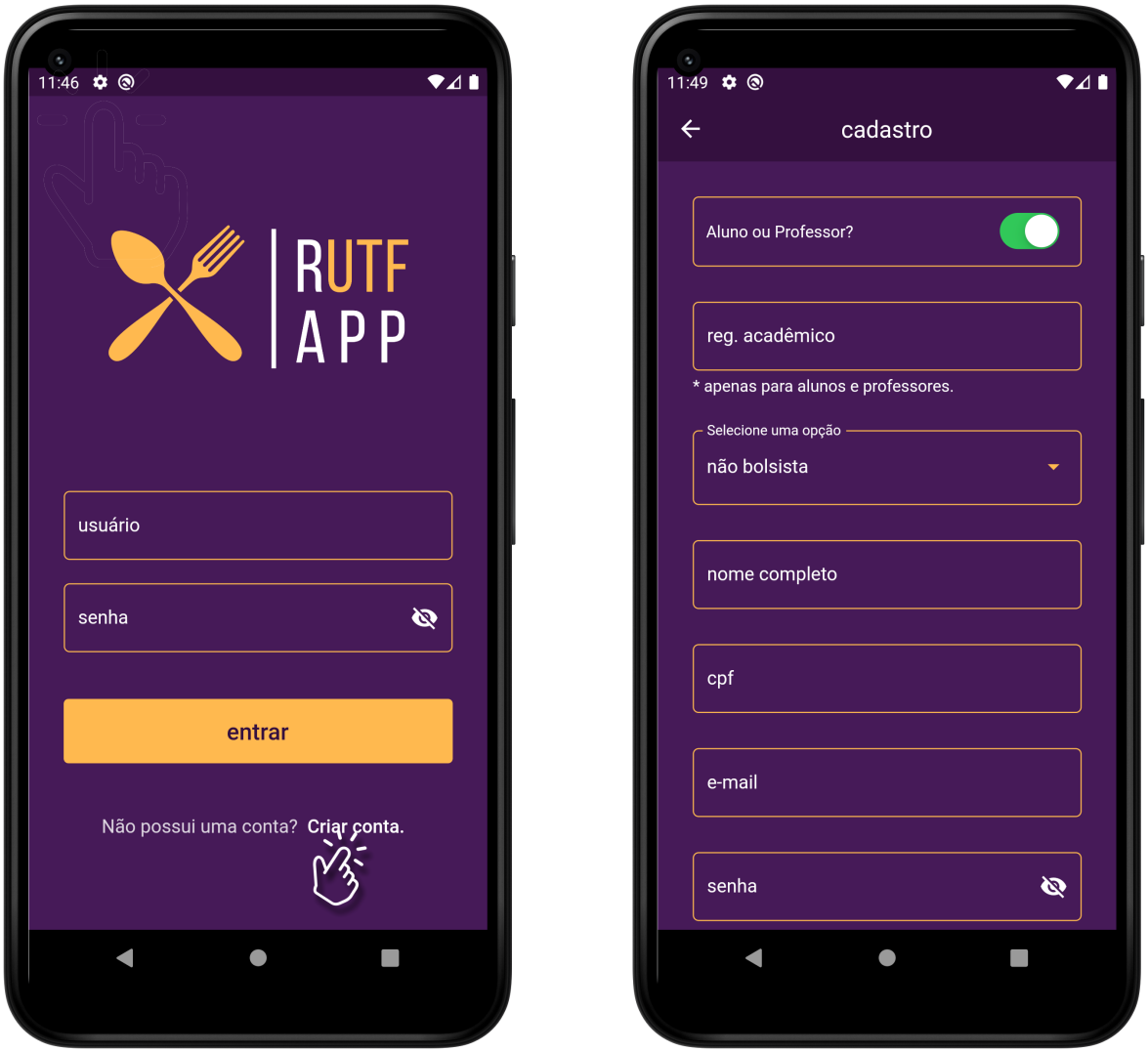
A seguir são apresentados amostras do ambiente *mobile*, exemplificando seu uso e suas funções, voltado para a utilização por parte dos usuários.

5.1.1 Tela de cadastro

Na Figura 25 é possível visualizar, a direita, após selecionar a opção de criação de conta na tela inicial do sistema. Nesta tela é possível preencher um formulário para realizar o cadastro e iniciar a utilização do sistema. A primeira opção, é utilizada para selecionar entre usuários com registro acadêmico (alunos ou professores) ou visitantes. Caso seja um usuário acadêmico, será

utilizado o RA para criação do *login*. Um RA como ”2058000” sera utilizado para criar o *login* como ”a2058000”’. E caso o usuário seja visitante, ele fornecerá um *login* próprio.

Figura 25 – Tela de cadastro do sistema

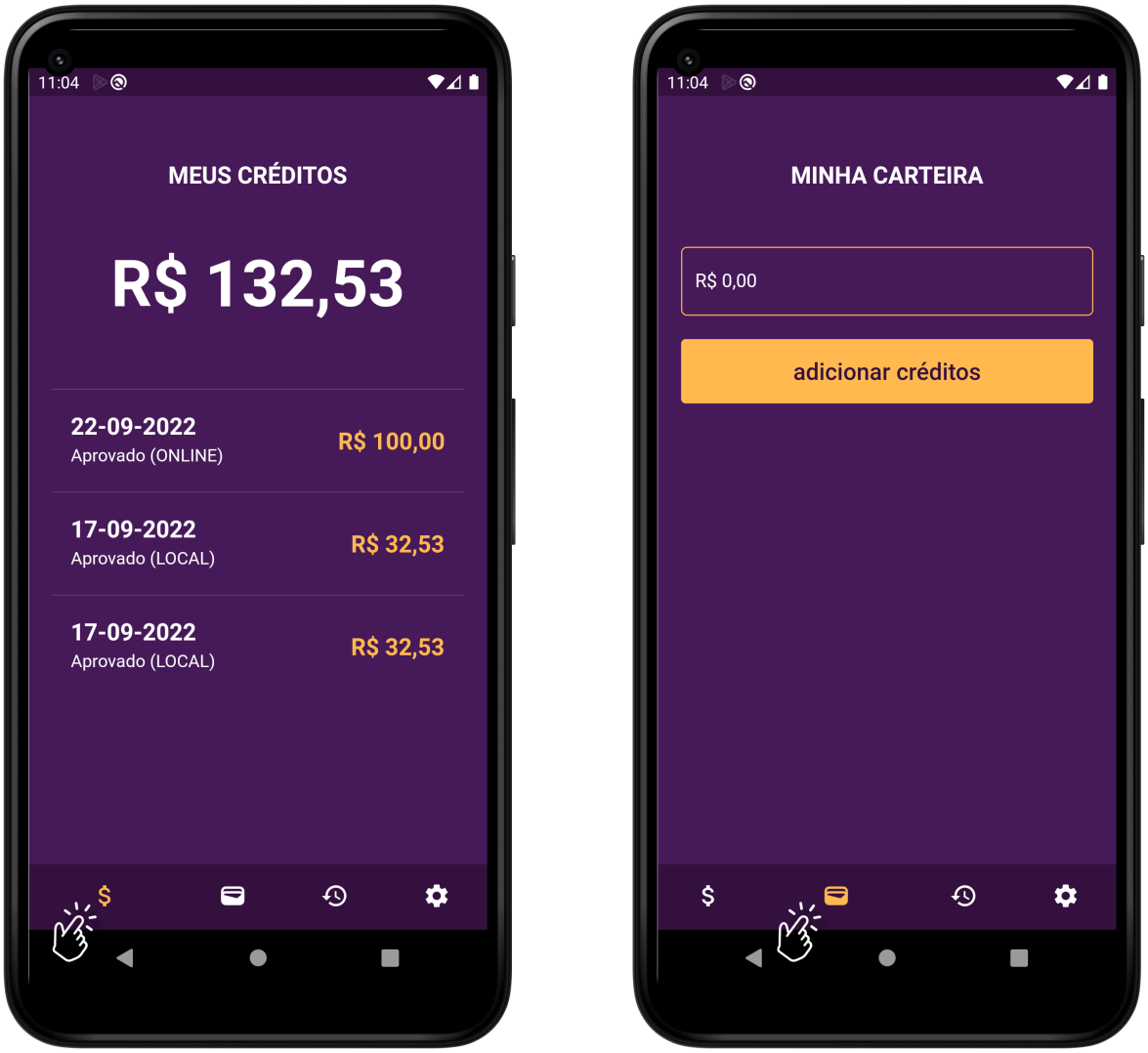


Fonte: Autoria própria (2022).

5.1.2 Telas referentes ao crédito

Na Figura 26 é possível visualizar duas telas, ambas após um *login* bem sucedido. A primeira, da esquerda, a tela inicial do usuário após um login, nela é possível visualizar os créditos disponíveis na conta. No topo está visível o total disponível, no rodapé do aplicativo, no formato de lista, estão as últimas recargas realizadas. A segunda tela, a direita, é a tela de recarga (*online*), utilizada para o usuário adicionar créditos de forma autônoma. Nesta tela, após digitar o valor, o usuário é reencaminhado para a tela de pagamento (explicado na subseção 4.3.3.3).

Figura 26 – Telas de créditos e recarga

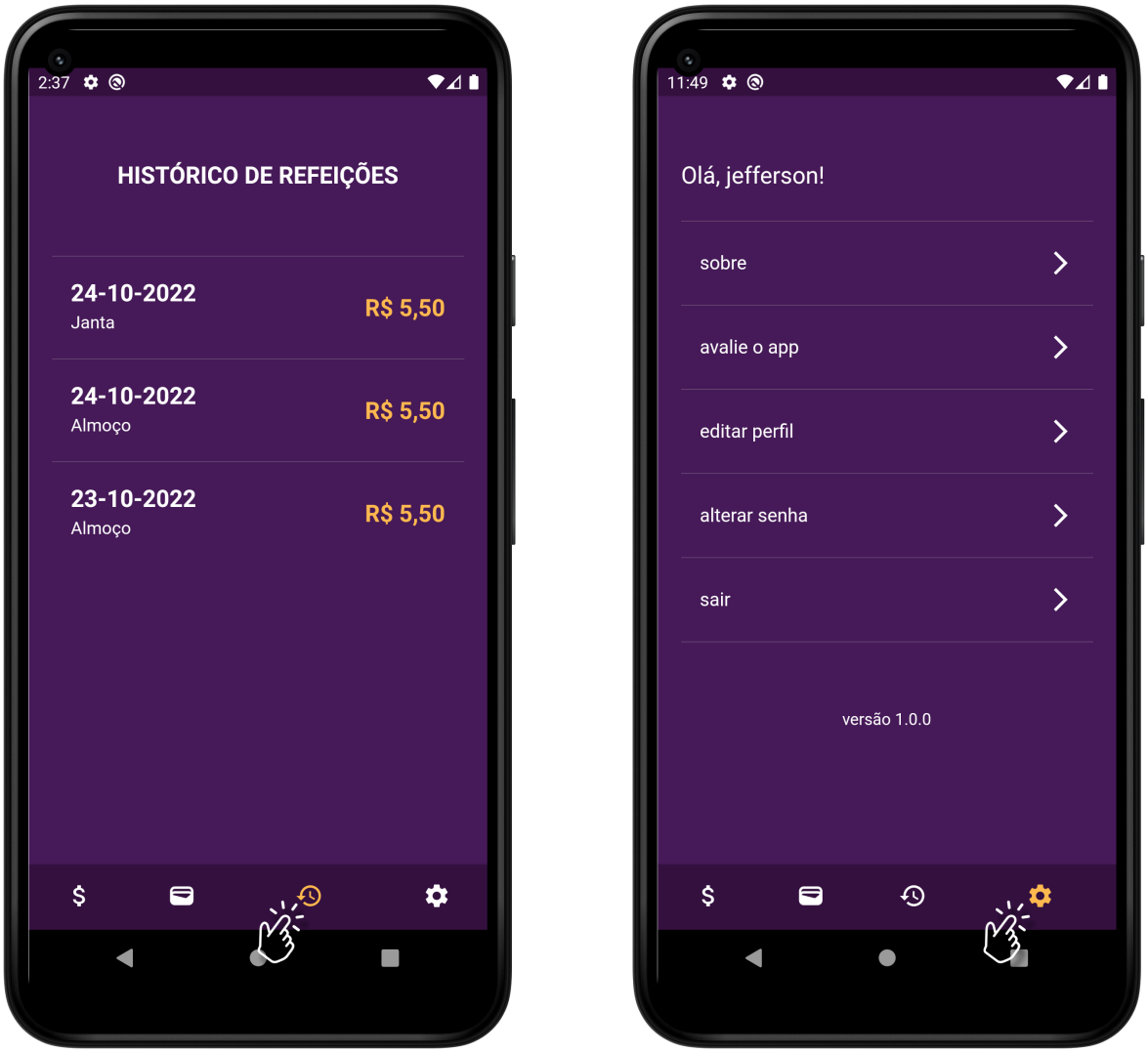


Fonte: Autoria própria (2022).

5.1.3 Tela de histórico de refeições e opções

Na Figura 27 é possível visualizar duas telas do sistema, ambas após um *login* bem sucedido. A primeira, da esquerda, a tela de histórico de refeições. Nela o usuário consegue visualizar as ultimas refeições realizadas e quanto foi descontado da sua conta. Lembrando que, no caso de bolsistas, esse valor aparece como zero. A segunda tela, a direita, é a tela principal de opções do sistema, nela é possível acessar edição de perfil, mudança de senha, dados sobre o aplicativo, entre outros.

Figura 27 – Telas de histórico de refeições e opções



Fonte: Autoria própria (2022).

5.2 Apresentação do ambiente *desktop*

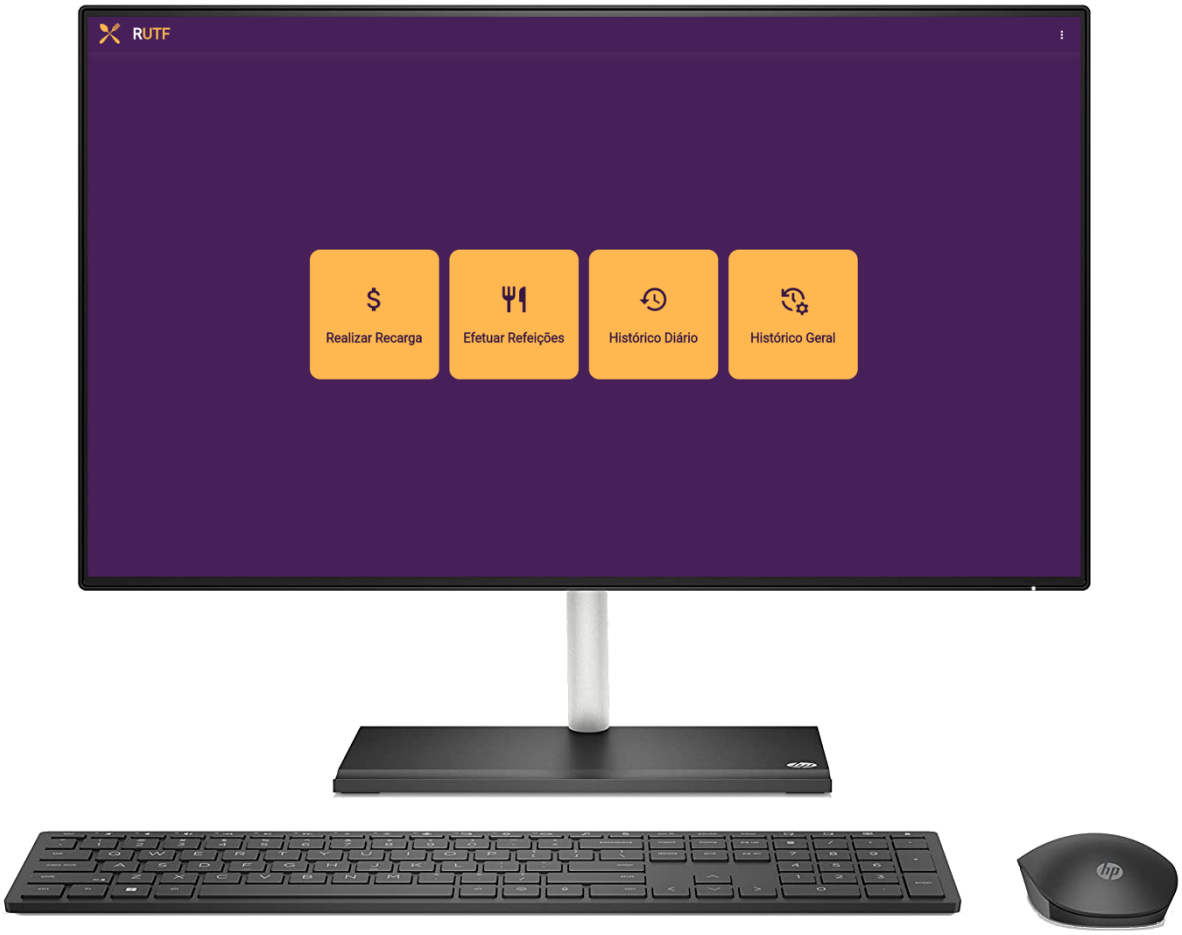
A seguir são apresentados amostras do ambiente *desktop*, explicando seus usos e funções, voltado para a utilização por parte dos operadores.

5.2.1 Tela inicial do ambiente *desktop*

Na Figura 28 é possível ver a tela inicial do ambiente *desktop*, apos um *login* bem sucedido por parte do operador. Foi utilizado botões grandes e auxílios visuais (ícones) para maior acessibilidade e melhor memorização dos comandos necessários. Os botões, respectivamente, são redirecionamentos para as telas: realizar recargar local, com método de pagamento próprio da empresa; efetuar refeições, tela principal para leitura do RA ou login dos usuários para que

os mesmos possam se alimenta; tela de histórico diário, contendo o histórico de recargas e refeições do dia atual; histórico geral, contendo todo o histórico.

Figura 28 – Telas inicial do ambiente *desktop*

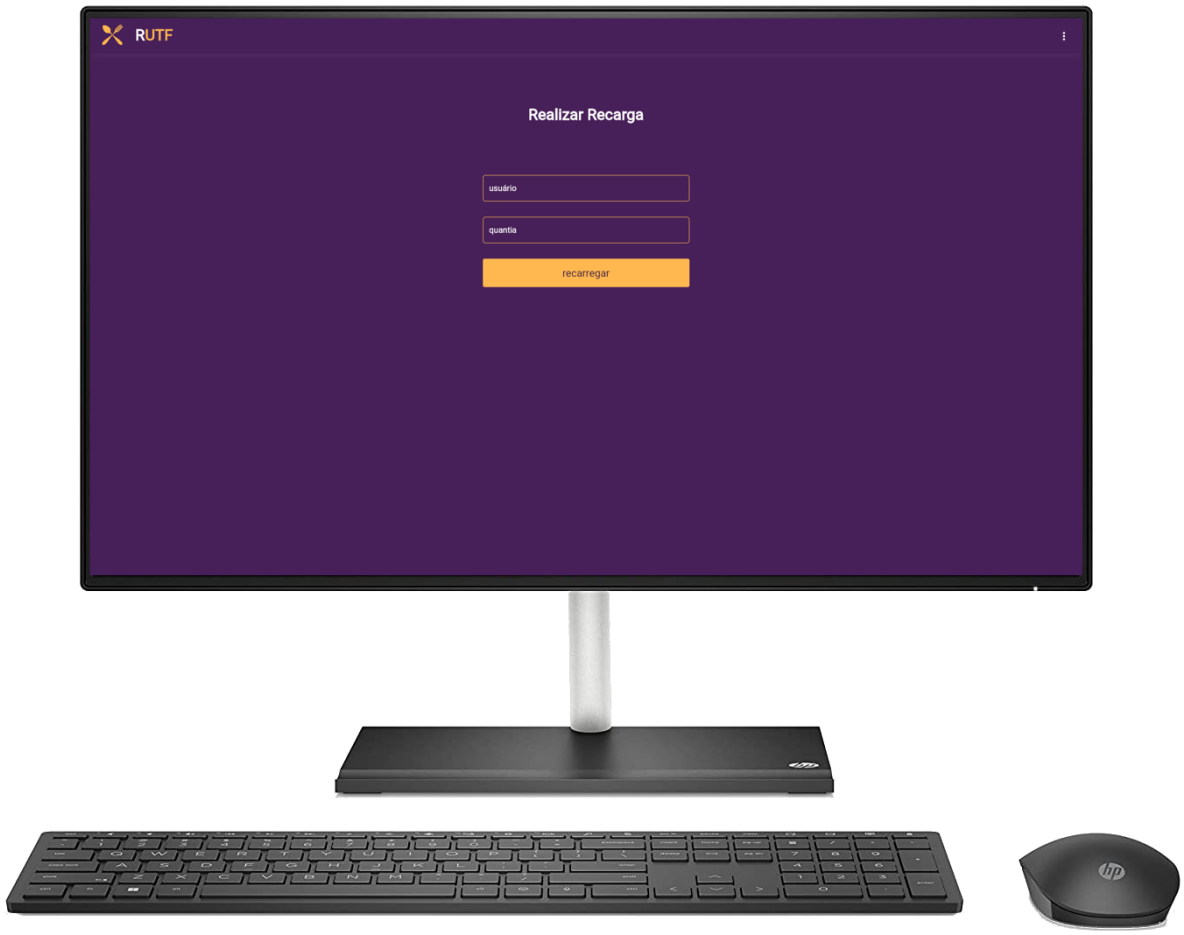


Fonte: Autoria própria (2022).

5.2.2 Tela de realizar recargas

Na Figura 29 é possível visualizar a tela de recargas *in-loco* feito por operadores. Nesta tela o operador digita de quanto será a recarga, e usuário em questão que está solicitando a mesma, que pode ser feito com o leitor de código de barras. A busca dentro do servidor, pelo nome de usuário, é feita de maneira inteligente, não sendo necessário, por parte do operador, adicionar por exemplo o carácter ”a” na frente do RA do aluno, pois o servidor se encarrega disso automaticamente. Em caso de sucesso uma mensagem de confirmação é exibida, o mesmo acontece em caso de falha.

Figura 29 – Telas realizar recargas



Fonte: Autoria própria (2022).

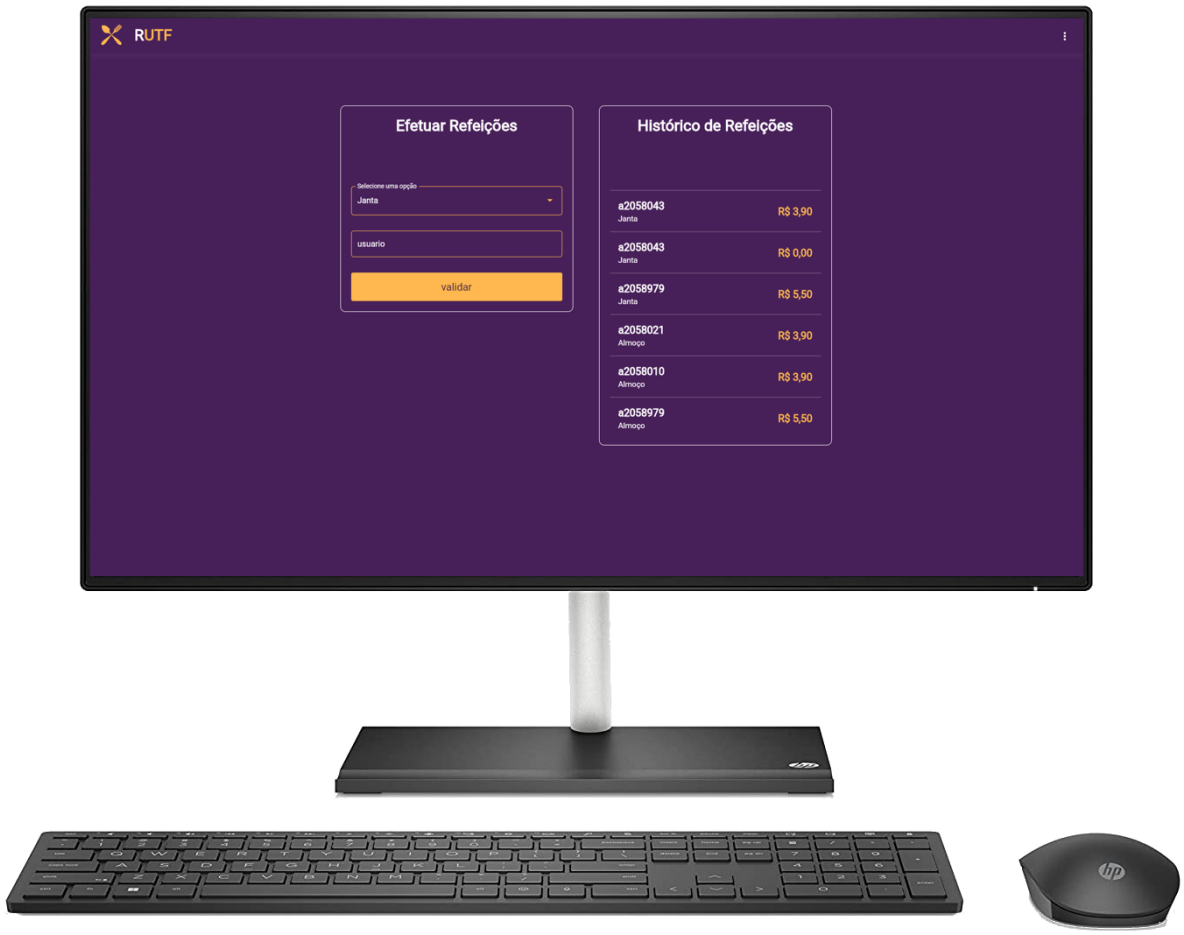
5.2.3 Tela de efetuar refeições

Na Figura 30 é possível visualizar a tela de efetuação de refeições. Nela, o operador seleciona qual o tipo da refeição (almoço ou jantar) e insere o *login* do usuário, que pode ser feito através do leitor de códigos de barra ou inserido manualmente. Uma mensagem de confirmação é exibida e o histórico, a direita, contendo as últimas refeições é atualizado, exibindo, além da mensagem, mais uma confirmação visual de que a operação foi bem sucedida. Em caso de erro, uma mensagem de usuário não encontrado é exibida.

5.2.4 Tela de histórico diário

Na Figura 31 é possível visualizar a tela de histórico diário de lançamentos. Nesta tela, o operador consegue observar os lançamentos de ambas refeições e recargas do dia atual. A tela de visualização geral é semelhante, agrupando o histórico de maneira completa.

Figura 30 – Telas efetuar refeições



Fonte: Autoria própria (2022).

5.3 Validação

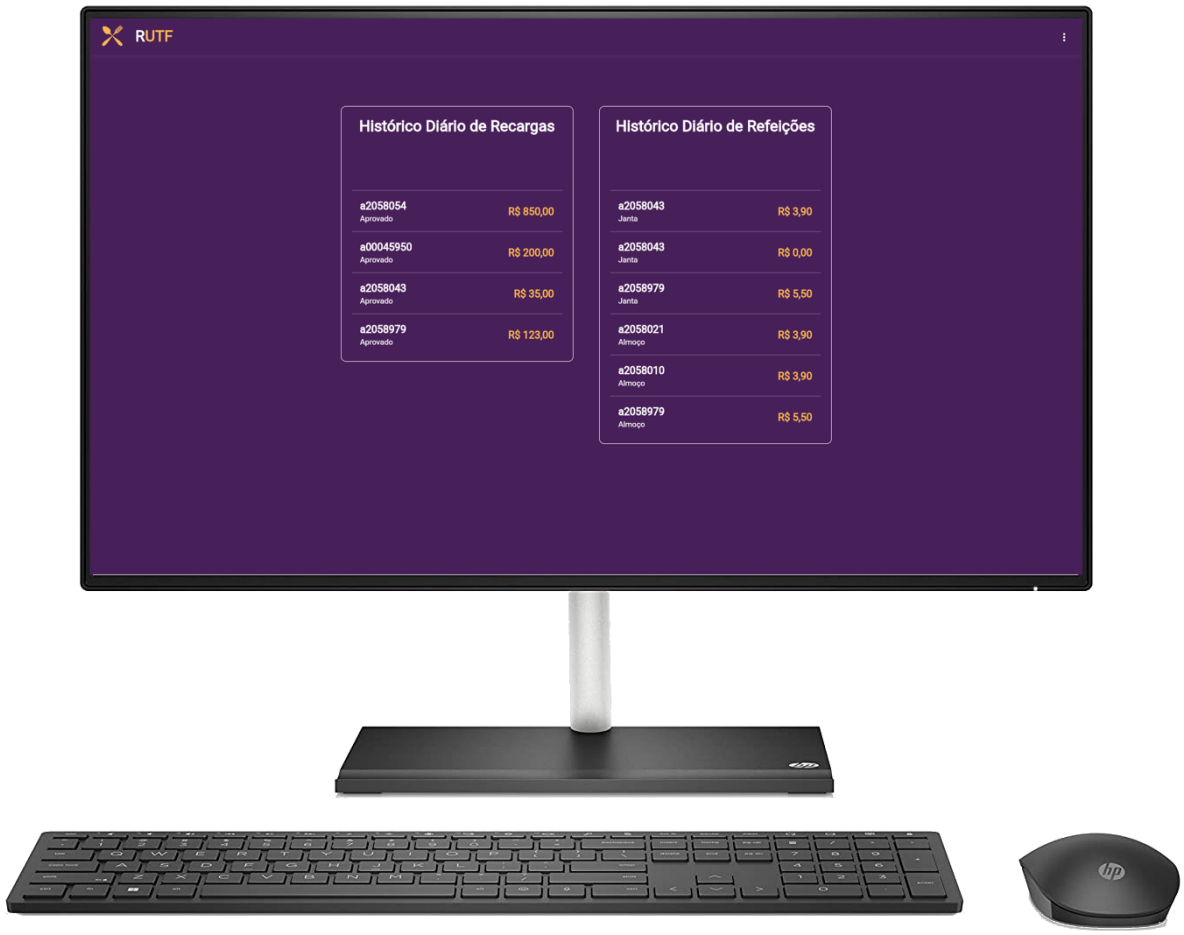
A seguir estão relatados as validações realizadas no sistema resultante. Cada uma com o foco nos seus respectivos nichos de usuários. Sendo uma com os usuários do ambiente *mobile* e a outra com os membros da gerencia do RU, ou seja, ambiente *desktop*.

5.3.1 Validação com usuários

Para realizar a validação com os usuários, um teste moderado foi aplicado. No qual foram dadas aos participantes do teste uma sequencia de tarefas, cada uma representando as principais funcionalidades do aplicativo móvel. Todas as ações do usuário eram monitoradas, e todas as dificuldades percebidas pelo avaliador eram anotadas.

Ao fim do teste, um questionário era fornecido ao participante que respondia avaliando o nível de acordo com a sua percepção de dificuldade da tarefa. As respostas eram definidas

Figura 31 – Telas de histórico diário



Fonte: Autoria própria (2022).

no intervalo de um a cinco, onde um representava uma tarefa considerada ”muito dificil” e cinco ”muito fácil”. E ao fim do questionário, um campo de sugestões foi adicionado.

O teste foi realizado com um total de cinco participantes, todos alunos da UTFPR e de cursos diversos. Os resultados dos testes, as dificuldades percebidas pelo avaliador e as sugestões estão listadas nas subseções a seguir.

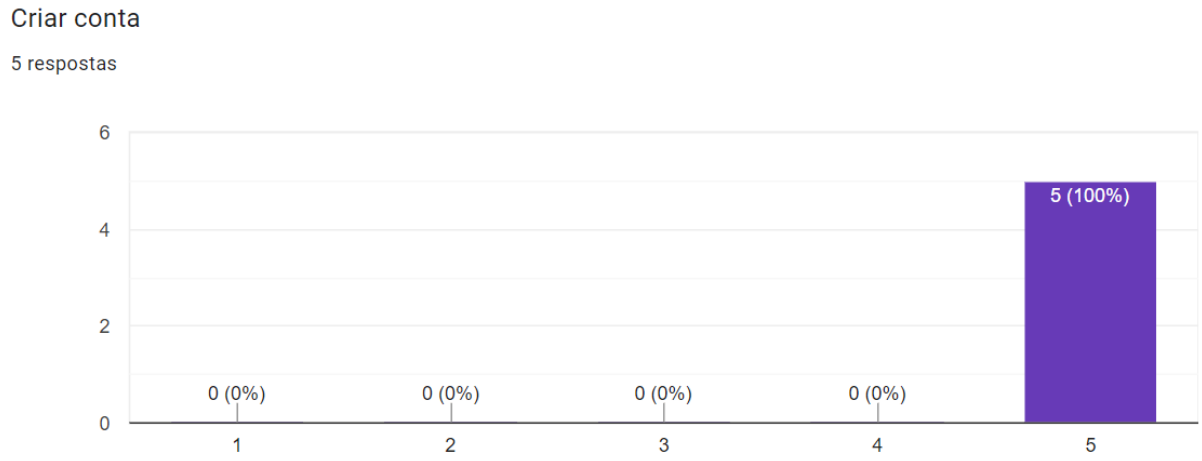
5.3.1.1 Criação de conta

A primeira tarefa dada ao avaliado no teste foi a de criação de conta. A finalidade desta tarefa é visualizar possíveis dificuldades na criação de contas. As respostas podem ser visualizadas no Gráfico 3.

5.3.1.2 Realização de recarga *online*

A segunda tarefa dada ao avaliado no teste foi a de realização de uma recarga *online*. A finalidade dessa tarefa é visualizar possíveis dificuldades na operação de recargas online,

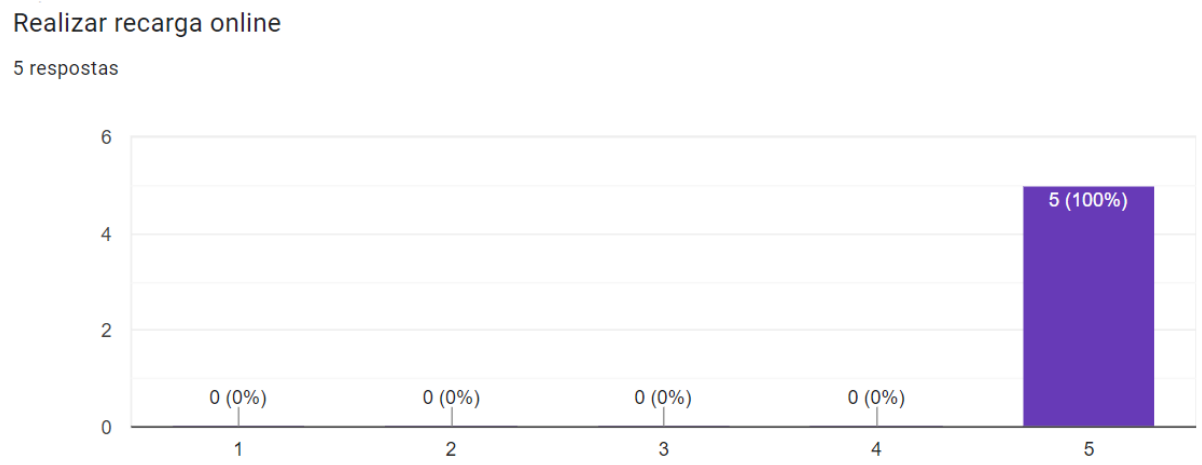
Gráfico 3 – Avaliação dos usuários na tarefa de criação de conta



Fonte: Autoria própria (2022).

incluindo o processo de encaminhamento a plataforma de pagamento externa. As respostas podem ser visualizadas no Gráfico 4.

Gráfico 4 – Avaliação dos usuários na tarefa de recarga *online*

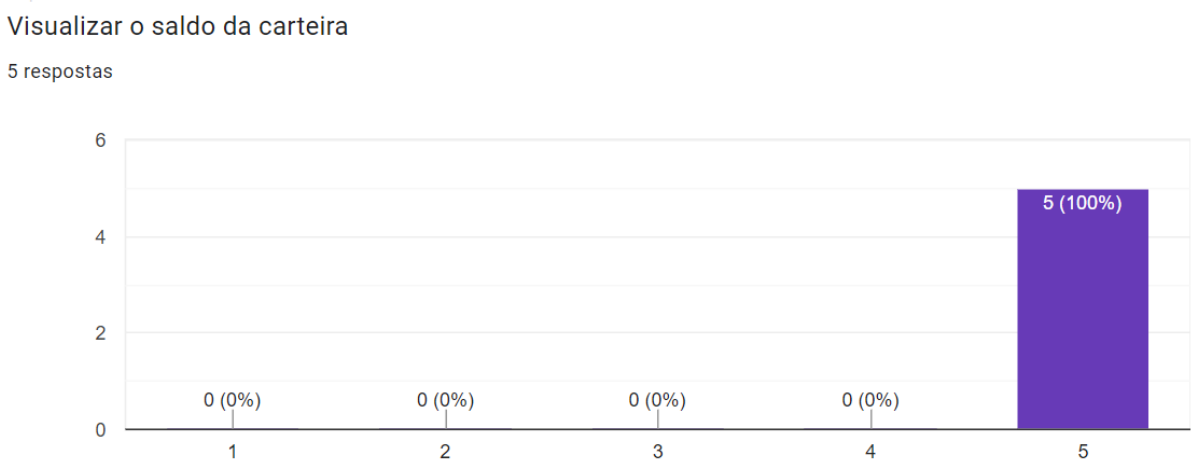


Fonte: Autoria própria (2022).

5.3.1.3 Visualizar saldo na carteira

A terceira tarefa dada ao avaliado no teste foi de visualizar o saldo adicionado na conta e informar ao avaliador. A finalidade dessa tarefa é visualizar possíveis dificuldades na operação de visualização das informações de saldo do usuário. As respostas podem ser visualizadas no Gráfico 5.

Gráfico 5 – Avaliação dos usuários no processo de visualização de saldo

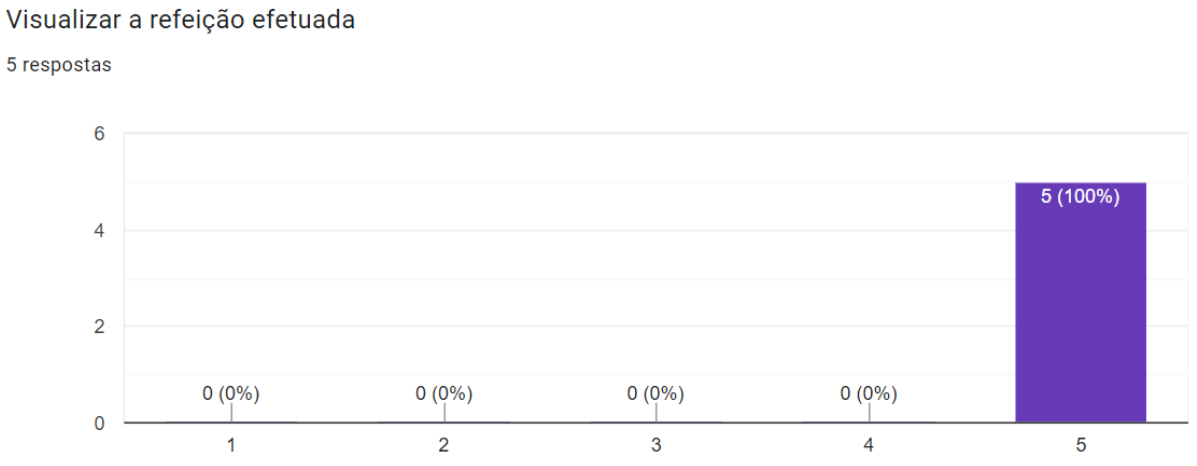


Fonte: Autoria própria (2022).

5.3.1.4 Visualizar refeição efetuada

A quarta tarefa dada ao avaliado no teste foi a de visualizar a refeição efetuada pelo avaliador na conta recém criada pelo usuário. A finalidade dessa tarefa é visualizar possíveis dificuldades na operação de visualização das informações de refeição do usuário. As respostas podem ser visualizadas no Gráfico 6.

Gráfico 6 – Avaliação dos usuários no processo de visualização de refeições



Fonte: Autoria própria (2022).

5.3.1.5 Sugestões fornecidas pelos avaliados

Na listagem a seguir estão descritas as sugestões fornecidas pelos participantes do teste no último campo de resposta do questionário.

- ”app topzin”
- ”Poderia ter um sistema de avaliação da refeição do dia, e mostrar o cardápio.”
- ”teste de usabilidade com a galera da utf :)”
- ”Talvez mudar o input na hora de inserir crédito e especificar melhor na hora de colocar o ra”
- ”Arrumar início das frases com letras maiúsculas”

5.3.1.6 Dificuldades observadas pelo avaliador

Algumas diretrizes foram seguidas durante a aplicação dos teste: Sugestões faladas pelo avaliado eram anotadas de acordo com a validade da mesma. E auxílios só eram fornecidas ao avaliado caso o mesmo solicitasse ou caso o avaliador percebesse que o mesmo ficou travado em alguma tarefa. As dificuldades percebidas e as sugestões fornecidas ao avaliador se encontram descritas a seguir.

Durante o processo de criação nenhuma dificuldade que impossibilitou o avanço do teste foi percebida. Alguns alunos, no momento de informar o registro acadêmico tentavam inserir o carácter ”a” como hábito aprendido em outras plataformas da universidade. No qual foi informado que não era necessário, pois o sistema fazia isso de maneira automática. Já no processo seguinte, o *login* do sistema, alguns usuários acadêmicos não sabiam o que inserir no campo ”usuário”. No qual era informado que a plataforma era responsável por atender tanto alunos como visitantes, e então indicado aos mesmo que o RA era o equivalente aos ”usuários” dos alunos.

Já durante a tarefa de realização de recarga *online*, como na anterior, nenhum problema que impossibilitasse o avanço dos testes foi percebido. Alguns avaliados, como costume aprendido em aplicativos terceiros, achavam que o campo de preenchimento do valor desejado na recarga era do tipo que se preenchia da direita para a esquerda. Ou seja, um usuário, por exemplo, digitou a sequência de algarismos ”700” e tinha expectativa de que o valor que seria recarregado era de R$7,00, oque não aconteceu, pois o valor fornecido ao meio de pagamento foi de R$700,00. As transações eram canceladas e refeitas pelos avaliados que acreditavam que o ”erro” eram seus e foi sugerido ao avaliador que o campo se comportasse dessa maneira no futuro.

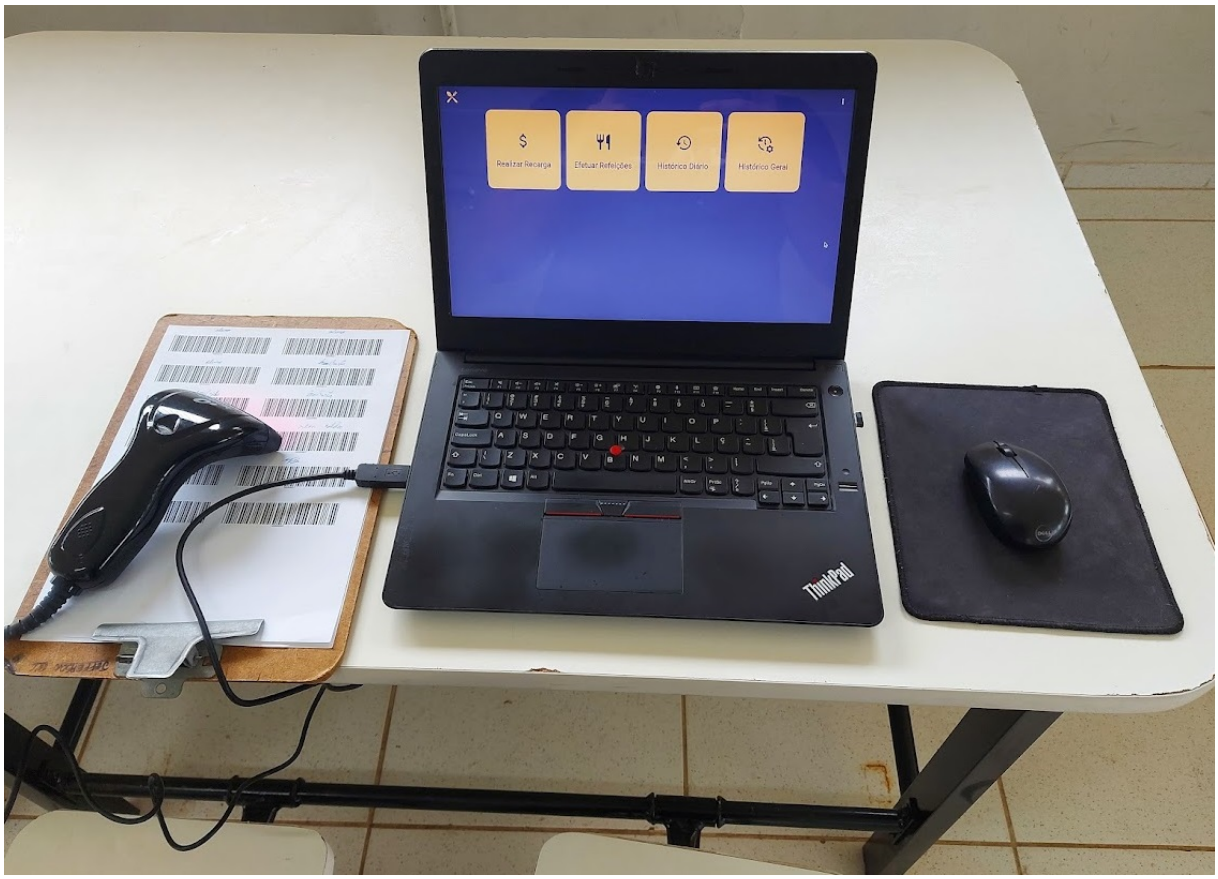
Como os processos de visualização (saldo e refeição) são semelhantes, as dificuldades encontradas foram as mesmas. Aqui foram encontrados adversidades que atrapalharam o

avanço do testes. O usuários realizavam de maneira intuitiva, como esperado pelo avaliador, nas telas de visualização, o movimento de *pull* para atualizar, comum em aplicativos terceiros. Porem, por funcionar somente em partes especificas, os avaliados se frustravam, e então era informado pelo avaliador que o movimento era aceito, porem somente na parte superior da tela. Como o movimento a tela de visualização de refeição era semelhante, os usuários já reproduziam o movimento na parte superior da tela. Os avaliados acreditavam que o erro era deles, mas sugeriam que a operação fosse aceita em qualquer parte da tela.

5.3.2 Validação com a gerencia

Como forma de validação do ambiente *desktop* foi realizado outra reunião com uma membra da equipe de gerencia. No qual foi apresentado o sistema *desktop* além de uma breve apresentação do sistema *mobile* para sanar duvidas a respeito do aplicativo. Durante a apresentação foi realizado um treinamento breve, e foram disponibilizados identificações fictícias e um leitor de códigos de barra, com a intenção de simular o ambiente real. O cenário apresentado a funcionária pode ser visto na Fotografia 3.

Fotografia 3 – Ambiente de testes utilizado durante a validação



Fonte: Autoria própria (2022).

Foram obtidas respostas positivas no que se diz respeita a usabilidade e aparência da aplicação. E, devido ao fato da reunião ter sido realizada com outro responsável da gerencia, no qual possui outras tarefas na empresa, requisitos que não foram devidamente abordados foram salientados. Foi apontado, pela membra da equipe, que ela é responsável por transferir o valores da planilha (Fotografia 3) para outra de responsabilidade da empresa, e que seria interessante no caso especifico dela por exemplo uma visualização resumida das informações.

6 CONCLUSÃO

Este trabalho teve como objetivo a implementação de um ambiente de gerenciamento de créditos, que considerasse a existência de todos os tipos de usuários levantados nos requisitos, sendo eles: usuários bolsistas, visitantes, alunos não bolsistas, servidores/funcionários e operadores do sistema. O foco do sistema desenvolvido foi no campus da UTFPR Ponta Grossa, porém o sistema resultante pode ser utilizado nos restaurantes de todos os campi e pode ser adaptado para outros cenários fora do contexto da UTFPR.

O sistema foi desenvolvido utilizando a linguagem de programação Python em conjunto com *framework* Flask, a SDK Flutter da Google, e o SGBD MySQL. Oferece um sistema com intuito de ser simples e fácil de usar. Sem distrações e informações em excesso.

Para alcançar o objetivo, foi realizada uma reunião com a equipe de gerencia do RU, levantando os requisitos e funcionalidades necessárias para o sistema resultante. Durante a reunião, foi descoberto que um sistema oficial da UTFPR já existia, porém sem funcionalidades de crédito, e utilizado apenas para o controle de repasse de verba, no qual também foi utilizado para aprimoramento dos requisitos deste trabalho.

Como forma de validação do sistema resultantes foi realizado teste de usabilidade monitorado com foco nos usuários do aplicativo, no qual foi recebido sugestões e avaliações do mesmos.Além de um reunião, seguida de apresentação com membra da gerência do restaurante. No qual, foram obtidos em ambos de maneira expressiva respostas positivas.

6.1 Dificuldades

Durante a fase de desenvolvimento foram encontradas dificuldades em relação aos conceitos inicialmente aplicados, devido, principalmente, à falta de conhecimento com as tecnologias inicialmente escolhidas. Pesquisas extensivas foram realizadas para tecnologias de banco de dados, que não foram utilizadas pois, posteriormente, foram descobertas, através de pesquisas e experimentações, tecnologias mais atuais para mapeamento de objetos utilizando linguagem de programação ao invés de comandos SQL.

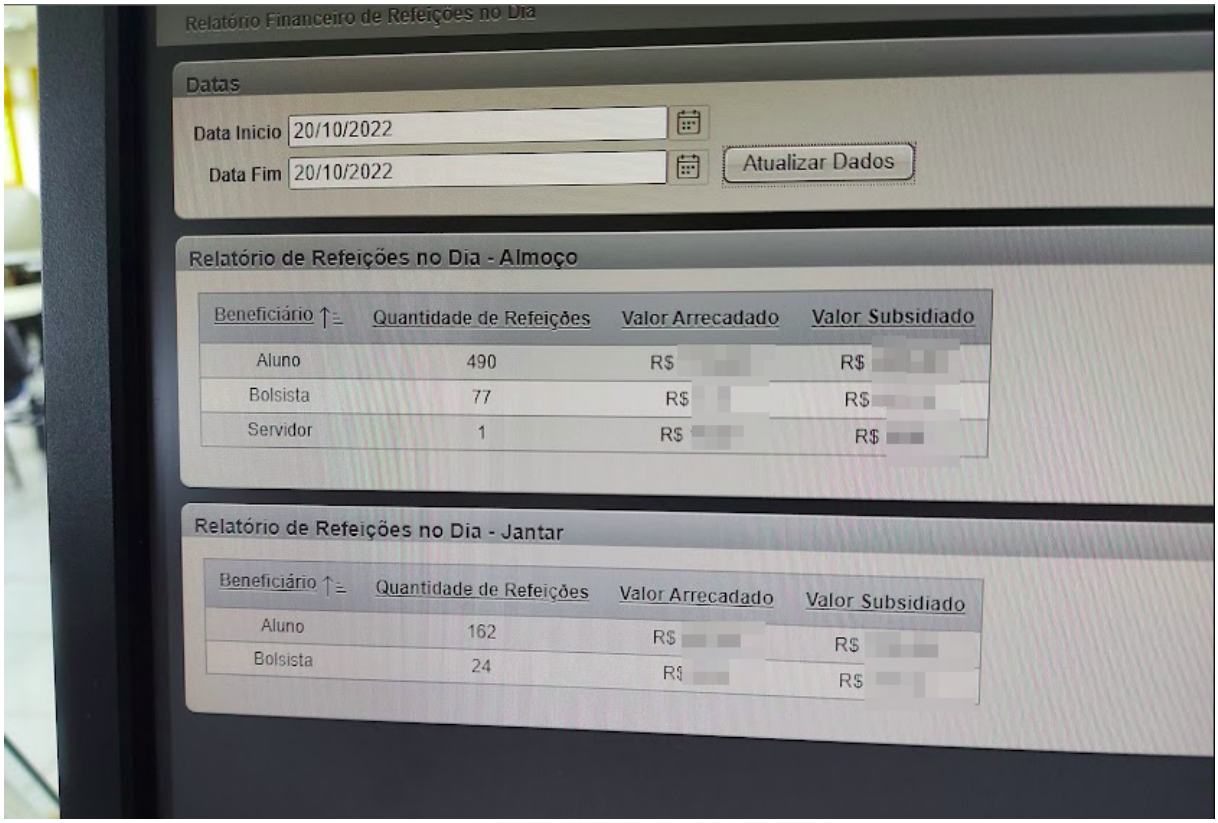
Outro empecilho foi na tentativa de utilização de tecnologias que não foram utilizadas, tais como Django e PyQt. Django se mostrou ser desnecessariamente complexa, pois fornece muitas ferramentas pré-embutidas, que tiveram efeito negativo no desenvolvimento da API REST. E PyQt, para desenvolvimento *desktop*, se mostrou desnecessário a partir da escolha do Flutter. Apesar das tecnologias utilizadas serem mais de acordo com o resultado do projeto, o tempo de aprendizado nas ferramentas não utilizadas afetou negativamente o cronograma do projeto.

6.2 Trabalhos futuros

Uma das principais requisições necessárias no aplicativo seria uma integração com a base de dados de alunos e servidores oficial da UTFPR. Com tal integração seria possível realizar o cadastro automaticamente de alunos e servidores na aplicação, principalmente para alunos bolsistas, no qual seria uma forma válida de confirmar seu benefício. Não foi possível obter essa integração durante o desenvolvimento desse projeto por motivos de conflito nas agendas. Outra fonte de trabalhos futuros seria aprofundar e implementar as sugestões fornecidas pelos usuários do aplicativo móvel, além de realizar testes mais extensivos e recorrentes.

Para mais, existe o novo requisito levantado, sugerido pela gerente do RU, as informações necessárias já estão devidamente inseridas no contexto da aplicação, bastando apenas implementar a geração da planilha como foi requerido. Um exemplo dessa planilha pode ser visto na Fotografia 4.

Fotografia 4 – Exemplo da planilha apontada na reunião de validação



Fonte: Autoria própria (2022).

REFERÊNCIAS

ANDERSON, D. J. **Kanban**: Successful evolutionary change for your technology. [*S.l.*]: Blue Hole Press, 2010.

ARIAWAN, D. **HTTP methods in spring restful services: Dariawan**. Dariawan, 2019. Disponível em: https://www.dariawan.com/tutorials/rest/http-methods-spring-restful-services/. Acesso em: 26 set. 2022.

ASAAS. **Asaas - Manual API**. Asaas, 2022. Disponível em: https://asaasv3.docs.apiary.io/#reference. Acesso em: 26 set. 2022.

BALDWIN, R.; COLT, J. **Your PC is ruining your vision. here's how to beat eye strain**. Conde Nast, 2018. Disponível em: https://www.wired.com/2013/09/flux-eyestrain/. Acesso em: 19 out. 2022.

BASSO, L. de O.; CHEIRAN, J. F. P. ; SANTAROSA, L. M. C. Testes de usabilidade com prototipação em papel: validação de ferramenta de ava acessível a pnes. *In*: **Taller Internacional de Software Educativo**. Santiago, Chile: Nuevas Ideas en Informática Educativa, 2009. v. 5, p. 151–158.

BATISTELLA, R. **Sistema web para controle de créditos em um restaurante universitário**. 2017 — Universidade Tecnológica Federal do Paraná, 2017.

BAYER, M. Sqlalchemy. *In*: BROWN, A.; WILSON, G. (Ed.). **The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks**. aosabook.org, 2012. Disponível em: http://aosabook.org/en/sqlalchemy.html.

BIANCUZZI, F. *et al.* **Masterminds of programming**: Conversations with the creators of major programming languages. [*S.l.*]: O'Reilly, 2009.

BODUCH, A. **React and React Native**. [*S.l.*]: Packt Publishing Ltd, 2017.

BORGES, G. D. S. **Crachá**. 2019. Disponível em: https://www.utfpr.edu.br/cursos/coordenacoes/stricto-sensu/ppgag-pb/documentos/academico/cracha.

CAMATA, T. P. **A técnica de prototipagem em papel com a participação de usuários**: o caso da plataforma integrada mec de recursos educacionais digitais. 2018. Dissertação (Mestrado) — Centro de Comunicação e Expressão, Universidade Federal de Santa Catarina, Florianópolis, 2018.

CHACON, S.; STRAUB, B. **Pro git**. [*S.l.*]: Springer Nature, 2014.

CLARKE, J. **SQL injection attacks and defense**. [*S.l.*]: Elsevier, 2009.

CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, ACM New York, NY, USA, v. 13, n. 6, p. 377–387, 1970.

CONNOLLY, T. M.; BEGG, C. E. **Database systems**: a practical approach to design, implementation, and management. [*S.l.*]: Pearson Education, 2005.

COSENTINO, V.; IZQUIERDO, J. L. C.; CABOT, J. A systematic mapping study of software development with github. **IEEE Access**, IEEE, v. 5, p. 7173–7192, 2017.

DRAKE, J. D.; WORSLEY, J. C. **Practical PostgreSQL**. [*S.l.*]: "O'Reilly Media, Inc.", 2002.

DUBEY, T. **Git branching strategy**. 2021. Disponível em: https://docs.wavemaker.com/learn/blog/2021/09/17/git-branching-strategy. Acesso em: 09 out. 2022.

EBY, P. J. **PEP 333**. 2003. Disponível em: https://peps.python.org/pep-0333/. Acesso em: 17 out. 2022.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 2000. Tese (Doutorado) — University of California, Irvine, 2000.

FIELDING, R. T.; NOTTINGHAM, M.; RESCHKE, J. **RFC 9110**. 2022. Disponível em: https://www.rfc-editor.org/rfc/rfc9110.html. Acesso em: 26 set. 2022.

FILHO, L. F. B. R. **RU digital: um sistema para gerenciamento de saldo financeiro de clientes de restaurantes universitários**. 2018 — Universidade Tecnológica Federal do Paraná, 2018.

FLUTTER. **Flutter documentation**. Google, 2022. Disponível em: https://docs.flutter.dev/. Acesso em: 17 out. 2022.

FLUTTER. **Flutter update**: Windows. Google, 2022. Disponível em: https://www.youtube.com/watch?v=g-0B_Vfc9qM. Acesso em: 17 out. 2022.

FOWLER, M. **UML distilled**: a brief guide to the standard object modeling language. [*S.l.*]: Addison-Wesley Professional, 2004.

GACKENHEIMER, C. **Introduction to React**. [*S.l.*]: Springer, 2015.

GITHUB. **Documentação do GitHub**: Obtenha ajuda, onde quer que você esteja em sua jornada no github. 2022. Disponível em: https://docs.github.com/pt. Acesso em: 09 out. 2022.

GITHUB. **Your AI pair programmer**: Github copilot uses the openai codex to suggest code and entire functions in real-time, right from your editor. 2022. Disponível em: https://github.com/features/copilot. Acesso em: 09 out. 2022.

GOOGLE. **Google Maps**. 2022. Disponível em: https://goo.gl/maps/PPAoqCoJYNvR3Suo6. Acesso em: 29 set. 2022.

GOV.BR. **Como É transmitido?**: Vírus pode ser transmitido durante um aperto de mão (seguido do toque nos olhos, nariz ou boca), por meio da tosse, espirro e gotículas respiratórias contendo o vírus. gov.br, 2021. Disponível em: https://www.gov.br/saude/pt-br/coronavirus/como-e-transmitido. Acesso em: 29 set. 2022.

GRINBERG, M. **Flask web development**: developing web applications with python. [*S.l.*]: O'Reilly, 2018.

JETBRAINS. **Python developers survey 2020**. 2020. Disponível em: https://www.jetbrains.com/lp/python-developers-survey-2020/. Acesso em: 17 out. 2022.

JOHNSON, R. *et al.* The spring framework–reference documentation. **interface**, v. 21, p. 27, 2004.

JONES, M.; BRADLEY, J.; SAKIMURA, N. Rfc 7519: Json web token (jwt). **IETF. May**, 2015.

JUNIOR, M.; FILHO, M. Variations of the kanban system: Literature review and classification. **International Journal of Production Economics**, v. 125, p. 13–21, 05 2010.

LADAS, C. **Scrumban-essays on kanban systems for lean software development**. [*S.l.*]: Modus Cooperandi Press, 2009. ISBN 978-0-578-00214-9.

LAL, R. **Digital design essentials**: 100 ways to design better desktop, web, and mobile interfaces. [*S.l.*]: Rockport, 2013.

LIMA, A. G. d. A. **Padrão SQL E Sua Evolução - Instituto de Computação**. 2007. Disponível em: https://www.ic.unicamp.br/~geovane/mo410-091/Ch05-PadraoSQL-art.pdf. Acesso em: 10 out. 2022.

LOELIGER, J.; MCCULLOUGH, M. **Version Control with Git**: Powerful tools and techniques for collaborative software development. [*S.l.*]: "O'Reilly Media, Inc.", 2012.

LORIA, S. **Flask-Marshmallow Documentation**. marshmallow-code, 2022. Disponível em: https://flask-marshmallow.readthedocs.io/en/latest/. Acesso em: 17 out. 2022.

LORIA, S. **Marshmallow**: simplified object serialization. marshmallow-code, 2022. Disponível em: https://marshmallow.readthedocs.io/en/stable/. Acesso em: 17 out. 2022.

LUCID. **O que é um diagrama UML?**: Por que usar um diagrama uml? 2022. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-uml. Acesso em: 30 nov. 2022.

LUCIDSPARK. **Ferramenta de quadro kanban online**. 2022. Disponível em: https://lucidspark.com/pt/criar/software-kanban-online. Acesso em: 20 jun. 2022.

LUTZ, M. **Learning python**: Powerful object-oriented programming. [*S.l.*]: O'Reilly, 2013.

MAKAI, M. **Object-relational Mappers**. 2022. Disponível em: https://www.fullstackpython.com/object-relational-mappers-orms.html. Acesso em: 10 out. 2022.

MARTIN, R. C. **UML for Java programmers**. [*S.l.*]: Prentice Hall PTR, 2003.

MARTIN, R. C. **Clean code: a handbook of agile software craftsmanship**. [*S.l.*]: Pearson Education, 2009.

MELO, R. d. **Flutter**: Widget tree and state management. DEV Community, 2019. Disponível em: https://dev.to/rubensdemelo/flutter-widget-tree-and-state-management-31an. Acesso em: 17 out. 2022.

MERCADOPAGO. **Mercado Pago SDK for Python**: Mercado pago's official python sdk. Mercado Pago, 2021. Disponível em: https://github.com/mercadopago/sdk-python. Acesso em: 26 set. 2022.

MERCADOPAGO. **Mercado Pago Developers**. Mercado Pago, 2022. Disponível em: https://www.mercadopago.com.br/developers/pt. Acesso em: 26 set. 2022.

MICROSOFT. **Q&A with guido van rossum**: inventor of python. Microsoft Reactor, 2021. Disponível em: https://www.youtube.com/watch?v=aYbNh3NS7jA. Acesso em: 10 out. 2022.

MILLER, j. **Which version control should developers use?**: On-time delivery is tantamount to growth, and growth is required for success. that means you must empower your engineers with every tool possible to make their job efficient and collaborative. 2021. Disponível em: https://www.bairesdev.com/blog/which-version-control-system-developers-use/. Acesso em: 09 out. 2022.

MONTEIRO, D. *et al.* Estudo sobre os fatores de influência na fila do restaurante universitário e sua otimização. **Revista Ciências do Ambiente On-Line**, v. 7, n. 1, 2011.

NANDAN, L. **Python libraries and frameworks**. Twitter, 2022. Disponível em: https://twitter.com/NandanLohitaksh/status/1505449747355561986. Acesso em: 17 out. 2022.

NAVONE, E. C. **What is python used for?**: 10+ coding uses for the python programming language. freeCodeCamp, 2022. Disponível em: https://www.freecodecamp.org/news/what-is-python-used-for-10-coding-uses-for-the-python-programming-language/. Acesso em: 10 out. 2022.

OHNO, T.; BODEK, N. **Toyota production system**: beyond large-scale production. [*S.l.*]: Productivity press, 1988.

OLIVEIRA, J. dos S. **Ru-Tec: automação de restaurante universitário**. 2021. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2021.

ORACLE. **MySQL**: Developer zone. 2022. Disponível em: https://dev.mysql.com/. Acesso em: 10 out. 2022.

PADILLA, J. **Welcome to PyJWT¶**. 2022. Disponível em: https://pyjwt.readthedocs.io/en/stable/. Acesso em: 26 set. 2022.

PAGSEGURO. **Pagseguro developers**. Pag Seguro, 2022. Disponível em: https://dev.pagseguro.uol.com.br/. Acesso em: 26 set. 2022.

PALLETS. **Jinja Documentation**. Pallets, 2007. Disponível em: https://jinja.palletsprojects.com/en/3.1.x/. Acesso em: 17 out. 2022.

PALLETS. **Werkzeug Documentation**. Pallets, 2007. Disponível em: https://werkzeug.palletsprojects.com/en/2.2.x/. Acesso em: 17 out. 2022.

PALLETS. **Flask Documentation**. Pallets, 2010. Disponível em: https://flask.palletsprojects.com/en/2.2.x/. Acesso em: 17 out. 2022.

PALLETS. **Flask-SQLAlchemy Documentation**. Pallets, 2010. Disponível em: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/. Acesso em: 17 out. 2022.

PETERS, T. **PEP 20 – The Zen of Python**. Python Software Foundation, 2004. Disponível em: https://peps.python.org/pep-0020/#the-zen-of-python. Acesso em: 17 out. 2022.

POPPENDIECK, M.; CUSUMANO, M. A. Lean software development: A tutorial. **IEEE software**, IEEE, v. 29, n. 5, p. 26–32, 2012.

PRAKASH, A. **Top 10 github alternatives to host your open source projects**. 2020. Disponível em: https://itsfoss.com/github-alternatives/. Acesso em: 09 out. 2022.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: Uma abordagem profissional. 9. ed. [*S.l.*]: McGraw Hill Brasil, 2021.

PYTHON. **Python Package Index**: Find, install and publish python packages. Python Software Foundation, 2022. Disponível em: https://pypi.org/. Acesso em: 10 out. 2022.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de gerenciamento de banco de dados**. [*S.l.*]: AMGH Editora, 2008.

RAMEZ, E.; B, N. S. *et al.* **Fundamentals of database systems**. 2016.

RELAN, K. **Building REST APIs with Flask**: Create python web services with mysql. [*S.l.*]: Apress, 2019.

RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. **RESTful Web APIs**: Services for a changing world. [*S.l.*]: "O'Reilly Media, Inc.", 2013.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [*S.l.*]: "O'Reilly Media, Inc.", 2008.

RIUT. **Repositório Institucional da Universidade tecnológica federal do Paraná**. 2022. Disponível em: https://repositorio.utfpr.edu.br/. Acesso em: 26 set. 2022.

RODRIGUEZ, A. Restful web services: The basics. **IBM developerWorks**, v. 33, p. 18, 2008.

RONACHER, A. **Opening the flask**: How an april fools'joke became a framework with good intentions. PyCon, 2011. Disponível em: http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf. Acesso em: 17 out. 2022.

ROSSUM, G. V.; JR, F. L. D. **Python tutorial**. [*S.l.*]: Python Software Foundation, 2020.

SANDHU, R. S. Role-based access control. *In*: **Advances in computers**. [*S.l.*]: Elsevier, 1998. v. 46, p. 237–286.

SEFELIN, R.; TSCHELIGI, M.; GILLER, V. Paper prototyping-what is it good for?: A comparison of paper-and computer-based low-fidelity prototyping. *In*: **CHI'03 extended abstracts on Human factors in computing systems**. [*S.l.*: *s.n.*], 2003. p. 778–779.

SELLS, C. **What's New in Flutter 2**: Flutter web and null safety move to stable, flutter desktop moves to beta and so much more! Flutter, 2021. Disponível em: https://medium.com/flutter/whats-new-in-flutter-2-0-fe8e95ecc65. Acesso em: 17 out. 2022.

SILBERSCHATZ, A. *et al.* **Database system concepts**. [*S.l.*]: McGraw-Hill New York, 2019. v. 7.

SMARTBEAR. **Swagger Petstore**. 2022. Disponível em: https://petstore.swagger.io/. Acesso em: 10 out. 2022.

SNEATH, T. **Flutter 1.0**: Google's portable ui toolkit. Google, 2018. Disponível em: https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html. Acesso em: 17 out. 2022.

SNEATH, T. **Announcing flutter 3.3 at Flutter Vikings**. Flutter, 2022. Disponível em: https://medium.com/flutter/announcing-flutter-3-3-at-flutter-vikings-6f213e068793. Acesso em: 17 out. 2022.

SNEATH, T. **Announcing Flutter for Windows**: Build high-quality windows apps that also run on mobile and web. Flutter, 2022. Disponível em: https://medium.com/flutter/announcing-flutter-for-windows-6979d0d01fed. Acesso em: 17 out. 2022.

SOHAN, S. *et al.* A study of the effectiveness of usage examples in rest api documentation. *In*: IEEE. **2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)**. [*S.l.*], 2017. p. 53–61.

SOMMERVILLE, I. **Software Engineering, Global Edition**. tenth. [*S.l.*]: Pearson, 2015. ISBN 1-292-09613-6.

SOUZA, J. V. de *et al.* Perfil dos alunos universitários dos cursos de educação física e fisioterapia em relação à alimentação e a atividade física. **Revista Práxis**, v. 6, n. 11, 2014.

STACKOVERFLOW. **Stack overflow developer survey 2020**. 2020. Disponível em: https://insights.stackoverflow.com/survey/2020. Acesso em: 10 out. 2022.

STAN, C. **Flutter 2020 - The State of Cross-Platform**: Promises (almost) delivered. The Startup, 2020. Disponível em: https://medium.com/swlh/flutter-2020-state-of-cross-platform-814f1d8ff16. Acesso em: 17 out. 2022.

STARUML. **StarUML documentation**. [*S.l.*], 2022. Disponível em: https://docs.staruml.io/. Acesso em: 22 jun. 2022.

THYMELEAF. **Thymeleaf**. 2022. Disponível em: https://www.thymeleaf.org/. Acesso em: 26 set. 2022.

TORRES, A. *et al.* Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. **information and software technology**, Elsevier, v. 82, p. 1–18, 2017.

UERJ. **Cartão Universitário**. 2019. Disponível em: http://www.restauranteuniversitario.uerj.br/teste/cartao_uerj.html. Acesso em: 03 out. 2022.

UNIFESP. **Dúvidas Frequentes sobre a Covid-19**: Tempo de proteção, efeitos colaterais e outros pontos. 2021. Disponível em: https://sp.unifesp.br/epm/banner/perguntas-respostas-covid#como-e-transmitido. Acesso em: 19 jun. 2022.

USP. **Cartão USP para acesso aos restaurantes**: Rucard (nova versão). 2017. Disponível em: https://prg.usp.br/alunos-2/jupiter-web-em-videos/cartao-usp-para-acesso-aos-restaurantes-rucard-nova-versao/. Acesso em: 03 out. 2022.

UTFPR. **Bolsas e Programas de Assistência Estudantil**. 2017. Disponível em: https://portal.utfpr.edu.br/alunos/servicos/assistencia-estudantil/assistencia-estudantil#aux-lio-estudantil-da-utfpr. Acesso em: 30 set. 2022.

UTFPR. **Redes Sociais**. 2019. Disponível em: http://portal.utfpr.edu.br/comunicacao/produtos/redes-sociais. Acesso em: 19 out. 2022.

UTFPR. **Restaurantes Universitários da UTFPR Voltam a funcionar no dia 3/3**: Com o subsídio da universidade, valor para estudantes é mantido em r$3,50. 2022. Disponível em: http://www.utfpr.edu.br/noticias/geral/restaurantes-universitarios-da-utfpr-voltam-ao-funcionamento-no-dia-3-de-marco. Acesso em: 19 jun. 2022.

VENNERS, B.; SOMMERS, F. **A Conversation with Guido van Rossum**. artima, 2003. Disponível em: https://www.artima.com/intv/guido.html. Acesso em: 10 out. 2022.

VESTERINEN, K. **SQLAlchemy-Utils Documentation**. 2022. Disponível em: https://sqlalchemy-utils.readthedocs.io/en/latest/. Acesso em: 19 out. 2022.

VINCENT, W. S. **Django for Beginners: Build websites with Python and Django**. [*S.l.*]: WelcomeToCode, 2021.

WHEELER, D. A. **Why open source software/free software (OSS/FS)?**: Look at the numbers. 2003. Disponível em: https://dwheeler.com/oss_fs_why.html. Acesso em: 10 out. 2022.

WINDMILL, E. **Flutter in action**. [*S.l.*]: Manning, 2020.

ZOLKIFLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. **Procedia Computer Science**, Elsevier, v. 135, p. 408–415, 2018.