

1. Capa

Nome do Estudante: Leonardo Raye e Wellerson Meredyk

Matrícula: Wellerson- 1325715. Leonardo Raye- 1325718

Disciplina: Qualidade de Software

Professor: Diego Sauter Possamai

Data de Entrega: 25/03/2025

2. Introdução

2.1 Apresentação do Projeto

Este documento apresenta o planejamento da construção do software "**LWCar**", uma aplicação web para locação de veículos. O objetivo principal é garantir uma experiência simples, rápida e confortável para facilitar o aluguel do carro desejado.

2.2 Justificativa

Muitos indivíduos e empresas enfrentam dificuldades na locação de veículos de forma ágil e eficiente. O **LWCar** visa resolver esse problema oferecendo uma plataforma intuitiva, com funcionalidades que facilitam a reserva, gestão e disponibilidade de veículos, garantindo praticidade e economia de tempo.

2.3 Público-Alvo

O software será utilizado por qualquer pessoa ou empresa que queira alugar veículos, do básico até veículos de luxo.

3. Modelagem do Software

3.1 Requisitos Funcionais

- RF01: Cadastro de usuários (clientes e administradores).
- RF02: Cadastro de veículos disponíveis para aluguel.
- RF03: Consulta de veículos por categoria, preço e disponibilidade.
- RF04: Reserva e confirmação de aluguel de veículos.
- RF05: Geração de relatórios sobre os alugueis realizados.
- RF06: Controle de status dos veículos (disponível, alugado, em manutenção).
-

3.2 Requisitos Não Funcionais

- RNF01: Interface responsiva para dispositivos móveis e desktop.
- RNF02: Armazenamento dos dados em um banco de dados.
- RNF03: Autenticação e segurança dos dados dos usuários.
- RNF04: Boa performance no carregamento das páginas.
- RNF05: Código bem estruturado para facilitar futuras manutenções.

•

3.3 Casos de Uso

Caso de Uso 1: Cadastro de Usuário

Ator: Cliente ou Administrador

Fluxo Principal:

1. O usuário acessa a página de cadastro.
2. Insere nome, e-mail, senha e outros dados necessários.
3. Confirma o cadastro.
4. O sistema armazena as informações e exibe uma mensagem de sucesso.

Caso de Uso 2: Cadastro de Veículo

Ator: Administrador

Fluxo Principal:

1. O administrador acessa a seção de cadastro de veículos.
2. Insere informações como marca, modelo, ano, categoria e status (disponível, alugado, manutenção).
3. Salva as informações.
4. O sistema registra o veículo no banco de dados.

Caso de Uso 3: Consulta de Veículos

Ator: Cliente

Fluxo Principal:

1. O cliente acessa a página de busca.
2. Insere filtros como categoria, preço e disponibilidade.
3. O sistema exibe uma lista de veículos correspondentes.
4. O cliente seleciona um veículo para ver mais detalhes.

Caso de Uso 4: Reserva de Veículo

Ator: Cliente

Fluxo Principal:

1. O cliente escolhe um veículo disponível.
2. Seleciona a data de retirada e devolução.
3. Confirma a reserva.
4. O sistema atualiza o status do veículo para "Alugado".

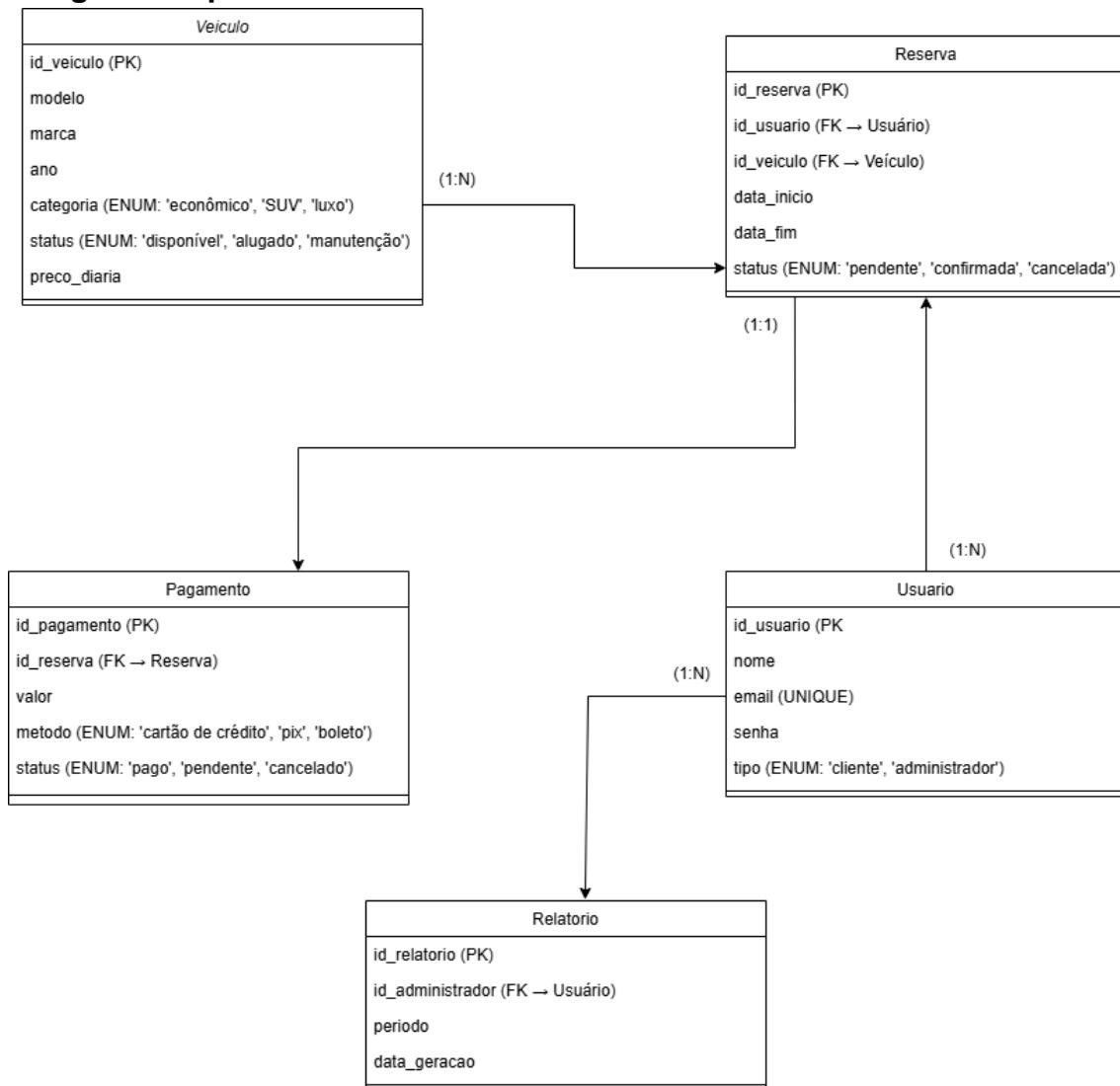
Caso de Uso 5: Geração de Relatórios

Ator: Administrador

Fluxo Principal:

1. O administrador acessa a seção de relatórios.
2. Escolhe um período específico.
3. O sistema gera um relatório detalhado em PDF com os alugueis realizados.

3.4 Modelagem de Dados (Diagrama Entidade-Relacionamento - ER) à Diagrama Aqui



3.5 Arquitetura do Sistema

O sistema será baseado em uma arquitetura monolítica de três camadas (MVC - Model, View, Controller), permitindo escalabilidade e modularidade.

- Model: Contém as regras de negócio e interação com o banco de dados.
- View: Interface do usuário desenvolvida em html e css.
- Controller: Implementado em Node.js, responsável pela lógica de requisições e respostas.
-

3.6 Planejamento da Infraestrutura

- Banco de Dados: MySQL

- Back-end: Node.js
- Front-end: html e css + Bootstrap
- Autenticação: JWT (JSON Web Token)
- Hospedagem: Localhost
- Monitoramento: Prometheus + Grafana

3.7 Diagramas

-> Diagramas UML completos, Wireframes de Interface, Diagrama de Casos de Uso

Diagrama UML

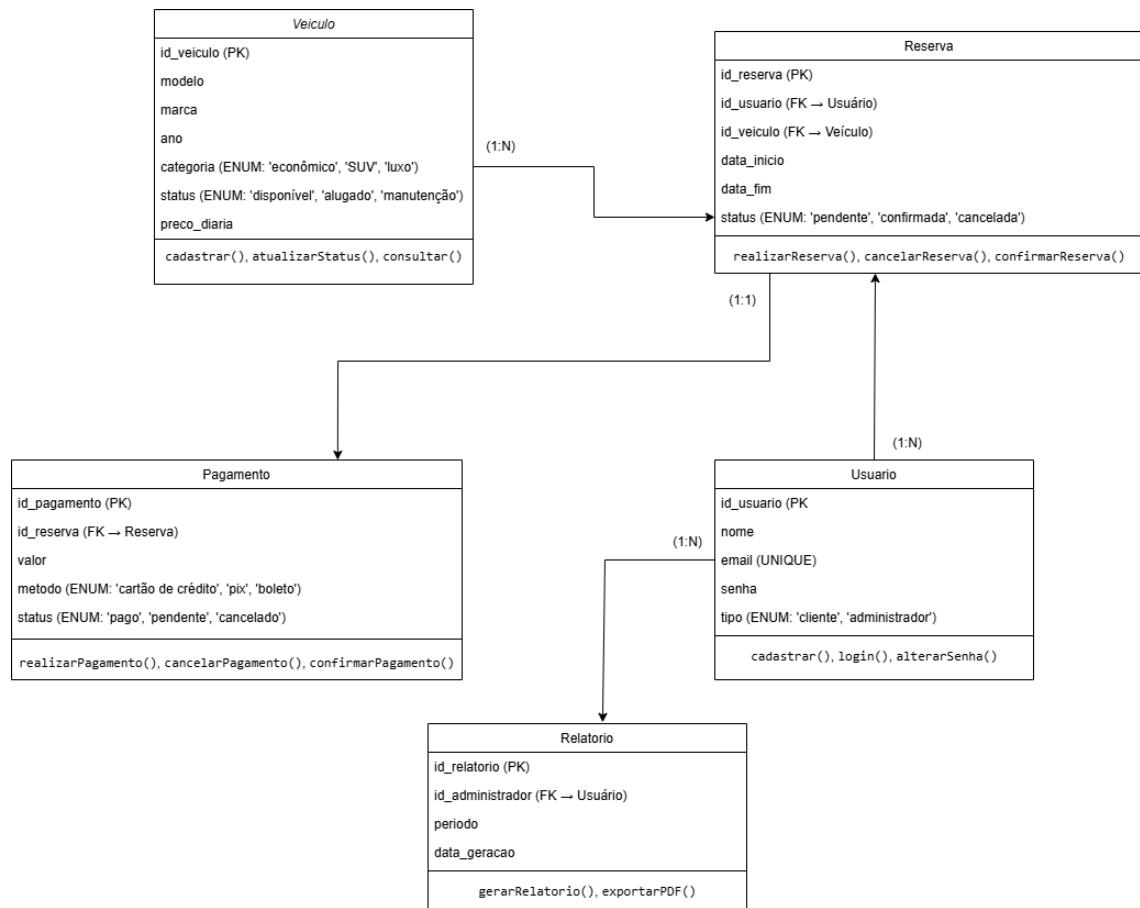
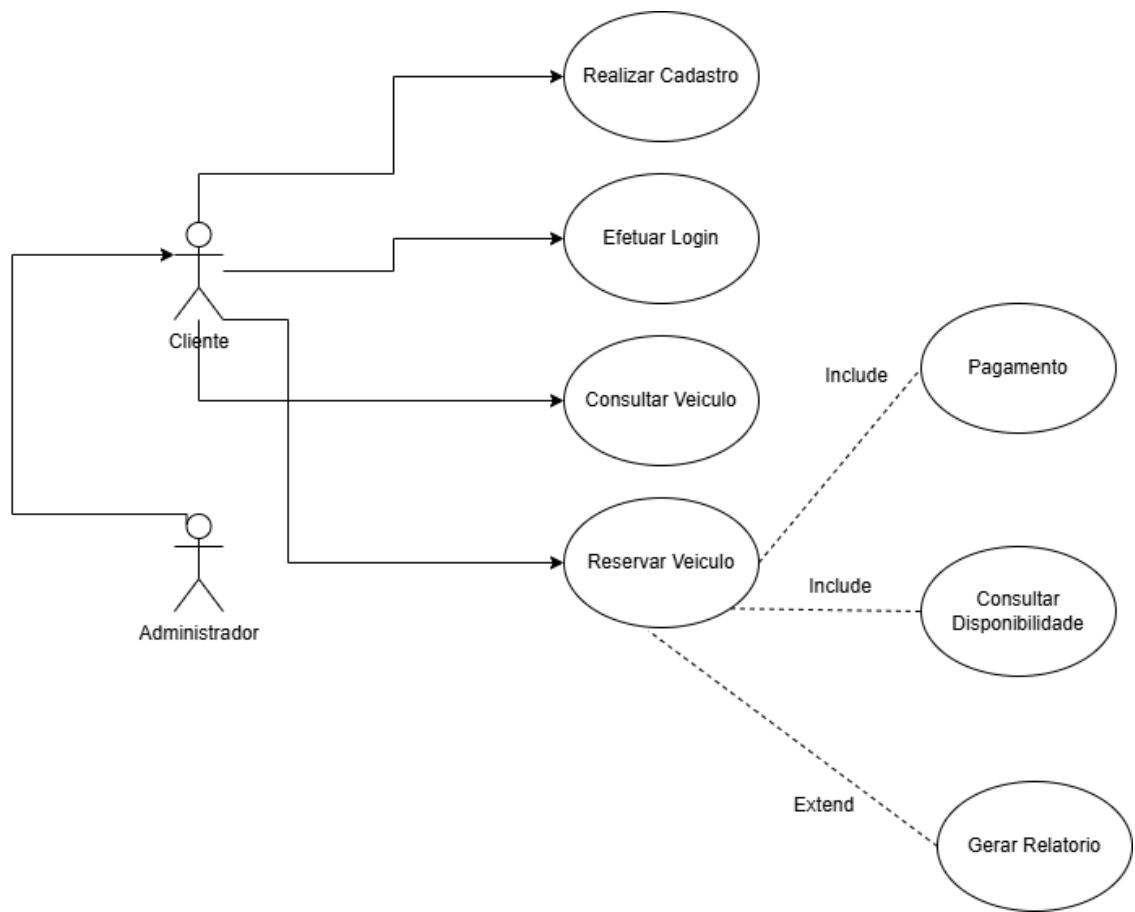


Diagrama de casos de uso



Wireframe de interface



4. Planejamento da Qualidade do Software

4.1 Normas e Padrões Utilizados

Para garantir a qualidade do software, serão seguidas normas e padrões reconhecidos internacionalmente:

- ISO/IEC 25010: Define critérios de qualidade para produtos de software, garantindo adequação funcional, confiabilidade e eficiência.
- ISO 9001: Padroniza a gestão da qualidade no desenvolvimento de software.
- ISO/IEC 12207: Especifica os processos do ciclo de vida do software.
- OWASP Top 10: Estabelece diretrizes para segurança em aplicações web.
- IEEE 829: Normatiza a documentação de testes de software.

4.2 Planejamento de Testes

O planejamento de testes será realizado para garantir que todas as funcionalidades do sistema estejam implementadas corretamente e sigam os padrões de qualidade definidos. Os seguintes tipos de testes serão aplicados:

4.2.1 Testes Unitários

- Objetivo: Garantir que cada componente do sistema funcione corretamente de forma isolada.
- Como será feito: Utilização do Jest para testar funções do backend e do frontend.
- Medição: Cobertura mínima de 80% do código-fonte.
- Validação: Resultados dos testes automatizados devem estar dentro dos critérios de aceitação definidos

4.2.2 Testes de Integração

- Objetivo: Avaliar a comunicação entre os diferentes módulos do sistema.
- Como será feito: Utilização do Postman para requisições REST API e Jest com supertest para validação de endpoints.
- Medição: Testes devem validar todas as rotas de API e possíveis interações entre serviços.
- Validação: Testes automatizados devem ser executados antes de cada implantação no ambiente de produção.

4.2.3 Testes de Usabilidade

- Objetivo: Avaliar a experiência do usuário e identificar dificuldades na navegação.
- Como será feito: Testes A/B com usuários reais e ferramentas como Hotjar para análise de interações.
- Medição: Taxa de sucesso das tarefas executadas por usuários em ambiente controlado.
- Validação: 80% dos usuários devem concluir tarefas comuns sem dificuldades.

4.2.4 Testes de Segurança

- Objetivo: Garantir que a aplicação está protegida contra ameaças e ataques comuns.
- Como será feito: Uso de Burp Suite, análise manual, como fuzzing e análise de

parâmetros de entrada.

- Medição:** Número de vulnerabilidades críticas detectadas deve ser zero antes da liberação do sistema.
- Validação:** Resultados das auditorias de segurança devem ser analisados e aplicadas correções conforme necessário.

- **Inserir aqui os casos de teste**

4.3 Métricas de Qualidade e Validação

Para garantir a qualidade do software ao longo do desenvolvimento, serão monitoradas e analisadas as seguintes métricas:

4.3.1 Métricas de Código

- Complexidade de Código (Manutenibilidade): A complexidade do código deve ser mantida abaixo de 10 para promover uma maior facilidade de manutenção, usando CodeClimate ou Codacy como ferramenta de análise de complexidade.
- Cobertura de Testes (Cobertura de Funções): Pelo menos 85% das funções/métodos devem ser cobertos por testes automatizados, incluindo testes unitários e de integração, monitorado via codecov.
- Legibilidade de Código: Métodos e funções não devem ter mais do que 40 linhas de código para garantir uma legibilidade e manutenção ideais, analisados por CodeClimate.

4.3.2 Métricas de Desempenho

- Tempo de Carregamento: O tempo de carregamento das páginas deve ser inferior a 1 segundo para a maior parte dos usuários, medido por Google Lighthouse.
- Consumo de Recursos (CPU): O uso de CPU deve ser monitorado para garantir que o sistema não exceda 70% de utilização durante períodos de carga, com monitoramento feito por Datadog ou Prometheus.

- Escalabilidade: O sistema deve ser capaz de suportar 5000 usuários simultâneos sem apresentar degradação no desempenho, medido com Apache Benchmark (ab) ou Artillery.

4.3.3 Métricas de Usabilidade

- Taxa de Conclusão de Tarefas: A 95% dos usuários devem ser capazes de concluir suas tarefas sem encontrar problemas, monitorado por FullStory ou Crazy Egg.
- Tempo de Tarefa Eficiente: O tempo médio necessário para completar uma tarefa deve ser inferior a 45 segundos, monitorado via Google Analytics.
- Taxa de Abandono de Tarefas: O número de usuários que abandonam a tarefa antes de completá-la deve ser inferior a 5%, com dados coletados através de Hotjar.

4.3.4 Métricas de Segurança

- Taxa de Vulnerabilidades Resolvidas: 100% das vulnerabilidades críticas detectadas durante o ciclo de desenvolvimento devem ser corrigidas antes da liberação final. Monitorado por SonarQube e OWASP Dependency-Check.
- Tempo de Resposta para Falhas: Falhas de segurança devem ser corrigidas em até 24 horas após sua descoberta, com controle via Jira ou Trello.
- Auditoria de Código de Segurança: Realização de auditorias automáticas semanalmente para garantir a detecção de falhas contínuas, utilizando ferramentas como Snyk ou OWASP Dependency-Check.
- .

4.4 Ferramentas Utilizadas para Garantia da Qualidade

Para facilitar o monitoramento e validação dos critérios de qualidade, serão utilizadas as seguintes ferramentas:

- SonarQube: Avaliação da qualidade do código, cobertura de testes e complexidade.
- Codecov: Ferramenta para monitorar a cobertura de código e a qualidade dos testes.

- Google Lighthouse: Medição do desempenho e acessibilidade das páginas web.
- Datadog/Prometheus: Monitoramento do uso de recursos como CPU e memória.
- FullStory/Crazy Egg: Análise de comportamento do usuário e usabilidade do sistema.
- OWASP ZAP/Snyk: Ferramentas para análise e auditoria de segurança.
- Hotjar/Crazy Egg: Análise de usabilidade e mapeamento de interações do usuário.

4.5 Diagramas

-> Inserir Diagramas aqui

❖ Diagrama de Planejamento da Qualidade

- Um diagrama hierárquico que mostra a estrutura de qualidade do software, incluindo testes, métricas e ferramentas utilizadas.
- Pode ser representado como um gráfico de fluxo conectando objetivos de qualidade → testes → métricas → ferramentas.

❖ Diagrama de Processo de Testes

- Mostra o fluxo de atividades no processo de testes.
- Exemplo: Etapas desde a criação dos casos de testes até a execução, relatório e correção de erros.

❖ Diagrama de Rastreabilidade de Requisitos e Testes

- Relaciona cada requisito funcional e não funcional com os testes associados.
- Ajuda a garantir que todos os requisitos possuem testes para validação.

❖ Diagrama de Fluxo de Defeitos

- Representa o ciclo de vida de um bug desde a sua identificação até a resolução.
- Mostra quem é responsável por cada etapa do fluxo.

4.6 Tabelas

❖ 4.6.1 Tabela de Critérios de Aceitação

Requisito	Critério de Aceitação	Métrica de Avaliação
-----------	-----------------------	----------------------

Cadastro de Usuário	O sistema deve permitir o cadastro de usuários com sucesso.	Taxa de sucesso em testes unitários ≥ 95%
Consulta de Veículos	O sistema deve retornar veículos correspondentes ao filtro.	Teste de integração para validar a busca
Reserva de Veículo	O sistema deve permitir a reserva de um veículo disponível.	Taxa de sucesso em testes de funcionalidade ≥ 90%
Geração de Relatórios	O sistema deve gerar um relatório detalhado de aluguéis.	Teste de unidade para gerar relatórios completos

❖ 4.6.2 Tabela de Planejamento de Testes

Tipo de Teste	Ferramenta	Ambiente	Responsável
Teste Unitário	Jest	_local	Equipe de Dev
Teste de integração	Postman Jest (supertest)	Servidor de desenvolvimento	QA
Teste de Performance	JMeter	Servidor de Staging	QA
Teste de segurança	OWASP ZAP	Servidor de Staging	QA

❖ 4.6.3 Matriz de Rastreabilidade de Testes

D do Requisito	Descrição	Teste Associado	Status
RF-001	Cadastro de usuário	TU-001, TI-001	Em execução
RF-002	Cadastro de veículo	TU-002, TI-002	Em execução
RF-003	Consulta de veículos	TU-003, TI-003	Em execução
RF-004	Reserva de veículo	TU-004, TI-004	Em execução

RF-005 Geração de relatórios TU-005, TI-005 Em execução

❖ 4.6.4 Tabela de Defeitos e Correções

D	Descrição	Severidade	Status	Responsável
BUG-001	Erro ao salvar cadastro de veículo	Alta	Em andamento	Dev Backend
BUG-002	Falha ao gerar relatório de aluguel	Média	Em andamento	Dev Backend
BUG-003	Carregamento lento de página de consulta	Baixa	Em andamento	Dev Frontend
BUG-004	Erro ao salvar tarefa	Alta	Em andamento	Dev Backend

❖ 4.6.5 Tabela de Métricas de Qualidade

Métrica	Como Medir	Ferramenta	Meta
Cobertura de Código	% de linhas testadas	SonarQube	≥ 80%
Tempo de Resposta	Tempo médio de carregamento	JMeter	< 2s
Taxa de Sucesso	Percentual de sucesso nos testes automatizados	Jest	< 95%
Número de Vulnerabilidades	Contagem de falhas de segurança	OWASP ZAP	0

Esses **diagramas e tabelas** ajudam a estruturar melhor o **planejamento de qualidade**, garantindo que os requisitos e validações estejam **claramente definidos e rastreados**.

CONCLUSÃO

O desenvolvimento do sistema LWCar foi planejado com o objetivo de oferecer uma plataforma eficiente e intuitiva para a locação de veículos, atendendo tanto a clientes quanto a administradores. Durante a elaboração deste documento, foram definidos os principais requisitos funcionais e não funcionais, garantindo que o software atenda às necessidades do público-alvo de forma segura, escalável e de fácil utilização.

A modelagem do software foi estruturada com base na arquitetura MVC (Model-View-Controller), utilizando tecnologias modernas para backend e frontend, além de um banco de dados robusto para armazenar as informações essenciais do sistema. O planejamento da infraestrutura e das métricas de qualidade assegura que o sistema será desenvolvido seguindo boas práticas de segurança, desempenho e manutenibilidade.

Além disso, um rigoroso plano de qualidade foi definido, incluindo normas e padrões reconhecidos internacionalmente, como ISO/IEC 25010, ISO 9001 e OWASP Top 10. A realização de testes unitários, de integração, usabilidade e segurança garantirá que o sistema funcione corretamente antes da sua implantação, minimizando falhas e otimizando a experiência do usuário.

Por fim, as tabelas de rastreabilidade, métricas de qualidade e planejamento de testes consolidam um processo estruturado para o desenvolvimento do LWCar, garantindo que a plataforma atinja um alto nível de confiabilidade e eficiência. Com esse planejamento, espera-se entregar um software de qualidade, contribuindo para a inovação e otimização no setor de locação de veículos.