

# ChevoTech

**Solução: Sensor OpalaVision**

Integrantes:

RM 99404 - Thiago Garcia Tonato

RM 55502 - Ian Madeira Gonçalves da Silva

RM 555109 - Murilo Ribeiro Santos

# Introdução

O projeto de automonitoramento de veículos da ChevoTech é uma iniciativa inovadora que visa eliminar a necessidade de monitoramento humano constante de automóveis. O sistema utiliza o sensor "OpalaVision", um sensor "master" capaz de capturar dados de diversos sensores instalados no veículo. Esses dados são então transmitidos para um aplicativo dedicado, que armazena e exibe as informações de forma acessível, permitindo ao usuário monitorar o estado do veículo em tempo real.

A API foi desenvolvida utilizando o framework Jersey com Maven para facilitar o gerenciamento de dependências e está conectada a um banco de dados Oracle para armazenamento persistente dos dados. Ela permite gerenciar usuários do sistema, com funcionalidades que incluem:

- **Cadastro de usuários**, com validação de CEP por meio da integração com a API do ViaCep.
- **Login de usuários**, com autenticação baseada em credenciais.

Tecnologias Utilizadas:

- **Java**: Linguagem principal da aplicação, usada para desenvolver toda a lógica da API e funcionalidades do sistema.
- **Oracle SQL**: Banco de dados relacional utilizado para armazenar informações dos usuários, carros e sensores.
- **JDBC (Java Database Connectivity)**: Tecnologia para conectar a aplicação Java ao banco de dados Oracle.
- **Oracle SQL Developer**: Ferramenta de gerenciamento e administração do banco de dados Oracle, usada para consultas e manutenção.
- **Maven**: Gerenciador de dependências e automação de build para simplificar o desenvolvimento e configuração do projeto.
- **Jersey**: Framework para facilitar a criação de serviços RESTful com Java, usado na construção da API.
- **OkHttp**: Biblioteca para realizar requisições HTTP, utilizada para integração com a API pública ViaCep.
- **Gson**: Biblioteca para conversão entre objetos Java e JSON, facilitando o processamento de respostas JSON da API ViaCep.

# Funcionalidades Chaves

## Gerenciamento de Usuários

- **Cadastrar Usuário:** Permite o registro de um novo usuário no sistema. O cadastro inclui dados como nome de usuário, senha e endereço, com validação do CEP fornecido através da API pública ViaCep.
- **Login de Usuário:** Permite que um usuário faça login no sistema com suas credenciais, validando nome de usuário e senha.

## Validação de Endereço

- **Validação de CEP:** Ao cadastrar um usuário, o CEP fornecido é validado em tempo real através da API ViaCep. Se o CEP for válido, as informações de endereço (logradouro e UF) são automaticamente preenchidas.

## Conexão com o Banco de Dados

- **ConnectionFactory:** A aplicação utiliza uma classe `ConnectionFactory` para gerenciar a comunicação com o banco de dados Oracle. Cada operação de cadastro, consulta ou autenticação estabelece uma conexão com o banco de dados, encerrando-a automaticamente após a execução.

## Persistência de Dados

- **Armazenamento de Usuários e Endereços:** A API armazena as informações dos usuários e seus endereços em um banco de dados Oracle, permitindo consultas, autenticações e operações de persistência de forma segura e eficiente.

# Procedimentos para Rodar a Aplicação

## Pré-requisitos

- **Java JDK 11+:** Certifique-se de que o Java Development Kit está instalado. Verifique a instalação executando `java -version` no terminal.
- **Maven:** Necessário para gerenciamento de dependências e build do projeto. Verifique com `mvn -version`.
- **Oracle Database:** A aplicação está configurada para se conectar a um banco de dados Oracle. Garanta que o Oracle Database está instalado e em execução.
- **Oracle SQL Developer:** Opcional para gerenciar o banco de dados Oracle.

## Configuração do Banco de Dados

Crie a tabela necessárias no banco de dados Oracle:

```
CREATE TABLE usuarios (  
  username VARCHAR(200),  
  password VARCHAR(200),  
  cep VARCHAR(10),  
  logradouro VARCHAR(100),  
  uf VARCHAR(50)  
);
```

Ajuste as credenciais de conexão no arquivo `ConnectionFactory.java` para que correspondam às credenciais do seu banco de dados Oracle:

```
String urlDeConexao = "jdbc:oracle:thin:@oracle.fiap.com.br:1521:ORCL";  
String login = "<username>";  
String senha = "<password>";
```

## Compilação e Build do Projeto

Abra o terminal na raiz do projeto e execute o comando abaixo para baixar as dependências e compilar o projeto:

```
mvn clean install
```

## Executando a Aplicação

Após o build, execute a aplicação com o comando:

```
mvn jetty:run
```

A aplicação estará disponível no endereço padrão `http://localhost:8080`.

## Testando os Endpoints

Utilize um cliente REST, como o **Postman** ou **cURL**, para testar os endpoints da API.

Exemplos de requisições:

- **Cadastro de Usuário:**

```
POST http://localhost:8080/api/users/register  
Content-Type: application/json  
Body:  
{  
  "username": "usuario1",  
  "password": "senha123",  
  "endereco": {  
    "cep": "01001000"  
  }  
}
```

- **Login de Usuário:**

```
POST http://localhost:8080/api/users/login  
Content-Type: application/json  
Body:  
{  
  "username": "usuario1",  
  "password": "senha123"  
}
```

## Log e Debug

- Verifique os logs no terminal para mensagens de erro ou sucesso.
- Para mais detalhes, ajuste o nível de logging nas configurações do projeto.

# Diagrama de Classes

