TQS: Product specification report

Ana Alexandra Antunes [876543], Bruno Bernardes [876543]

v2025-12-15

[1 Introduction 1](#_Toc213750830)

[1.1 Overview of the project 1](#_Toc213750831)

[1.2 Known limitations 1](#_Toc213750832)

[1.3 References and resources 2](#_Toc213750833)

[2 Product concept and requirements 2](#_Toc213750834)

[2.1 Vision statement 2](#_Toc213750835)

[2.2 Personas and scenarios 2](#_Toc213750836)

[2.3 Project epics and priorities 2](#_Toc213750837)

[3 Domain model 3](#_Toc213750838)

[4 Architecture notebook 3](#_Toc213750839)

[4.1 Key requirements and constrains 3](#_Toc213750840)

[4.2 Architecture view 3](#_Toc213750841)

[4.3 Deployment view 3](#_Toc213750842)

[5 API for developers 3](#_Toc213750843)

[This report is the main source of technical documentation on the project, clarifying the functional scope and architectural choices. Provide concise, but informative content, **allowing other software engineers to understand the product**.

Tips on the expected content placed along the document are meant to be removed!

You may use English or Portuguese; do not mix.]

# Introduction

## Overview of the project

<contextualize the objectives of this project assignment in the scope of the TQS course>

<introduce your application/product: brief overview of the solution concept. What is it good for? For whom? Introduce the name of the product if it has one>

O presente projeto foi desenvolvido no âmbito da unidade curricular de Testes e Qualidade de Software (TQS) do curso de Engenharia Informática da Universidade de Aveiro, no ano letivo de 2025/2026. O principal objetivo abrange temas como estratégias de teste (unitários, integração, sistema, aceitação), automação de testes, análise estática de código, métricas de qualidade e práticas de integração e entrega contínuas (CI/CD).

O MineForNow é uma plataforma digital inovadora que facilita o aluguer peer-to-peer (P2P) de veículos privados subutilizados. A solução visa criar uma ponte entre proprietários de veículos que desejam rentabilizar os seus carros quando não estão a utilizá-los e pessoas que necessitam de um veículo por períodos específicos, oferecendo uma alternativa mais flexível e económica às empresas tradicionais de rent-a-car.

Público-alvo:

- Proprietários de Veículos (Owners): Indivíduos que possuem um ou mais veículos e desejam obter rendimento extra disponibilizando-os para aluguer temporário.

- Inquilinos (Renters): Utilizadores que necessitam de um veículo para viagens, deslocações ou situações pontuais, procurando opções diversificadas a preços competitivos.

- Administradores: Gestores da plataforma responsáveis pela supervisão das operações, aprovação de novos proprietários e manutenção da qualidade do serviço.

Principais Funcionalidades:

- Sistema de autenticação e gestão de perfis com diferentes níveis de acesso

- Registo e gestão de veículos com fotos e especificações técnicas detalhadas

- Motor de pesquisa avançado com filtros por localização, datas, tipo de veículo, preço e características

- Sistema de reservas com gestão de disponibilidade

- Painel de controlo (dashboard) personalizado para cada tipo de utilizador

- Backoffice administrativo para gestão da plataforma

Proposta de Valor:

O MineForNow diferencia-se por combinar a conveniência de uma plataforma digital moderna com a economia colaborativa, permitindo que veículos parados gerem rendimento para os seus proprietários enquanto oferecem aos inquilinos uma maior variedade de opções a preços mais acessíveis do que as soluções tradicionais de aluguer de viaturas.

A arquitetura da aplicação segue uma abordagem moderna de desenvolvimento web, com um backend em Spring Boot (Java) que expõe uma API RESTful, e um frontend desenvolvido em React, comunicando através de endpoints seguros com autenticação JWT. Todo o projeto foi desenvolvido com foco na qualidade, incorporando testes em múltiplas camadas e análise contínua de código.

## Project limitations and know issues

 <explain the known limitations, especially the **features that were planned/expected but not implemented** (and why…)

**To be reviewed and completed by the end of the project** >

> **Nota:** Esta secção será revista e completada até ao final do projeto.

Sistema de Upload de Imagens

- Planeado: Upload direto de fotografias dos veículos pelos proprietários

- Estado atual: Utilização de URLs de imagens externas como placeholder

- Motivo: Priorização das funcionalidades core do MVP e complexidade adicional de gestão de ficheiros/storage

Sistema de Pagamentos Real

- Planeado: Integração com gateway de pagamentos (ex: Stripe, PayPal)

- Estado atual: Simulação de pagamentos sem processamento real

- Motivo: Fora do âmbito académico, requereria configuração de contas merchant e tratamento de dados financeiros sensíveis

Sistema de Avaliações e Reviews

- Planeado: Permitir que renters avaliem veículos e owners após cada reserva

- Estado atual: Não implementado

- Motivo: Funcionalidade relegada para iterações futuras por limitações de tempo

Notificações Push/Email

- Planeado: Alertas automáticos para confirmações de reserva, lembretes e comunicações

- Estado atual: Não implementado

- Motivo: Requereria integração com serviços externos de email/push notifications

Chat/Mensagens entre Utilizadores

- Planeado: Sistema de comunicação direta entre owners e renters

- Estado atual: Não implementado

- Motivo: Complexidade de implementação de sistema de mensagens em tempo real

Limitações Técnicas Conhecidas

- Sessões de utilizador: Tokens JWT expiram após período definido, requerendo novo login

- Responsividade: Algumas páginas podem não estar totalmente otimizadas para dispositivos móveis de menores dimensões

- Dados de demonstração: A aplicação inclui dados mock para fins de teste e demonstração

- Concorrência de reservas: O sistema não implementa locks otimistas para reservas simultâneas do mesmo veículo

Melhorias Futuras Identificadas

- Implementação de sistema de favoritos para veículos

- Histórico detalhado de reservas com exportação

- Dashboard de analytics para owners

- Suporte multi-idioma (internacionalização)

- Integração com APIs de geolocalização para cálculo de distâncias

## References and resources

<document the key components (e.g.: libraries, web services) or key references (e.g.: blog post) used that were really helpful and certainly would help other students pursuing a similar work>

> **Nota:** Tentar compor esta merda

Backend (Java/Spring)

| Componente | Versão | Descrição |

|------------|--------|-----------|

| Spring Boot | 3.3.6 | Framework principal para desenvolvimento da API REST |

| Spring Security | - | Autenticação e autorização com suporte a JWT |

| Spring Data JPA | - | Abstração para acesso a dados com Hibernate |

| H2 Database | - | Base de dados em memória para testes |

| Lombok | - | Redução de código boilerplate (getters, setters, builders) |

| JJWT | 0.11.5 | Biblioteca para geração e validação de tokens JWT |

Frontend (React)

| Componente | Versão | Descrição |

|------------|--------|-----------|

| React | 18.2.0 | Biblioteca principal para construção da interface |

| Vite | 7.2.6 | Build tool e dev server de alta performance |

| React Router DOM | 6.20.0 | Gestão de rotas e navegação SPA |

| Tailwind CSS | 3.3.6 | Framework CSS utility-first para estilização |

| Lucide React | 0.294.0 | Biblioteca de ícones SVG |

| Axios | 1.13.2 | Cliente HTTP para comunicação com a API |

| date-fns | 2.30.0 | Manipulação de datas em JavaScript |

Ferramentas de Testes e Qualidade

| Ferramenta | Versão | Propósito |

|------------|--------|-----------|

| JUnit 5 | - | Framework de testes unitários para Java |

| Mockito | - | Mocking de dependências em testes unitários |

| Cucumber | 7.18.0 | Testes de aceitação BDD (Behavior-Driven Development) |

| Playwright | 1.48.0 (Backend) / 1.57.0 (Frontend) | Automação de browser para testes E2E |

| Selenium | 4.26.0 | Alternativa para testes de interface web |

| WebDriverManager | 6.1.0 | Gestão automática de drivers de browser |

| JaCoCo | 0.8.12 | Análise de cobertura de código |

| SonarQube | Cloud | Análise estática de código e métricas de qualidade |

| Xray | 1.0.0 | Integração de testes com Jira |

DevOps e Infraestrutura

| Ferramenta | Propósito |

|------------|-----------|

| Docker | Containerização da aplicação |

| GitHub Actions | CI/CD pipeline para build e testes automatizados |

| PostgreSQL | Base de dados de produção |

| Maven | Gestão de dependências e build do backend |

| npm | Gestão de dependências do frontend |

Recursos e Referências Úteis

Documentação Oficial

- [Spring Boot Reference](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/) - Documentação completa do Spring Boot

- [Spring Security](https://docs.spring.io/spring-security/reference/index.html) - Guia de autenticação e autorização

- [React Documentation](https://react.dev/) - Documentação oficial do React

- [Tailwind CSS](https://tailwindcss.com/docs) - Referência de classes utility

- [Playwright](https://playwright.dev/docs/intro) - Guia de testes E2E

- [Cucumber](https://cucumber.io/docs/guides/) - Documentação de BDD

Tutoriais e Artigos Recomendados

- [Baeldung - Spring Security JWT](https://www.baeldung.com/spring-security-oauth-jwt) - Implementação de autenticação JWT

- [Baeldung - Testing in Spring Boot](https://www.baeldung.com/spring-boot-testing) - Estratégias de teste

- [Testing JavaScript](https://testingjavascript.com/) - Boas práticas de testes em JS/React

- [Cucumber + Spring Boot Integration](https://cucumber.io/docs/guides/10-minute-tutorial/) - Setup de testes BDD

Ferramentas de Desenvolvimento

- IntelliJ IDEA / VS Code - IDEs utilizadas no desenvolvimento

- Postman / Insomnia - Testes manuais de API REST

- Chrome DevTools - Debugging de frontend

# Product concept and requirements

## Vision statement

<functional (black-box) description of the application: Which is the high-level/business problem being solved by your system? Which are the key features you promise to address it?>

<if needed, clarify what was planned/expected to be included but was changed to a different approach/concept >

<optional: how is your system different or similar to other well-known products?>

<optional: additional details on the process for the requirements gathering and selection (how did we developed the concept? Who helped us with the requirements? etc)>

## Personas and scenarios

<“Personas are fictional people. They have names, likenesses, clothes, occupations, families, friends, pets, possessions, and so forth. They have age, gender, ethnicity, educational achievement, and socioeconomic status. They have life stories, goals and tasks. Scenarios can be constructed around personas, but the personas come first. They are not ‘agents’ or ‘actors’ in a script, they are people. Photographs of the personas and their workplaces are created and displayed. […] It is to obtain a more powerful level of identification and engagement that enable design, development, and testing to move forward more effectively”. Adapted from Grudin, J. and Pruitt, J., 2002, June. Personas, participatory design and product development: An infrastructure for engagement. In Proc. PDC (Vol. 2).

Sample personas: [secção 4.1, neste artigo](http://www.mdpi.com/1424-8220/18/4/1285) (open access)] >

<Develop one or more representative scenarios for each persona. You don’t need to include all possible details. Pick the main scenarios, related to the core value of the system.>

<The scenarios tell the story of the Personas in their lives, doing their daily/professional activities that are relevant to find the points of contact with the system under specification.

Scenarios are somewhat similar to use cases (they have a goal and tell a story), but, unlike use cases, they capture a larger process, with activities that may not use the software. Scenarios don’t require a “template”, like the usual use cases description.>

Sample: [secção 4.2 neste artigo](http://www.mdpi.com/1424-8220/18/4/1285) (open access)] >

## Project epics and priorities

**[**Apresentar um plano indicativo para a implementação incremental da solução ao longo de várias iterações/releases, explicando as funcionalidades a atingir por [*epics*](https://www.atlassian.com/agile/project-management/epics-stories-themes) ]

# Domain model

<which information concepts will be managed in this domain? How are they related?>

<use a logical model (UML classes) to explain the concepts of the domain and their attributes, not a entity-relationship relational database model>

# Architecture notebook

## Key requirements and constrains

<**Identify issues that will drive the choices for the architecture** such as: Are there hardware dependencies that should be isolated from the rest of the system? Does the system need to function efficiently under unusual conditions? Are there integrations with external systems? Is the system to be offered in different user-interfacing platforms (web, mobile devices, big screens,…)?

For a more systematical approach:

* Note the collection of [Architectural Characteristics](https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/ch04.html) the software architect should be aware
* [Identify architectural characteristics](https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/ch05.html) that are relevant for your project (will drive the key design decisions). Note the [case study](https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/ch05.html" \l "ch-identifying-ss) and the explicit characteristics related to users and extensibility. This will support later non-functional tests.

## Architecture view

→ Discuss architecture planned for the software solution: what are the main building blocks? [include a diagram](https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/ch23.html) (a logical view, such as a package or block diagram). Avoid implementation technology or deployment references, but protocols/standards can be included.

→ refer to the [architecture style](https://learning.oreilly.com/library/view/fundamentals-of-software/9781098175504/part02.html) applied, if any

explain how the identified modules will interact. Use a sequence diagram to clarify the interactions along time, when needed

→ discuss more advanced app design issues: integration with Internet-based external services, data synchronization strategy, distributed workflows, push notifications mechanism, distribution of updates to distributed devices, etc.>

## Deployment view (production configuration)

[Explain the planned organization of the solution in terms of production configuration (deployment). Note the implementation technologies in the diagram, e.g.: place the PostgreSQL symbol in the Database, etc.]. Indicate the existence of containers (Docker), IP addresses, and ports, etc.

This part will be completed when deployments are actually in place.

# API for developers

[Explain the API organization and main collections in general terms. Details/documentation of methods should be included in a hosted API documentation solution, such as Swagger, Postman documentation, or included in the development itself (e.g., Maven site).

Be sure to use [best practices for REST Api design](https://learning.oreilly.com/library/view/mastering-rest-apis/9798868803093/html/617304_1_En_7_Chapter.xhtml). Keep minda REST API applies a resource-oriented design (APIs should be designed around resources, which are the key entities your application exposes, not actions)