DriverPlan by Rigel

DOCUMENTO DE ARQUITETURA

Versão 1.0

Histórico de Revisão

Data	Versão	Descrição	Autor(es)
20/05/2024	1.0	Primeiro versão do documento de arquitetura	Rigel

Autores:

Matrícula	Nome	Descrição do papel assumido na equipe	% de contribuição ao trabalho
211062179	Marcelo de Araújo Lopes	Membro	12,5 %
211062473	Samuel Afonso da Silva Santos	Membro	12,5 %
221031120	Arthur Fonseca Vale	Membro	12,5 %
221008679	Pablo Serra Carvalho	Membro	12,5 %
221008786	Mateus Villela Consorte	Membro	12,5 %
221037803	Letícia Kellen Ramos Paiva	Membro	12,5 %
202045820	Karolina Vieira Barbosa	Membro	12,5 %
202017067	Raul Falluh Fragoso de Mendonça	Membro	12,5 %

Sumário

1 Introdução

1.1 Propósito

Este documento descreve a arquitetura do sistema "DriverPlan", desenvolvido no âmbito da disciplina de MDS - Métodos de Desenvolvimento de Software - no primeiro semestre de 2024, com o objetivo de proporcionar uma compreensão abrangente do sistema para desenvolvedores, testadores e demais interessados.

1.2 Escopo

1. Gestão de Cronogramas
    - Descrição: O sistema deve permitir a gestão de cronogramas diários, semanais, mensais e anuais (opcional), com capacidade para alterações sem inconsistências.
    - Objetivo: Oferecer uma interface de usuário clara e funcional para a criação, visualização e modificação de cronogramas de viagem.
2. Integração com Google Maps
    - Descrição: O sistema deve integrar com Google Maps para calcular pontos de parada, distâncias e tempo de chegada.
    - Objetivo: Fornecer informações precisas e em tempo real para os usuários e motoristas, melhorando a eficiência das viagens.
3. Mensagens Automatizadas
    - Descrição: O sistema deve enviar mensagens automatizadas para os motoristas.
    - Objetivo: Manter os motoristas informados sobre suas rotas e horários de forma automática e eficiente.
4. Cálculo de Taxas
    - Descrição: O sistema deve aplicar critérios específicos para o cálculo de taxas com base em horário, dia, frequência, deslocamento, gasolina e taxa de aguardo.
    - Objetivo: Implementar um sistema de cálculo de taxas flexível e preciso que atenda às regras de negócio definidas.
5. Espaço para Visualização de Dados
    - Descrição: O sistema deve permitir o armazenamento e visualização de Dados do cliente.
    - Objetivo: Facilitar a gestão de dados importantes relacionados às viagens e contratos dos clientes.

2 Representação Arquitetural

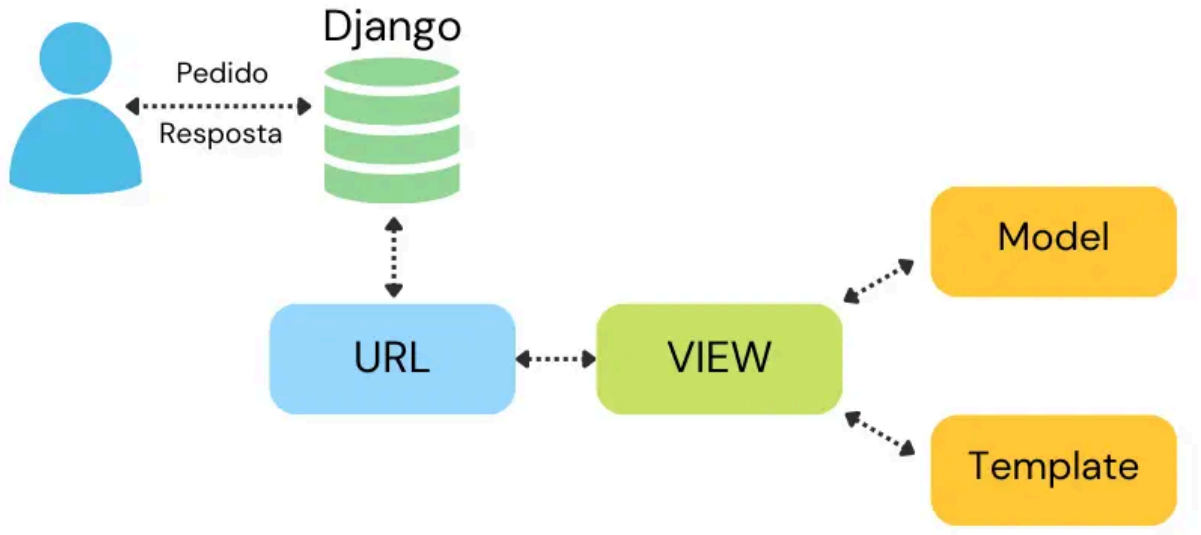
2.1 Definições

O sistema seguirá uma arquitetura Model-View-Template (MVT), um estilo arquitetural que facilita a separação das responsabilidades dentro da aplicação web, garantindo uma organização clara e eficiente do código.

2.2 Justificativa:

A escolha da arquitetura MVT com Python e Django é justificada por sua capacidade de atender de forma eficiente aos requisitos do produto e do projeto, conforme descrito nos documentos de Visão e Declaração de Escopo. Esta arquitetura promove uma separação clara de responsabilidades, facilita a escalabilidade e manutenção, e proporciona uma base sólida e segura para o desenvolvimento da aplicação, garantindo que o produto final seja robusto, seguro e flexível.

2.3 Detalhamento



O Modelo de Arquitetura MVT é uma variação do Modelo de Arquitetura MVC (Model-View-Controller) usado em muitos frameworks web, como Django (para Python).

View (Visualização): É responsável por apresentar os dados aos usuários. Ele exibe a interface do usuário e interage com o modelo para obter dados:

- Responsável por apresentar os cronogramas aos usuários e receber suas interações, como criação e edição.
- Exibe as informações do Google Maps na interface do usuário e permite interações, como inserção de pontos de parada.
- Envia mensagens automatizadas com base em eventos, como atualizações de rotas e horários.
- Aplica as regras de cálculo de taxas e exibe o valor calculado ao usuário.
- Apresenta os dados do cliente aos usuários e permite interações, como adição ou edição de dados

Model (Modelo): Representa os dados e a lógica de negócios. Ele interage com o banco de dados para buscar, armazenar e manipular os dados.

- Definirá a estrutura dos cronogramas e as operações de busca, criação, atualização e exclusão no banco de dados.
- Pode lidar com a lógica de integração com a API do Google Maps e armazenar informações relevantes no banco de dados.
- Pode armazenar modelos de mensagens e informações de contato dos motoristas.
- Define as regras e critérios para o cálculo de taxas e armazena informações relacionadas às viagens e taxas no banco de dados.
- Define a estrutura dos dados do cliente e suas relações, armazenando essas informações no banco de dados.

Template (Modelo de Apresentação): Define a aparência da interface do usuário. Ele combina dados do modelo com a visualização para produzir a saída final que é enviada ao cliente.

- Define a aparência dos cronogramas na interface do usuário.
- Pode incluir mapas interativos na interface do usuário para visualização das rotas.
- Não é diretamente aplicável, a menos que mensagens automatizadas sejam visualizadas na interface do usuário.
- Pode incluir formulários para entrada de informações relacionadas às taxas na interface do usuário.
- Define a aparência dos dados do cliente na interface do usuário.

2.4 Metas e restrições arquiteturais

Metas Arquiteturais:

1. Escalabilidade

- Descrição: O sistema deve ser capaz de crescer e acomodar um número crescente de usuários e dados sem comprometer o desempenho.
- Objetivo: Garantir que a arquitetura suporte a adição de novos recursos e a expansão de capacidade conforme a demanda aumenta.

2. Manutenibilidade
    - Descrição: O sistema deve ser fácil de manter e atualizar.
    - Objetivo: Facilitar a implementação de novas funcionalidades, correção de bugs e atualizações de segurança através de uma separação clara de responsabilidades (Model, View, Template).
3. Segurança
    - Descrição: Proteger os dados do usuário e de seus clientes.
    - Objetivo: Implementar práticas de segurança.
4. Desempenho
    - Descrição: O sistema deve responder às requisições dos usuários de maneira eficiente e com tempo de resposta aceitável.
    - Objetivo: Otimizar consultas ao banco de dados e minimizar o tempo de processamento nas views.

5. Reusabilidade
    - Descrição: Os componentes do sistema devem ser reutilizáveis em diferentes partes do projeto.
    - Objetivo: Promover a reutilização de código para reduzir redundâncias e acelerar o desenvolvimento.
6. Flexibilidade
    - Descrição: A arquitetura deve permitir ajustes e personalizações conforme as necessidades dos usuários evoluem.
    - Objetivo: Facilitar a adaptação a novos requisitos sem a necessidade de grandes refatorações.

Restrições Arquiteturais

1. Tecnologia
    - Descrição: A utilização de Python como linguagem de programação e Django como framework web é mandatória.
    - Objetivo: Alavancar a simplicidade, robustez e comunidade de suporte de Python e Django.
2. Banco de Dados
    - Descrição: O sistema deve utilizar o banco de dados MySQL.
    - Objetivo: Garantir a compatibilidade e a facilidade de integração com o MySQL, que é escalável e amplamente suportado.
3. Servidor Web
    - Descrição: O sistema deve ser implementado utilizando o servidor Apache.
    - Objetivo: Assegurar uma configuração robusta, segura e bem suportada para hospedar a aplicação Django.
4. Hospedagem

- Descrição: O sistema deve ser implementado em uma infraestrutura que suporte Django, utilizando Apache como servidor web.
- Objetivo: Garantir que a solução possa ser facilmente implantada em ambientes de produção com Apache e MySQL.

5. Compatibilidade
    - Descrição: O sistema deve ser compatível com os navegadores web modernos e dispositivos móveis.
    - Objetivo: Oferecer uma experiência de usuário consistente e acessível em diversas plataformas e dispositivos.
6. Regulamentações e Conformidade
    - Descrição: O sistema deve estar em conformidade com as regulamentações aplicáveis, como GDPR para proteção de dados pessoais.
    - Objetivo: Garantir que o sistema atenda às normas legais e de conformidade relevantes.
7. Usabilidade
    - Descrição: A interface do usuário deve ser intuitiva e fácil de usar.
    - Objetivo: Proporcionar uma experiência de usuário positiva e minimizar a necessidade de treinamento ou suporte.

8. Desenvolvimento Ágil
    - Descrição: O sistema deve ser desenvolvido seguindo práticas ágeis.
    - Objetivo: Permitir entregas incrementais e contínuas, adaptando-se rapidamente às mudanças nos requisitos.

2.5 Visão de Casos de uso (escopo do produto)

O DriverPlan é um sistema de gestão e agendamento de viagens projetado para motoristas particulares. Seu escopo inclui funcionalidades essenciais como agendamento de viagens, integração com o Google Maps para cálculo de rotas, criação e alteração de cronogramas, cadastro de motoristas e clientes, além de recursos de segurança e usabilidade. A escolha da arquitetura MVT (Model-View-Template), foi com base dos requisitos específicos, como a necessidade de uma interface intuitiva e fácil de usar (Teste de usabilidade), a segurança dos dados dos usuários, a integração com o Google Maps para cálculo de rotas (Integração com o Google Maps), e a criação e alteração de cronogramas e também A experiência prévia da equipe no desenvolvimento em python e web e a clara divisão de responsabilidades proporcionada por essa arquitetura também foram fatores-chave que influenciam na escolha da arquitetura MVT

2.6 Visão lógica

O sistema é subdividido nos seguintes módulos:

1. Módulo de Autenticação e Autorização
    - Função: Gerenciar o login e logout de usuários, verificar permissões de acesso.
    - Razão Lógica: Garantir que somente usuários autenticados possam acessar o sistema e realizar operações conforme suas permissões.
2. Módulo de Gestão de Usuários
    - Função: Gerenciar informações dos usuários, incluindo motoristas e clientes.
    - Razão Lógica: Manter um registro atualizado e acessível de todos os usuários que interagem com o sistema.
3. Módulo de Calendário e Cronogramas
    - Função: Gerenciar os cronogramas de viagens, permitindo criação, visualização e modificação.
    - Razão Lógica: Facilitar a organização e planejamento das viagens dos clientes.
4. Módulo de Mensagens Automatizadas
    - Função: Enviar notificações automáticas para motoristas sobre suas viagens e horários.
    - Razão Lógica: Manter os motoristas informados e pontuais, melhorando a eficiência do serviço.

5. Módulo de Integração com Google Maps
    - Função: Calcular rotas, pontos de parada, distâncias e tempos de chegada.
    - Razão Lógica: Fornecer informações precisas e em tempo real para otimizar as viagens.
6. Módulo de Cálculo de Taxas
    - Função: Calcular taxas baseadas em horário, dia, frequência, deslocamento, gasolina, e taxa de aguardo.
    - Razão Lógica: Aplicar corretamente as regras de negócio para o cálculo de taxas, garantindo precisão e transparência.
7. Módulo de Relatórios e Documentação
    - Função: Gerar relatórios e armazenar documentos de contrato.
    - Razão Lógica: Facilitar a geração de relatórios administrativos e manter um repositório organizado de documentos.
8. Módulo de Persistência de Dados
    - Função: Gerenciar a persistência dos dados no banco de dados MySQL.
    - Razão Lógica: Assegurar que todos os dados do sistema sejam armazenados de forma segura e eficiente.
9. Comunicação entre Módulos - Interfaces
    - API Interna: Os módulos se comunicam entre si através de uma API interna que facilita a troca de informações e chamadas de funções.

- Front-end / Back-end: O front-end faz chamadas aos endpoints do back-end para realizar operações como login, gerenciamento de cronogramas, envio de mensagens e cálculos de rotas.
- Serviços Externos: A integração com Google Maps é feita através de APIs externas fornecidas pelo Google.

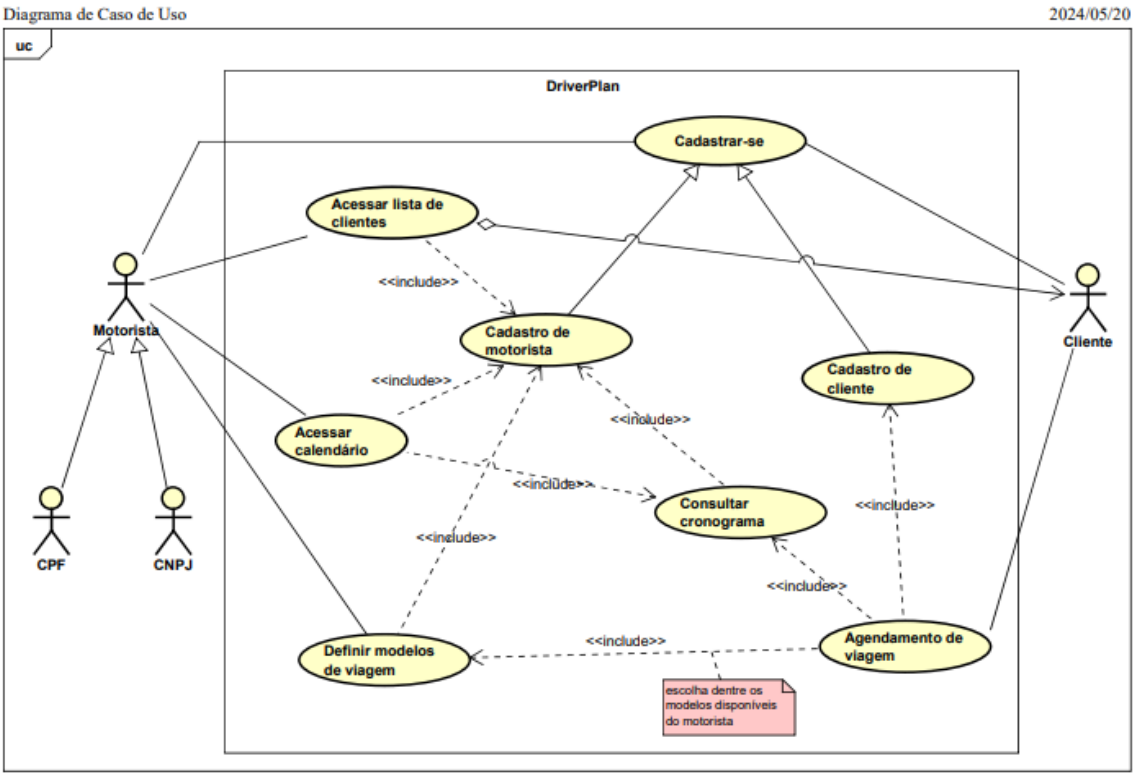
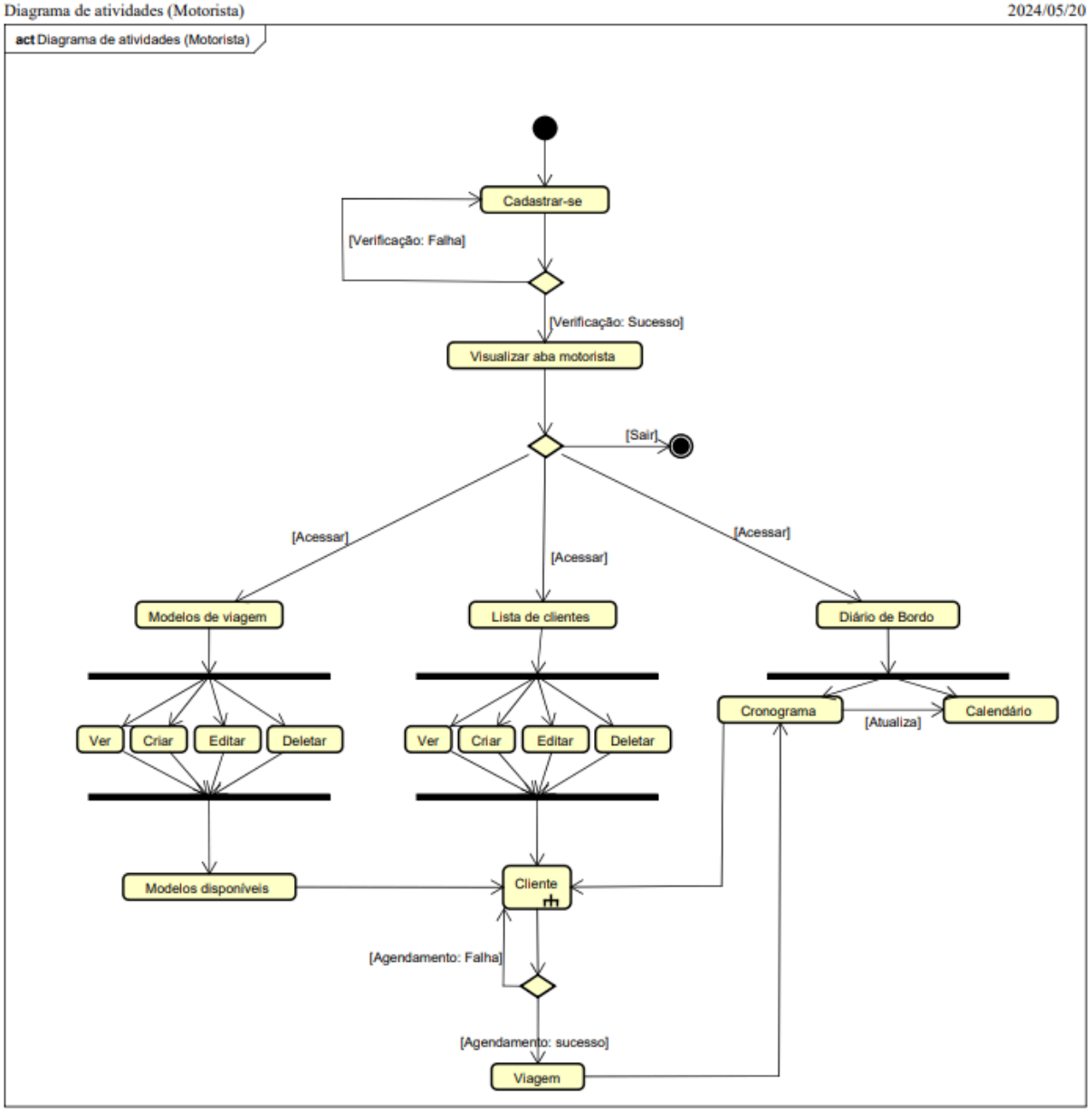


Diagrama de atividades (Motorista) 2024/05/20



Cliente

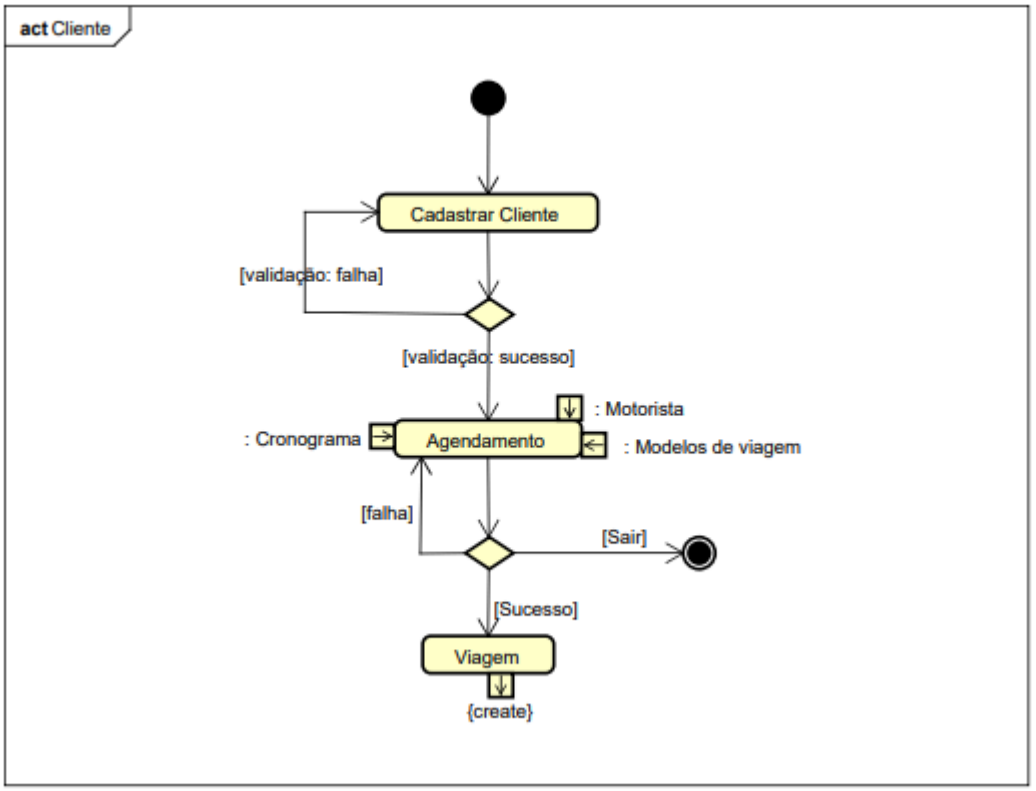
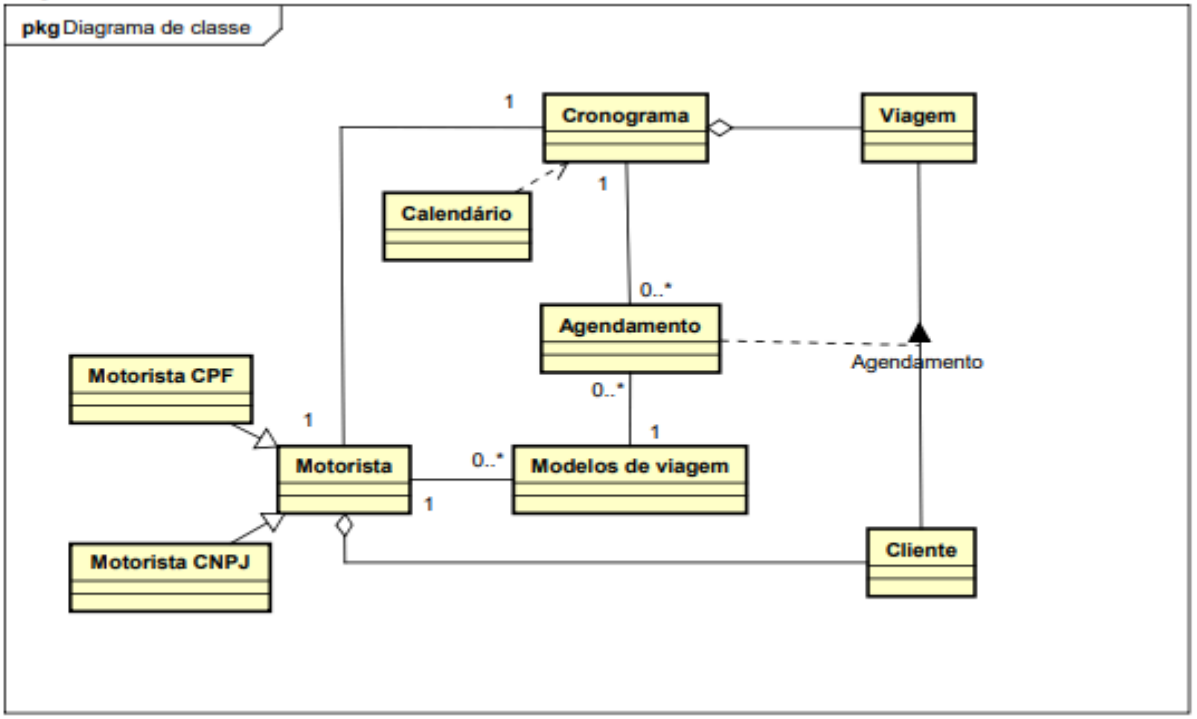
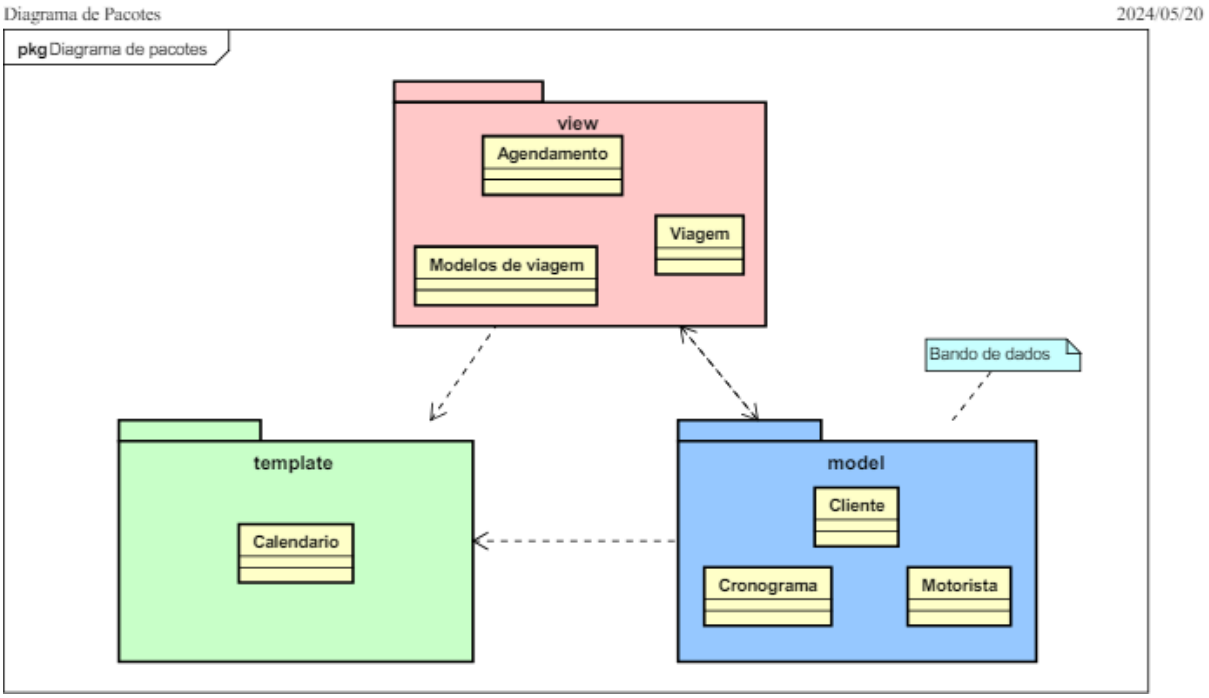


Diagrama de classe



2.7 Visão de Implementação



A camada “Model” contém as definições dos dados e regras de negócios da aplicação. Os modelos são usados para definir a estrutura do banco de dados, bem como para definir as operações que podem ser realizadas sobre esses dados:

- Cliente: Este modelo representa os clientes do sistema, armazenando informações pessoais e detalhes de contato.
- Cronograma: Representa o cronograma das viagens, incluindo horários de partida e chegada, além de pontos de origem e destino.
- Motorista: Este modelo representa os motoristas, armazenando suas informações pessoais, detalhes de licença, e outras informações relevantes.

Além de conter a lógica de negócios, a camada de “Model” é responsável por interagir diretamente com o banco de dados. O Django ORM (Object-Relational Mapping) traduz as operações realizadas nos modelos em comandos SQL que são executados no banco de dados:

- Cliente: Quando um cliente é criado, atualizado ou deletado, essas operações são refletidas no banco de dados.
- Cronograma: A criação, modificação e exclusão de cronogramas são gerenciadas pelo ORM, mantendo a consistência dos dados.
- Motorista: As operações nos dados dos motoristas são convertidas em operações SQL pelo ORM.

2.8 Visão de Implantação

Infraestrutura de hardware

Para hospedar o software DriverPlan, estaremos utilizando um servidor web Apache. O servidor de aplicação deve possuir um processador multi-core com alta frequência de clock para lidar com a carga de processamento da aplicação. Além disso, é importante ter uma quantidade adequada de memória RAM, sendo recomendado pelo menos 8 GB, dependendo do tamanho da carga de trabalho. O armazenamento deve ser em disco SSD para garantir tempos de acesso mais rápidos e melhor desempenho da aplicação. Quanto ao sistema operacional, podemos utilizar uma distribuição Linux, como Ubuntu Server, Debian ou Windows , devido à estabilidade e segurança que oferecem.

Para o banco de dados, será usado um sistema de gerenciamento de banco de dados relacional (RDBMS), como MySQL, para armazenar dados de forma estruturada e segura. O servidor de banco de dados deve ter um processador multi-core para lidar com consultas complexas e cargas de trabalho intensivas. É necessário alocar uma quantidade significativa de memória RAM para cache de consultas e otimização de desempenho, sendo recomendado pelo menos 4 GB, dependendo do tamanho da carga de trabalho. O armazenamento deve ser em disco SSD para garantir tempos de resposta rápidos e alta disponibilidade dos dados.

Para garantir alta disponibilidade e escalabilidade, é recomendável utilizar técnicas de balanceamento de carga, como um balanceador de carga, para distribuir o tráfego entre vários servidores de aplicação. Além disso, é importante implantar redundância em todos os componentes críticos, incluindo servidores de aplicação, bancos de dados e armazenamento, para evitar pontos únicos de falha e garantir a continuidade do serviço em caso de falha de hardware.

Tecnologias

Será usada a linguagem de programação Python e o Framework Django. A linguagem Python foi escolhida por sua simplicidade, legibilidade e pelo fato de ser a linguagem que a maioria do grupo sabia, o que acelera o desenvolvimento e a manutenção do código. Além disso, possui uma vasta biblioteca padrão e uma comunidade ativa, oferecendo suporte e recursos para uma ampla gama de funcionalidades. Já o Django foi escolhido por ser um framework web de alto nível que promove o desenvolvimento rápido e limpo. Segue o princípio DRY, ajudando a reduzir a redundância no código. Django oferece ferramentas integradas para autenticação, roteamento, formulários e ORM (Object-Relational Mapping), facilitando a criação de aplicações web robustas e seguras. Além disso, vamos usar o Google Maps API, que é crucial para o cálculo de rotas, distâncias e tempos de chegada, oferecendo uma solução confiável e precisa para o planejamento de viagens. O Google Maps API fornece dados atualizados e precisos, o que é essencial para o funcionamento eficiente do DriverPlan, especialmente para motoristas que precisam planejar suas rotas com antecedência.

Outras tecnologias a serem utilizadas serão as ferramentas de desenvolvimento, como o Visual Studio Code, que é um editor de código poderoso e extensível, suportando uma ampla variedade de linguagens de programação e ferramentas de desenvolvimento, o Git, um sistema de controle de versão distribuído que permite rastrear alterações no código e colaborar com outros desenvolvedores, e o GitHub, plataforma de hospedagem de repositórios Git, facilitando a colaboração em projetos, gerenciamento de código e revisão de alterações. Essas ferramentas são essenciais para o controle de versão e a colaboração eficiente da equipe de desenvolvimento.

Por fim, usaremos o MySQL, escolhido pela sua robustez, escalabilidade e eficiência na gestão de dados. MySQL é um sistema de gerenciamento de banco de dados relacional que suporta uma ampla variedade de aplicações, desde sistemas pequenos até grandes aplicações empresariais. Ele oferece segurança robusta, alta performance e confiabilidade, além de ser amplamente suportado e documentado, o que facilita a manutenção.

2.9 Restrições adicionais

O aplicativo é composto de um sistema simples e objetivo para seus usuários que só pode ser acessado via navegador web e integrado em APIs que não sejam necessariamente do Google, reforçando a relevância do software enquanto a sua usabilidade, navegação, padronização e portabilidade de suas funcionalidades.

3 Bibliografia

Django. Disponível em: <https://docs.djangoproject.com/en/5.0/>. Acesso em: 19 maio. 2024.

Entendendo o MVT (Model View Template). Disponível em: <https://www.usandopy.com/pt/artigo/entendendo-o-mvt-model-view-template/>. Acesso em: 19 maio. 2024.