**MOVBR**

**MOVBR**

**Arquitetura de Software**

**Versão <1.0>**

* **Histórico de Revisão**

| **Data** | **Versão** | **Descrição** | **Autor** |
| --- | --- | --- | --- |
| 10/05/2025 | 1.0 | Elaboração inicial do documento | Equipe MOVBR |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Sumário**

1. Introdução----------------------------------------------------------------------------------  
    1.1 Finalidade  
    1.2 Escopo  
    1.3 Descrição do Problema  
    1.4 Definições, Acrônimos e Abreviações
2. Representação Arquitetural------------------------------------------------------------  
    2.1 Visão Geral  
    2.2 Componentes Principais
3. Metas e Restrições da Arquitetura---------------------------------------------------  
    3.1 Metas  
    3.2 Restrições
4. Visão de Casos de Uso------------------------------------------------------------------
5. Visão Lógica--------------------------------------------------------------------------------
6. Visão de Implantação--------------------------------------------------------------------
7. Visão da Implementação----------------------------------------------------------------
8. Tamanho e Desempenho---------------------------------------------------------------
9. Qualidade-----------------------------------------------------------------------------------

**1. Introdução**

**1.1 Finalidade**

Este documento descreve a arquitetura de software do projeto **MOVBR**, cujo objetivo é entregar uma aplicação web responsiva que otimize e agilize a experiência de uso do transporte público no Distrito Federal. Serve como referência para desenvolvedores, stakeholders e novos membros da equipe.

**1.2 Escopo**

Este documento auxilia os envolvidos no projeto a captar aspectos arquiteturais do sistema que são necessários para o desenvolvimento de uma solução que atenda às necessidades dos usuários finais. Além de auxiliar no entendimento do sistema por novos membros da equipe.

**1.3 Descrição do Problema**

| **Problema** | 1. **Desinformação dos usuários,** que não conseguem acessar com facilidade horários atualizados, localização dos veículos e condições do trânsito em tempo real; 2. **Atrasos frequentes e mudanças inesperadas de rota que não são comunicadas de forma eficaz;** 3. **Desaproveitamento do potencial turístico, já que visitantes não têm acesso integrado a informações sobre pontos turísticos próximos às rotas de transporte.** |
| --- | --- |
| **Afeta** | Os usuários do Transporte público. |
| **Impacto** | Uso ineficiente dos modais de transporte. |
| **Solução** | * **Roteiro Inteligente: geração de itinerários personalizados com base na origem e destino informados pelo usuário, utilizando múltiplos modais. O sistema sugere a melhor opção de trajeto e informa a previsão de chegada ao destino;** * **Monitoramento em Tempo Real: exibição da localização atual dos veículos, previsão de chegada ao ponto de embarque do usuário e alertas sobre mudanças de rota, atrasos ou interrupções no serviço;** * **Informações Turísticas Integradas: apresentação de pontos turísticos próximos às paradas de transporte, com sugestões de passeios e mapas interativos.** |

**1.4 Definições, Acrônimos e Abreviações**

* **API**: Application Programming Interface
* **GPS**: Global Positioning System
* **MVC**: Model-View-Controller
* **UI**: User Interface
* **DB** – Banco de Dados
* **REST** – Representational State Transfer
* **API** – Application Programming Interface

**1.5 Visão Geral**

Este documento apresenta diferentes visões arquiteturais que detalham como o sistema MOVBR deve se comportar em seus diversos processos. São descritos os aspectos relacionados à implantação, implementação, bem como as restrições de desempenho e qualidade que o sistema deve atender.

**1.5.1 Tabela de Funcionalidades**

| **Funcionalidades** |
| --- |
| •  Pesquisa de rotas por nome e visualização de detalhes. |
| •  Visualização de paradas e seus horários previstos. |
| • Geração de itinerários com integração entre diferentes modais |
| • Mapas interativos com marcadores personalizados (Leaflet). |
| •  Localização de paradas próximas com base na geolocalização do usuário. |
| •  Exibição de pontos turísticos próximos às paradas. |
| •  Interface web responsiva baseada em templates HTML + CSS. |
| • API RESTful com dados georreferenciados via MongoDB. |

**1.5.2 Levantamento e Análise de Requisitos**

**2.1 Entendimento do Problema**

Para o desenvolvimento do MOVBR, foram investigadas as principais dificuldades enfrentadas pelos usuários do transporte público do Distrito Federal. As informações foram coletadas a partir de análises de reclamações comuns, entrevistas com usuários e consulta a bases públicas de dados.

Entre os cenários de uso e "dores" mais citados, destacam-se:

* **Falta de informações em tempo real** sobre localização e horários dos veículos;
* **Mudanças inesperadas de rota** sem aviso;
* **Dificuldade de integração** entre diferentes modais de transporte;
* **Ausência de informações turísticas integradas** às rotas e pontos de parada.

**2.1 Requisitos Funcionais e Não-Funcionais**

**Funcionais:**

* Pesquisa e geração de itinerários entre origem e destino;
* Visualização de localização em tempo real dos ônibus;
* Emissão de alertas sobre atrasos ou mudanças de rota;
* Exibição de pontos turísticos próximos às rotas;
* Acesso a mapas interativos;
* Cadastro e autenticação de usuários.

**Não-Funcionais:**

* **Desempenho:** respostas em menos de 2 segundos para consultas de rota;
* **Segurança:** autenticação de usuários e proteção dos dados pessoais;
* **Usabilidade:** interface acessível e intuitiva para usuários de diferentes perfis;
* **Disponibilidade:** sistema operando 24/7 com alta confiabilidade;
* **Escalabilidade:** possibilidade de expansão para outras regiões ou inclusão de mais modais.

2. **Resumo dos Usuários**

| **Nome** | **Responsabilidade** | **Descrição** |
| --- | --- | --- |
| **Usuário Comum** | Consultar rotas e horários, visualizar localização dos veículos em tempo real, explorar pontos turísticos. | Utiliza o sistema para se locomover com eficiência, receber alertas sobre mudanças nas rotas e acessar informações úteis sobre o trajeto. |
| **Turista** | Descobrir pontos turísticos próximos às paradas, planejar deslocamentos integrando diferentes modais. | Explora atrações do DF com ajuda do sistema, que sugere passeios, exibe mapas interativos e fornece estimativas de tempo de chegada. |
| **Administrador do Sistema** | Manter dados atualizados, gerenciar fontes externas, garantir funcionamento dos serviços e qualidade das informações. | Responsável técnico pelo backend, coleta e integração de APIs públicas, verificação da precisão dos dados geoespaciais e desempenho da aplicação. |

**3. Necessidade do cliente**

**4. Representação Arquitetural**

**4.1 Visão Geral**

A arquitetura do MOVBR é baseada no modelo **cliente-servidor**, com backend em **Django** (Python) e frontend em **React**, organizados por camadas e comunicação via API RESTful. Serviços assíncronos e geolocalização também fazem parte da arquitetura.

* Backend: Flask + PyMongo
* Frontend: HTML + CSS + JavaScript + Jinja2
* Banco de Dados: MongoDB com suporte a geolocalização (2dsphere)
* Integração: Leaflet.js para mapas + localização via navegador
* Organização: Rotas divididas por interface (HTML) e API (JSON)

**4.2 Visão de Casos de Uso**

Esta visão apresenta as funcionalidades principais do sistema e os perfis dos usuários que interagem com ele. Destacam-se os seguintes casos:

* Pesquisa e geração de rotas entre pontos informados pelo usuário.
* Visualização em tempo real da localização dos veículos.
* Recebimento de alertas sobre atrasos ou mudanças nas rotas.
* Exploração de pontos turísticos próximos às paradas.

**4.3 Visão Lógica**

Define a organização estática do sistema, mostrando a estrutura de classes e módulos, e o padrão arquitetural adotado (MVC):

* **Model:** Entidades e regras de negócio (ex: Rotas, Veículos, Usuários).
* **View:** Interface do usuário, construída em React.
* **Controller:** API RESTful em Django que orquestra as interações entre Model e View.

**4.4 Visão de Processos**

Mostra o comportamento dinâmico do sistema diante de eventos, incluindo:

* Processamento de requisições do usuário para busca de rotas.
* Atualização periódica das informações dos veículos via tarefas assíncronas.
* Geração e envio de notificações em tempo real.

**4.5 Visão de Implantação**

Descreve a configuração física e lógica do ambiente onde o sistema será executado, incluindo:

* Servidores ou containers para backend e frontend.
* Banco de dados geoespacial.
* Serviços de fila para processamento assíncrono (Celery + Redis).
* Integração com APIs externas para dados de transporte e turismo.

**4.6 Visão de Implementação**

Detalha a distribuição do código e componentes do sistema em unidades executáveis, destacando:

* Módulos backend organizados em pacotes Django.
* Aplicação frontend React consumindo a API REST.
* Workers Celery para tarefas em background.
* Configuração e deployment via containers Docker.

**4.7 Componentes Principais**

* **Frontend Web (React):** Exibe rotas, localização de veículos e pontos turísticos.
* **Backend RESTful (Django + DRF):** Fornece dados e lógica de negócio.
* **Banco de Dados (PostgreSQL + PostGIS):** Armazena rotas, paradas, horários e locais turísticos.
* **Celery + Redis:** Processamento assíncrono de dados externos e notificações.
* **APIs externas:** Coleta de dados de transporte e mapas.

**5. Metas e Restrições de Arquitetura**

* 1. **Metas**
* **Escalabilidade Modular:** O sistema deve permitir fácil inclusão de novas funcionalidades, modais de transporte ou regiões.
* **Desempenho:** As respostas do sistema devem ser rápidas, especialmente para consultas de rotas e localização.
* **Alta Disponibilidade:** O sistema deve estar disponível continuamente, com mínimo tempo de inatividade.
* **Usabilidade:** A interface deve ser simples, responsiva e acessível a diversos perfis de usuário, incluindo turistas.

**5.2 Restrições Técnicas**

* **Paradigma de Programação:** Utilização do paradigma orientado a objetos.
* **Padrão Arquitetural:** Estrutura baseada em MVC (Model-View-Controller).
* **Linguagem de Programação (Backend):** Python 3.10+.
* **Framework Backend:** Django com Django REST Framework.
* **Frontend**: Implementado com React (JavaScript).
* **Multiplataforma:** A aplicação deve funcionar em qualquer navegador moderno (web responsiva).
* **Banco de Dados:** PostgreSQL, com extensão PostGIS para suporte a dados geoespaciais.

**6. Arquitetura do Sistema e Design de Software**

**6.1 Visão Geral da Arquitetura**

A arquitetura do sistema **MOVBR** adota um padrão **monolítico modular**, permitindo a separação lógica de responsabilidades dentro do mesmo código-base. Essa estrutura favorece a manutenibilidade e organização do projeto durante a fase inicial. Na **versão 2 do sistema**, está prevista a migração gradual para uma arquitetura baseada em **micro-serviços**, com componentes independentes e escaláveis de forma isolada.

**Tecnologias Utilizadas:**

* Python 3.10+
* Flask
* PyMongo
* MongoDB
* Jinja2
* Leaflet.js
* HTML5 / CSS3
* JavaScript
* Geolocalização via navegador

**6.2 Design de Software (Diagramas)**

A seguir estão os principais artefatos utilizados para representar o design técnico do sistema:

* **Diagrama UML de Classes:** Descreve as entidades principais como Usuário, Veículo, Rota, PontoTurístico, Itinerário, Notificação, e os relacionamentos entre elas.
* **Diagrama de Sequência:** Representa o fluxo de execução de um caso de uso típico, como a **consulta de rotas**, incluindo requisição feita pelo frontend, processamento no backend, e retorno dos dados ao usuário.
* **Diagrama de Componentes:** Apresenta os principais módulos da aplicação e suas responsabilidades, como:

Cadastro — gerenciamento de contas e autenticação

Itinerários — geração e simulação de rotas

Monitoramento — localização em tempo real

Turismo — exibição de pontos turísticos

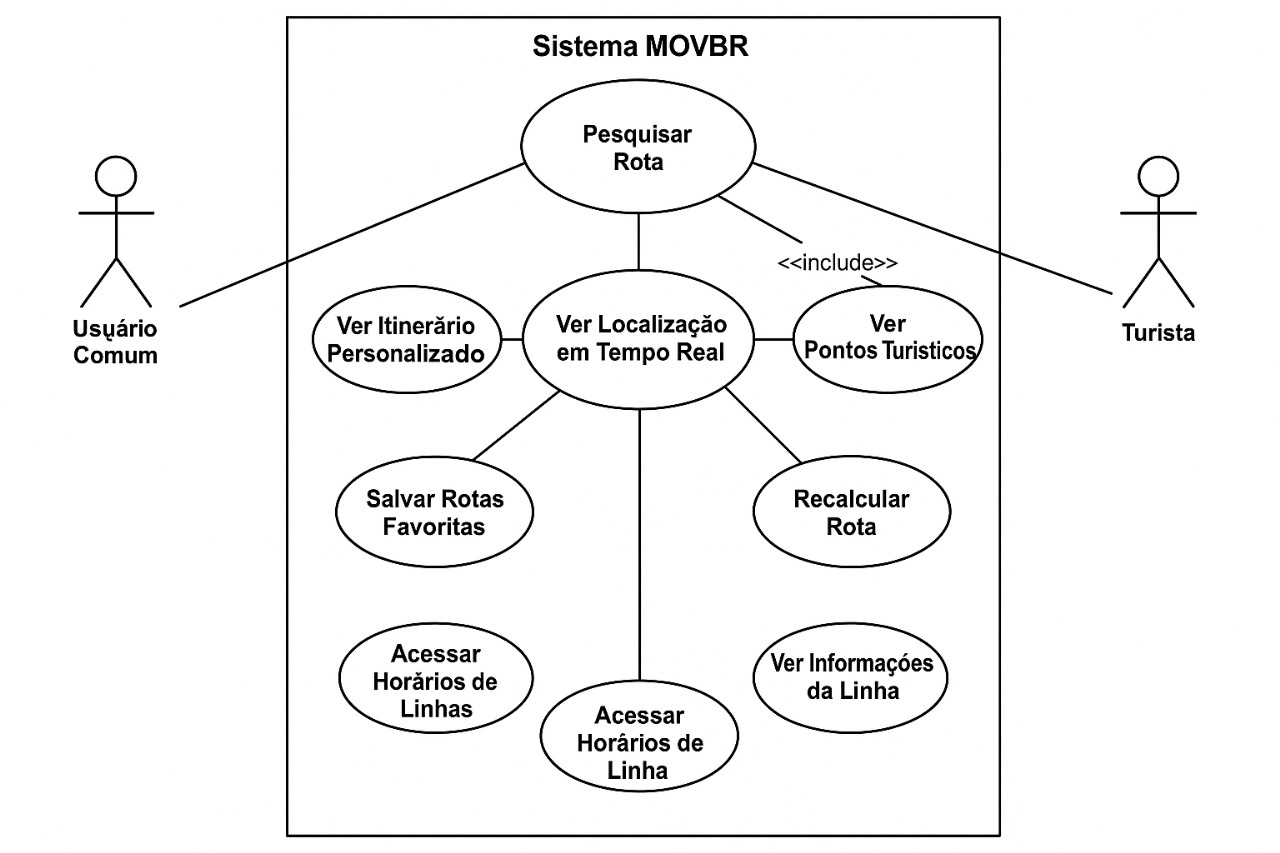
Notificações — alertas sobre atrasos e mudanças

* **Modelos de Banco de Dados:**

**Modelo Conceitual** — entidades e associações em alto nível.

**Modelo Lógico** — tabelas, chaves primárias e estrangeiras, tipos de dados e índices geoespaciais.

**6. Visão de Casos de Uso**



**7. Visão Lógica**

**7.1 Visão Geral**

A visão lógica descreve a estrutura da arquitetura do sistema MOVBR em termos de camadas e componentes, com foco na organização interna do software. O padrão arquitetural adotado é o **MVC (Model-View-Controller)**, que promove a separação de responsabilidades e facilita a manutenção e evolução do sistema.

* 1. **Estrutura em Camadas (MVC)**
* **Model (Modelo):** Representa as entidades do sistema, como Usuário, Veículo, Rota, Itinerário, Ponto Turístico e Notificação. Também contém a lógica de negócios associada, como validações, cálculos de tempo estimado, e regras de relacionamento.
* **View (Visão):** Componente de interface com o usuário, implementado em **React**. Responsável por exibir as informações ao usuário e enviar requisições ao backend (API RESTful).
* **Controller (Controle):** Intermediário entre as Views e os Models. No caso do MOVBR, está representado pelas **Views do Django REST Framework**, que processam requisições HTTP, validam dados, executam regras de negócio e retornam respostas em formato JSON.

**7.3 Organização dos Pacotes**

**Backend (Django):**

* core.models: modelos do sistema (Usuário, Veículo, Rota, etc.)
* core.serializers: serialização e validação de dados
* core.views: controladores das requisições da API
* core.tasks: tarefas assíncronas com Celery
* core.utils: utilitários e serviços auxiliares

**Frontend (React):**

* components/: componentes reutilizáveis da UI
* pages/: telas da aplicação (Home, Rotas, Turismo, etc.)
* services/: chamadas para a API
* hooks/: lógica de estado e efeitos personalizados

**8. Visão de Implantação**

| **Componente** | **Hospedagem** | **Container** | **Escalável** |
| --- | --- | --- | --- |
| Frontend (React) | Vercel / Netlify | Opcional | Sim |
| Backend (Django) | Heroku / Railway | Sim | Sim |
| PostgreSQL + PostGIS | Supabase / AWS RDS | Sim | Sim |
| Redis + Celery | VPS / Docker | Sim | Sim |
| APIs externas | Dados GTFS / OSM | — | — |

**9. Visão da Implementação**

O sistema MOVBR será implementado utilizando os princípios da Programação Orientada a Objetos, com base na arquitetura MVC (Model-View-Controller), promovendo a separação de responsabilidades e facilitando a manutenção e evolução do software. A aplicação é construída com tecnologias modernas tanto no frontend quanto no backend, e organizada de forma modular para possibilitar uma futura migração para arquitetura de micro-serviços.

O backend será desenvolvido em Python 3.10+, utilizando o framework Django em conjunto com o Django REST Framework, responsável pela construção da API RESTful que fornece os dados ao frontend. A estrutura interna será organizada em pacotes coesos como models, views, serializers, services (para lógica de negócio), tasks (para tarefas assíncronas com Celery + Redis) e utils.

O frontend será implementado com React (JavaScript), garantindo uma interface web responsiva, intuitiva e acessível. Essa camada será composta por páginas (pages), componentes reutilizáveis (components), serviços de comunicação com a API (services) e o roteamento da aplicação (routes).

O sistema utilizará o PostgreSQL como banco de dados relacional, com a extensão PostGIS para suporte a dados geoespaciais, essenciais para o cálculo e exibição de rotas, localizações de veículos e pontos turísticos. Toda a aplicação será containerizada por meio do Docker, garantindo portabilidade, consistência entre ambientes e facilidade de implantação em servidores em nuvem como Heroku, Railway, Render ou VPS próprios.

Durante sua primeira versão, a arquitetura adotada será monolítica modular. No entanto, toda a organização dos pacotes e serviços foi planejada para permitir a evolução para uma arquitetura de micro-serviços, com separação dos módulos como autenticação, itinerários, monitoramento e turismo em serviços independentes.

A integração com APIs externas públicas de transporte e turismo permitirá o enriquecimento dos dados apresentados ao usuário final. A comunicação entre os módulos será realizada de forma assíncrona quando necessário, garantindo desempenho e escalabilidade.

Em resumo, a implementação do MOVBR segue uma abordagem moderna, robusta e extensível, garantindo qualidade, organização e viabilidade para crescimento futuro da aplicação.

A implementação segue boas práticas de POO e REST:

* **Frameworks:** Django, DRF, React, Celery
* **Containers:** Docker para backend, banco e workers
* **Controle de versão:** Git + GitHub
* **Deploy:** Automatizado via CI/CD (ex: GitHub Actions, Railway Deploy)

**10. Tamanho e Desempenho**

A aplicação MOVBR será acessada por meio de um **site próprio responsivo**, compatível com navegadores modernos em **dispositivos desktop e móveis**, garantindo que tanto **usuários locais quanto turistas** possam utilizá-la com facilidade a partir de computadores, smartphones ou tablets, sem necessidade de instalação.

**Desempenho Esperado:**

* O tempo de resposta para consultas de rotas e localização de veículos deve ser inferior a **2 segundos**, mesmo com múltiplos usuários simultâneos.
* A atualização de dados externos, como posição dos veículos e alertas de transporte, será realizada por meio de tarefas assíncronas com **execução em background**, para não impactar a experiência do usuário.
* O sistema deverá suportar o uso contínuo 24/7, com alta disponibilidade e tolerância a falhas em serviços críticos, como notificações e geração de itinerários.

**Tamanho do Sistema:**

* O banco de dados armazenará:
  + Informações de usuários, rotas, veículos e pontos turísticos;
  + Registros de itinerários gerados e notificações enviadas;
  + Dados geoespaciais indexados para consultas eficientes.
* A arquitetura modular e containerizada permite fácil replicação do sistema em ambientes de teste, homologação e produção.
* O sistema será escalável horizontalmente, permitindo a adição de novos servidores para atender à demanda crescente.

**11. Qualidade**

O padrão arquitetural adotado no projeto MOVBR visa garantir uma melhor organização do código-fonte, promovendo **manutenibilidade, portabilidade e escalabilidade** do sistema.

**Aspectos de Qualidade Considerados:**

* **Manutenibilidade:** A estrutura modular e o uso do padrão MVC facilitam a compreensão, correção e evolução do código, permitindo que novas funcionalidades sejam adicionadas com menor esforço.
* **Portabilidade:** A aplicação é desenvolvida com tecnologias amplamente suportadas e containerizada via Docker, o que permite sua execução em diferentes ambientes e plataformas, desde máquinas locais até servidores em nuvem.
* **Escalabilidade:** A arquitetura monolítica modular preparada para futura migração a micro-serviços assegura que o sistema possa crescer em capacidade e funcionalidade conforme a demanda aumenta.
* **Reutilização de Código:** Componentes bem definidos e pacotes reutilizáveis reduzem a duplicação de código e aumentam a produtividade da equipe.
* **Testabilidade:** A separação clara entre camadas facilita a implementação de testes unitários e de integração, garantindo maior confiabilidade e menor incidência de erros em produção.
* **Segurança:** Práticas como validação de dados, autenticação segura e uso de protocolos HTTPS são incorporadas para proteger a integridade e confidencialidade das informações dos usuários.
* **Usabilidade:** Interfaces desenvolvidas com foco na experiência do usuário, garantindo acessibilidade e facilidade de uso para diversos perfis, incluindo turistas e pessoas com necessidades especiais.