

Sistema de Logística - AvaSystem

Sistema completo de gestão logística para controle de frota, motoristas, manutenções e documentos de transporte (CT-e).

Índice

- [Características](#)
- [Tecnologias](#)
- [Pré-requisitos](#)
- [Instalação](#)
- [Configuração](#)
- [Executando o Projeto](#)
- [Estrutura do Projeto](#)
- [API Endpoints](#)
- [Perfis de Usuário](#)
- [Deploy](#)

Características

-  **Autenticação JWT** com múltiplos perfis de acesso
-  **Gestão de Veículos** - Cadastro e controle de frota
-  **Gestão de Motoristas** - Controle de CNH e dados pessoais
-  **Manutenções** - Sistema completo de registro e acompanhamento
-  **CT-e** - Upload e gestão de documentos fiscais
-  **Dashboard** - Indicadores e estatísticas em tempo real
-  **Responsivo** - Interface adaptada para mobile (motoristas)
-  **Performance** - Otimizado com índices de banco e cache

Tecnologias

Backend

- Node.js 18+
- Express.js
- MySQL 8.0
- Sequelize ORM
- JWT (jsonwebtoken)
- Bcrypt
- Multer (upload de arquivos)

Frontend

- React 18
- Axios

- React Router
- Context API
- Lucide Icons
- CSS3 (Flexbox/Grid)

Pré-requisitos

Antes de começar, certifique-se de ter instalado:

- [Node.js](#) (versão 18 ou superior)
- [MySQL](#) (versão 8.0 ou superior)
- [Git](#)
- npm ou yarn

Instalação

1. Clone o repositório

```
git clone https://github.com/seu-usuario/sistema-logistica.git  
cd sistema-logistica
```

2. Configurar o Banco de Dados

```
# Entre no MySQL  
mysql -u root -p  
  
# Execute o script SQL  
source database-schema.sql  
  
# Ou se preferir:  
mysql -u root -p < database-schema.sql
```

O script cria automaticamente:

- Banco de dados `sistema_logistica`
- Todas as tabelas necessárias
- Índices para performance
- Views úteis
- Stored procedures
- Dados iniciais (usuários de teste)

3. Instalar Dependências do Backend

```
cd backend  
npm install
```

4. Configurar Variáveis de Ambiente

```
# Copiar arquivo de exemplo  
cp .env.example .env
```

```
# Editar com suas configurações  
nano .env
```

Exemplo de .env:

```
NODE_ENV=development  
PORT=5000  
  
DB_HOST=localhost  
DB_PORT=3306  
DB_NAME=sistema_logistica  
DB_USER=root  
DB_PASS=sua_senha_mysql  
  
JWT_SECRET=chave_super_secreta_min_32_caracteres_aqui  
JWT_EXPIRES_IN=7d  
  
MAX_FILE_SIZE=5242880  
UPLOAD_PATH=./uploads  
  
FRONTEND_URL=http://localhost:3000
```

5. Instalar Dependências do Frontend

```
cd ../sistema-logistica  
npm install
```

⚙️ Configuração

Usuários Padrão

O sistema vem com 3 usuários pré-cadastrados para teste:

Usuário	Senha	Perfil
gerente	123	Gerente
assistente	123	Assistente
motorista	123	Motorista

⚠️ IMPORTANTE: Altere estas senhas em produção!

Estrutura de Perfis

- **Gerente:** Acesso total ao sistema
- **Assistente:** Cadastros e visualizações (sem deletar)
- **Motorista:** Visualiza seus próprios dados e registra informações

⌚ Executando o Projeto

Desenvolvimento

Terminal 1 - Backend

```
cd backend  
npm run dev  
  
# Servidor rodará em: http://localhost:5000
```

Terminal 2 - Frontend

```
cd sistema-logistica  
npm start  
  
# Aplicação abrirá em: http://localhost:3000
```

Produção

Backend

```
cd backend  
npm start
```

Frontend

```
cd sistema-logistica  
npm run build  
  
# Servir com servidor web (nginx, apache, etc)
```

📁 Estrutura do Projeto

```
sistema-logistica/  
└── backend/  
    ├── config/  
    │   └── database.js      # Configuração MySQL  
    ├── controllers/  
    │   ├── authController.js # Autenticação  
    │   ├── vehicleController.js # Veículos  
    │   ├── maintenanceController.js # Manutenções  
    │   ├── cteController.js    # CT-e  
    │   └── dashboardController.js # Dashboard  
    ├── middleware/  
    │   ├── auth.js          # JWT  
    │   ├── upload.js         # Multer  
    │   ├── errorHandler.js  # Erros  
    │   └── validators.js     # Validações  
    ├── models/  
    │   ├── User.js  
    │   ├── Vehicle.js  
    │   ├── Maintenance.js  
    │   ├── Cte.js  
    │   └── index.js  
    └── routes/  
        ├── auth.routes.js  
        ├── user.routes.js  
        ├── vehicle.routes.js  
        ├── maintenance.routes.js  
        └── cte.routes.js
```

```

    └── dashboard.routes.js
    ├── uploads/                      # Arquivos (gitignored)
    ├── .env                           # Variáveis (gitignored)
    ├── .env.example
    ├── package.json
    └── server.js

    └── sistema-logistica/           # Frontend React
        ├── public/
        └── src/
            ├── components/
            ├── context/
            ├── services/
            ├── App.js
            └── index.js
        └── package.json

    └── database-schema.sql          # Schema do banco
    └── README.md

```

API Endpoints

Autenticação

POST	/api/auth/register	# Registrar usuário
POST	/api/auth/login	# Login
GET	/api/auth/me	# Dados do usuário logado
PUT	/api/auth/change-password	# Alterar senha
POST	/api/auth/logout	# Logout

Usuários

GET	/api/users	# Listar usuários
GET	/api/users/:id	# Obter usuário
PUT	/api/users/:id	# Atualizar usuário
DELETE	/api/users/:id	# Desativar usuário

Veículos

GET	/api/vehicles	# Listar veículos
GET	/api/vehicles/:id	# Obter veículo
POST	/api/vehicles	# Criar veículo
PUT	/api/vehicles/:id	# Atualizar veículo
PUT	/api/vehicles/:id/km	# Atualizar KM
DELETE	/api/vehicles/:id	# Desativar veículo

Manutenções

GET	/api/maintenances	# Listar manutenções
GET	/api/maintenances/urgent	# Manutenções urgentes
GET	/api/maintenances/:id	# Obter manutenção
POST	/api/maintenances	# Criar manutenção
PUT	/api/maintenances/:id	# Atualizar manutenção
PUT	/api/maintenances/:id/status	# Atualizar status
DELETE	/api/maintenances/:id	# Deletar manutenção

CT-e

```
GET /api/ctes # Listar CT-e
GET /api/ctes/:id # Obter CT-e
POST /api/ctes # Criar CT-e (multipart/form-data)
PUT /api/ctes/:id # Atualizar CT-e
GET /api/ctes/:id/download # Download do arquivo
DELETE /api/ctes/:id # Deletar CT-e
```

Dashboard

```
GET /api/dashboard/stats # Estatísticas gerais
GET /api/dashboard/urgent-maintenances # Manutenções urgentes
GET /api/dashboard/recent-activities # Atividades recentes
GET /api/dashboard/maintenances-chart # Gráfico de manutenções
```

Autenticação nas Requisições

Incluir header em todas as rotas protegidas:

```
Authorization: Bearer <seu_token_jwt>
```

Perfis de Usuário

Gerente

- Acesso completo
- Criar/editar/deletar todos os registros
- Visualizar todos os dados
- Gerenciar usuários

Assistente

- Criar e editar registros
- Visualizar todos os dados
- Não pode deletar
- Não pode gerenciar usuários

Motorista

- Visualizar seus próprios dados
- Registrar abastecimentos
- Fazer checklist de veículos
- Acesso limitado

Deploy

Backend (Produção)

1. Configure variáveis de ambiente no servidor

2. Instale PM2 para gerenciamento:

```
npm install -g pm2
pm2 start server.js --name api-logistica
pm2 save
pm2 startup
```

Frontend (Produção)

1. Build do projeto:

```
npm run build
```

2. Configure nginx:

```
server {
    listen 80;
    server_name seu-dominio.com;

    root /caminho/para/build;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

🔒 Segurança

- Senhas hasheadas com bcrypt (salt rounds: 10)
- JWT tokens com expiração configurável
- Validação de inputs com express-validator
- CORS configurado
- Helmet.js para headers de segurança
- Proteção contra SQL injection (Sequelize)
- Upload de arquivos com validação de tipo e tamanho

Guia de Deploy - Sistema de Logística

Checklist Pré-Deploy

- [] Todos os testes locais passando
- [] Variáveis de ambiente configuradas
- [] Banco de dados criado e populado
- [] Backup do banco de dados realizado
- [] SSL/HTTPS configurado
- [] Domínio apontando para servidor
- [] Firewall configurado

Opção 1: Deploy em VPS (Ubuntu/Debian)

1. Preparar Servidor

```
# Atualizar sistema
sudo apt update && sudo apt upgrade -y

# Instalar Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs

# Instalar MySQL
sudo apt install -y mysql-server

# Instalar PM2 globalmente
sudo npm install -g pm2

# Instalar Nginx
sudo apt install -y nginx

# Instalar Certbot (SSL)
sudo apt install -y certbot python3-certbot-nginx
```

2. Configurar MySQL

```
# Configurar MySQL
sudo mysql_secure_installation

# Entrar no MySQL
sudo mysql

# Criar banco e usuário
CREATE DATABASE sistema_logistica CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
CREATE USER 'logistica_user'@'localhost' IDENTIFIED BY
'SuaSenhaSegura123!';
GRANT ALL PRIVILEGES ON sistema_logistica.* TO
'logistica_user'@'localhost';
FLUSH PRIVILEGES;
```

```
EXIT;

# Importar schema
mysql -u logistica_user -p sistema_logistica < database-schema.sql
```

3. Deploy do Backend

```
# Criar pasta para aplicação
sudo mkdir -p /var/www/sistema-logistica
cd /var/www/sistema-logistica

# Clonar repositório ou transferir arquivos
git clone seu-repositorio.git .

# Instalar dependências do backend
cd backend
npm install --production

# Criar arquivo .env
nano .env
```

.env de Produção:

```
NODE_ENV=production
PORT=5000

DB_HOST=localhost
DB_PORT=3306
DB_NAME=sistema_logistica
DB_USER=logistica_user
DB_PASS=SuaSenhaSegura123!

JWT_SECRET=chave_super_secreta_production_min_32_caracteres_muito_long
a
JWT_EXPIRES_IN=7d

MAX_FILE_SIZE=5242880
UPLOAD_PATH=/var/www/sistema-logistica/backend/uploads

FRONTEND_URL=https://seudominio.com
# Criar pasta de uploads
mkdir uploads
sudo chown -R www-data:www-data uploads

# Iniciar com PM2
pm2 start server.js --name api-logistica
pm2 save
pm2 startup
```

4. Deploy do Frontend

```
# Ir para pasta do frontend
cd ../sistema-logistica

# Instalar dependências
npm install

# Criar .env.production
echo "REACT_APP_API_URL=https://seudominio.com/api" > .env.production
```

```
# Build de produção
npm run build

# Mover build para pasta do Nginx
sudo cp -r build /var/www/sistema-logistica-frontend
```

5. Configurar Nginx

```
# Criar arquivo de configuração
sudo nano /etc/nginx/sites-available/sistema-logistica
server {
    listen 80;
    server_name seudominio.com www.seudominio.com;

    # Frontend
    root /var/www/sistema-logistica-frontend;
    index index.html;

    # Logs
    access_log /var/log/nginx/logistica_access.log;
    error_log /var/log/nginx/logistica_error.log;

    # Frontend - SPA
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Backend API
    location /api {
        proxy_pass http://localhost:5000/api;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;

        # Timeout para uploads
        client_max_body_size 10M;
        proxy_connect_timeout 600;
        proxy_send_timeout 600;
        proxy_read_timeout 600;
        send_timeout 600;
    }

    # Arquivos estáticos (uploads)
    location /uploads {
        alias /var/www/sistema-logistica/backend/uploads;
        autoindex off;
    }

    # Gzip
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css text/xml text/javascript
                application/x-javascript application/xml+rss
    }
```

```

        application/javascript application/json;
    }
    # Habilitar site
    sudo ln -s /etc/nginx/sites-available/sistema-logistica
    /etc/nginx/sites-enabled/

    # Remover site padrão
    sudo rm /etc/nginx/sites-enabled/default

    # Testar configuração
    sudo nginx -t

    # Reiniciar Nginx
    sudo systemctl restart nginx

```

6. Configurar SSL (Certbot)

```

# Obter certificado SSL
sudo certbot --nginx -d seudominio.com -d www.seudominio.com

# Renovação automática já está configurada
# Testar renovação:
sudo certbot renew --dry-run

```

7. Firewall

```

# Habilitar UFW
sudo ufw enable

# Permitir SSH
sudo ufw allow 22

# Permitir HTTP e HTTPS
sudo ufw allow 80
sudo ufw allow 443

# Verificar status
sudo ufw status

```

Opção 2: Deploy com Docker

1. Criar Dockerfile para Backend

backend/Dockerfile:

```

FROM node:18-alpine

WORKDIR /app

# Copiar package.json e instalar dependências
COPY package*.json ./
RUN npm ci --production

# Copiar código
COPY . .

# Criar pasta de uploads

```

```

RUN mkdir -p uploads
EXPOSE 5000
CMD ["node", "server.js"]

```

2. Criar Dockerfile para Frontend

sistema-logistica/Dockerfile:

```

FROM node:18-alpine AS build

WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY ..
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

3. Docker Compose

docker-compose.yml:

```

version: '3.8'

services:
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: sistema_logistica
      MYSQL_USER: logistica_user
      MYSQL_PASSWORD: logistica_pass
    volumes:
      - mysql_data:/var/lib/mysql
      - ./database-schema.sql:/docker-entrypoint-initdb.d/schema.sql
    ports:
      - "3306:3306"
    networks:
      - logistica-network

  backend:
    build: ./backend
    environment:
      NODE_ENV: production
      PORT: 5000
      DB_HOST: mysql
      DB_NAME: sistema_logistica
      DB_USER: logistica_user
      DB_PASS: logistica_pass
      JWT_SECRET: ${JWT_SECRET}
    ports:
      - "5000:5000"
    depends_on:

```

```

        - mysql
volumes:
    - ./backend/uploads:/app/uploads
networks:
    - logistica-network

frontend:
    build: ./sistema-logistica
    ports:
        - "80:80"
    depends_on:
        - backend
    networks:
        - logistica-network

volumes:
    mysql_data:

networks:
    logistica-network:
        driver: bridge
# Iniciar containers
docker-compose up -d

# Ver logs
docker-compose logs -f

# Parar containers
docker-compose down

```

Opção 3: Deploy em Plataformas Cloud

Heroku

```

# Instalar Heroku CLI
npm install -g heroku

# Login
heroku login

# Criar app
heroku create nome-do-app

# Adicionar MySQL (ClearDB)
heroku addons:create cleardb:ignite

# Configurar variáveis de ambiente
heroku config:set NODE_ENV=production
heroku config:set JWT_SECRET=sua_chave_secreta

# Deploy
git push heroku main

# Executar migrations
heroku run node migrate.js

```

DigitalOcean App Platform

1. Conectar repositório GitHub
2. Configurar:
 - o **Backend:** Node.js service (porta 5000)
 - o **Frontend:** Static site (npm run build)
 - o **Database:** Managed MySQL 8.0
3. Configurar variáveis de ambiente
4. Deploy automático

AWS EC2 + RDS

Similar ao deploy em VPS, mas usando:

- EC2 para servidor de aplicação
- RDS para MySQL
- S3 para uploads de arquivos
- CloudFront para CDN
- Route 53 para DNS

Monitoramento

PM2 Monitoramento

```
# Status dos processos
pm2 status

# Logs em tempo real
pm2 logs api-logistica

# Monitoramento de recursos
pm2 monit

# Reiniciar aplicação
pm2 restart api-logistica

# Salvar configuração
pm2 save
```

Logs do Sistema

```
# Logs do Nginx
sudo tail -f /var/log/nginx/logistica_access.log
sudo tail -f /var/log/nginx/logistica_error.log

# Logs do MySQL
sudo tail -f /var/log/mysql/error.log

# Logs da aplicação (PM2)
pm2 logs --lines 100
```

Backup

Script de Backup Automático

backup.sh:

```
#!/bin/bash

# Configurações
BACKUP_DIR="/var/backups/sistema-logistica"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="sistema_logistica"
DB_USER="logistica_user"
DB_PASS="SuaSenhaSegura123!"

# Criar pasta se não existir
mkdir -p $BACKUP_DIR

# Backup do banco de dados
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME | gzip >
"$BACKUP_DIR/db_backup_$DATE.sql.gz"

# Backup dos uploads
tar -czf "$BACKUP_DIR/uploads_backup_$DATE.tar.gz" /var/www/sistema-
logistica/backend/uploads

# Manter apenas os últimos 7 dias de backup
find $BACKUP_DIR -name "*.gz" -mtime +7 -delete

echo "Backup concluído: $DATE"
# Dar permissão de execução
chmod +x backup.sh

# Configurar cron para backup diário às 2h da manhã
crontab -e

# Adicionar linha:
0 2 * * * /caminho/para/backup.sh >> /var/log/backup.log 2>&1
```

Manutenção

Atualizar Aplicação

```
# Fazer backup primeiro!
./backup.sh

# Parar aplicação
pm2 stop api-logistica

# Atualizar código
cd /var/www/sistema-logistica
git pull origin main

# Atualizar dependências
cd backend
npm install --production

# Reiniciar
pm2 restart api-logistica

# Verificar logs
pm2 logs api-logistica --lines 50
```

Otimização de Performance

```
# Analisar uso de recursos
pm2 monit

# Aumentar PM2 instances (clustering)
pm2 start server.js -i max --name api-logistica

# Configurar Nginx cache
# Adicionar no nginx.conf:
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m
max_size=1g;

# Otimizar MySQL
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
# Adicionar:
# max_connections = 200
# innodb_buffer_pool_size = 1G
```

Troubleshooting

Aplicação não inicia

```
# Verificar logs do PM2
pm2 logs api-logistica

# Verificar porta em uso
sudo lsof -i :5000

# Testar conexão com banco
mysql -u logistica_user -p sistema_logistica
```

Erro 502 Bad Gateway

```
# Verificar se backend está rodando
pm2 status

# Verificar logs do Nginx
sudo tail -f /var/log/nginx/logistica_error.log

# Testar backend diretamente
curl http://localhost:5000/health
```

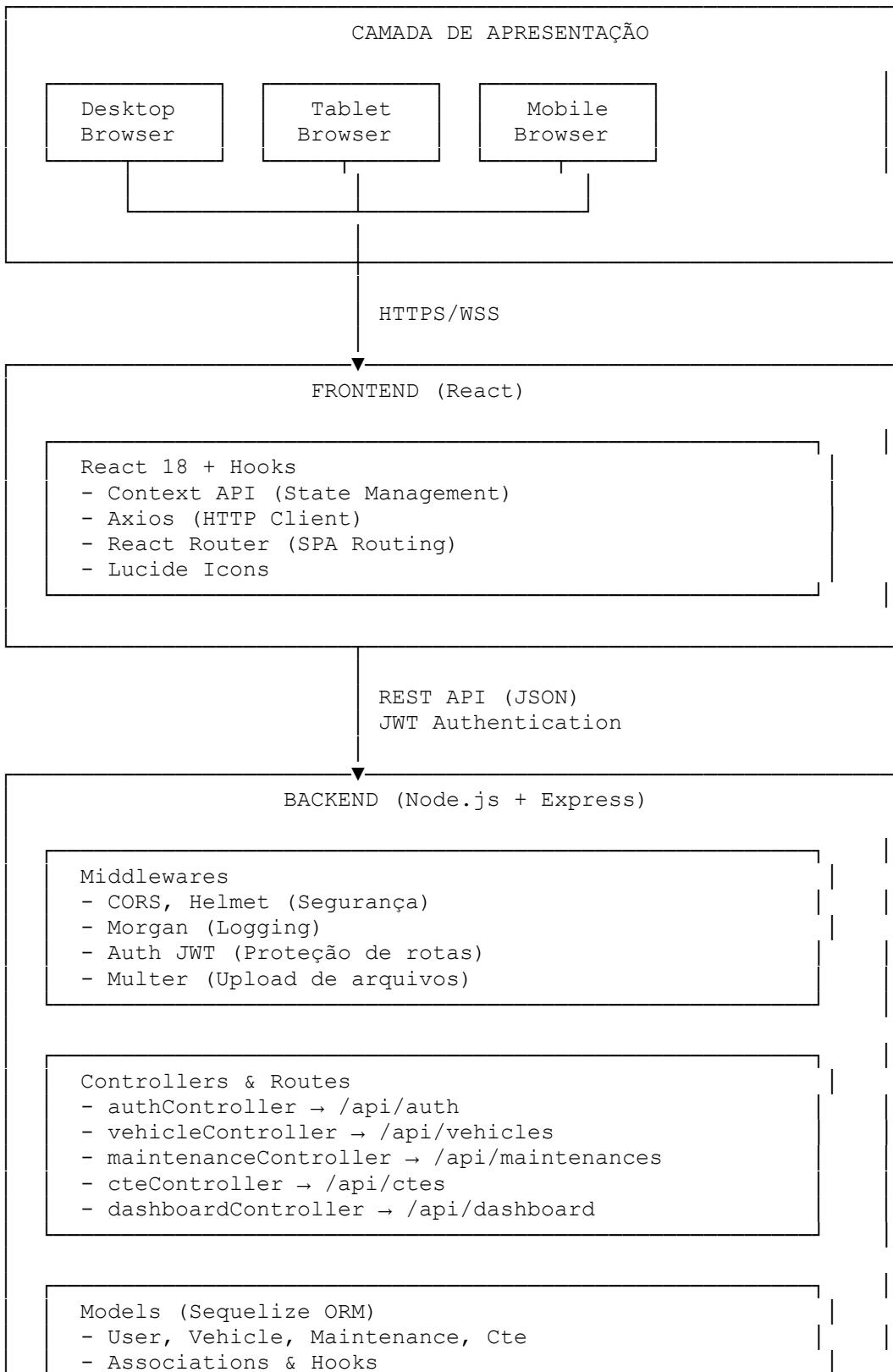
Banco de dados lento

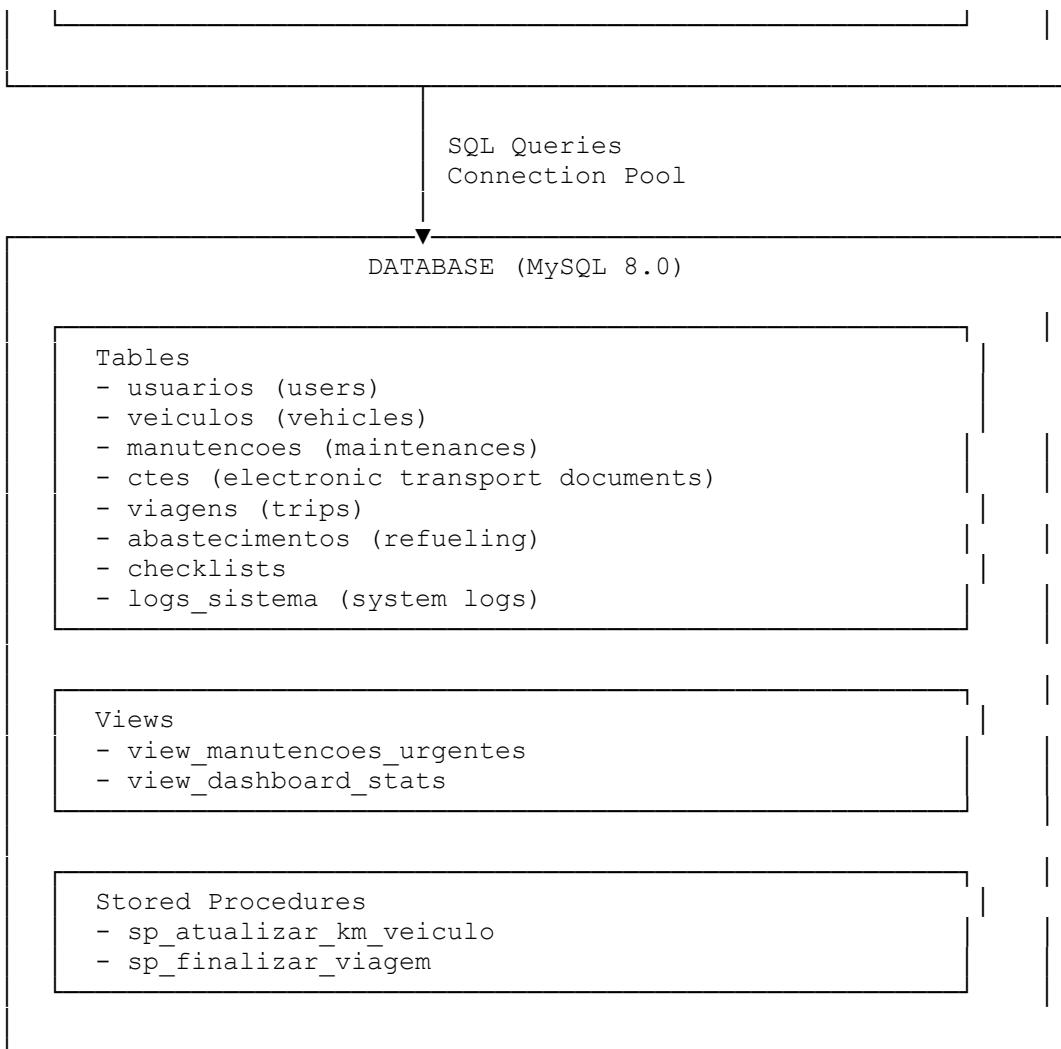
```
# Analisar queries lentas
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
# Adicionar:
# slow_query_log = 1
# slow_query_log_file = /var/log/mysql/slow.log
# long_query_time = 2

# Verificar queries lentas
sudo tail -f /var/log/mysql/slow.log
```

E Arquitetura do Sistema - AvaSystem

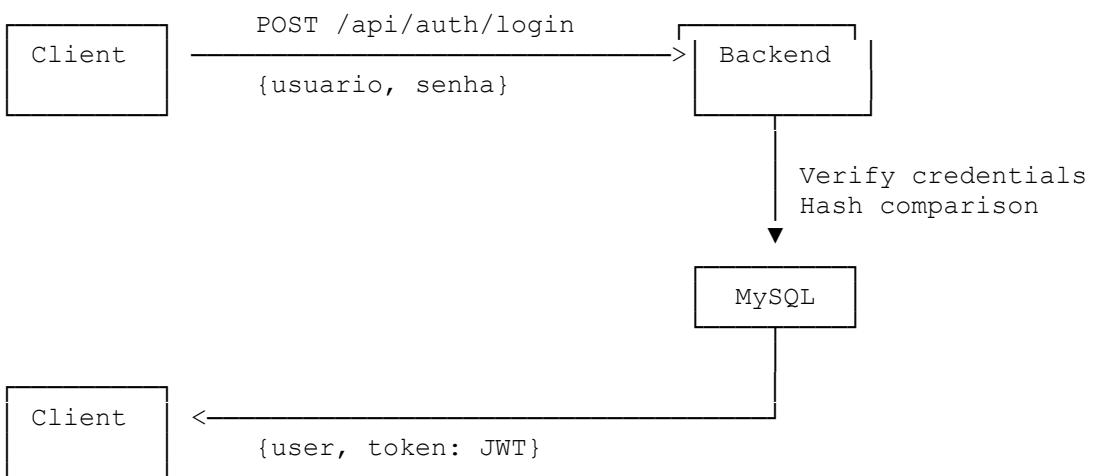
Visão Geral



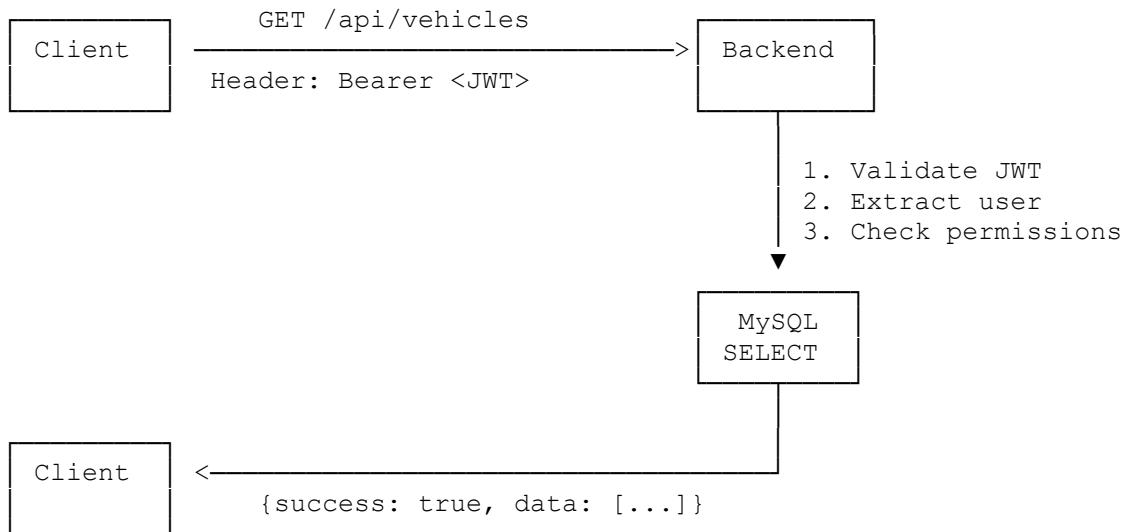


Fluxo de Dados

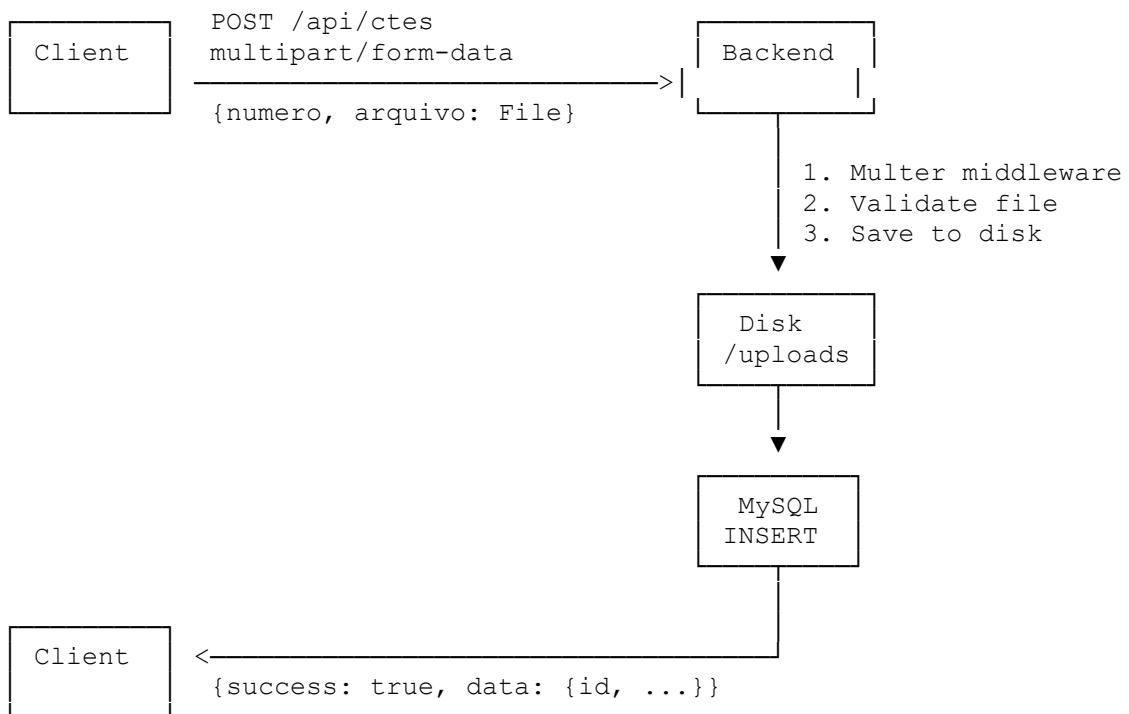
1. Autenticação



2. Requisição Protegida



3. Upload de Arquivo (CT-e)



Estrutura de Pastas Completa

```

projeto-sistema-logistica/
    └── backend/
        ├── config/
        │   └── database.js          # Sequelize config
        └── controllers/
            ├── authController.js    # Login, registro, JWT
            ├── vehicleController.js  # CRUD veículos
            ├── maintenanceController.js # CRUD manutenções
            └── cteController.js      # CRUD CT-e + upload

```

```
    └── dashboardController.js      # Estatísticas

  ├── middleware/
  │   ├── auth.js                  # JWT verify + authorize
  │   ├── upload.js                # Multer config
  │   ├── errorHandler.js          # Global error handler
  │   └── validators.js            # Express-validator

  ├── models/
  │   ├── User.js                  # Model + bcrypt hooks
  │   ├── Vehicle.js
  │   ├── Maintenance.js
  │   ├── Cte.js
  │   └── index.js                 # Associations

  ├── routes/
  │   ├── auth.routes.js
  │   ├── user.routes.js
  │   ├── vehicle.routes.js
  │   ├── maintenance.routes.js
  │   ├── cte.routes.js
  │   └── dashboard.routes.js

  ├── uploads/                    # Arquivos (gitignored)
  │   └── .gitkeep

  ├── .env                         # Config (gitignored)
  ├── .env.example
  ├── .gitignore
  ├── package.json
  └── server.js                   # Entry point

  └── sistema-logistica/          # Frontend React
      ├── public/
      │   ├── index.html
      │   ├── manifest.json
      │   └── robots.txt

      ├── src/
      │   ├── components/             # (futuro)
      │   │   ├── Dashboard.jsx
      │   │   ├── VehicleList.jsx
      │   │   ...
      │   |
      │   ├── context/               # (futuro)
      │   │   └── AuthContext.js
      │   |
      │   ├── services/              # (futuro)
      │   │   └── api.js
      │   |
      │   ├── App.css
      │   ├── App.js
      │   ├── index.css
      │   └── index.js               # Estilos principais
                                    # Componente principal
                                    # Entry point

      ├── .env
      ├── package.json
      └── README.md                 # Config frontend

  └── database-schema.sql          # Schema completo MySQL
  └── README.md                   # Documentação principal
```

```

├── DEPLOYMENT.md          # Guia de deploy
├── ARCHITECTURE.md        # Este arquivo
├── install.sh              # Script de instalação
├── backup.sh                # Script de backup
├── start-backend.sh        # Helper script
├── start-frontend.sh       # Helper script
└── start-all.sh            # Helper script (tmux)

```

Tecnologias e Versões

Backend

Tecnologia	Versão	Função
Node.js	18+	Runtime JavaScript
Express	4.18+	Framework web
MySQL	8.0+	Banco de dados relacional
Sequelize	6.35+	ORM
bcrypt	5.1+	Hash de senhas
jsonwebtoken	9.0+	Autenticação JWT
multer	1.4+	Upload de arquivos
express-validator	7.0+	Validação de inputs
helmet	7.1+	Segurança HTTP headers
cors	2.8+	CORS policy
morgan	1.10+	HTTP logging

Frontend

Tecnologia	Versão	Função
React	18+	Library UI
axios	1.6+	HTTP client
lucide-react	0.263+	Ícones

Segurança

Camadas de Segurança Implementadas

1. Autenticação JWT
 - o Tokens com expiração configurável
 - o Refresh tokens (implementação futura)
 - o Logout com invalidação de token
2. Autorização por Perfil
 - o Gerente: Acesso total
 - o Assistente: Leitura e escrita (sem deletar)
 - o Motorista: Acesso limitado aos seus dados
3. Proteção de Senhas
 - o Bcrypt com salt rounds = 10

- Nunca retorna senha nas respostas
 - Validação de força de senha
- 4. Proteção de Inputs**
- Express-validator em todas as rotas
 - Sanitização de dados
 - Proteção contra SQL Injection (Sequelize)
 - Proteção contra XSS
- 5. Headers de Segurança**
- Helmet.js configurado
 - CORS policy restritiva
 - Content Security Policy
- 6. Upload de Arquivos**
- Validação de tipo MIME
 - Limite de tamanho (5MB)
 - Nomes de arquivo sanitizados
 - Path traversal protection

Performance

Otimizações Implementadas

- 1. Banco de Dados**
- Índices em colunas de busca frequente
 - Índices compostos para queries complexas
 - Connection pooling (10 conexões)
 - Views pré-calculadas
- 2. API**
- Compressão gzip de respostas
 - Cache de queries (futuro: Redis)
 - Paginação de resultados (futuro)
 - Lazy loading de relações
- 3. Frontend**
- Code splitting (React.lazy - futuro)
 - Memoization com useMemo/useCallback
 - Debounce em buscas
 - Otimização de re-renders

Escalabilidade

Estratégias de Escalabilidade

- 1. Horizontal Scaling**
- Backend stateless (JWT)
 - Load balancer (Nginx)
 - Multiple backend instances (PM2 cluster)
- 2. Vertical Scaling**
- Otimização de queries
 - Caching estratégico
 - Database tuning

3. Database Scaling

- Read replicas (futuro)
- Sharding por região (futuro)
- Archive de dados antigos

Monitoramento

Métricas Importantes

1. Sistema

- CPU usage
- Memory usage
- Disk I/O
- Network traffic

2. Aplicação

- Request/response time
- Error rate
- Active connections
- Queue size

3. Banco de Dados

- Query execution time
- Slow query log
- Connection pool usage
- Table locks

Roadmap Futuro

Funcionalidades Planejadas

- [] Dashboard com gráficos (Chart.js/Recharts)
- [] Notificações em tempo real (Socket.io)
- [] Sistema de agendamento de manutenções automático
- [] Relatórios em PDF (jsPDF)
- [] Integração com APIs de rastreamento GPS
- [] App mobile nativo (React Native)
- [] Sistema de chat interno
- [] Integração com WhatsApp Business API
- [] OCR para leitura de documentos
- [] Machine Learning para previsão de manutenções
- [] Assinatura digital de documentos
- [] Integração com ERP/SAP

Melhorias Técnicas

- [] Implementar Redis para cache
- [] Adicionar testes unitários (Jest)
- [] Adicionar testes E2E (Cypress)
- [] CI/CD pipeline (GitHub Actions)
- [] Docker Compose para desenvolvimento

- [] Kubernetes para produção
 - [] GraphQL API (alternativa REST)
 - [] Microservices architecture
 - [] Event-driven architecture (RabbitMQ/Kafka)
-

Última atualização: 2025-10-29

Versão: 1.0.0

Autor: Sistema AvaSystem