PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

SISTEMAS DE INFORMAÇÃO

SARA DE ALMEIDA PASOLINI

REQUISITOS PARA O RELATÓRIO DE FUNCIONAMENTO

Documentação de Software

BETIM

2025

SARA DE ALMEIDA PASOLINI

REQUISITOS PARA O RELATÓRIO DE FUNCIONAMENTO

Documentação de Software

Documentação de testes de funcionamento à disciplina Interação Tecnologia e Desenvolvimento de Sistemas, do curso de Sistemas de Informação , terceiro período, na Pontifícia Universidade Católica de Minas Gerais.

BETIM

2025

Resumo

Segundo Pressman, a documentação de software serve para comunicar ideias e requisitos, garantindo um entendimento compartilhado entre a equipe e as partes interessadas, além de ser um componente essencial do próprio software, que engloba instruções, dados e documentos. Nesta documentação busco explicitar, com clareza e assertividade, o resultado de testes e funcionamento do sistema produzido para a disciplina.

Palavras-chave: testes; documentação; sistema de software; funcionamento.

SUMÁRIO

1 INTRODUÇÃO

O objetivo principal desta documentação é apresentar o desenvolvimento e funcionamento de um sistema de locadora de veículos, destacando sua relevância para a automação de processos de aluguel, cadastro de clientes e controle de veículos.

A elaboração deste sistema justifica-se pela necessidade crescente de otimizar a gestão de locadoras de veículos, garantindo maior eficiência e precisão no gerenciamento de informações sobre clientes, veículos disponíveis e histórico de locações. Nesse contexto, torna-se fundamental avaliar critérios como organização de dados, controle de classes, interface intuitiva para administração da locadora , a fim de identificar os pontos fortes e limitações do sistema desenvolvido.

O objetivo geral desta análise é compreender a implementação prática de um sistema informatizado para locadora de veículos, permitindo o gerenciamento de veículos disponíveis para aluguel, o registro de clientes e locações, bem como o controle de datas de retirada e devolução e do valor total do aluguel. Além disso, busca-se oferecer uma interface intuitiva para administração da locadora e documentar tecnicamente as principais funcionalidades e escolhas tecnológicas adotadas.

O sistema foi desenvolvido utilizando as tecnologias C# 9.0+, ASP.NET Core, Entity Framework, SQL Server Express, Swagger para documentação de API, GitHub para controle de versão e Automapper. Suas funcionalidades incluem o cadastro de veículos e clientes, registro de locações, controle de quilometragem, oferecendo assim uma solução integrada e eficiente para o gerenciamento da locadora.

2 REQUISITOS FUNCIONAIS

Os requisitos de software constituem elementos essenciais no processo de desenvolvimento de sistemas, uma vez que definem as funcionalidades, restrições e características que um software deve possuir para atender às necessidades e expectativas dos usuários. Conforme Pressman (2011), os requisitos de software podem ser classificados em requisitos funcionais, que descrevem o que o sistema deve fazer, e requisitos não funcionais, que especificam propriedades e restrições do sistema, tais como desempenho, usabilidade e segurança.

No presente trabalho prático de locadora de veículos, os requisitos funcionais identificados para o sistema incluem:

1. Cadastro de veículos: O sistema deve permitir a inclusão de novos veículos, registrando informações como modelo, fabricante, ano e quilometragem atual, cor e placa.

2. Edição de veículos: O sistema deve possibilitar a atualização de informações de veículos já cadastrados.

3. Exclusão de veículos: O sistema deve permitir a remoção de veículos do cadastro, garantindo a integridade dos dados relacionados a locações.

4. Cadastro de clientes: O sistema deve permitir o registro de novos clientes, com dados como ID do cliente, nome , CPF e e-mail..

5. Edição de clientes: O sistema deve possibilitar a atualização das informações dos clientes cadastrados.

6. Exclusão de clientes: O sistema deve permitir a remoção de clientes, mantendo a consistência das locações já registradas.

7. Registro de locações: O sistema deve possibilitar a criação de locações, registrando veículo, cliente, data de retirada, data de devolução e valor total do aluguel.

8. Registro de devoluções: O sistema deve permitir a atualização do status do veículo após a devolução, incluindo quilometragem final e confirmação do pagamento.

9. Consulta de histórico de locações: O sistema deve permitir a pesquisa e visualização do histórico de locações por cliente ou veículo.

10. Registro de pagamento: O sistema deve permitir registrar um pagamento utilizando o ID do cliente e o ID da locação correspondente.

11. Edição de pagamento: O sistema deve possibilitar a edição de pagamentos já registrados, permitindo ajustes em valores ou forma de pagamento.

12. Exclusão de pagamento: O sistema deve permitir a exclusão de registros de pagamento, desde que o mesmo não comprometa o histórico financeiro da locadora.

13. Consulta de pagamentos por cliente: O sistema deve permitir consultar todos os pagamentos realizados por um cliente, utilizando o ID do cliente como parâmetro de busca.

14. Listagem geral de pagamentos: O sistema deve possibilitar visualizar todos os pagamentos realizados, facilitando o controle e auditoria financeira

15. Cadastro de fabricante: O sistema deve permitir o registro de fabricantes, informando nome e país de origem.

16. Edição de fabricante: O sistema deve possibilitar editar os dados de fabricantes já cadastrados.

17. Exclusão de fabricante: O sistema deve permitir excluir fabricantes do sistema, mantendo a integridade dos dados vinculados a veículos.

18. Busca de fabricante por ID: O sistema deve permitir buscar fabricantes por seu identificador único (ID).

19. Busca de fabricante por nome: O sistema deve permitir localizar fabricantes pelo nome, facilitando o gerenciamento de marcas cadastradas.

20. Filtros de busca e consulta por ID: O sistema deve oferecer filtros de pesquisa em todas as rotas, permitindo localizar veículos, clientes e locações, pagamentos , através de seus identificadores únicos.

2.1 Casos de Uso Principais

Para melhor compreensão das funcionalidades do sistema, destacam-se os seguintes casos de uso principais:

- UC01: Registrar novo cliente – Permite o cadastro completo de informações do cliente no sistema.

- UC02: Registrar veículo – Possibilita o cadastro de veículos disponíveis para locação, incluindo dados essenciais como modelo, fabricante e ano.

- UC03: Registrar locação – Realiza o registro de uma nova locação, associando cliente e veículo, datas de retirada e devolução, e valor do aluguel.

- UC04: Registrar devolução – Registra quilometragem final e confirmação de pagamento.

- UC05: Consultar histórico de locações – Permite visualizar todas as locações de um cliente ou veículo, possibilitando relatórios detalhados.

- UC06: Editar informações de cliente ou veículo – Atualiza dados já cadastrados, mantendo o histórico e integridade do sistema.

- UC07: Excluir cliente ou veículo – Remove registros do sistema, garantindo que locações anteriores sejam preservadas de forma consistente.

- UC08: Filtrar registros por ID – Localiza rapidamente clientes, veículos ou locações por identificadores únicos.

- UC09: Gerar relatórios de locação – Produz relatórios de veículos disponíveis, locações ativas e histórico de clientes.

- UC10 : Buscar Registros por ID em Todas as Rotas – processo de busca de registros específicos em todas as rotas do sistema, utilizando o identificador único (ID) de cada entidade cadastrada. O sistema deve permitir ao usuário realizar consultas diretas e obter informações detalhadas sobre clientes, veículos, fabricantes, alugueis e pagamentos.

Esses casos de uso garantem que o sistema atenda às necessidades da locadora de veículos, oferecendo funcionalidades completas e integradas, com foco em eficiência, confiabilidade e facilidade de operação.

3 ARQUITETURA DO SISTEMA

A Arquitetura Limpa (Clean Architecture), proposta por Robert C. Martin (Uncle Bob), é um padrão arquitetural que visa desenvolver sistemas com alto grau de reutilização de código, coesão, independência de tecnologias e facilidade de testes, promovendo uma separação clara entre regras de negócio e detalhes de implementação.

A arquitetura do sistema de locadora de veículos foi desenvolvida com base no padrão MVC (Model–View–Controller), amplamente utilizado em aplicações ASP.NET Core por sua capacidade de organizar o código de forma modular, escalável e de fácil manutenção. Esse modelo tem como principal objetivo separar as responsabilidades entre as diferentes camadas da aplicação, permitindo que a lógica de negócio, a interface e o controle de fluxo funcionem de maneira independente e coesa.

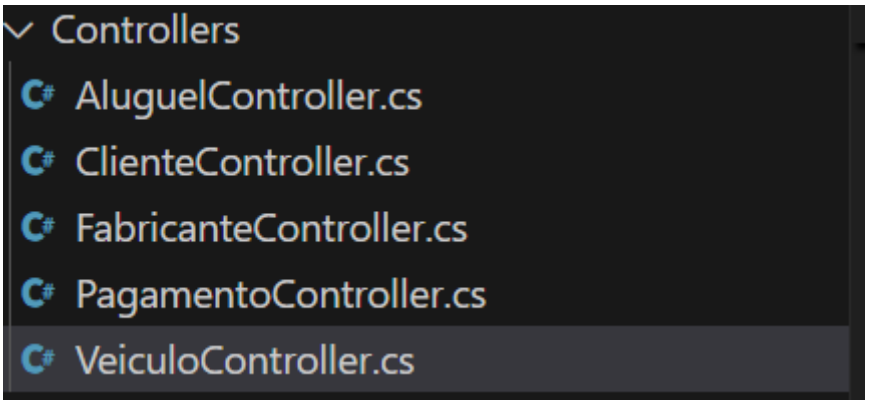
Estrutura Geral do Sistema

A aplicação foi estruturada em **camadas** interconectadas, cada uma desempenhando uma função específica dentro do sistema:

- **Models (Modelo):**
  Esta camada é responsável pela representação dos dados e das regras de negócio. Contém as classes que representam as entidades principais do sistema — como *Cliente*, *Veículo*, *Fabricante*, *Locação* e *Pagamento*.
  Além disso, esta camada inclui os DTOs (Data Transfer Objects), utilizados para transferir dados entre as camadas de forma segura, e as entidades do Entity Framework, que fazem o mapeamento entre as classes e as tabelas do banco de dados SQL Server Express.

- **Controller (Controlador):**
  A camada de controle tem a função de intermediar a comunicação entre o usuário e o sistema.
- Os controladores são responsáveis por receber as requisições HTTP (GET, POST, PUT, DELETE), processá-las e encaminhá-las à camada de modelo. Após o processamento, retornam as respostas adequadas ao cliente (usuário ou sistema consumidor da API).

- Exemplo de controladores presentes no sistema:



Fonte: Captura de tela do autor (2025).

- **View (Visão):**
  Embora o sistema seja uma API Web, a camada de visão é representada pelas respostas JSON retornadas ao cliente, permitindo integração com diferentes interfaces gráficas ou sistemas externos.
  A documentação das rotas e operações foi implementada utilizando o Swagger, o que facilita a visualização e teste das funcionalidades do sistema de forma interativa.

- **Camada de Dados (Data Access Layer):**
  Essa camada é composta pelo Entity Framework Core, que atua como um *ORM (Object-Relational Mapper)*, responsável por gerenciar o acesso e a persistência dos dados no banco SQL Server Express.
  Ela abstrai a manipulação direta de SQL, permitindo que o desenvolvedor trabalhe com objetos e classes em vez de comandos manuais, garantindo segurança e eficiência nas operações de CRUD (Create, Read, Update, Delete).

- **Camada de Serviços (Service Layer):**
  Responsável pela **lógica de negócio** e **validações específicas**, essa camada contém os métodos que aplicam as regras antes que as informações sejam salvas ou atualizadas no banco. Por exemplo, antes de registrar uma locação, o sistema valida se o veículo está disponível, se o cliente está ativo e se há quilometragem compatível.

Divisão de Responsabilidades entre as Camadas

- A seguir, é apresentada a divisão funcional das camadas do sistema:

Camada	Responsabilidade Principal	Tecnologias Utilizadas
Model	Representação dos dados, entidades e regras de negócio	C# , Entity Framework Core
Controller	Receber e processar requisições HTTP, coordenar ações e retornar respostas	ASP.NET Core
View	Apresentação dos dados em formato JSON e documentação via Swagger	Swagger UI, JSON
Data Access	Acesso ao banco de dados, persistência e consultas	SQL Server Express
Service	Implementação da lógica de negócio e validações	C# e boas práticas de arquitetura

Fonte : Tabela elaborada pelo próprio autor (2025).

Fluxo de Comunicação entre as Camadas

- O fluxo de funcionamento da aplicação segue a seguinte sequência lógica:

1. O usuário realiza uma requisição HTTP através de uma rota definida na API.
2. O Controller correspondente recebe a requisição e a valida.
3. O Controller aciona o Service, responsável por aplicar as regras de negócio e validações.
4. O Service comunica-se com o Model e, por meio do Entity Framework, realiza operações de leitura, inserção, atualização ou exclusão no banco de dados.
5. Após o processamento, a resposta é enviada de volta ao Controller.
6. O Controller retorna o resultado ao usuário em formato JSON, exibido de forma amigável no Swagger.

4 IMPLEMENTAÇÃO TÉCNICA

A implementação Técnica segundo Pressman se refere ao conjunto de atividades realizadas para transformar os requisitos de software em um produto operacional, que inclui a construção do software (codificação), testes para garantir a qualidade e sua eventual implantação e entrega ao cliente, seguindo princípios e métodos de engenharia de software para assegurar um desenvolvimento eficiente e confiável.

A implementação do sistema foi realizada utilizando a linguagem C# 9.0, em conjunto com o framework ASP.NET Core, que oferece uma estrutura robusta para o desenvolvimento de aplicações web baseadas em serviços RESTful. O Entity Framework Core foi empregado como ORM (Object-Relational Mapper), permitindo a interação entre as classes do sistema e o banco de dados SQL Server Express, de forma a simplificar o gerenciamento das operações de persistência de dados.

O **Swagger** foi integrado à aplicação para disponibilizar uma interface interativa de documentação e teste das APIs, facilitando o processo de desenvolvimento, depuração e integração com outras aplicações.

A estrutura do projeto foi organizada conforme os princípios da Arquitetura Limpa (Clean Architecture), que promove a separação de responsabilidades e a independência entre as camadas. Dessa forma, o sistema foi dividido em três principais camadas:

- **Model (Domínio):** responsável pelas entidades, regras de negócio e validações;

- **Controller (Apresentação):** encarregada de gerenciar as requisições HTTP, controlar o fluxo de informações e retornar as respostas adequadas;

- **Repository/Service (Aplicação e Infraestrutura):** responsável pela comunicação com o banco de dados, execução de consultas e manipulação dos dados.

As principais classes do sistema representam entidades fundamentais para o domínio da aplicação, como **Cliente**, **Veículo**, **Aluguel**, **Pagamento**, **Fabricante** , cada uma com seus respectivos atributos e relacionamentos. O padrão DTO (Data Transfer Object) foi adotado para garantir a segurança e integridade dos dados durante a comunicação entre as camadas, evitando o acoplamento direto das entidades do domínio.As escolhas de design priorizaram a modularidade, escalabilidade e facilidade de manutenção do código, permitindo que novas funcionalidades possam ser implementadas sem comprometer a estrutura existente. Além disso, a utilização de boas práticas de programação e convenções do .NET assegurou a legibilidade e padronização do projeto.

Tabela 1 – Tecnologias utilizadas no desenvolvimento do sistema:

Tecnologia	Descrição	Versão
C#	Linguagem principal utilizada para desenvolvimento da aplicação.	9.0
ASP.NET Core	Framework para construção de APIs e aplicações web seguindo o padrão MVC.	.NET 9.0
Entity Framework Core	ORM utilizado para mapear classes e gerenciar o acesso ao banco de dados.	9.0.9
SQL Server Express	Banco de dados relacional utilizado para persistência de dados da aplicação.	2022
Swagger (Swashbuckle)	Ferramenta para documentação interativa das APIs REST.	Mais recente disponível
Visual Studio Code	Ambiente de desenvolvimento integrado (IDE) utilizado para codificação e depuração.	August 2025 (version 1.104)
.NET SDK	Plataforma base para compilação e execução do projeto.	9.0
Postman/Insomnia	Ferramentas utilizadas para teste e validação das rotas da API.	Atual
Git e GitHub	Controle de versão e hospedagem do código-fonte do projeto.	Atual
AutoMapper	Mapear automaticamente dados entre objetos de diferentes tipos.	12.0.1

Fonte : Tabela elaborada pelo próprio autor (2025).

Tabela 2 – Principais classes do sistema :

Classe	Descrição	Responsabilidade Principal
Veículo	Representa os veículos cadastrados na locadora.	Armazena informações como modelo, marca, placa, quilometragem e status de disponibilidade.
Cliente	Define os dados dos clientes da locadora.	Controla informações pessoais, documentos e histórico de locações.
Fabricante	Gerencia informações sobre as montadoras dos veículos.	Permite cadastro, edição, exclusão e busca por nome ou ID.
Aluguel	Representa o processo de aluguel de um veículo.	Registra o cliente, veículo, datas de retirada e devolução, quilometragem e valor total.
Pagamento	Controla as transações financeiras relacionadas às locações.	Realiza, edita, exclui e consulta pagamentos com base no ID do cliente ou da locação.

Fonte : Tabela elaborada pelo próprio autor (2025).

5 DESCRIÇÃO DO BANCO DE DADOS

O banco de dados utilizado no sistema foi desenvolvido com o SQL Server Express, visando atender aos requisitos de persistência e integridade das informações da locadora de veículos. O modelo de dados foi estruturado de forma relacional, utilizando o Entity Framework Core como ferramenta de mapeamento objeto-relacional (ORM), garantindo a integração eficiente entre as classes do sistema e as tabelas do banco.

O modelo de dados foi projetado para representar as principais entidades do domínio, como veículos, clientes, locações, pagamentos e fabricantes, permitindo o controle eficiente das operações de cadastro, atualização, exclusão e consulta. Cada entidade foi mapeada para uma tabela correspondente, com chaves primárias, estrangeiras e restrições de integridade que asseguram a consistência das informações.

A seguir, descrevem-se as principais tabelas e seus relacionamentos:

Tabela	Descrição	Relacionamentos e Restrições de Integridade
Veículos	Armazena informações sobre os veículos disponíveis na locadora.	Chave primária: IdVeiculo. Chave estrangeira: IdFabricante.
Clientes	Contém os dados cadastrais dos clientes.	Chave primária: IdCliente. Relacionamento 1:N com Locacoes e Pagamentos.
Fabricantes	Registra as montadoras responsáveis pelos veículos.	Chave primária: IdFabricante. Relacionamento 1:N com Veiculos.
Locações	Controla as informações referentes às locações realizadas.	Chaves estrangeiras: IdCliente, IdVeiculo.

Pagamentos	Registra as transações financeiras associadas às locações.	Chaves estrangeiras: IdCliente, IdLocacao.
Funcionários	(Opcional) Armazena dados dos administradores do sistema.	Chave primária: IdFuncionario.

Fonte : Tabela elaborada pelo próprio autor (2025).

A integridade referencial é garantida por meio da utilização de chaves primárias e estrangeiras, que impedem a exclusão de registros vinculados, evitando inconsistências entre as tabelas relacionadas.
A seguir, apresentam-se **consultas SQL exemplificativas** que ilustram operações comuns no sistema:

```
-- Consulta de todas as locações de um cliente específico
SELECT * FROM Locacoes
WHERE IdCliente = 3;

-- Consulta dos pagamentos realizados por um cliente
SELECT * FROM Pagamentos
WHERE IdCliente = 3;

-- Consulta dos veículos disponíveis para aluguel
SELECT * FROM Veiculos
WHERE Disponivel = 1;

-- Consulta de veículos de um determinado fabricante
SELECT * FROM Veiculos
WHERE IdFabricante = 2;
```

Essas consultas demonstram a aplicabilidade do modelo de dados na recuperação de informações essenciais para o funcionamento do sistema, permitindo à locadora realizar o controle eficiente de sua frota, clientes e movimentações financeiras.

6 TESTES E VALIDAÇÃO

Os testes realizados no sistema de locadora de veículos tiveram como objetivo principal garantir o correto funcionamento das funcionalidades implementadas, assegurando a confiabilidade e a integridade dos dados processados. As validações foram conduzidas de forma manual, utilizando o software Postman, que permitiu testar as requisições HTTP de cada rota da API e observar o comportamento das respostas enviadas pelo servidor.

Durante a etapa de testes, foram verificadas as principais operações CRUD (Create, Read, Update, Delete), aplicadas às entidades Clientes, Veículos, Fabricantes, Locações e Pagamentos. Para cada rota, foram realizadas requisições GET, POST, PUT e DELETE, a fim de confirmar a integridade das operações de cadastro, edição, consulta e exclusão de registros.

Entidade	Método HTTP	Descrição do Teste	Resultado Esperado	Situação
Clientes	POST	Cadastrar novo cliente	Cliente criado com ID único e retorno HTTP 201 (Created)	Sucesso
Clientes	GET	Consultar lista de clientes	Retorna lista completa em formato JSON	Sucesso
Veículos	PUT	Atualizar quilometragem após locação	Valor de KM atualizado corretamente	Sucesso
Fabricante	DELETE	Excluir fabricante inexistente	Retorno HTTP 404 (Not Found)	Sucesso

Aluguel	POST	Registrar nova locação vinculada a cliente e veículo	Lançamento gravado com ID e datas corretas	Sucesso
Pagamento	GET	Listar pagamentos de um cliente específico	Retorna todos os pagamentos vinculados ao ID informado	Sucesso
Pagamento	PUT	Atualizar valor de um pagamento	Valor atualizado e salvo no banco de dados	Sucesso
Veículos	GET	Consultar veículo por ID	Retorno com informações completas do veículo	Sucesso
Aluguel	DELETE	Excluir locação com dependência de pagamento	Bloqueio por integridade referencial	Sucesso
Clientes	GET	Consultar histórico de locações de um cliente	Lista de locações vinculadas retornada corretamente	Sucesso

Fonte : Tabela elaborada pelo próprio autor (2025).

7 RESULTADOS E CONCLUSÕES

Os resultados obtidos com o desenvolvimento do sistema de locadora de veículos demonstram a eficácia da aplicação em automatizar e otimizar os processos de gestão da frota, controle de locações, registro de clientes e processamento de pagamentos. A implementação das funcionalidades previstas atendeu de forma satisfatória aos requisitos funcionais estabelecidos, proporcionando uma experiência fluida e eficiente para o administrador da locadora.

Durante o desenvolvimento, observou-se que a integração entre as camadas da aplicação, aliada ao uso do Entity Framework Core, permitiu o gerenciamento consistente dos dados, reduzindo erros de inserção e exclusão. Além disso, o uso do Swagger como ferramenta de documentação contribuiu para a visualização clara e testável das rotas da API, facilitando a validação e manutenção do sistema.

A conformidade dos requisitos foi comprovada por meio dos testes realizados no Postman, os quais confirmaram o funcionamento correto de todas as operações CRUD das entidades Clientes, Veículos, Fabricantes, Aluguel e Pagamentos. As rotas apresentaram retornos adequados, com status HTTP compatíveis e persistência adequada no banco de dados SQL Server Express, validando o comportamento esperado para cada caso de uso.

Conclui-se que o sistema desenvolvido atinge plenamente os objetivos propostos, oferecendo uma solução confiável e escalável para o gerenciamento de locações de veículos. A aplicação apresenta arquitetura modular, organização consistente e boa manutenção do código, refletindo boas práticas de desenvolvimento. Dessa forma, o projeto se mostra eficaz para uso em ambientes reais, podendo ser expandido futuramente com novas funcionalidades, como autenticação de usuários, relatórios analíticos e interface gráfica para clientes.

8 CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema de locadora de veículos representou uma experiência significativa de aprendizado e aprimoramento técnico. Ao longo do processo, foi possível consolidar conhecimentos sobre arquitetura de software, integração com banco de dados, implementação de APIs e boas práticas de desenvolvimento em C# e ASP.NET Core.

Apesar dos resultados positivos, o percurso apresentou diversos desafios, como a compreensão das relações entre as entidades, o uso do Entity Framework e a configuração correta das rotas e migrações. Essas dificuldades, entretanto, contribuíram de forma essencial para o crescimento acadêmico e profissional, permitindo uma melhor compreensão sobre o ciclo de vida do desenvolvimento de software e a importância da persistência diante de problemas técnicos.

Como sugestão para melhorias futuras, recomenda-se a implementação de autenticação e autorização de usuários, interface gráfica para o cliente final, e testes automatizados, a fim de ampliar a segurança, acessibilidade e robustez do sistema. Além disso, a integração com serviços externos, como gateways de pagamento ou plataformas de notificação, poderia tornar o sistema ainda mais completo e próximo de um ambiente de produção real.

Por fim, destaca-se que, embora o desenvolvimento tenha exigido esforço e superação de diversas barreiras, a experiência adquirida foi extremamente valiosa para o aprendizado prático na área de desenvolvimento web, fortalecendo competências técnicas e pessoais fundamentais para a formação em Sistemas de Informação.

REFERÊNCIAS

Pressman, R. S. *Engenharia de Software: uma abordagem profissional*. São Paulo: McGraw-Hill, [2011].

“Requisitos de Software: análise, elicitação e técnicas.” *DIO — Digital Innovation One*. Disponível em: https://www.dio.me/articles/requisitos-de-software-analise-elicitacao-e-tecnicas. Acesso em: 04 out. 2025.

Valente, Marco Túlio. “Engenharia de Software Moderna: Arquitetura Limpa.” *Engenharia de Software Moderna*. Disponível em: https://engsoftmoderna.info/artigos/arquitetura-limpa.html. Acesso em: 05 out. 2025.

“Estudo de caso no contexto da Engenharia de Software: SGCOPEX / Case study in the context of Software Engineering: SGCOPEX.” *Brazilian Journal of Development*. Acesso em: 05 out. 2025.