TQS: Product specification report

Diogo Gaitas [73259], Giovanni Santos [115637], Rafael Semedo [115665]

v2025-06-09

1 Introduction

1.1 Overview of the project

O objetivo deste projeto é propor, conceptualizar e implementar uma aplicação em múltiplas camadas, baseada em frameworks corporativos, além de aplicar uma estratégia de Software Quality Assurance (SQA) e integrar pipelines de CI/CD ao longo de todo o ciclo de desenvolvimento.

Portanto, a **VoltUnity** será desenvolvida para aprimorar a gestão de carregamento de veículos elétricos em ambientes com múltiplos operadores. A plataforma permitirá que motoristas façam buscas e filtros de estações de carregamento próximas em tempo real, reservar vagas com antecedência para evitar filas, desbloqueiem o carregador diretamente pelo aplicativo e acompanhem seu histórico de consumo e a economia de CO_2 gerada.

Já o time de operações poderá cadastrar novas estações e gerenciar status de manutenção, monitorar tendências de uso e gerar relatórios diários, semanais ou mensais exportar dados para análises externas.

1.2 Known limitations

Durante o ciclo de desenvolvimento identificámos um conjunto de limitações que, por constrangimentos de tempo e de complexidade técnica, não foi possível ultrapassar na presente iteração.

Em primeiro lugar, a integração do Xray ao Jira para transformá-lo num centro de gestão de testes.

Em segundo lugar, a camada de mapa de geolocalização, prevista para consumir a API pública do OpenStreetMap/Leaflet com clustering de estações não avançou além de um *mock*.

Outra lacuna reside na ausência total de testes de performance: não chegámos a elaborar scripts k6 ou cenários-alvo, pelo que não existe evidência de latência, throughput ou resiliência sob carga

Em termos de autenticação, estava previsto disponibilizar uma página de login que distinguisse os perfis Admin e Driver (com autorização condicional na UI);

1.3 References and resources

Bibliotecas e Frameworks Utilizados

- **Spring Boot**
  Estrutura principal da aplicação backend, com suporte nativo para APIs REST, segurança, persistência e testes.
- **Spring Data JPA (Hibernate)**
  Abstração para acesso ao banco de dados (PostgreSQL), com suporte a consultas e relacionamentos.
- **PostgreSQL**
  Base de dados principal da aplicação.
- **JUnit 5**
  Estrutura padrão para todos os testes unitários e de integração.
- **Cucumber + Gherkin**
  Documentação dos endpoints REST para facilitar integração.
- **Mockito**
  Framework usado para simular dependências em testes unitários (por exemplo, repositórios ou serviços externos).
- **MockMvc (Spring Test)**
  Utilizado para testes de endpoints REST sem subir o servidor completo, garantindo verificação dos controllers de forma rápida e eficaz.
- **TestContainers**
  Utilizado para executar testes de integração com uma instância real de PostgreSQL dentro de containers Docker, garantindo confiabilidade e isolamento.

- **JaCoCo**
  Ferramenta usada para gerar relatórios de cobertura de testes. O objetivo definido foi atingir ≥ 80% de cobertura nos módulos principais.
- **GitHub Actions**
  Automatização do build, testes e verificação de qualidade a cada push ou pull request para as branch main e dev. Qualquer falha nos testes ou violação das regras de qualidade bloqueia o merge.
- **SonarQube (local ou SonarCloud)**
  Utilizado para análise estática do código, e no workflow CI para falhar o merge em casos de quality gate falhar.
- **Swagger/OpenAPI**
  Documentação dos endpoints REST para facilitar integração.
- **Docker**
  Orquestração de containers para backend e banco de dados.
- **React**
  Utilizado para construir a interface web do utilizador e admin.

2 Product concept and requirements

2.1 Vision statement

Motoristas dos carros elétricos enfrentam diversas barreiras ao planejar e executar recargas: estações com disponibilidade imprevisível, múltiplos operadores e interfaces díspares, filas inesperadas e ausência de transparência nos custos e impactos ambientais. Do lado dos operadores, há dificuldades para cadastrar estações de forma centralizada, monitorar uso em tempo real e gerar relatórios consolidados para tomada de decisão.

VoltUnity propõe uma plataforma única que unifica toda a jornada de recarga de um *E.V.*, do ponto de vista tanto do motorista quanto do operador, sem expor detalhes de implementação interna

Principais Funcionalidades do Sistema

- **Descoberta de Estações**
  - Busca geoespacial em tempo real, com filtros por localização, tipo de carregador e disponibilidade.
- **Reserva de Slot**
  - Agendamento antecipado de vagas, com prevenção de conflitos (double-booking) e confirmação instantânea.
- **Desbloqueio e Início de Carga**
  - Autenticação via token/mobile, solicitação ao carregador e confirmação de início de sessão diretamente no app.
- **Monitoramento e Relatórios de Consumo**

    - Dashboards que exibem kWh consumidos por sessão, semana e mês; cálculo de economia de CO_2 versus equivalente a combustíveis fósseis.
- **Modelos de Pagamento**
    - Fluxo “pay-per-use” ou planos de assinatura, com gestão de cartões, descontos e renovação automática.
- **Gestão de Estações (Operador)**
    - Cadastro de novas estações, sinalização de manutenção, relatórios de uso com exportação em *CSV*/*PDF*.
- **Integração Externa**
    - API pública documentada e sandbox de testes para desenvolvedores terceiros.

2.2 Personas and scenarios

Personas:

1. Emily – Motorista de um E.V. (Viajante Urbana)

- **Idade:** 35
- **Metas:** Localizar, reservar e efetuar carregamentos com eficiência em estações de veículos elétricos durante o trajeto diário.
- **Pontos problemáticos:** Disponibilidade inconsistente de carregadores, longos tempos de espera, preços pouco claros.

2. Raj – Operador de estação (gerente com conhecimento técnico)

- **Idade:** 42
- **Metas:** Monitorar o uso da estação, gerenciar a manutenção e garantir alto tempo de atividade.
- **Pontos problemáticos:** Falta de insights em tempo real, monitoramento manual, horários de menor movimento subutilizados.

3. Lea – Integrador terceirizado (Desenvolvedor de aplicativos)

- **Idade:** 29
- **Metas:** Criar serviços focados em *E*.V. através de APIs e dados da ***VoltUnity***.
- **Pontos problemáticos:** Ambientes de teste inacessíveis, documentação de API deficiente.

Scenarios:

US 1.1 – Buscar estações próximas

Cenário Principal

1. Motorista abre a tela “Buscar Estações”.

2. App obtém localização do GPS e mostra mapa centrado.
3. Motorista aplica filtros (tipo de carregador “CCS”, disponibilidade “sim”).
4. Sistema retorna lista de estações que atendem aos critérios.
5. Motorista seleciona uma estação no mapa; app exibe detalhes (nome, distância, status).

US 1.2 – Reservar slot de carregamento

Cenário Principal

1. Motorista visualiza horários disponíveis na estação selecionada.
2. Escolhe um horário de início (ex.: 15:00–15:30).
3. O sistema bloqueia esse slot e retorna a confirmação imediata.
4. Agenda aparece como “Reservado” na UI.

US 2.1 – Desbloquear ponto de carregamento

Cenário Principal

1. Motorista acessa “Minhas Reservas” e seleciona o slot ativo.
2. Clica em “Desbloquear” no app.
3. App envia requisição ao backend e à API da estação.
4. Carregador libera fisicamente o conector; app mostra “Carregamento iniciado” com contador de kWh.

US 3.1 – Escolher pagamento ou assinatura

Cenário Principal

1. Antes de confirmar a reserva, a app apresenta opções “Pay-per-use” e “Assinatura Mensal”.
2. Motorista escolhe “Assinatura” e revisa valor mensal e benefícios.
3. Clica em “Confirmar” e é redirecionado ao checkout.

US 4.1 – Marcar estação em manutenção

Cenário Principal

1. Operador/Admin faz login no “Portal de Operações".
2. Em “Gerenciamento de Estações”, ele ativa o toggle “Em Manutenção” na estação desejada.
3. O Status atualiza instantaneamente na API.

4. Motoristas veem a estação como “Indisponível” em buscas e reservas.

US 4.2 – Relatórios de tendência de uso

Cenário Principal

1. Operador acessa “Relatórios de Uso”.
2. Seleciona período (diário, semanal, mensal).
3. Dashboard renderiza gráfico de linha/trend.
4. Clica em “Exportar” e baixa CSV ou PDF.

US 5.1 – Visualizar economia de CO_2 (motorista)

Cenário Principal

1. Motorista abre “Impacto Ambiental” no app.
2. O sistema calcula CO_2 evitado com base no total de kWh e fator de emissão (e.g. 0,233 kg CO_2/kWh).
3. Exibe indicadores diários, semanais, mensais e gráfico comparativo com gasolina.

US 5.2 – Agregação de dados de CO_2 (admin)

Cenário Principal

1. Admin acessa o painel “Sustentabilidade”.
2. Selecionar filtros de localização e período.
3. O painel mostra gráficos de barras/linhas agregados (todas as estações).

2.3 Project epics and priorities

Epic 1: Pesquisa de estações e reserva de horários

User Story 1.1

Como motorista de veículo elétrico, quero procurar estações de carregamento disponíveis perto de mim para poder planejar minha rota de carregamento com eficiência.

- **Critérios de aceitação:**
    - O utilizador pode filtrar estações por localização, disponibilidade e tipo de carregador.
    - Resultados atualizados em tempo real.

User Story 1.2

Como motorista de veículo elétrico, quero reservar um horário de carregamento com antecedência para evitar tempos de espera.

- **Critérios de aceitação:**
    - A reserva mostra os horários disponíveis.
    - Reservas duplicadas são evitadas.

Epic 2: Rastreamento de carregamento e uso

User Story 2.1

Como motorista de veículo elétrico, quero desbloquear um ponto de carregamento com meu telefone para poder começar a carregar rapidamente.

- **Critérios de aceitação:**
    - Somente reservas válidas ou horários disponíveis podem ser usados para desbloquear.
    - Mensagem de confirmação após o desbloqueio.

User Story 2.2

Como motorista de veículo elétrico, quero visualizar meu histórico de consumo de energia para poder monitorar o uso e o custo.

- **Critérios de aceitação:**
    - O painel exibe kWh por sessão/semana/mês.
    - As economias de CO2 são mostradas em comparação com alternativas à gasolina.

Epic 3: Pagamentos e Assinaturas

User Story 3.1

Como motorista de veículo elétrico, quero escolher entre pagamento por uso e assinaturas para poder controlar meus gastos.

- **Critérios de aceitação:**
    - O modelo de pagamento pode ser selecionado antes da reserva.
    - As assinaturas são refletidas na fatura e permitem descontos/acesso ilimitado.

Epic 4: Gestão da Estação (Operador)

User Story 4.1

Como operador de estação, quero marcar uma estação como em manutenção para que os motoristas não reservem horários indisponíveis.

- **Critérios de aceitação:**
    - O painel do operador inclui botões de alternância de status.
    - As atualizações de status são refletidas instantaneamente em toda a plataforma.

User Story 4.2

Como operador de estação, quero relatórios de tendências de uso para poder planejar a manutenção e os preços de forma eficaz.

- **Critérios de aceitação:**
    - Estatísticas diárias/semanais/mensais visualizadas.
    - Relatórios CSV ou PDF exportáveis.

User Story 4.3

Como operador de estação, desejo registrar novas estações de carregamento na plataforma para que elas fiquem disponíveis para uso dos motoristas.

- **Critérios de aceitação:**
    - As estações recém-cadastradas aparecem na busca por motoristas após aprovação ou sincronização.
    - Formulário de inscrição com local, número de vagas e tipo de carregador.
    - Validação da entrada.

Epic 5: Impacto e Análise de CO2

User Story 5.1

Como motorista de veículo elétrico, quero ver quanto CO2 economizei ao longo do tempo para entender o impacto ambiental.

- **Critérios de aceitação:**
    - O painel mostra a economia de CO2 por dia, semana e mês.
    - Comparação com emissões equivalentes de gasolina.

User Story 5.2

*Como administrador do **VoltUnity**, quero dados agregados de CO2 e uso para mostrar o impacto da plataforma na sustentabilidade.*

- **Critérios de aceitação:**
    - Visuais e estatísticas sobre uso e benefícios ambientais.
    - Filtrar por local e período de tempo.

Epic 6*:* Integração com APIs externas

User Story 6.1

*Como integrador de serviços, quero ter acesso a uma API bem documentada com um ambiente de teste para poder desenvolver aplicativos que se integrem à plataforma **VoltUnity.***

- **Critérios de aceitação:**
    - Sandbox acessível com dados simulados.
    - Documentação clara da API com exemplos de chamadas.

3 Domain model

Conceitos de informação

O sistema ***VoltUnity EV Platform*** gerencia o processo de reserva de slots de carregamento para veículos elétricos, conectando motoristas a estações de carregamento. Os principais conceitos de informação no domínio são:

- **User (Utilizador/Motorista):**

  Representa uma pessoa que utiliza o sistema para reservar slots de carregamento. Possui atributos como nome, e-mail e papel (ex.: motorista). Um utilizador pode estar associado a múltiplas reservas e assinaturas, e pode estar vinculado a um ou mais carros por meio da relação *DriverCar*.

- **Car (Carro):**

  Representa o veículo elétrico do utilizador, com atributos como matrícula, modelo/marca, capacidade da bateria e economia de CO2. Um carro está associado a um utilizador por meio da relação *DriverCar*.

- **Station (Estação de Carregamento):**

  Representa uma estação física de carregamento, com atributos como nome, status (ativo ou inativo), localização (latitude e longitude), endereço, número total de slots e potência máxima. Uma estação pode conter vários pontos de carregamento (slots).

- **ChargerPoint (Ponto de Carregamento):**

  Representa um slot específico dentro de uma estação, com atributos como status (disponível, em uso, etc.). Um ponto de carregamento pertence a uma única estação e pode ser reservado por uma reserva.

- **Booking (Reserva):**

  Representa uma reserva de um ponto de carregamento feita por um utilizador. Inclui atributos como status (confirmada, etc.), horário de início e fim, e preço na reserva. Uma reserva está associada a um único ponto de carregamento e a um único utilizador.

- **Payment (Pagamento):**

  Representa uma transação de pagamento relacionada a uma sessão de carregamento ou reserva, com atributos como valor, método de pagamento, e referências a sessões de carregamento e reservas.

- **Subscription (Assinatura):**

  Representa um plano de assinatura de um utilizador, com atributos como nome do plano, datas de início e fim, e tipo de pagamento. Uma assinatura está vinculada a um único utilizador.

- **DriverCar**:

  Uma relação associativa entre User e Car, indicando quais carros estão associados a um motorista. Inclui referências a DriverID e CarID.

- **ChargingSession (Sessão de Carregamento):**

  Representa uma sessão de carregamento associada a uma reserva, com atributos como energia consumida (em kWh), horários de início e fim. Uma sessão está vinculada a uma única reserva.

Relações entre conceitos

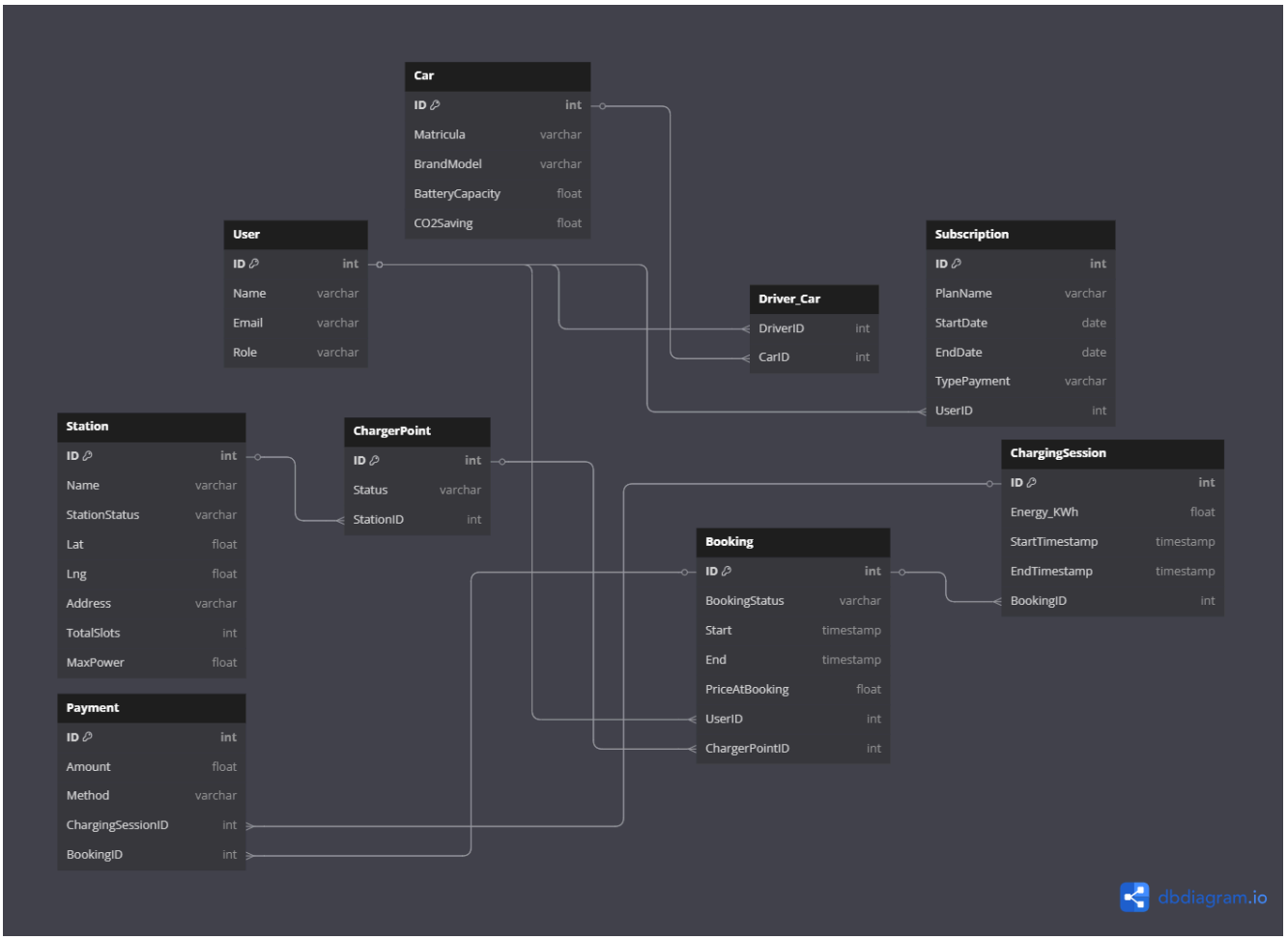
Cada utilizador pode possuir múltiplos veículos por meio da entidade “DriverCar”, que funciona como ponte entre um registro de utilizador e seus carros associados. Assim, um único utilizador consegue cadastrar diversos automóveis, enquanto cada carro cadastrado na plataforma pode estar vinculado a mais do que um utilizador, caso seja compartilhado ou co-possuído, configurando dessa forma uma relação de um para muitos tanto de utilizador quanto de carros para *DriverCar*.

Além disso, cada utilizador pode manter várias assinaturas ativas simultaneamente. Isso significa que, ao se cadastrar, um utilizador pode optar por planos diferentes e gerenciar todas elas sob seu perfil, exemplificando a relação **um-para-muitos** entre “User” e “Subscription”. De maneira análoga, um mesmo utilizador tem a possibilidade de efetuar múltiplas reservas ao longo do tempo; cada reserva realizada gera um registro na entidade “Booking”, consolidando também uma relação de **um-para-muitos** entre “User” e “Booking”.

No que diz respeito à infraestrutura de recarga, cada estação (“Station”) pode conter diversos pontos de carregamento (“ChargerPoint”), traduzindo-se noutra relação de **um-para-muitos**. Cada ponto de carregamento, por sua vez, pode ser reservado várias vezes ao longo do tempo, uma vez que diferentes utilizadores agendam horários para utilizar o mesmo equipamento, criando a relação de **um-para-muitos** entre “ChargerPoint” e “Booking”. Em relação às sessões propriamente ditas, cada reserva pode gerar uma sessão de carregamento única, mas não há obrigatoriedade de que toda reserva leve a uma sessão efetiva caracterizando uma relação de **um-para-um** ou de **um-para-zero-ou-um** entre “Booking” e “ChargingSession”.

Por fim, os pagamentos podem se vincular tanto a uma sessão de carregamento específica quanto a uma reserva em si. Em geral, quando o utilizador conclui uma sessão de carregamento, um único pagamento é registado para aquela sessão, estabelecendo uma relação de **um-para-um**. De forma semelhante, há cenários em que o pagamento é associado diretamente à reserva realizada, também numa relação de um para um ou de um para zero ou um, caso a reserva seja cancelada ou não resulte em pagamento. Dessa maneira, todas as entidades financeiras ligadas a “Payment” refletem relações diretas, restritas a uma única sessão de carregamento ou reserva por transação.

Modelo Lógico (Diagrama de Classes UML)



4 Architecture notebook

4.1 Key requirements and constrains

O projeto VoltUnity exige uma arquitetura modular, escalável e segura para atender múltiplos perfis de usuário (motorista, operador e integrador). As decisões de design foram guiadas por requisitos funcionais e não funcionais importantes:

Integrações externas

- APIs de geolocalização, pagamento e interoperabilidade (OCPI/OCPP)
- Requer isolamento de serviços e design desacoplado

Multiplataforma

- Acesso via web responsivo e possível app móvel
- APIs REST separadas do frontend

Alta concorrência

- Uso intenso em horários de pico
- Sistema precisa ser escalável e evitar reservas duplicadas

Isolamento de componentes

- Serviços como pagamentos e autenticação isolados por segurança e flexibilidade

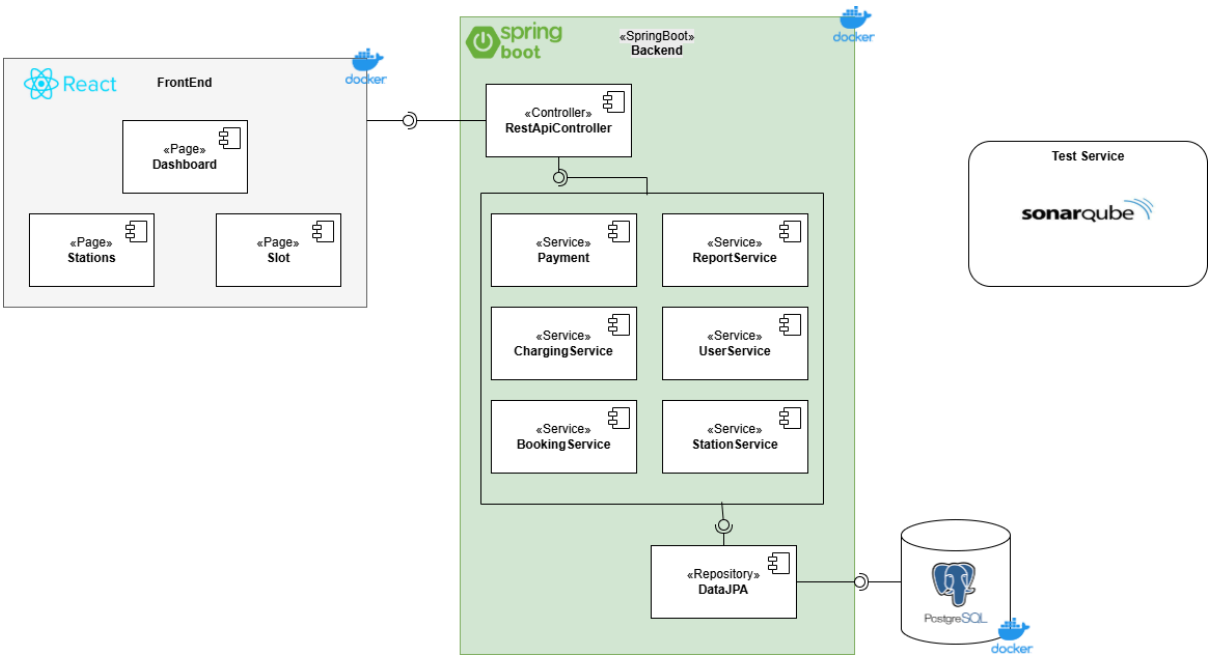
Características arquiteturais principais

- **Desempenho**: respostas rápidas em filtros e reservas
- **Escalabilidade**: modularidade e deploy independente
- **Disponibilidade**: tolerância a falhas parciais
- **Segurança**: autenticação JWT e controle de acesso por perfil
- **Extensibilidade**: fácil adição de novos recursos e integrações

4.2 Architecture view

Arquitetura Lógica da Solução

A solução foi concebida segundo um estilo de arquitetura em camadas, no qual cada bloco de responsabilidade exerce uma função bem definida, facilitando manutenção, testes e evolução. O diagrama a seguir apresenta uma visão lógica dos principais módulos, sem fazer referência direta a tecnologias ou detalhes de implantação:



Presentation Layer:

- User Interface (UI): Responsável pela interação com usuários finais (motoristas e administradores). Permite criar, consultar e cancelar reservas, gerenciar pagamentos, e visualizar relatórios.
- API Gateway: Ponto de entrada para requisições externas, roteando-as para os módulos apropriados.

Business Logic Layer:

- **Reservation Management**: Gerência de reservas, validando horários e disponibilidade de slots.
- **Slot Management**: Administra slots de carregamento, incluindo status (ex.: disponível, em uso).
- **Station Management**: Gerencia informações sobre estações de carregamento (ex.: localização, tipos de carregadores).
- **Payment Processing**: Processa transações financeiras relacionadas a reservas.
- **User Management**: Gerencia perfis dos utilizadores e autenticação/autorização.
- **Report Generation**: Produz relatórios.
- **Security Enforcement**: Garante segurança e controle de acesso.
- **Charging Session Management**: Controla sessões de carregamento ativas.
- **Maintenance Scheduling**: Gerência de manutenção de estações.
- ***Subscription Management***: Administra planos de assinatura para utilizadores.

Data Access Layer:

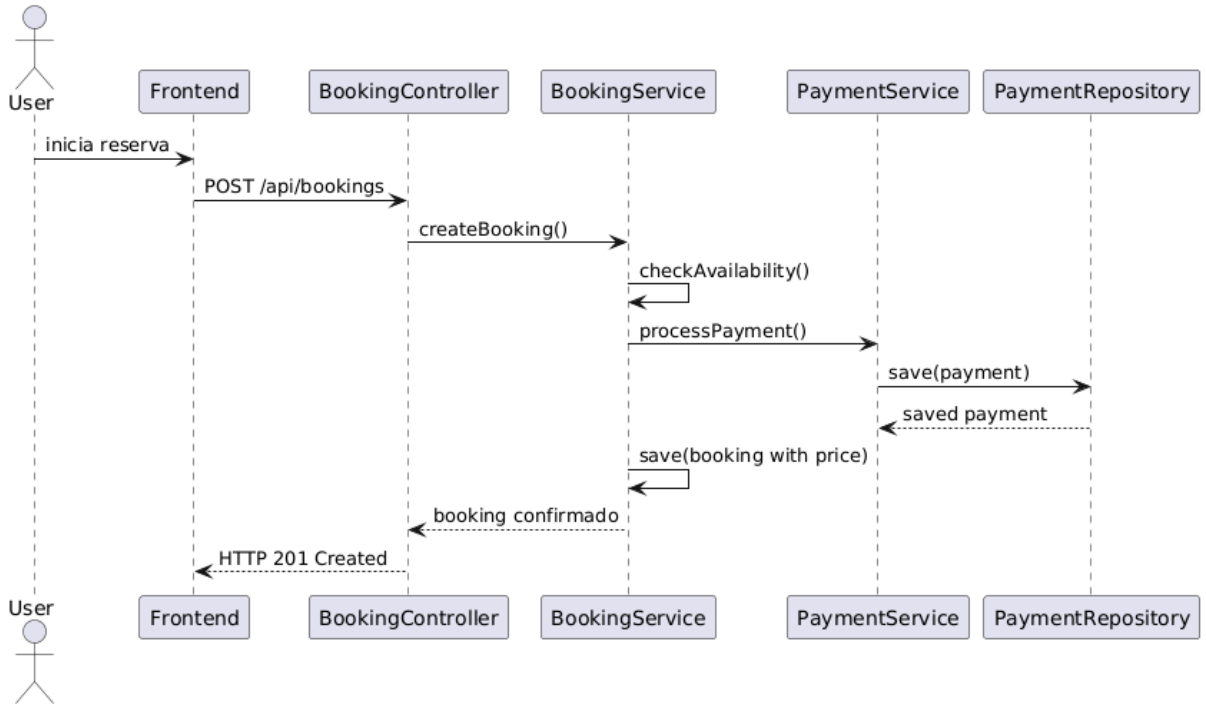
- Data Repository: Abstrai o acesso aos dados, fornecendo operações CRUD para entidades como reservas, slots, estações, utilizadores, etc.

Além disso, o padrão **MVC** (Model-View-Controller) é implícito na estrutura, onde:

- View: Representada pela UI.
- Controller: Implícita na API Gateway e nos pontos de entrada dos módulos de Business Logic.
- Model: Representado pelos módulos de lógica de negócios e Data Repository.

Diagrama de Sequências

A interação entre os módulos segue um fluxo síncrono via chamadas REST (no frontend) e chamadas internas entre serviços (no backend). O seguinte diagrama de sequência exemplifica a criação de uma reserva com pagamento:



O utilizador inicia uma reserva através da interface web, que envia os dados ao *BookingController* no backend. O *BookingService* valida a disponibilidade de slots e, se houver disponibilidade, aciona o *PaymentService* para registar o pagamento (simulado como concluído). Após confirmação do pagamento, a reserva é persistida com o valor pago associado (priceAtBooking) e status “confirmed”. Finalmente, a resposta de sucesso é enviada ao frontend, permitindo ao utilizador visualizar a confirmação na interface.

4.3 Deployment view

A solução é entregue em Docker Compose com os seguintes serviços:

- **springboot_app**: imagem *openjdk:21-jdk-slim*, exposta internamente na porta 8080;
- **postgres**: imagem postgres:15-alpine, porta 5432, volume persistente postgres-data;
- **react_app**: interface Web, servida em http://deti-tqs-16.

A análise de qualidade de código é feita em SonarCloud (pipeline CI). Cada pull request na branch main dispara a GitHub Actions; após merge bem-sucedido, a pipeline executa

docker compose up -d no servidor, garantindo zero-downtime para a aplicação em produção.

5 API for developers

Tendo em conta que diferentes perfis de utilizador (Condutor, Administrador) acedem ao mesmo backend, a **API REST** foi desenhada com limites de responsabilidade bem definidos.

Os endpoints seguem o padrão */{role}/{recurso}* ou, quando o recurso é comum, apenas */{recurso}*, sendo a autorização garantida por scopes JWT.

A documentação é gerada automaticamente com Springdoc-OpenAPI (Swagger 3). Através das anotações **@Operation** e **@ApiResponses** nos controllers, a UI interactiva é criada em cada build, mantendo-se sempre actualizada.

A documentação completa pode ser consultada em:

- Ambiente local: http://localhost:8080/swagger-ui/index.html
- Servidor de integração: http://deti-tqs-16:8080/swagger-ui/index.html#/

maintenance-controller

- PUT /api/v1/stations/{stationId}/pricing
- GET /api/v1/maintenance/plan

station-controller

- PUT /api/v1/stations/{id}
- PUT /api/v1/stations/{id}/status
- GET /api/v1/stations
- POST /api/v1/stations
- GET /api/v1/stations/{id}/slots
- GET /api/v1/stations/search

slot-controller

- PUT /api/v1/slots/{id}/status
- GET /api/v1/slots
- POST /api/v1/slots

reservation-controller

- PUT /api/v1/reservations/{id}/cancel
- POST /api/v1/reservations

charging-session-controller

- GET /api/v1/charging-sessions/{id}
- PUT /api/v1/charging-sessions/{id}
- PUT /api/v1/charging-sessions/{id}/cancel
- GET /api/v1/charging-sessions
- POST /api/v1/charging-sessions
- GET /api/v1/charging-sessions/user/{userId}

subscription-controller

- GET /api/v1/subscriptions
- POST /api/v1/subscriptions
- GET /api/v1/subscriptions/users/{userId}

payment-controller

- GET /api/v1/payments
- POST /api/v1/payments
- GET /api/v1/payments/users/{userId}

charging-controller

- POST /api/v1/charges/unlock

car-controller

- GET /api/v1/cars
- POST /api/v1/cars
- GET /api/v1/cars/users/{userId}
- DELETE /api/v1/cars/{carId}

report-controller

- GET /api/v1/reports/users/{userId}/co2
- GET /api/v1/reports/sustainability
- GET /api/v1/reports/consumption
- GET /api/v1/reports/co2