



UNIVERSIDADE PAULISTA

ICET - INSTITUTO DE CIÊNCIAS EXATAS E TECNOLOGIA

**CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS**

PROJETO INTEGRADO MULTIDISCIPLINAR

PIM IV

**Desenvolvimento de um sistema integrado para o controle de
operações em uma fazenda urbana, visando apoiar uma startup
inovadora na área de segurança alimentar**

Nome	R.A
João Victor Ramos do Nascimento	G7308C1
Nathalia Jacque Mendes Lima	R025063
Renan Pereira Diniz	N921714
Ruan dos Santos Oliveira	N066CF4
Vitor Antony de Marchi Castro	G775AG7

SÃO JOSÉ DOS CAMPOS – SP

NOVEMBRO / 2024

Nome	RA
João Victor Ramos do Nascimento	G7308C1
Nathalia Jacque Mendes Lima	R025063
Renan Pereira Diniz	N921714
Ruan dos Santos Oliveira	N066CF4
Vitor Antony de Marchi Castro	G775AG7

Desenvolvimento de um sistema integrado para o controle de operações em uma fazenda urbana, visando apoiar uma startup inovadora na área de segurança alimentar

Projeto Integrado Multidisciplinar (PIM) desenvolvido como exigência parcial dos requisitos obrigatórios à aprovação semestral no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da UNIP (Universidade Paulista), orientado pelo corpo docente do curso.

São José dos Campos – SP

NOVEMBRO / 2024

RESUMO

O objetivo desse projeto foi realizar a criação de um sistema disponível na *web*, *desktop* e *mobile* capaz de realizar o gerenciamento de uma fazenda urbana chamada Estrela do Vale, utilizando-se da metodologia *scrum* para separação de tarefas e a junção da metodologia PMI para gerenciamento de riscos do projeto, juntamente com a documentação e separação de áreas específicas que a fazenda vai atuar desde regras seguidas até o gerenciamento de recursos humanos do projeto. Todos os objetivos do projeto foram atingidos na parte de desenvolvimento utilizando-se do SQL Server em seu *back-end* juntamente com a linguagem Python, já para o *Front-end* foi utilizado o electron baseado na linguagem *JavaScript*, a parte *mobile* foi desenvolvida na linguagem flutter juntamente com a ferramenta FlutterFlow, integrado a um banco de dados SQL chamado Supabase para envio de avaliações de diferentes áreas da fazenda. Foram obtidos resultados satisfatórios e lições valiosas no projeto juntamente com conhecimentos chave para futuros desenvolvimentos e projetos, os dados obtidos durante o desenvolvimento foram principalmente relacionados a como a localização de um empreendimento desse quanto o custo de desenvolvimento podem causar um impacto monetário e social de grande porte. O grupo concluiu que foi um projeto trabalhoso, porém satisfatório de ser realizado, pois todos conseguiram aprender conceitos e informações que não seriam passadas no dia a dia apenas durante um projeto parecido, além de acostumar a todos a trabalharem em grupo, assim se preparando também para o mercado de trabalho.

Palavras-Chave:

PMI, gerenciamento, fazenda urbana, metodologia, desenvolvimento, documentação, SQL, avaliações.

Sumário

1. INTRODUÇÃO	5
2. PROGRAMAÇÃO ORIENTADA A OBJETOS II	7
3. DESENVOLVIMENTO DE SOFTWARE PARA INTERNET	8
4. TÓPICOS ESPECIAIS DE PROGRAMAÇÃO ORIENTADA A OBJETOS	9
5. PROJETO DE SISTEMAS ORIENTADO A OBJETOS	10
6. GERENCIAMENTO DE PROJETO DE SOFTWARE	11
7. EMPREENDEDORISMO.....	13
8. GESTÃO DA QUALIDADE	14
9. DESENVOLVIMENTO DO PROJETO	15
9.1 Programação Orientada a Objetos II	15
9.1.1 Aplicativo desktop	18
9.1.2 Diagrama entidade relacionamento	19
9.2 Desenvolvimento de Software para Internet.....	22
9.2.1 Sistema <i>web</i> responsivo.....	23
9.2.3 Mecanismos de segurança.....	23
9.2.3 Teste de usabilidade do protótipo	24
9.2.4 Autorização de Privacidade do teste de usabilidade	26
9.3 Tópicos Especiais de Programação Orientada a Objetos	27
9.3.1 Herança.....	27
9.3.2 Polimorfismo.....	28
9.3.3 Abstração	29
9.3.4 Testes de unidade	31
9.3.5 Aplicativo mobile	34
9.4 Projeto de Sistemas Orientados a Objetos.....	37
9.5 Gerenciamento de Projeto de Software.....	40
9.5.1 Partes interessadas do sistema	40

9.5.2 Escopo do projeto	41
9.5.3 Cronograma do projeto.....	41
9.5.4 Estimativa de custo do projeto	42
9.5.5 Desenvolvimento de recursos humanos.....	42
9.5.6 Sistemas de comunicação utilizados.....	43
9.5.7 Gerenciamento de risco do projeto	43
9.5.8 Encerramento do projeto	43
9.6 Empreendedorismo	45
9.6.1 Missão Principal e objetivos	45
9.6.2 Público alvo e sistemas concorrentes	45
9.7 Gestão de Qualidade	47
10. CONCLUSÃO	48

1. INTRODUÇÃO

Com o crescimento acelerado da população urbana global, as preocupações com a segurança alimentar se tornaram mais intensas. Este aumento populacional nas cidades exige uma oferta constante e acessível de alimentos frescos e saudáveis. As fazendas urbanas representam uma resposta moderna e sustentável a este desafio. Essas iniciativas, muitas vezes implementadas em espaços compactos e infraestruturas urbanas, buscam cultivar uma diversidade de produtos agrícolas, desde hortaliças até frutas, diretamente dentro das cidades. Com isso o desenvolvimento de um sistema de gestão para fazendas urbanas é uma necessidade crescente, considerando a urbanização acelerada e a demanda por soluções sustentáveis e inovadoras na produção de alimentos.

O presente projeto tem como objetivo a criação de um sistema de gerenciamento disponível nas plataformas *web*, *mobile* e *desktop*, permitindo assim um controle abrangente e acessível para os gestores da fazenda, que é chamada de Estrela do Vale, utilizando-se dos protótipos e requisitos levantados no PIM III do 3 semestre de 2024, além da continuidade em seu desenvolvimento e aplicação. Neste contexto a criação de um sistema de gestão eficaz é essencial para otimizar as operações, aumentar a produtividade e garantir a qualidade dos produtos.

Este projeto integrará as disciplinas: Programação Orientada a Objetos II, Desenvolvimento de *Software* para *Internet*, Tópicos Especiais de Programação Orientada a Objetos, Projeto de Sistemas Orientado a Objetos e Gerenciamento de Projetos de Software. Além das disciplinas complementares de Empreendedorismo e Gestão da Qualidade que serão também utilizadas no desenvolvimento do projeto.

OBJETIVO GERAL

Desenvolver um sistema de gerenciamento disponível nas plataformas *web*, *mobile* e *desktop* para o controle de produção, fornecedores, venda aos clientes, relatórios do negócio, com suas devidas separações de funcionalidades e requisitos.

Objetivos específicos

- Desenvolver e aplicar os conhecimentos adquiridos em sala de aula envolvendo as disciplinas cursadas presencialmente e a distância;
- Desenvolver a capacidade de identificar as necessidades do usuário e traduzi-las em soluções de negócios;
- Argumentar e discutir as tecnologias utilizadas nos projetos de sistemas computacionais;
- Fomentar o hábito de trabalho em equipe e execução de projetos envolvendo múltiplas disciplinas.

2. PROGRAMAÇÃO ORIENTADA A OBJETOS II

A Programação Orientada a Objetos (POO) é um paradigma de programação essencial que se tornou a espinha dorsal do desenvolvimento de *software* moderno. Desenvolvida como uma evolução dos métodos procedurais tradicionais, a POO organiza o código em torno de objetos, que são instâncias de classes e representam entidades do mundo real ou conceitos abstratos com propriedades (atributos) e comportamentos (métodos). De acordo com FARINELLI.

A proposta da orientação a Objetos é representar o mais fielmente possível as situações do mundo real nos sistemas computacionais. Nós entendemos o mundo como um todo composto por vários objetos que interagem uns com os outros. Da mesma maneira, a Orientação a Objetos consiste em considerar os sistemas computacionais não como uma coleção estruturada de processos, mas sim como uma coleção de objetos que interagem entre si. (FARINELLI, 2007, p. 4)

A adoção da POO traz inúmeras vantagens, como maior modularidade, reutilização de código, facilidade de manutenção e escalabilidade. Esses benefícios são alcançados através de princípios fundamentais como:

- **Encapsulamento:** Protege os dados internos dos objetos, permitindo acesso apenas através de métodos definidos.
- **Herança:** Permite a criação de novas classes baseadas em classes existentes.
- **Polimorfismo:** Habilita a utilização de uma interface comum para diferentes tipos de objetos.
- **Abstração:** Simplifica a complexidade através da definição de interfaces e classes abstratas.

Além desses conceitos, a POO é frequentemente complementada pelo uso de ferramentas de modelagem como a UML. Esse assunto será mais aprofundado no capítulo de Projeto de Sistemas Orientado a Objetos. Ao longo do desenvolvimento serão aplicados conceitos de POO para o produto ser um sistema robusto e eficiente em sua função.

3. DESENVOLVIMENTO DE SOFTWARE PARA INTERNET

A alta busca por produtos e serviços no ambiente da *web*, fez aumentar a necessidade de refinamentos e mais organização no desenvolvimento dessas plataformas sendo importante ao ponto de um *site* conseguir mais ou menos vendas conforme Flávio Waltz(2024, slide 37 - aula 2 desenvolvimento de *software* para *internet*)"A Amazon mostrou que a redução do tempo de carregamento da página em apenas 100ms resultou em um aumento de 1% na receita."

A experiência do usuário (UX) tornou-se um fator determinante , uma navegação confusa ou conteúdo desatualizado pode afastar os clientes e prejudicar a reputação da marca conforme Flávio Waltz(2024, p.29) - aula 2 desenvolvimento de *software* para *internet*" Sites acessíveis também são indexados de forma mais eficaz por mecanismos de busca como o Google".

4. TÓPICOS ESPECIAIS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

A importância dos tópicos especiais em Python reside na capacidade de equipar os programadores com ferramentas e técnicas que vão além do básico. Ao explorar esses temas, os desenvolvedores podem criar aplicações mais eficientes, escaláveis e de fácil manutenção. Por exemplo, os conceitos de herança e associação conforme Thiago Leite e Carvalho(2024, p. 22)"Eles criam um relacionamento que possibilita o reuso de forma mais prática e menos propícia a erros."

Python é uma linguagem versátil e poderosa, amplamente utilizada em diversas áreas, desde desenvolvimento web até análise de dados e inteligência artificial conforme Nilo Ney Coutinho Menezes(2024, p.37)"[...] Python é famosa por ter baterias inclusas, fazendo referência a um produto completo que pode ser usado prontamente".

5. PROJETO DE SISTEMAS ORIENTADO A OBJETOS

A disciplina de Projeto de Sistemas Orientados a Objetos (POO) é fundamental para o desenvolvimento de software moderno, fornecendo aos alunos as ferramentas necessárias para projetar sistemas robustos e escaláveis. De acordo com BACURAU (2022, p. 1) "A Programação Orientada a Objetos (POO) é o paradigma mais utilizado atualmente. Os recursos da POO, da UML e dos processos de desenvolvimento, permitem modelar e desenvolver software com maior eficiência e qualidade."

A principal parte desta disciplina é a utilização de diagramas UML (*Unified Modeling Language*), que são ferramentas poderosas para visualizar, especificar, construir e documentar os artefatos de um sistema orientado a objetos. Os diagramas UML permitem que os desenvolvedores representem graficamente a estrutura e o comportamento do sistema, facilitando a compreensão e a comunicação entre os membros da equipe.

De acordo com Santander e Castro (2000), a dificuldade na definição de requisitos se deve principalmente a dois fatores: a dificuldade da comunicação entre o usuário e o desenvolvedor e o fato de os clientes não saberem o que realmente desejam do sistema.

Utilizar UML no desenvolvimento de sistemas garante que todos os aspectos do projeto sejam bem planejados e documentados, reduzindo a possibilidade de erros e retrabalho. A integração dos conceitos de POO com a utilização de diagramas UML não só melhora a qualidade do *software* desenvolvido, quanto também alinha as funcionalidades do sistema entre a equipe de desenvolvimento e o gerente do projeto.

6. GERENCIAMENTO DE PROJETO DE SOFTWARE

A complexidade do desenvolvimento de *software* moderno exige métodos eficazes de gerenciamento de projetos para garantir a entrega de produtos de alta qualidade dentro do prazo e orçamento estipulados. De acordo com LAGO.

Com o passar do tempo o gerenciamento de projetos foi amadurecendo, problemas foram surgindo ao longo do caminho, e várias soluções foram tentadas até se chegar a que hoje parece ser a ideal; surgiram profissões específicas ligadas à prática de gerenciamento de projetos; em algumas empresas foram criados departamentos específicos para cuidar do gerenciamento de projetos, e em 1969, em Filadélfia, Pensilvânia, EUA, surgiu o *Project Management Institute* (PMI®). (LAGO 2007. P.)

Neste contexto, a adoção da Metodologia PMI (*Project Management Institute*) destaca-se como uma abordagem sistemática e amplamente reconhecida para o gerenciamento de projetos. Com base em referências bibliográficas devidamente citadas e relacionadas, este estudo apresentará os métodos de gerenciamento de projetos de *software*, focando especialmente na utilização da metodologia PMI.

A Metodologia PMI, conforme descrito no PMBOK (*Project Management Body of Knowledge*), oferece um conjunto de melhores práticas que podem ser aplicadas ao gerenciamento de projetos em diversas áreas, incluindo o desenvolvimento de *software*. Este trabalho descreverá o processo de utilização dessa metodologia no suporte às atividades de gerenciamento do projeto de *software*, abordando as questões relacionadas às demandas do sistema proposto.

O objetivo é demonstrar como a aplicação desta metodologia pode melhorar o gerenciamento do projeto, trazendo benefícios tanto para o sistema quanto para as pessoas envolvidas. Os ganhos esperados incluem maior clareza na definição dos objetivos do projeto, melhor controle dos prazos e custos, e uma comunicação mais eficiente entre as partes interessadas.

Durante o desenvolvimento uma das partes mais importantes é ciclo de vida do projeto, que pode ser dividida em Iniciação, planejamento, execução, monitoramento e controle e finalmente o encerramento.

Na fase de iniciação, o foco está na definição do escopo e dos objetivos do projeto, identificando as partes interessadas e estabelecendo um quadro inicial que guiará todo o processo.

O planejamento envolve a criação de um plano detalhado que inclui cronogramas, alocação de recursos, e identificação de riscos, assegurando que todos os aspectos do projeto sejam cuidadosamente considerados e organizados.

A execução é a fase em que as atividades planejadas são implementadas, e os entregáveis do projeto começam a tomar forma. Durante esse período, é crucial manter um controle rigoroso para assegurar que o progresso esteja alinhado com o plano.

O monitoramento e controle serve para acompanhar o andamento do projeto em tempo real, permitindo ajustes necessários e gestão de mudanças para manter o projeto nos trilhos.

Já o encerramento marca a conclusão formal do projeto, onde os resultados são entregues, avaliados e documentados, e as lições aprendidas são registradas para futuros projetos. Este ciclo de vida do projeto não só facilita uma gestão mais organizada e eficiente, mas também contribui para a melhoria contínua dos processos e métodos aplicados.

7. EMPREENDEDORISMO

Empreendedorismo é uma força motriz essencial que pode revolucionar o desenvolvimento de sistemas. Ele promove a inovação, incentiva a busca por soluções criativas e eficazes, e impulsiona a implementação de novas ideias. Aplicar princípios empreendedores no desenvolvimento de um sistema significa adotar uma mentalidade proativa e voltada para a resolução de problemas, onde a identificação de oportunidades e a capacidade de adaptação são primordiais.

De acordo com Ruiz, 2019:

A literatura sobre empreendedorismo descreve há décadas o importante papel econômico dos empreendedores ligado ao desenvolvimento de inovações. Tendo os empreendedores um incentivo econômico (lucro) para o desenvolvimento de novos produtos e serviços (pela geração de novas receitas e novos clientes), estes ficam mais propensos para tal atividade e ajudam no desenvolvimento das economias em que estão inseridos (Ruiz, 2019, p. 6).

Os empreendedores trazem uma perspectiva única, focada em agregar valor ao usuário final e em manter a competitividade no mercado. Essa abordagem permite não apenas a criação de sistemas mais eficientes e ajustados às necessidades dos usuários, mas também a exploração de novos mercados e tecnologias. Com a mentalidade de um empreendedor, o desenvolvimento de um sistema pode transcender a simples implementação técnica, transformando-se em uma jornada de inovação contínua e de melhoria constante.

8. GESTÃO DA QUALIDADE

A qualidade é o pilar fundamental para o sucesso de qualquer projeto, visando atender ao que foi acordado com o cliente. Conforme Crosby (1986, p. 31) “qualidade é a conformidade do produto às suas especificações”. Entender a importância da qualidade é crucial para aqueles que lideram projetos, pois está intrinsecamente ligada à reputação da empresa, à satisfação dos clientes e à sustentabilidade do negócio.

Escolher ferramentas e metodologias adequadas, aliada a uma equipe que entende das normas do produto, conforme ' (OAKLAND, 1994, p. 459) “[...]estabelecer a infraestrutura necessária para garantir o melhoramento anual da qualidade; identificar as necessidades específicas de melhorias, ou seja, os projetos de melhoramento; estabelecer para cada projeto uma equipe com claras responsabilidades, para levá-lo a uma conclusão bem-sucedida”, permite a entrega final com qualidade e possibilidades de melhorias.

9. DESENVOLVIMENTO DO PROJETO

Nesta parte do projeto será feita a implementação de todas as pesquisas realizadas, além de toda a análise do cenário físico e lógico de todos os sistemas necessários para o bom funcionamento da fazenda urbana, além da apresentação de todos os diagramas, atendendo sempre a todos os requisitos apresentados pelos professores e documentos disponibilizados.

9.1 Programação Orientada a Objetos II

O projeto da Fazenda Urbana Estrela do Vale foi desenvolvido com Python e Django para o *back-end*, visando atender às necessidades de gestão operacional da fazenda. O *framework* Django foi escolhido por sua estrutura *Model-View-Template* (MVT), que organiza a aplicação de forma eficiente, facilitando a implementação de operações *CRUD* (*Create, Read, Update, Delete*) e a integração com o banco de dados (Django *Documentation...*, 2024), além de prover ferramentas para autenticação e autorização de usuários.

Na arquitetura *MVT*, o *Model* faz o mapeamento do banco de dados, permitindo a estruturação e manipulação dos dados da aplicação, o *Template* geralmente serve como a interface visual, renderizando o HTML para navegação dos usuários, e o *View* representa a lógica de negócio, definindo o que acontece em cada interação do usuário.

No entanto, neste projeto, o *Template* não foi utilizado, uma vez que o *front-end* foi desenvolvido separadamente com HTML, JavaScript e CSS. Dessa forma, o Django atua como *back-end* exclusivo, oferecendo uma estrutura bem-organizada e modular para gerenciar todas as operações e dados necessários ao funcionamento da Fazenda Urbana Estrela do Vale.

A linguagem Python foi selecionada pela sua simplicidade e robustez, enquanto Django oferece uma plataforma segura e escalável para o *back-end*. Para a interface de usuário, foram utilizados HTML, CSS e JavaScript, proporcionando uma experiência intuitiva e amigável, com fácil navegação e usabilidade.

Com o avanço do desenvolvimento do projeto, optou-se pelo uso do Docker para hospedar o banco de dados. O Docker é uma plataforma que possibilita a criação, o envio e a execução de aplicações em contêineres, o que proporciona um ambiente isolado e consistente para a execução de serviços. A utilização do Docker para hospedar o banco de dados apresenta diversas vantagens, tais como a facilidade de configuração e gerenciamento, a portabilidade entre diferentes ambientes e a capacidade de garantir que o banco de dados seja executado de maneira uniforme em todas as etapas do desenvolvimento.

Neste projeto, o banco de dados Microsoft SQL Server foi configurado em um contêiner Docker, facilitando a integração com a aplicação Django. O processo foi realizado por meio de um arquivo “docker-compose.yml”, que define a imagem do SQL Server, as variáveis de ambiente e as configurações de rede. A comunicação entre a aplicação em Python e o banco de dados foi possibilitada pela biblioteca pyodbc, que estabelece uma conexão eficiente entre o Python e o SQL Server.

Para criar APIs *RESTful* foi utilizado o Django REST Framework (DRF), permitindo a comunicação entre o *front-end* e o *back-end* de forma estruturada em arquivos dentro dos arquivos “views.py”. No projeto, o DRF é utilizado para serializar os dados dos modelos e construir *endpoints* para operações CRUD respeitando os métodos HTTP. Métodos como “post_criar”, “get_lista”, “get_detalhe” e “put_editar” são implementados nas *views* para gerenciar as operações de criação, leitura, atualização e inativação de registros.

Na operação *Create* “post_criar”, utilizamos um método *POST*, apresentado na figura 1 na próxima página, para inserir novos registros no banco de dados. Primeiro, os dados da requisição são carregados e validados. Se um usuário associado não for informado ou encontrado, uma mensagem de erro é retornada. Caso contrário, o objeto é criado, validado com “full_clean()” para garantir que os dados atendam às regras do modelo, e salvo no banco de dados.

Figura 1 - Codificação do método *POST* para criar um produto

```
@csrf_exempt
@gerente_required
@login_required_middleware
def post_criar(request):
    if request.method != "POST":
        return HttpResponseNotAllowed(["POST"], "Método não permitido")

    try:
        data = json.loads(request.body)
        fornecedor = Fornecedor.objects.get(id=data["fornecedor"])
        data["fornecedor"] = fornecedor
        produto = Produto(**data) # instancia um objeto do tipo Produto
        produto.full_clean() # valida os dados do objeto
        produto.save() # salva o objeto no banco de dados
        return JsonResponse({"id": produto.id}, status=201)

    except json.JSONDecodeError:
        return JsonResponse({"erro": "Corpo da requisição inválido"}, status=400)
    except ObjectDoesNotExist:
        return JsonResponse({"erro": "Fornecedor não encontrado"}, status=404)
    except ValidationError as e:
        return JsonResponse({"erro": e.message_dict}, status=400)
    except Exception as e:
        return JsonResponse({"erro": f"Erro inesperado {str(e)}"}, status=500)
```

Fonte: Os autores, 2024.

Na operação *Read* “get_lista” e “get_detalhe”, temos duas funções: “get_lista”, para listar registros de acordo com filtros recebidos na requisição e “get_detalhe”, que recupera e retorna os dados de um objeto específico usando seu identificador primário “pk” pelo método *GET*. O “get_lista” utiliza um filtro construído dinamicamente com “build_filters”, permitindo busca personalizada, enquanto “get_detalhe” verifica a existência do registro antes de retornar os dados.

Para a operação *Update* “put_editar”, a função “put_editar” permite a atualização de registros específicos com método *PUT*. Ela carrega os dados do corpo da requisição e atualiza apenas os atributos válidos no objeto. Após a atualização, o objeto é validado com “full_clean()” e salvo no banco de dados. Em caso de erro, como atributos inválidos ou JSON (*JavaScript Object Notation*) malformado, mensagens de erro são retornadas para informar o problema.

Finalmente, na operação *Delete*, em vez de excluir o registro, ele é marcado como inativo, promovendo a segurança e a integridade dos dados. Essa abordagem evita a perda definitiva de informações e mantém um histórico de registros para referência futura.

9.1.1 Aplicativo desktop

O aplicativo *desktop* foi desenvolvido utilizando-se do *framework* Electron juntamente com JavaScript e HTML. Essa linguagem foi escolhida devido a capacidade de junção do *Web* e *desktop* permitindo assim uma significativa aceleração no processo de desenvolvimento, economizando recursos e permitindo a capacidade de focar em realizar a documentação dos processos de desenvolvimento.

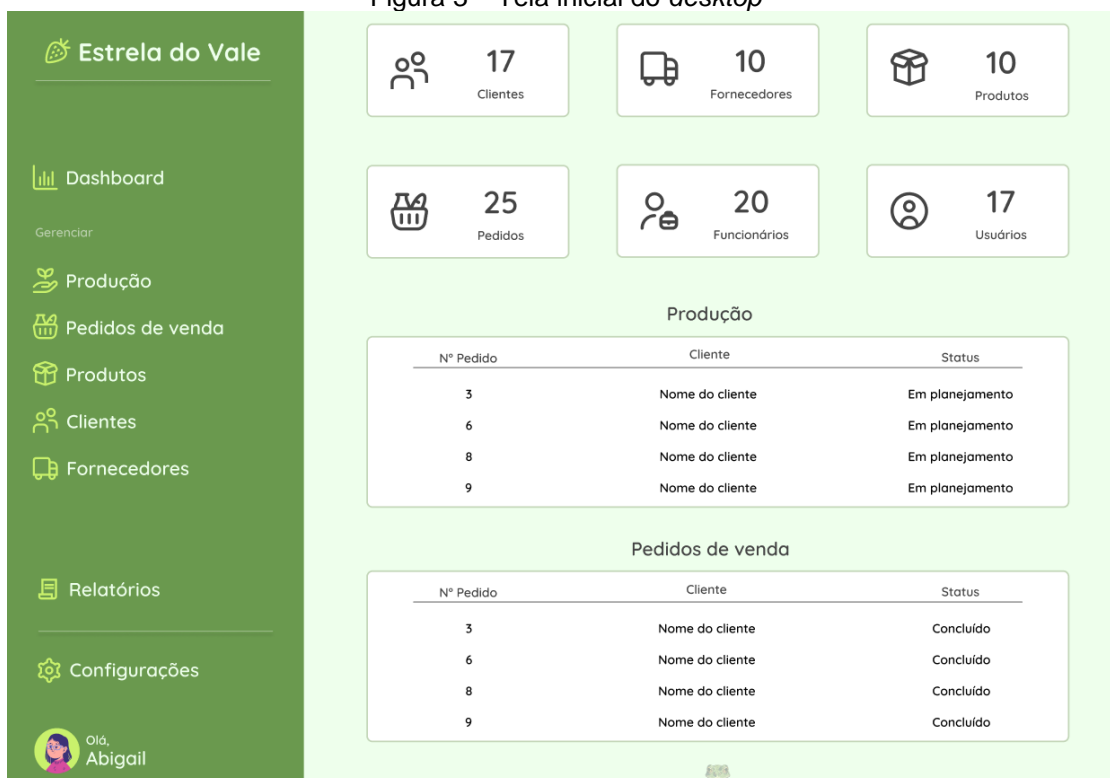
Na figura 2 abaixo está a captura da tela de *login* do aplicativo, já em sua versão final.

Figura 2 – Tela de *login* do desktop



Fonte: Os autores, 2024.

Na figura 3 abaixo se encontra a tela inicial do aplicativo *desktop* onde o usuário é redirecionado logo após realizar o login no aplicativo com seu *login*(endereço de *e-mail*) e senha de acesso.

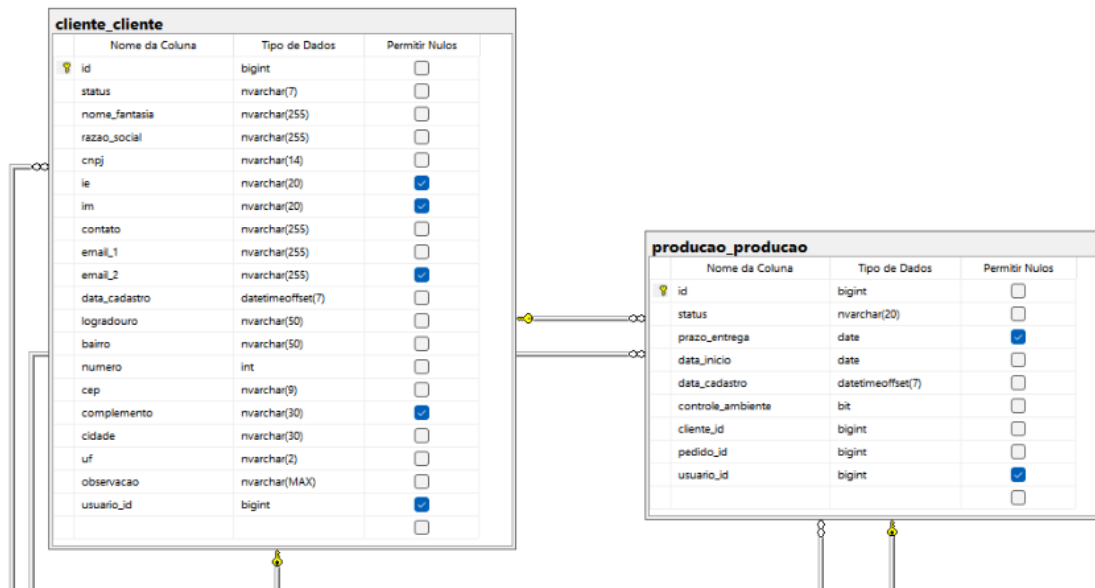
Figura 3 – Tela inicial do *desktop*


Fonte: Os autores, 2024.

9.1.2 Diagrama entidade relacionamento

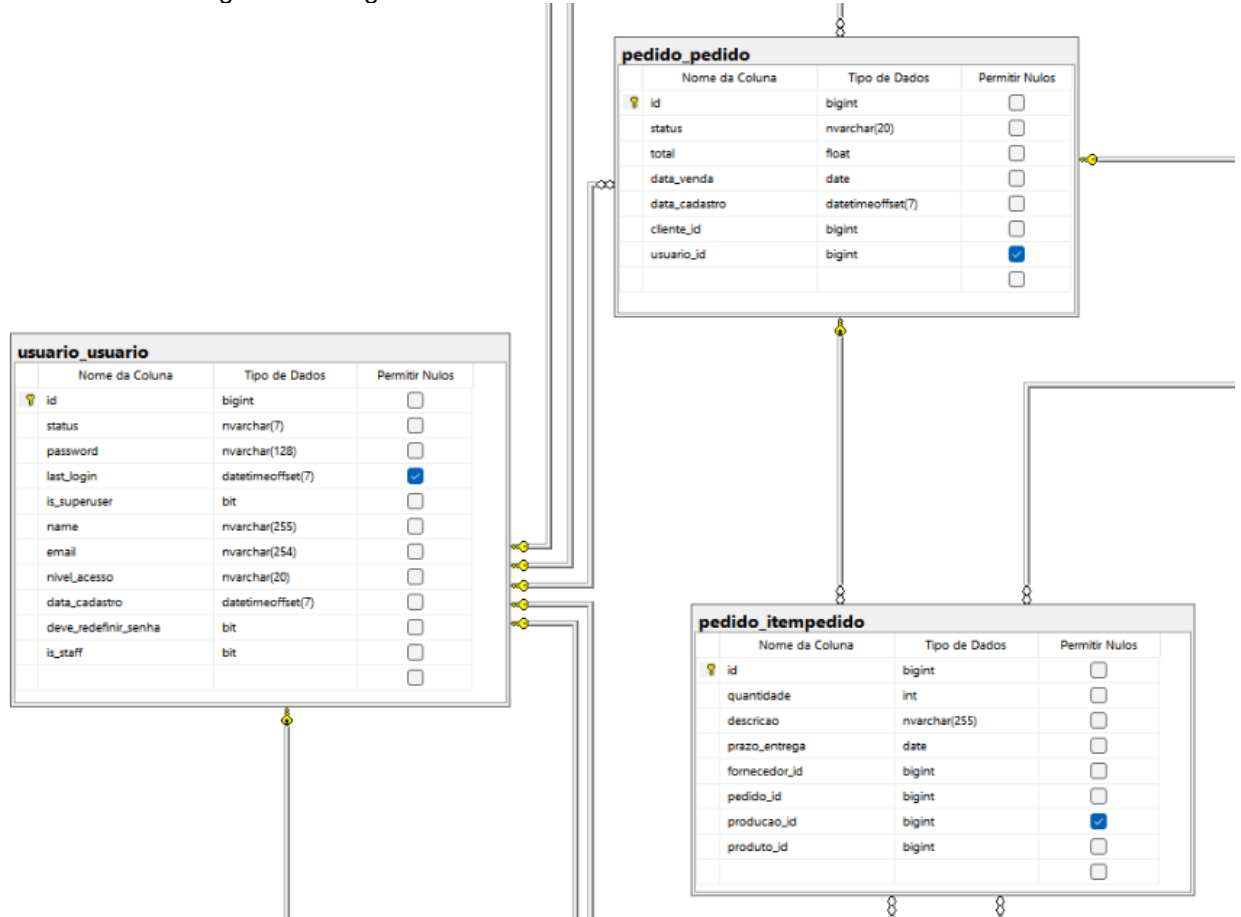
O desenvolvimento foi guiado pelos diagramas de modelagem, assegurando a conformidade com a estrutura de classes e relacionamentos, de forma que o código implementado reflete a modelagem inicial do projeto. Os modelos principais como Usuario, Cliente, Fornecedor, Funcionario, Produto, Pedido e Producao foram construídos para representar as entidades do sistema. O diagrama de classes UML detalhou cada entidade com seus respectivos atributos e relacionamentos, proporcionando uma base estruturada para o desenvolvimento, com foco em manter a consistência entre a modelagem inicial e o código final conforme as figuras 4 5 6 e 7 apresentadas nas próximas páginas.

Figura 4 – Diagrama entidade relacionamento do sistema. Parte 1



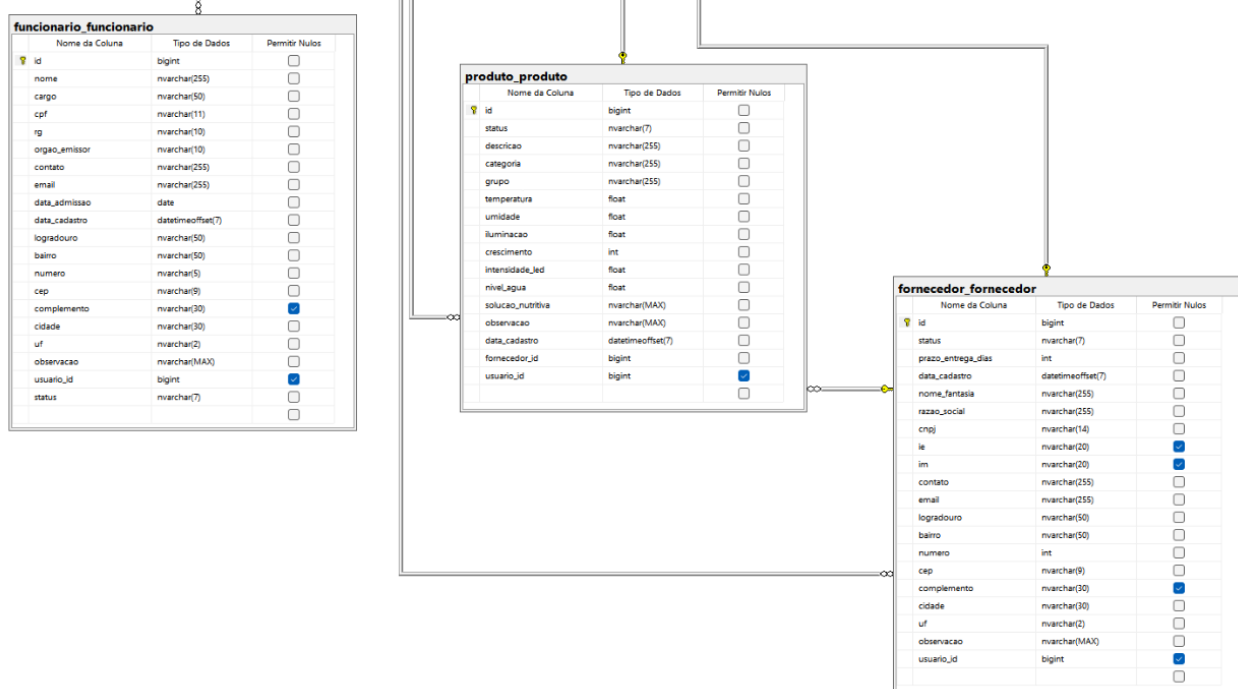
Fonte: Os autores, 2024.

Figura 5 – Diagrama entidade relacionamento do sistema. Parte 2



Fonte: Os autores, 2024.

Figura 6 – Diagrama entidade relacionamento do sistema. Parte 3



Fonte: Os autores, 2024.

[illegible]

Fonte: Os autores, 2024.

No desenvolvimento da parte *web* do projeto foi utilizado a tecnologia Electron, a aplicação oferece uma experiência consistente e otimizada em ambas as plataformas *web* e *Desktop*. Assim providenciando um desenvolvimento mais ágil e com melhor integração entre plataformas.

- Atualizações instantâneas exibidas na interface do usuário, permitindo a tomada de decisões informadas.
- Design amigável e intuitivo, facilitando o uso por usuários com diferentes níveis de habilidade técnica.

- Atualizações instantâneas exibidas na interface do usuário, permitindo a tomada de decisões informadas.
- Design amigável e intuitivo, facilitando o uso por usuários com diferentes níveis de habilidade técnica.

9.2.1 Sistema web responsivo

Durante o desenvolvimento foi realizado a implementação de um sistema web responsivo que se adapta a diferentes telas, sem se desconfigurar ou vazar da tela. A figura 8 abaixo apresenta uma parte do código implementando um container em *JavaScript*.

Figura 8 – Container em JavaScript

```
<style>
.container {
  width: 500px;
  height: 300px;
  overflow: hidden; /* Esconde qualquer parte da imagem que ultrapassar o contêiner */
  position: relative;
}
body, html {
  margin: 0;
  padding: 0;
  overflow: hidden; /* Remove a barra de rolagem */
  height: 100%;
}
</style>
</style>

</head>
<body>
  <div class="container2">
    <div class="login">
      <div class="container">
        
      </div>

      <div class="desktop-15">
        <img class="menu-principal-icon" alt="" >

        <div class="gerenciar">Gerenciar</div>
        <div class="produo-wrapper">
          <div class="produo">Produção</div>
        </div>
      </div>
    </div>
  </div>
</body>
```

Fonte: Os autores, 2024.

9.2.3 Mecanismos de segurança

Uma senha segura é fundamental em qualquer programa que envolve autenticação de usuários, pois é a primeira linha de defesa contra acessos não autorizados e violações de segurança. A figura 9 abaixo apresenta um trecho do código onde é implementado uma camada de segurança para as senhas.

Figura 9 – Segurança do sistema web

```
async function registerUser(username, password) {
  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(password, salt);

  users.push({ email: email, password: hashedPassword });
  console.log('Usuário registrado com sucesso!');
}

async function loginUser(email, password) {
  const user = users.find(user => user.username === username);
  if (!user) {
    console.log('Usuário não encontrado!');
    return;
  }

  const isMatch = await bcrypt.compare(password, user.password);
  if (isMatch) {
    console.log('Login bem-sucedido!');
  } else {
    console.log('Senha incorreta!');
  }
}

async function main() {
  await registerUser('usuario1', 'senhaSegura123');
  await loginUser('usuario1', 'senhaSegura123');
  await loginUser('usuario1', 'senhaIncorreta');
}

main();
```

Fonte: Os autores, 2024.

O código apresentado acima na figura 9 implementa um sistema de login seguro utilizando JavaScript e a biblioteca bcryptjs. Inicialmente, simula um banco de dados de usuários através de uma lista. A função `registerUser` registra novos usuários, gerando um *hash* seguro para a senha usando `bcrypt` e armazenando o nome de usuário junto com a senha criptografada. A função “`loginUser`” verifica a existência do nome de usuário e compara a senha fornecida com a senha armazenada de forma segura.

9.2.3 Teste de usabilidade do protótipo

Durante o desenvolvimento foi realizado o teste de usabilidade do sistema com uma pessoa, sendo elas não ligadas a área de TI, para maior fidelidade dos testes.

O participante abaixo trabalha na empresa na integrante do grupo Nathalia, porém em uma área diferente, não tão focada em desenvolvimento.

Durante o teste de usabilidade, foi solicitada ao participante a tarefa de verificar a listagem de usuários do sistema. Ao iniciar, o usuário logou no sistema e foi direcionado automaticamente para a tela de dashboard. Na dashboard, identificou um informativo que exibe a quantidade total de usuários cadastrados, o que o levou a acreditar que seria possível acessar a listagem completa de usuários por meio desse indicativo. Contudo, essa não é a maneira principal de acessar a funcionalidade de gerenciamento de usuários. O caminho correto seria pelo menu lateral, através da opção Configurações > Gerenciar usuários. A ausência de um atalho direto ou de uma opção específica para Usuários no menu lateral pode influenciar negativamente na usabilidade, visto que o usuário se sentiu inclinado a buscar a funcionalidade diretamente pela dashboard.

O participante a seguir também trabalha na empresa na integrante do grupo Nathalia, porém com uma área diferente, não tão focada em desenvolvimento.

No segundo teste de usabilidade, foi solicitado ao participante a tarefa de cadastrar um novo fornecedor no sistema. O usuário iniciou o processo realizando login e foi direcionado automaticamente para a tela de *dashboard*. Sem dificuldades, o participante identificou no menu lateral a opção "Fornecedores" e, ao clicar nela, foi direcionado para a página de gerenciamento de fornecedores. Na tela de fornecedores, o participante localizou rapidamente o botão "Novo", que estava destacado no canto superior direito da página. Ao clicar nesse botão, o sistema exibiu o formulário de cadastro de fornecedor, permitindo que o usuário preenchesse as informações necessárias para concluir a tarefa. Todo o processo foi realizado de forma intuitiva e sem interrupções, evidenciando que a disposição lógica e a nomenclatura clara das opções no menu facilitaram o fluxo da tarefa.

A conclusão bem-sucedida da tarefa e a facilidade encontrada pelo participante durante o teste reforçam a eficácia do design atual para essa funcionalidade. A acessibilidade e organização do menu, aliadas à presença de botões claros e destacados, contribuem positivamente para a experiência do usuário.

O termo assinado pelos participantes dos testes se encontra na próxima página.

9.2.4 Autorização de Privacidade do teste de usabilidade

Autorização de privacidade

Eu abaixo assinado, autorizo a participação no teste de usabilidade do sistema da fazenda Estrela do Vale, desenvolvido pelo time de desenvolvimento SAPOO. Compreendo e aceito os termos e condições estabelecidos abaixo:

Objetivo do Teste: O teste de usabilidade visa avaliar a eficácia, eficiência e satisfação dos usuários ao interagir com o sistema. Os dados coletados serão utilizados para melhorar o design e a funcionalidade do sistema.

Coleta de Dados: Durante o teste, serão coletadas as seguintes informações:

Vídeos e áudios das sessões de teste, incluindo a tela e a voz do participante, para fins de análise. Os dados coletados serão utilizados exclusivamente para fins de análise e melhoria do sistema. Garantimos a confidencialidade total de todas as informações adquiridas.

Direitos do Participante:

A participação no teste é totalmente voluntária. O participante pode desistir a qualquer momento sem qualquer penalidade. O participante tem o direito de solicitar o anonimato de suas informações pessoais. O participante pode solicitar acesso aos dados coletados durante o teste.

Termo de Consentimento: Ao assinar este documento, declaro que compreendo e concordo com os termos aqui descritos e autorizo a coleta e o uso dos meus dados conforme especificado.

Assinatura do Participante: _____

9.3 Tópicos Especiais de Programação Orientada a Objetos

Nesta parte do projeto serão mostrados todos os requisitos solicitados pela disciplina, esses incluem, desenvolvimento de classes que estejam em total conformidade com o diagrama de classe a ser apresentado na documentação do projeto, aplicados os conceitos fundamentais de herança, polimorfismo e abstração, e para garantir a qualidade e a robustez do sistema, serão desenvolvidos testes de unidade automatizados.

9.3.1 Herança

A herança, no exemplo da figura 10 abaixo, descreve como a classe “Produto” herda as funcionalidades da classe abstrata “StatusModel”, que define um campo de status com as variáveis predefinidas “ATIVO” e “INATIVO”. Por ser abstrata, “StatusModel” serve apenas como base para outras classes, sem gerar uma tabela própria no banco de dados. Essa herança facilita a reutilização do campo status e dos valores padrão em qualquer classe que a estenda, simplificando o código e mantendo a consistência.

Figura 10 – Seção de código com exemplo de herança.

```
from django.db import models

class StatusModel(models.Model):
    STATUS_CHOICES = [
        ("ATIVO", "Ativo"),
        ("INATIVO", "Inativo"),
    ]

    status = models.CharField(max_length=7, choices=STATUS_CHOICES, default="ATIVO")
    ...

You, 6 minutes ago | 1 author (You)
class Produto(StatusModel):
    CATEGORIA_CHOICES = [
        ("HORTALICA_DE_FLOR", "Hortaliça de Flor"),
        ("HORTALICA_DE_FOLHA", "Hortaliça de Folha"),
        ("TUBERCULO", "Tubérculo"),
    ]

    descricao = models.CharField(max_length=255)
    categoria = models.CharField(max_length=255, choices=CATEGORIA_CHOICES)
    ...
```

Fonte: Os autores, 2024.

Na classe `Pedido`, a herança de `StatusModel` permite que o campo `status` seja incorporado com as mesmas opções e funcionalidade, associando a cada instância de `Pedido` um `status` específico.

Essa abordagem reduz a repetição de código e torna a manutenção mais eficiente, uma vez que alterações no modelo de `status` se aplicam automaticamente a todas as classes que herdarem de `StatusModel` sendo predominantemente utilizado pelas demais classes desenvolvidas.

9.3.2 Polimorfismo

O polimorfismo se aplica no tratamento das exceções na *view* de cliente, especificamente no bloco *try-except*. O método “`post_criar`” trata várias exceções de maneira diferente, de acordo com o tipo de erro que ocorre. Embora todas as exceções sejam tratadas dentro de um bloco *except*, elas possuem comportamentos distintos dependendo do tipo da exceção:

1. ***json.JSONDecodeError***: Quando o corpo da requisição não é um *JSON* válido, uma resposta específica é gerada, com uma mensagem de erro como “Corpo da requisição inválido”.
2. ***ValidationError***: Quando ocorre um erro de validação nos dados, a exceção é capturada, e os erros de validação são retornados como um dicionário com detalhes específicos sobre os campos inválidos.
3. **Exceção genérica (*Exception*)**: Para qualquer outro erro inesperado, uma mensagem genérica de erro é retornada, informando o tipo do erro com um código de *status* 500.

Cada uma dessas exceções é tratada de forma única, mas todas seguem o mesmo padrão de retornar uma resposta *JSON* com a mensagem de erro apropriada. O comportamento do código (a forma como a resposta é construída) varia dependendo da exceção capturada, o que é um exemplo clássico de polimorfismo, onde um único método “*except*” se comporta de maneiras diferentes dependendo do tipo de exceção.

A Figura 11 apresentada na página seguinte apresenta um exemplo de polimorfismo no código utilizado em uma parte de nosso projeto.

Figura 11 – Trecho de código com polimorfismo.

```
@csrf_exempt
@gerente_required
@login_required_middleware
def post_criar(request):
    if request.method != "POST":
        return HttpResponseNotAllowed(["POST"], "Método não permitido")

    try:
        data = json.loads(request.body)
        usuario_id = data.pop("usuario", None)

        if not usuario_id:
            return JsonResponse({"erro": "Usuário não informado"}, status=400)

        try:
            usuario = Usuario.objects.get(id=usuario_id)
        except Usuario.DoesNotExist:
            return JsonResponse({"erro": "Usuário não encontrado"}, status=400)

        cliente = Cliente(usuario=usuario, **data)
        cliente.full_clean()
        cliente.save()
        return JsonResponse({"id": cliente.id}, status=201)

    except json.JSONDecodeError:
        return JsonResponse({"erro": "Corpo da requisição inválido"}, status=400)
    except ValidationError as e:
        return JsonResponse({"erro": e.message_dict}, status=400)
    except Exception as e:
        return JsonResponse({"erro": f"Erro inesperado {str(e)}"}, status=500)
```

Fonte: Os autores, 2024.

O polimorfismo está presente na maneira como o método `post_criar` lida com diferentes tipos de exceções (`JSONDecodeError`, `ValidationError`, `Exception`), retornando respostas apropriadas para cada situação. Isso permite que a *view* de cliente tenha um tratamento flexível e adequado para diversas situações de erro, garantindo que as respostas sejam sempre claras e específicas conforme o tipo de falha encontrada.

9.3.3 Abstração

No código da Figura 12 apresentada na próxima página, observa-se o conceito de abstração aplicado na classe “`StatusModel`”, definida como uma classe abstrata.

Figura 12 – Exemplo de abstração no código

```
from django.db import models

class StatusModel(models.Model):
    STATUS_CHOICES = [
        ("ATIVO", "Ativo"),
        ("INATIVO", "Inativo"),
    ]
    status = models.CharField(max_length=10, choices=STATUS_CHOICES, default="ATIVO")

    class Meta:
        abstract = True # Define a classe como abstrata
```

Fonte: Os autores, 2024.

Essa classe encapsula o campo *status* com opções predefinidas ("ATIVO" e "INATIVO"), permitindo que outras classes compartilhem essa funcionalidade sem precisar redefinir o campo. A abstração ocorre ao se separar essa funcionalidade comum, proporcionando um modelo genérico que pode ser reutilizado sem gerar uma tabela própria no banco de dados.

Ao herdar de *StatusModel*, as classes *Produto* e *Usuario* por exemplo adquirem o campo *status*, o que simplifica o código e favorece a manutenção. Essa estrutura abstrai a funcionalidade comum de *status*, tornando o modelo mais organizado e evitando a repetição, permitindo que cada classe foque em atributos e comportamentos específicos.

A figura 13 apresentada na próxima página, apresenta a implementação do conceito acima apresentado.

Figura 13 – Exemplo de abstração no código

```
class Produto(StatusModel):
    descricao = models.CharField(max_length=255)
    categoria = models.CharField(max_length=255, choices=[
        ("HORTALICA_DE_FLOR", "Hortaliça de Flor"),
        ("HORTALICA_DE_FOLHA", "Hortaliça de Folha"),
        ("TUBERCULO", "Tubérculo"),
    ])
    fornecedor = models.ForeignKey(Fornecedor, models.CASCADE)
    data_cadastro = models.DateTimeField(auto_now_add=True, editable=False)
    usuario = models.ForeignKey(
        Usuario, on_delete=models.SET_NULL, null=True, editable=False
    )
    # Outros campos e métodos específicos do produto

    class Meta:
        verbose_name = "Produto"
        verbose_name_plural = "Produtos"

class Usuario(StatusModel, AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(unique=True)
    nome = models.CharField(max_length=255)
    # Outros campos e métodos específicos do usuário

    class Meta:
        verbose_name = "Usuário"
        verbose_name_plural = "Usuários"
```

Fonte: Os autores, 2024.

9.3.4 Testes de unidade

Os testes apresentados das figura 14 presente na próxima página, verifica a funcionalidade da API relacionada ao modelo Fornecedor, cobrindo ações como criação, consulta, atualização e tratamento de erros para *endpoints* específicos.

A classe *FornecedorAPITests* utiliza o Django *TestCase* em conjunto com o *framework pytest* para simular requisições e avaliar a resposta da API, assegurando que os *endpoints* retornem os status e dados esperados em diferentes cenários.

Figura 14 – Trecho de código definindo os testes de unidade

```
You, 3 weeks ago | 2 authors (You and one other)
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth import get_user_model
from .models import (
    Fornecedor,
    Usuario,
)
import json

Usuario = get_user_model()

You, 3 weeks ago | 2 authors (You and one other)
class FornecedorAPITests(TestCase):
    def setUp(self):
        self.client = Client()

        # Criar um usuário de teste
        self.user = Usuario.objects.create_user(
            email="testuser@example.com", password="testpassword", nivel_acesso="ADMINISTRADOR"
        )

        # Autenticar o usuário de teste
        self.client.login(email="testuser@example.com", password="testpassword")

        self.fornecedor_data = {
            "nome_fantasia": "Fornecedor Teste",
            "razao_social": "Fornecedor Teste LTDA",
            "cnpj": "12345678000195", # Um CNPJ fictício válido
            "ie": "1234567890",
            "im": "1234567890",
            "contato": "Contato Teste",
            "email": "contato@test.com",
            "logradouro": "Rua Teste",
            "bairro": "Bairro Teste",
            "numero": "123",
            "cep": "12345-678",
            "complemento": "Apto 1",
            "cidade": "Cidade Teste",
            "uf": "SP",
            "observacao": "Observação teste",
            "prazo_entrega_dias": 5,
            "usuario": self.user, # Associa o fornecedor ao usuário de teste
        }
        self.fornecedor = Fornecedor.objects.create(**self.fornecedor_data)
```

Fonte: Os autores, 2024.

O trecho abaixo se refere a Figura 15 apresentada na próxima página.

Primeiramente, o teste `test_get_lista` válida se a lista de fornecedores é retornada corretamente, verificando a resposta com status 200 e a presença do nome do fornecedor.

O teste `test_get_detalhe` foca na visualização de um fornecedor específico, confirmando que a resposta inclui os dados corretos e retorna status 200.

Já o teste `test_post_criar` examina o comportamento do *endpoint* de criação de fornecedores, enviando dados válidos para a API e verificando se o fornecedor é criado com sucesso, o que é indicado pelo status 201 e pela presença de um "id" no JSON de resposta.

O teste `test_put_editar` assegura que a atualização de dados de um fornecedor específico funcione corretamente, alterando o nome do fornecedor e verificando se a atualização é refletida no banco de dados.

Por fim, os testes `test_get_detalhe_not_found` e `test_put_editar_not_found` garantem que o sistema lida apropriadamente com IDs inexistentes, retornando um status 404 quando o fornecedor solicitado não é encontrado, o que reforça a robustez e o tratamento de erros da API.

Figura 15 – Trecho de código apresentando os testes de unidade

```
def test_get_lista(self):
    response = self.client.get(reverse("fornecedor:get_lista"))
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, self.fornecedor.nome_fantasia)

def test_get_detalhe(self):
    response = self.client.get(
        reverse("fornecedor:get_detalhe", args=[self.fornecedor.id])
    )
    self.assertEqual(response.status_code, 200)
    self.assertEqual(
        response.json()["nome_fantasia"], self.fornecedor_data["nome_fantasia"]
    )

def test_post_criar(self):
    self.fornecedor_data["cnpj"] = "59598388000117"
    self.fornecedor_data["usuario"] = self.user.id

    response = self.client.post(
        reverse("fornecedor:post_criar"),
        data=json.dumps(self.fornecedor_data),
        content_type="application/json",
    )

    print(response.content)
    self.assertEqual(response.status_code, 201)
    self.assertIn("id", response.json())

def test_put_editar(self):
    updated_data = {"nome_fantasia": "Fornecedor Atualizado"}
    response = self.client.put(
        reverse("fornecedor:put_editar", args=[self.fornecedor.pk]),
        data=json.dumps(updated_data),
        content_type="application/json",
    )
    self.assertEqual(response.status_code, 200)
    self.fornecedor.refresh_from_db()
    self.assertEqual(self.fornecedor.nome_fantasia, "Fornecedor Atualizado")

def test_get_detalhe_not_found(self):
    response = self.client.get(
        reverse("fornecedor:get_detalhe", args=[999])
    )
    # ID que não existe
    self.assertEqual(response.status_code, 404)

def test_put_editar_not_found(self):
    updated_data = {"nome_fantasia": "Fornecedor Atualizado"}
    response = self.client.put(
        reverse("fornecedor:put_editar", args=[999]),
        data=json.dumps(updated_data),
        content_type="application/json",
    )
    self.assertEqual(response.status_code, 404)
```

Fonte: Os autores, 2024.

Com o uso de `pytest`, a execução dos testes se torna mais ágil e permite uma fácil integração com outras ferramentas de análise de cobertura e qualidade do código.

9.3.5 Aplicativo mobile

O aplicativo *mobile* foi desenvolvido pela ferramenta FlutterFlow utilizando-se da linguagem de programação Flutter. Nele é possível realizar a avaliação dos setores da fazenda e visualizar gráficos das quantidades de avaliação por ano e mês da fazenda, porém apenas o gerente e um técnico do TI tem acesso a esse aplicativo, pois será uma tarefa de gerenciamento disponível apenas para a visão dos gestores, a tela de *login* do sistema está apresentada na figura 16 abaixo.

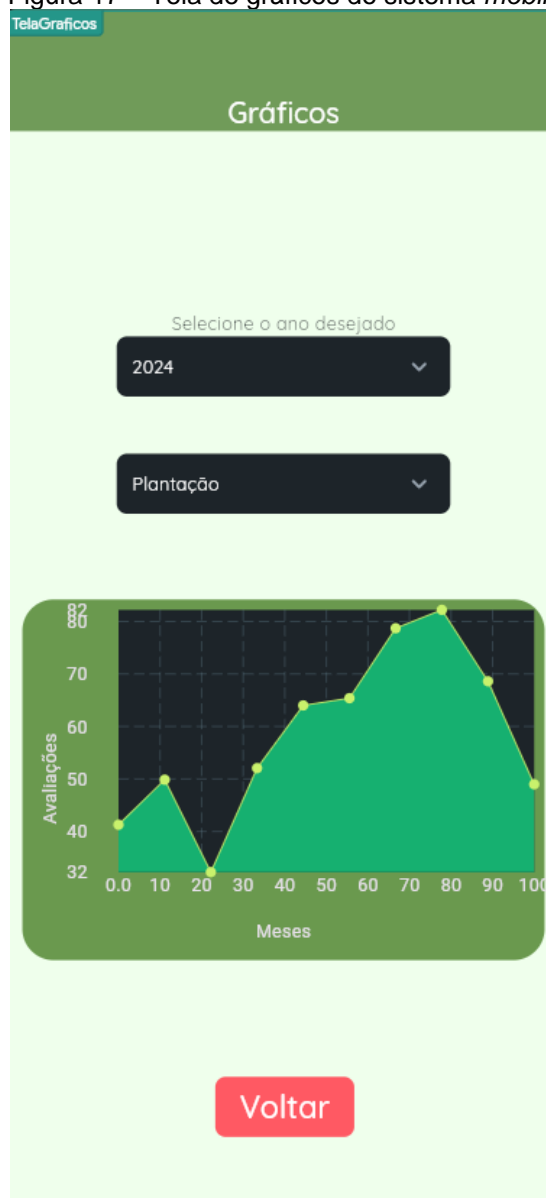
Figura 16 - Tela de login do aplicativo



Fonte: Os autores, 2024.

O sistema em sua versão mobile conta também com a visualização em gráficos para a quantidade de avaliações de cada setor, assim possibilitando uma visão ampla para gerenciamento. Na figura 17 abaixo se encontra a tela de acompanhamento das avaliações.

Figura 17 – Tela de gráficos do sistema *mobile*



Fonte: Os autores, 2024.

O técnico do TI tem acesso a esse sistema para realizar o cadastro dos gestores no sistema pelo banco de dados que está hospedado no Supabase, assim permitindo acesso de qualquer local do mundo, basta acesso à internet e a chave de acesso principal do painel de gerenciamento do banco.

A questão da segurança no aplicativo mobile, o banco de dados utilizado foi o Supabase que possui ferramentas para a segurança dos dados, que uma delas é a presença de servidores no Brasil que faz estar de acordo com um dos requisitos da LGPD que são o armazenamento de dados de brasileiros em solo nacional, outra característica desse banco é o armazenamento de senhas criptografada, como apresentado na tabela 1 abaixo.







Tabela 1 – Exemplo de senha criptografada no banco Supabase

teste@estreladovale.com.br	\$2a\$10\$KV0S1mPSZc74f4OpE.4KtePOIEsRow6phDH2zfu1PAL2oy9QMHSrO
teste@teste.com	\$2a\$10\$.T.uRGLeB52lr.VDLIsMuOX51sTAVmPwlhmIP823UP5WNLwMEskoO

Fonte: Os autores, 2024.

As avaliações que serão enviadas para o banco de dados *mobile* serão armazenadas da forma apresentada na tabela 2 abaixo. Essa tabela será usada para ser representada no gráfico presente no aplicativo, mostrando a quantidade de avaliações de cada mês durante um determinado ano.

Tabela 2 – Tabela de avaliações dentro do Supabase

notasedv	
  #	id int8
	nome text
	setor text
	nota_av numeric
	data_av date

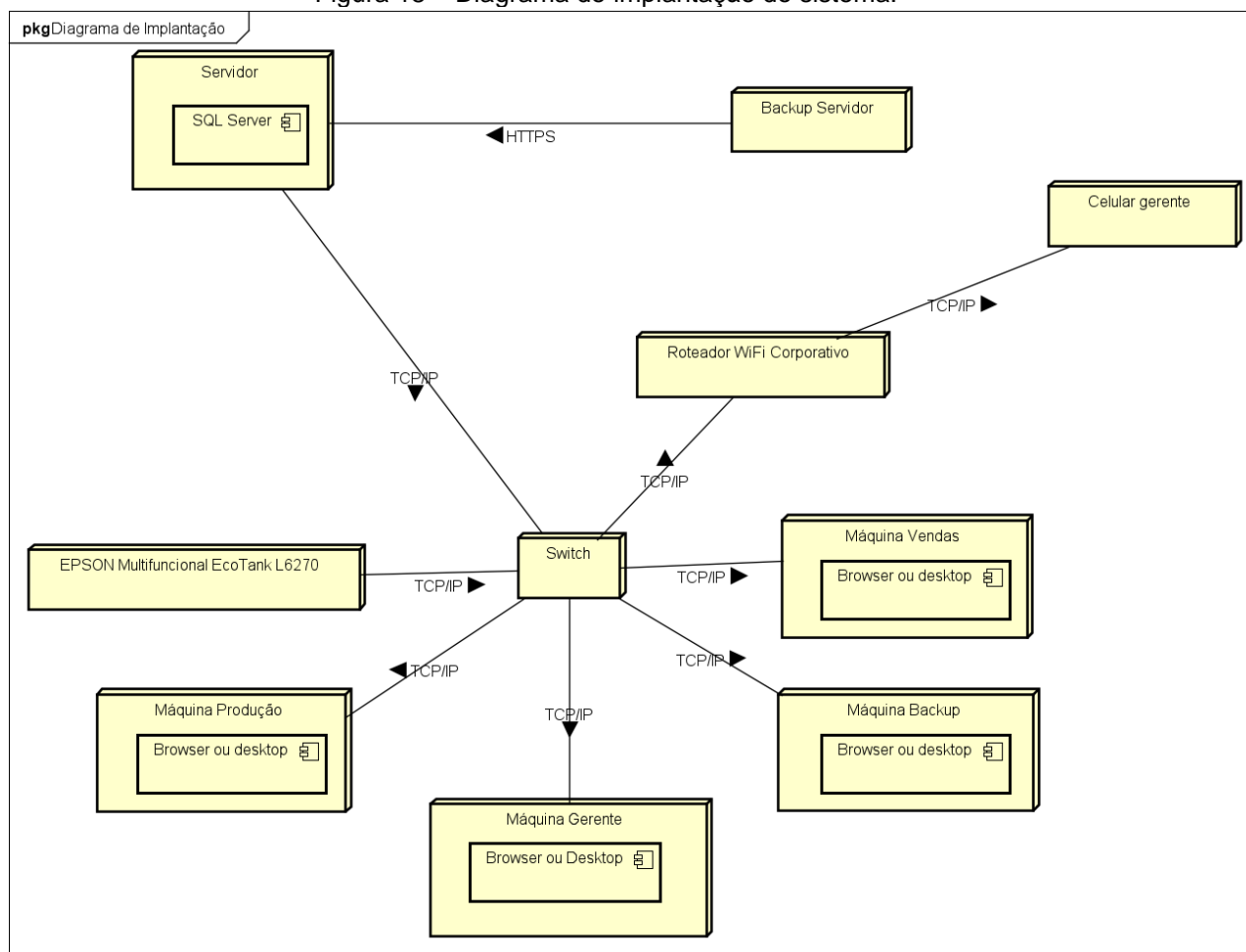
Fonte: Os autores, 2024.

9.4 Projeto de Sistemas Orientados a Objetos

No desenvolvimento do projeto a matéria de Projeto de Sistemas Orientados a Objetos, teve seu papel fundamental em definição de requisitos e modelagem do sistema, sempre levando em conta que os diagramas do sistema são a base para seu desenvolvimento.

Durante o desenvolvimento do projeto um dos primeiros diagramas desenvolvidos e melhorados foi o diagrama de implantação do sistema, apresentado na figura 18 abaixo.

Figura 18 – Diagrama de implantação do sistema.

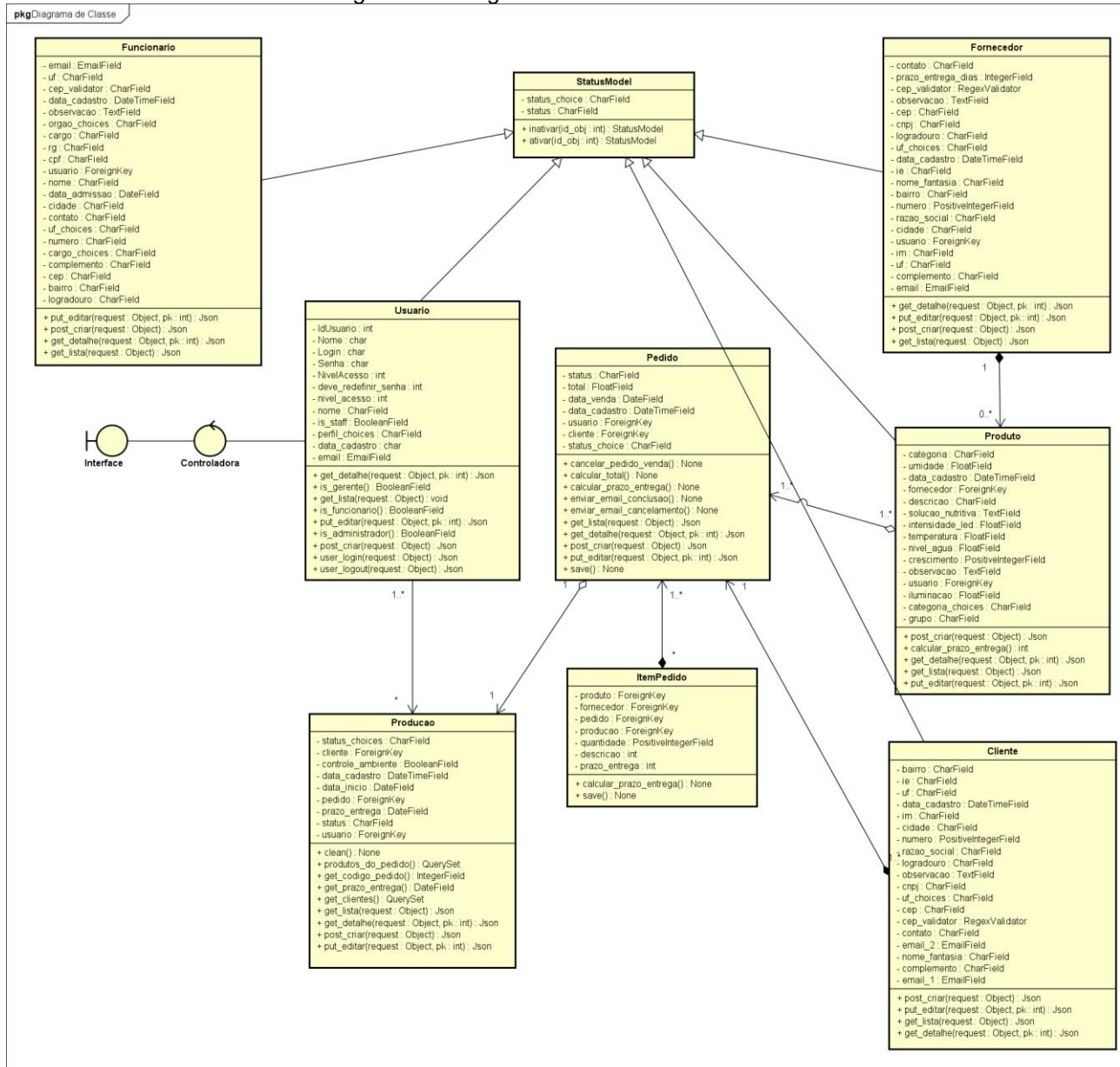


Fonte: Os autores, 2024.

O diagrama de implantação é usado como base da estrutura física do sistema, mostrando tanto quantos dispositivos foram utilizados, quanto como a estrutura de um modo geral da fazenda é implementada e configurada.

Na figura 19 apresentada na próxima folha é representado o diagrama de classe do projeto, devidamente implementado na documentação.

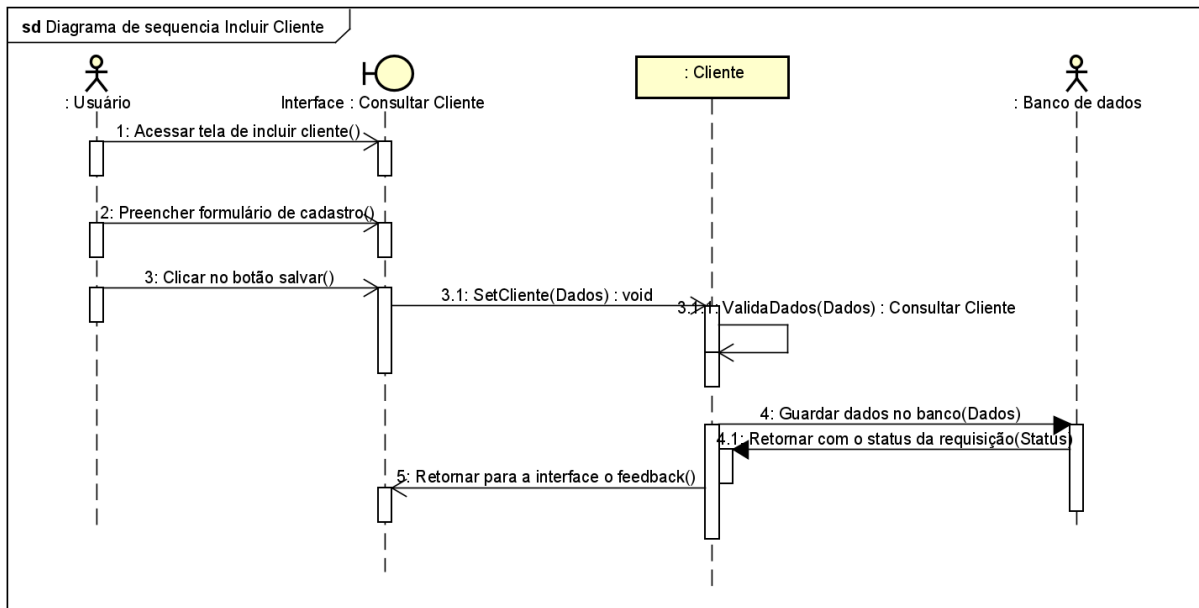
Figura 19 - Diagrama de classes Geral



Fonte: Os autores, 2024.

Durante o desenvolvimento foram também desenvolvidos diagramas de sequência para partes dos aplicativos, como na figura 20, apresentada na próxima página, que representa o diagrama de sequência da inclusão de um cliente no programa *desktop*.

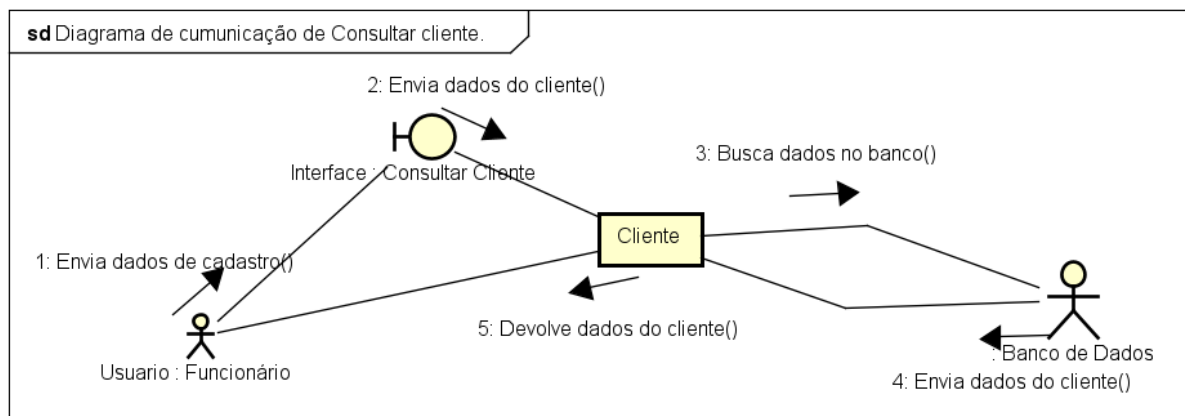
Figura 20 – Diagrama de sequência de inclusão de cliente.



Fonte: Autoria própria

Outro diagrama desenvolvido foi o de comunicação, utilizado para explorar o comportamento dinâmico do sistema, esse diagrama está apresentado na figura 21 abaixo.

Figura 21 – Diagrama de comunicação



Fonte: Autoria própria

9.5 Gerenciamento de Projeto de Software

Na iniciação do projeto, foram definidas as partes interessadas juntamente com a importância de desenvolver um sistema de gerenciamento de fazenda urbana que atende às necessidades das partes interessadas e contribui para a sustentabilidade, eficiência e segurança alimentar urbana.

Também levando em conta a geração de capital que um sistema desse trás, tanto para os desenvolvedores quanto para os compradores dele.

9.5.1 Partes interessadas do sistema

As partes interessadas deste sistema podem ser divididas em:

- **Agricultores urbanos:** De pequeno e médio porte que necessitam de um sistema robusto que atenda às suas necessidades.
- **Consumidores locais:** Que podem ser lojas de produtos veganos e vegetarianos, ou também pessoas físicas, caso sua demanda atenda ao pedido mínimo necessário.
- **Startups e empresas de cultivo:** Empresas que desenvolvem soluções tecnológicas para a agricultura, que podem integrar o sistema a mais sensores e automações proprietárias.
- **Governos:** Por conta do seu nicho de desenvolvimento não amplamente conhecido, pode ser necessário que algum governo adquira o sistema para a implantação em fazendas ou órgãos especializados.

Na parte da importância do desenvolvimento desse *software* e de sua necessidade, pode ser evidenciados vários tópicos como:

- **Eficiência Operacional:** O sistema ajuda a centralizar a gestão das atividades agrícolas, otimizando recursos e reduzindo desperdícios.
- **Sustentabilidade:** Facilidade de práticas sustentáveis, como o uso eficiente de água e energia.
- **Produção Local:** Ajuda a reduzir a dependência de transporte de longa distância, garantindo produtos mais frescos.
- **Inovação Tecnológica:** Realiza a Integração de tecnologias modernas, melhorando monitoramento e controle.

9.5.2 Escopo do projeto

Objetivo: Implementar um sistema de gestão completa nas plataformas (*Web* e *desktop*) e um aplicativo Android para gestão de riscos e qualidade.

Entregas:

Aplicativo completo *desktop* e *web* integrados

Aplicativo *android*

Elaboração de diagramas(caso de uso, classe e implementação)

Limites:

Aplicativo apenas *android*.

Utilizar somente banco SQL

Prazo de entrega: mínimo de 30 máximo de 32 dias.

9.5.3 Cronograma do projeto

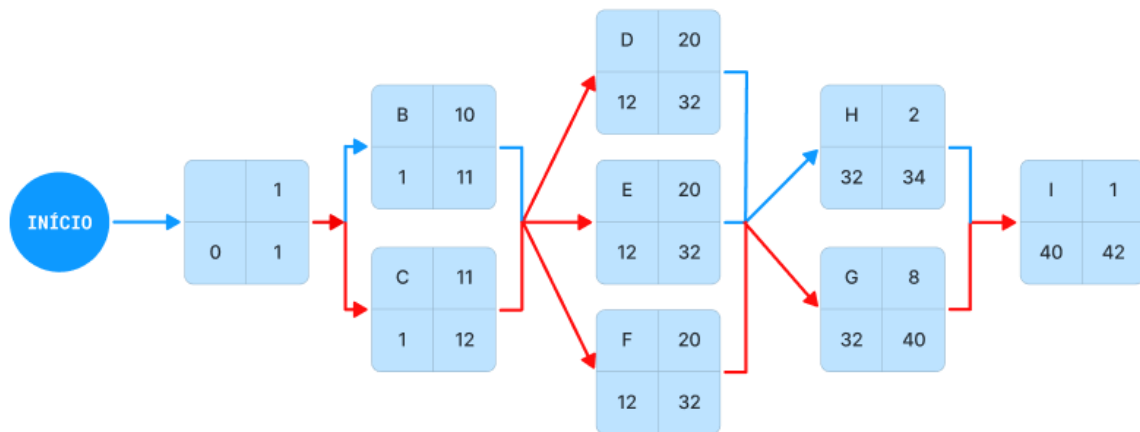
Na figura 22 abaixo se encontra a divisão de atividades do projeto juntamente com o diagrama representando o caminho crítico do projeto, figura 23.

Figura 22 – Atividades do projeto e respectivos tempo.

Aividade	Atividade predecessora	Duração(dias)	Descrição
A	-	1	Organizar requisitos
B	A	10	Prototipar telas no Figma
C	A	11	Criar diagramas astah
D	B,C	20	Desenvolver APP Desktop
E	B,C	20	Desenvolver APP Web
F	B,C	20	Desenvolver APP Mobile
G	D,E,F	8	Configurar Data base
H	D,E,F	2	Testar aplicações
I	G,H	1	Apresentar

Fonte: Autoria própria

Figura 23 – Caminho crítico do projeto



LEGENDA: — CAMINHO CRÍTICO -> A,C,D,E,F,G,I

Fonte: Autoria própria.

9.5.4 Estimativa de custo do projeto

Na tabela 1 abaixo pode ser encontrado o orçamento do projeto que leva em conta os gastos com materiais e com horas das pessoas envolvidas, sendo ele o custo final envolvido.

Tabela 3 – Orçamento do Projeto

Orçamento do projeto					
Orçamento do mês de		Porcentagem do orçamento já gasto: 99%			
Setembro a novembro					
Orçamento alimentar total para o mês		Valor já gasto			
R\$ 18.170		R\$ 17.920			
	Alimentação	Transporte	Upgrades/melhorias	Planos/serviços	Custo pessoas
Orçamento Planejado	R\$ 100	R\$ 4.200	R\$ 500	R\$ 500	R\$ 12.870
Despesas reais	R\$ 100	R\$ 4.200	R\$ 250	R\$ 500	R\$ 12.870

Fonte: Autoria própria

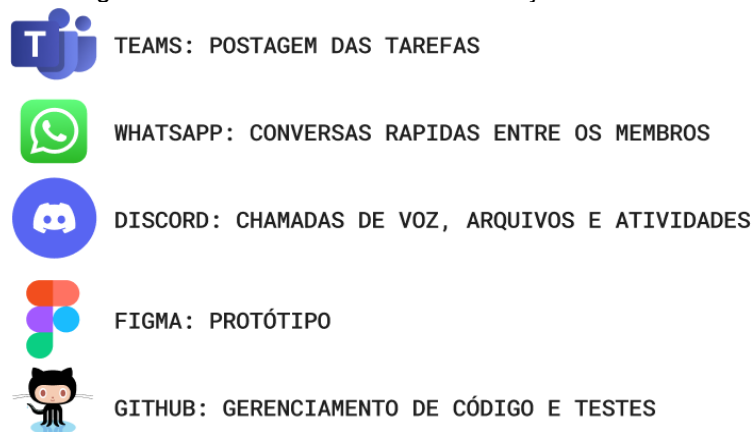
9.5.5 Desenvolvimento de recursos humanos

A equipe trabalhou com o *scrum* sendo liderada por um *scrum* mestre que foi escolhido através de uma votação entre os membros, as outras atividades como: programação e elaboração de diagramas foi dada abertamente para os membros poderem escolher quais eles tinham mais afinidade com base em experiências passadas e estudos recentes, assim aproveitando da eficiência do direcionamento de pessoas para áreas que elas possuem mais afinidade e consequentemente adiantando e otimizando o desenvolvimento do projeto como um todo.

9.5.6 Sistemas de comunicação utilizados

Durante o projeto foram utilizados vários métodos para gerenciamento de atividades, reuniões e desenvolvimento, eles estão listados na Figura 24 abaixo.

Figura 24 – Ferramentas de comunicação utilizadas



Fonte: Autoria própria

Durante o desenvolvimento foi realizado a consulta com os professores sobre os requisitos e andamento do desenvolvimento em uma frequência semanal, para assim garantir o atendimento de todos os requisitos necessários e deixá-los cientes de como o andamento do projeto se encontra.

9.5.7 Gerenciamento de risco do projeto

Para preparação do gerenciamento de riscos do projeto foram realizados certos preparos e planos, como *backups* dos arquivos utilizados, como documentos planilhas e códigos, tanto em nuvem quanto local.

Em preparação para a apresentação, todo o time é capaz de explicar tecnicamente o *software* e os diagramas desenvolvidos, sem ter uma pessoa única exclusiva, pensando em medidas de segurança sobre o *hardware* a equipe carrega um segundo *notebook* caso algo aconteça com o principal e todo sistema tem um *backup* físico e em nuvem.

9.5.8 Encerramento do projeto

Durante o projeto, foi essencial organizar e arquivar toda a documentação, incluindo: Relatórios finais, detalhando todas as fases do projeto, decisões tomadas e resultados alcançados e os registros de comunicação, mantendo um histórico das comunicações internas, assegurando a transparência e a clareza de todas as interações.

Realizamos também uma avaliação de desempenho abrangente, a identificação de sucessos, destacamos as áreas onde o projeto superou as expectativas, áreas de melhoria, identificamos pontos que poderiam ser aprimorados em futuros projetos.

Transferência de Conhecimento: Para garantir a continuidade e eficiência no uso do sistema, asseguramos que: O treinamento da equipe de operações recebesse o treinamento completo sobre o sistema, capacitando-os a gerenciar e manter o sistema com autonomia.

Esses passos garantem que o projeto não apenas atenda às necessidades imediatas, mas também que a equipe esteja preparada para mantê-lo e evoluí-lo no futuro.

9.6 Empreendedorismo

Desde o início do desenvolvimento do projeto, era bem claro o seu principal objetivo, do desenvolvimento de um *software* capaz de realizar o gerenciamento de uma fazenda urbana chamada estrela do vale, tanto em uma plataforma *web*, *desktop* e *mobile*.

9.6.1 Missão Principal e objetivos

A missão principal do projeto foi gerar um software que é capaz de atender os requisitos, que foram levantados com base na análise do mercado das fazendas urbanas, sendo eles gerenciamento de fornecedores, funcionários, produção e etc.

Junto com o objetivo principal do projeto foram averiguados também os objetivos de curto, médio e longo prazo, sendo eles uma visão do sistema no presente e futuro, como especificados abaixo.

- Curto prazo: Desenvolvimento de uma versão estável do sistema capaz de atender todos os requisitos pré-estabelecidos em todas as versões.
- Médio prazo: Gestão bem próxima do sistema para correção de erros no cenário de implementação dele dentro de uma fazenda urbana.
- Longo Prazo: Realizar a transformação do sistema para uma plataforma completa de gerenciamento com a implementação de mais funcionalidades e suporte aprimorado.

9.6.2 Público alvo e sistemas concorrentes

O público-alvo do sistema é principalmente áreas urbanas com alta densidade populacional. Pequenos e médios agricultores urbanos, *startups* de agricultura, e cooperativas agrícolas com a necessidade de soluções tecnológicas para maximizar a produção em espaços limitados, reduzir desperdícios e melhorar a qualidade dos produtos, além de necessitarem de cada vez mais inovação em um cenário competitivo e tecnológico que é o mundo das fazendas urbanas.

Os principais concorrentes do sistema desenvolvido são, plataformas de gestão agrícola tradicionais, aplicativos de monitoramento de cultivos específicos e soluções de IoT para agricultura.

Esses sistemas podem ser mais baratos para aquisição, porém acabam tendo funcionalidades mais básicas de serem utilizadas, porém não oferecem as funcionalidades necessárias a um sistema mais complexo de gerenciamento, e podem acabar atrasando ou limitando a operação da fazenda.

Já as estratégias para vencê-los no mercado podem ser simplificadas em:

- Diferenciação pelo foco em agricultura urbana e integração multiplataforma (*web, mobile e desktop*)
- Oferta de funcionalidades específicas para desafios urbanos como gestão de espaços pequenos e otimização de recursos hídricos.
- Implementação de uma interface otimizada e mais amigável e de fácil uso para atrair usuários com diferentes níveis de conhecimento tecnológico.

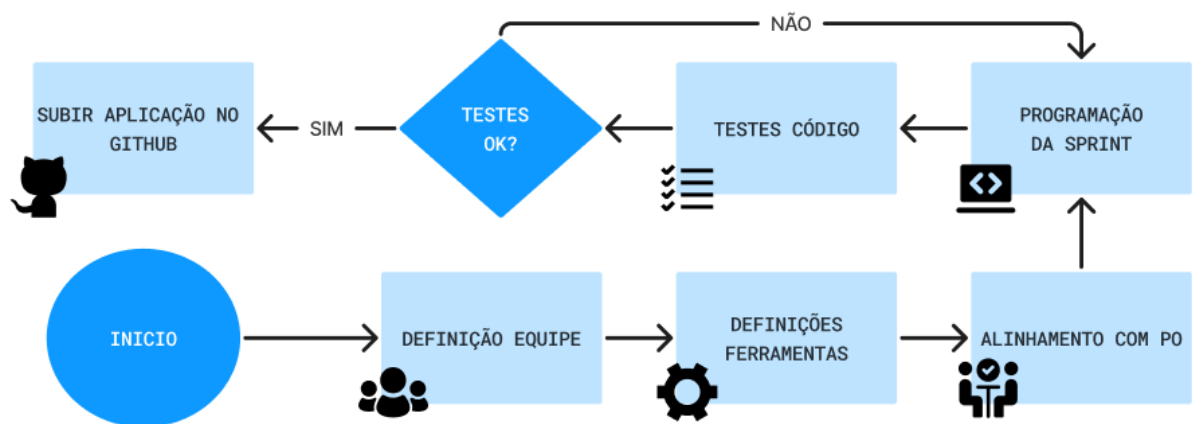
Nas estratégias de comercialização para o sistema, várias opções estão disponíveis, como:

- Licenças: Modelos de licenciamento flexíveis, incluindo opções de entrada para pequenos agricultores e versões com funcionalidades avançadas para fazendas maiores.
- Venda: Pode ser tanto direta ou via parcerias com distribuidores de tecnologia agrícola.
- Suporte: Suporte técnico disponível online e por telefone, com a opção de contato via WhatsApp para abertura de chamados.

9.7 Gestão de Qualidade

O grupo identificou a necessidade de mitigar problemas de diferentes gravidades, como erros de código e, principalmente, o não cumprimento dos requisitos. Para isso, o líder da equipe definiu um fluxo de desenvolvimento que visa reduzir esses problemas, assegurando a entrega do produto conforme especificado, esse método está especificado na figura 25 a seguir.

Figura 25 – Fluxo de desenvolvimento



Fonte: Autoria própria.

Essa abordagem não apenas melhora a qualidade do *software*, mas também promove uma comunicação mais eficaz entre os membros da equipe, garantindo um resultado que atende às expectativas exigidas no escopo do projeto.

Além de também promover um ambiente de desenvolvimento mais organizado e com menos falhas e erros pois realizando esses controles e verificações o sistema desenvolvido acaba sendo mais completo, organizado e de melhor qualidade geral.

10. CONCLUSÃO

Desde o levantamento de requisitos, foi identificado a necessidade de aplicativos simples, porém com funcionalidades específicas que são necessárias para gestão eficiente da fazenda, sempre observando e analisando quais seriam as ferramentas mais eficientes para seu desenvolvimento.

A equipe desenvolveu um conjunto de aplicativos que automatiza a logística e a produção da fazenda dentro do *desktop*, *web* e *mobile*, tornando-se ferramentas essenciais no dia a dia dos funcionários. Durante o desenvolvimento foi identificado a necessidade de tornar o aplicativo móvel mais dinâmico, permitindo que a liderança avalie em tempo real a performance dos funcionários em relação à segurança e qualidade.

Ao final do projeto, constatamos que os aplicativos, além de otimizar a gestão, servem como um registro detalhado das atividades, facilitando futuras auditorias, e ainda possibilitando a integração futura com muitas ferramentas de gestão. Com todos os processos monitorados, a próxima etapa consiste em automatizar a geração de relatórios personalizados para cada tipo de auditoria.

11. REFERÊNCIAS

ANDRADE E SILVA, Diandra. Como funciona a arquitetura MTV (Django). **Medium**, 2020. Disponível em: <https://diandrasilva.medium.com/como-funciona-a-arquitetura-mtv-django-86af916f1f63>. Acesso em: 10, novembro e 2024.

BACURAU, Rodrigo M.; LEAL, Brauliro G.; RAMOS, Ricardo A. Uma Abordagem para a Construção de Diagramas da UML Concomitante à Prototipação de Interface [em linha]. set. 2022.

CARVALHO, Thiago Leite. Orientação a Objetos. Edição atualizada. São Paulo - SP, Casa do código, 2024.

CRISTÓVÃO, Ana Martins. Gestão da Qualidade. São Paulo - SP, Editora Sol, 2013.

DJANGO. Django *Documentation*. Disponível em: <https://docs.djangoproject.com/en/5.1/faq/general/#djangoappears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>. Acesso em: 05 nov. 2024.

FARINELLI, Fernanda. Conceitos básicos de programação orientada a objetos. Instituto Federal Sudeste de Minas Gerais, 2007.

LAGO, Decio; MINGOSS, Rubens Aparecido. Gerenciamento de projetos segundo as normas preconizadas pelo PMI®-Um. **Revista de Ciências Exatas e Tecnologia**, v. 2, n. 2, p. 38-52, 2007.

MENEZES, Nilo Ney Coutinho. Introdução a programação com *Python*. 4ª Edição. São Paulo - SP, Novatec, 2024.

RUIZ, Fernando Martinson. **Empreendedorismo**. Senac, 2019

SANTANDER, V. F. A.; CASTRO, J. F. B. Desenvolvendo Use Cases a Partir da Modelagem Organizacional. III *Workshop* de Engenharia de Requisitos, Rio de Janeiro - RJ, Brasil. 2000.

WALTZ, Flavio. *Slideshow* aula 2 - desenvolvimento de software para internet. São José dos Campos - SP, 2024.

FICHA DE CONTROLE DO PIM

Grupo Nº _____ Ano: 2 Período: 4 Orientador: _____

Tema: Desenvolvimento de um sistema integrado para o controle de operações em uma fazenda urbana.

Alunos:

RA	Nome	E-mail	Curso	Visto do aluno
G7308C1	João Victor Ramos do Nascimento	joao.nascimento2@aluno.unip.br	CST em ADS	João Victor R
R025063	Nathalia Jacque Mendes Lima	nathalia.lima56@aluno.unip.br	CST em ADS	Nathalia J
N921714	Renan Pereira Diniz	renan.diniz1@aluno.unip.br	CST em ADS	Renan P
N066CF4	Ruan dos Santos Oliveira	ruan.oliveira17@aluno.unip.br	CST em ADS	Ruan S
G775AG7	Vitor Antony de Marchi Castro	vitor.castro10@aluno.unip.br	CST em ADS	Vitor A

Registros:

Data do encontro	Observações
26/08/2024	Inicialização de atividades e escolha de Lider.
31/08/2024	Separação de papéis e tarefas para desenvolvimento.
01/09/2024	Organização de documentação e requisitos entre membros do grupo.
05/09/2024	Verificação de ferramentas para Desenvolvimento Mobile.
08/09/2024	Inicialização das pesquisas para o desenvolvimento e ferramentas.
15/09/2024	Equipe do Web e Desktop inicializaram o desenvolvimento.
22/09/2024	Plataforma Mobile foi selecionada e iniciado o desenvolvimento.
29/09/2024	Alinhamento de grupo para discussão sobre o desenvolvimento.
06/10/2024	Ferramenta de desenvolvimento Mobile definida.
20/10/2024	Documentação em 20% de conclusão junto com APP mobile em 40%
27/10/2024	Ferramenta mobile alterada para o FlutterFlow por conta de problemas com Android Studio.
28/10/2024	Alinhado com o professor Flávio sobre a utilização do FlutterFlow.
03/11/2024	Documentação em 40% e formatação sendo realizada juntamente com ela.
10/11/2024	Documentação 75% pronta. Mobile em 60% e desktop/web em 55% e <i>back-end</i> em 70%.
13/11/2024	Mobile em conformidade com requisitos juntamente com web/desktop. Alinhado com o professor Flávio.
14/11/2024	Gerenciamento do projeto em conformidade com requisitos. Alinhado com o Professor Alexandre.
15/11/2024	Revisão da documentação por conta do adiamento da data de entrega para dia 20/11/2024.
17/11/2024	Revisão da documentação e correção de erros.
18/11/2024	Revisão final da documentação para envio no site da Unip.