CENTRO UNIVERSITÁRIO CATÓLICA DE SANTA CATARINA
ENGENHARIA DE SOFTWARE

LARISSA HOFFMANN DE SOUZA

SISTEMA PARA RESERVA DE CARROS CORPORATIVOS

JOINVILLE
2025

Resumo

Neste projeto é apresentado a especificação de um sistema voltado para o controle de uso de veículos da frota corporativa. Baseado em uma arquitetura de micro serviços, com React.js no frontend e NestJS no backend, PostgreSQL como banco de dados e infraestrutura em nuvem (Azure), o projeto aborda desde os requisitos funcionais e não funcionais até os aspectos de segurança, usabilidade e monitoramento, propondo uma solução escalável e segura.

1. Introdução

1.1. Contexto

Empresas que mantêm uma frota de veículos corporativos enfrentam desafios relacionados à organização de reservas, controle de uso, rastreabilidade e conformidade documental. Processos manuais ou descentralizados tendem a gerar retrabalho, riscos operacionais e baixa eficiência. Este projeto propõe o desenvolvimento de uma aplicação web responsiva para centralizar e automatizar a gestão de reservas de carros, oferecendo controle completo sobre veículos, rotas, documentos e usuários.

1.2. Justificativa

No contexto da engenharia de software, este projeto representa uma aplicação prática de diversos conceitos fundamentais da área. Além disso, a solução possui aplicabilidade real, podendo ser adaptada a qualquer organização que requeira o controle do uso de veículos de forma digital, segura e escalável. Dessa forma, o projeto possui relevância acadêmica e aplicabilidade prática.

1.3. Objetivos

1.3.1 Objetivo Geral

Desenvolver um sistema web de gerenciamento de reservas de veículos corporativos, com controle de acesso baseado em perfis e suporte à auditoria de uso.

1.3.2 Objetivos Específicos:

- Definir e implementar requisitos funcionais e não funcionais.
- Aplicar arquitetura de micro serviços com C4 Model.
- Desenvolver interfaces responsivas com React.js, priorizando usabilidade.
- Implementar backend com NestJS e banco de dados PostgreSQL.
- Integrar cache com Redis e autenticação com JWT + OAuth2.
- Automatizar CI/CD com GitHub Actions.
- Monitorar a aplicação com Prometheus e Grafana.
- Garantir segurança com criptografia, controle de sessão e logs de auditoria.
- Disponibilizar o sistema em ambiente em nuvem escalável (Azure).

2. Descrição do Projeto

2.1. Tema do Projeto

O projeto propõe o desenvolvimento de uma plataforma web para gerenciamento de reservas de veículos corporativos.

2.2. Problemas a Resolver

- Ausência de automatização na gestão e controle de reservas veiculares.
- Inexistência de histórico e rastreabilidade de reservas e documentos.
- Dificuldade de padronizar o processo de solicitação e aprovação de uso de carros.
- Falta de relatórios consolidados sobre o uso de veículos.

2.3. Limitações

- Não suportará integração com órgãos de trânsito para consulta ou emissão de multas em tempo real.
- Não haverá módulo de manutenção preventiva ou corretiva dos veículos.
- Não suportará aplicativos móveis nativos (iOS/Android) — a solução será responsiva via navegador.
- Não haverá controle financeiro completo com integração a sistemas de reembolso ou folha de pagamento.

- Não haverá módulo para auditoria jurídica de uso indevido de veículos ou incidentes legais.

3. Especificação Técnica

3.1. Requisitos de Software

Requisitos Funcionais (RF):

- RF01: O sistema deve permitir ao administrador criar, atualizar e remover usuários.
- RF02: O sistema deve permitir ao administrador atribuir e revogar permissões de aprovador.
- RF03: O sistema deve autenticar os usuários com base em login/senha.
- RF04: O sistema deve permitir que administradores e aprovadores cadastrem novos carros informando placa, modelo, cor e quilometragem.
- RF05: O sistema deve permitir que administradores e aprovadores atualizem os dados dos carros.
- RF06: O sistema deve permitir que administradores e aprovadores removam carros.
- RF07: O sistema deve permitir que administradores e aprovadores cadastrem postos credenciados.
- RF08: O sistema deve permitir que administradores e aprovadores atualizem e removam dos postos cadastrados.
- RF09: O sistema deve permitir que usuários solicitem reservas informando local de saída, chegada, data e horário.
- RF10: O sistema deve permitir que o administrador exclua solicitações de reserva.
- RF11: O sistema deve permitir que aprovadores aprovem ou cancelem reservas pendentes.
- RF12: O sistema deve permitir que usuários façam upload de documentos após o uso do carro (CNH, recibo de posto, foto da quilometragem, outros gastos).
- RF13: O sistema deve permitir que administradores e aprovadores consultem e validem os documentos enviados.

- RF14: O sistema deve permitir que administradores cadastrem, atualizem e removam modelos de checklist.
- RF15: O sistema deve permitir que usuários preencham o checklist ao devolver o carro.
- RF16: O sistema deve permitir que os aprovadores preencham um checklist de validação na devolução do carro.
- RF17: O sistema deve permitir a geração de relatórios filtráveis por: Usuário; Carro; Filial; Período.
- RF18: O sistema deve permitir que usuários visualizem seu histórico pessoal de uso de veículos.
- RF19: O sistema deve permitir ao usuário visualizar os postos cadastrados no caminho entre origem e destino da reserva.

Requisitos Não-Funcionais (RNF):

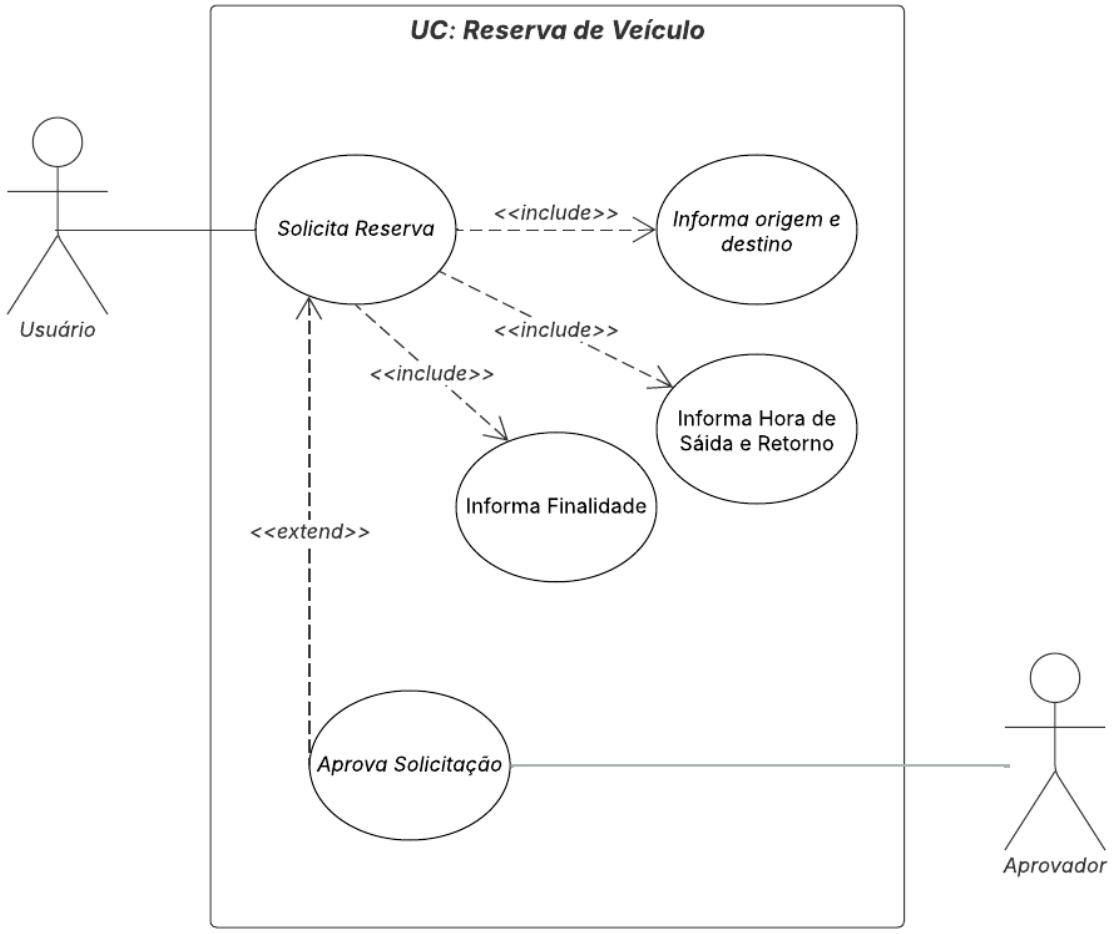
- RNF01: O sistema deve ser acessível via navegador web responsivo.
- RNF02: A autenticação dos usuários deve ser realizada de forma segura, utilizando JWT e OAuth2.
- RNF03: O backend deve ser implementado utilizando o framework NestJS.
- RNF04: O frontend deve ser desenvolvido com React.js.
- RNF05: O sistema deve utilizar banco de dados relacional PostgreSQL para persistência de dados.
- RNF06: O sistema deve utilizar Redis como mecanismo de cache.
- RNF07: A aplicação deve seguir arquitetura baseada em micro serviços.
- RNF08: O sistema deve manter logs de auditoria com rastreabilidade de ações por usuário.
- RNF09: A interface do usuário deve ser intuitiva, seguindo boas práticas de UX/UI.
- RNF10: A aplicação deve possuir tempo médio de resposta inferior a 2 segundos para operações comuns.
- RNF11: O sistema deve seguir o padrão arquitetural MVC.
- RNF12: O sistema deve garantir alta disponibilidade.

3.1.1 Funcionalidades Futuras

- RF20 (Futuro): Sugestão de rota com API do Google Maps.
- RF21 (Futuro): Vincular multas a usuários com base na reserva ativa.
- RF22 (Futuro): Integração com Active Directory (AD).
- RF23 (Futuro): Integração com Microsoft Teams e Outlook.

3.1.2 Diagrama de Caso de Uso

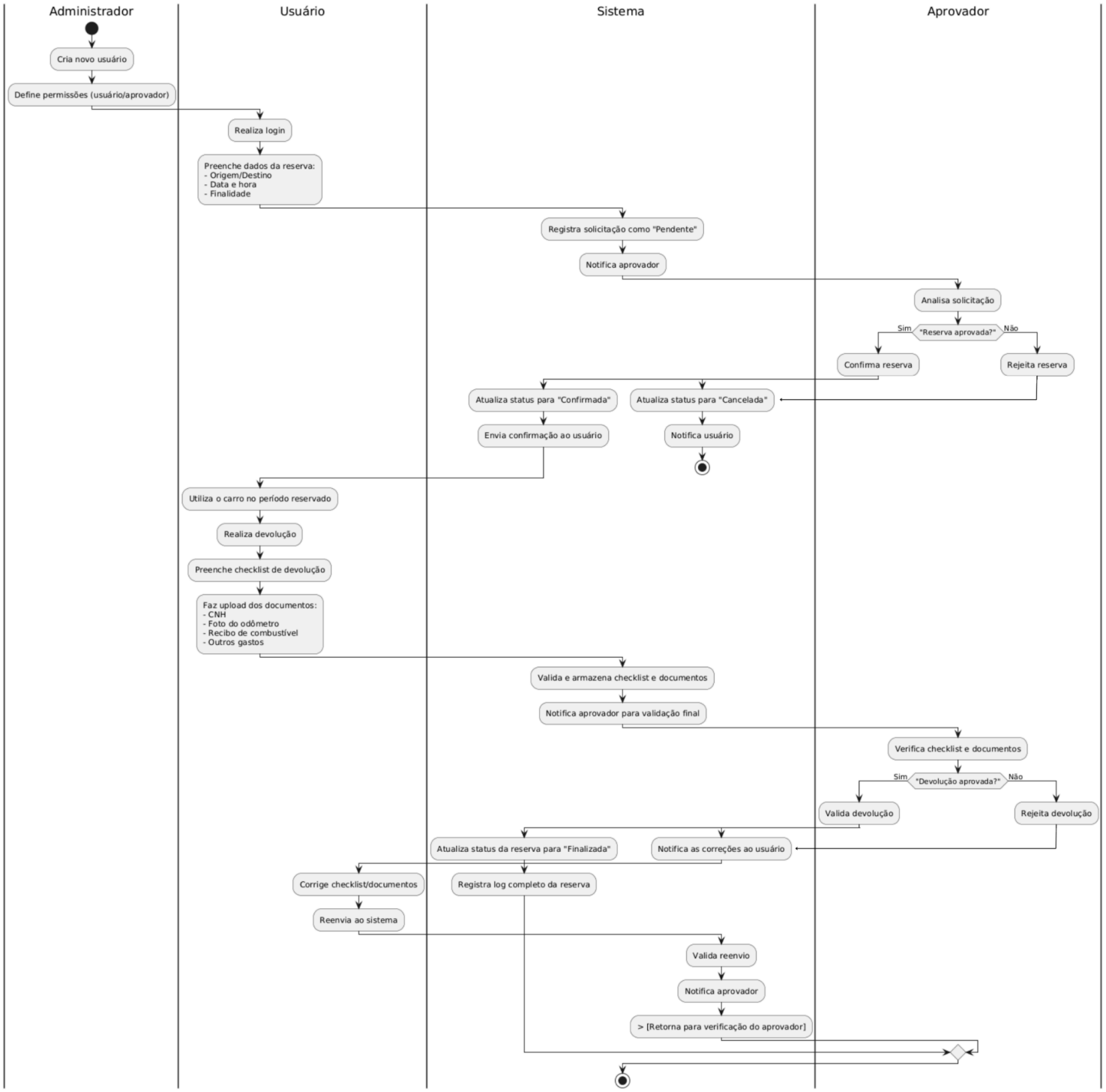
O diagrama a seguir representa uma das funcionalidades centrais do sistema: o processo de reserva de veículo. Nele, os papéis de Usuário e Aprovador são destacados, com seus respectivos casos de uso.



3.2. Considerações de Design

O design do sistema foi orientado por princípios de modularidade, escalabilidade, manutenibilidade e segurança. Os protótipos de interface do usuário serão desenvolvidos no Figma, permitindo validação da experiência antes da implementação com React.js.

3.2.1 Diagramas de Atividades



3.2.2 Visão Inicial da Arquitetura

- **Frontend**: React.js.
- **Backend**: NestJS.
- **Banco de Dados**: PostgreSQL.
- **Cache**: Redis.
- **Integrações**: OAuth2.
- **Infraestrutura**: Docker + Azure.

3.2.3 Padrões de Arquitetura

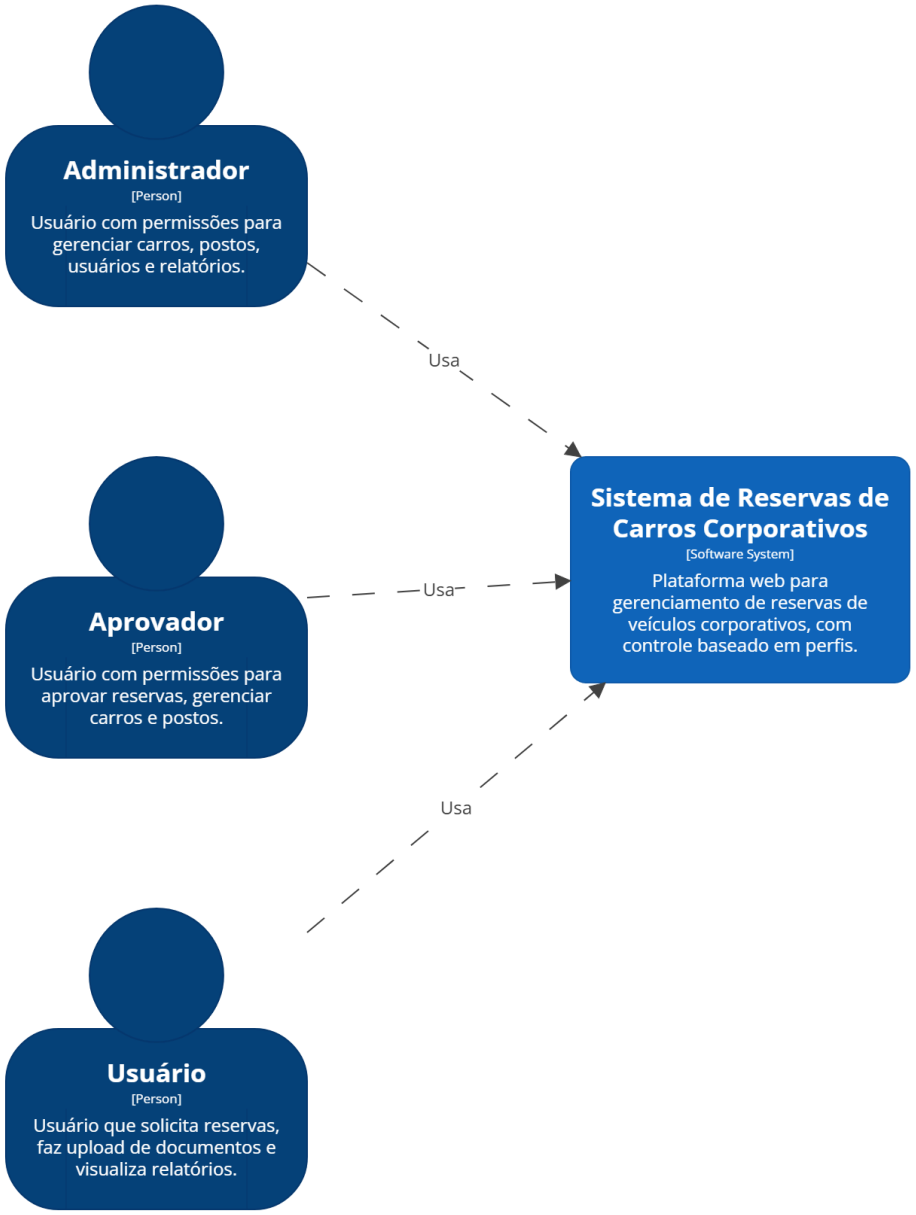
- **Microserviços**: Serviços independentes com responsabilidade única.
- **MVC (Model-View-Controller)**: Aplicado tanto no frontend (React) quanto backend (NestJS).
- **RESTful API**: Comunicação entre frontend e backend.
- **Autenticação com OAuth2 + JWT**: Para segurança e controle de acesso.

3.2.4 Modelos C4 (Visão da Arquitetura)

Esta seção apresenta a arquitetura da aplicação nos três níveis de abstração do C4 Model: Diagrama de Contexto, Contêineres e Componentes.

3.2.4.1 Diagrama de Contexto - Nível 1

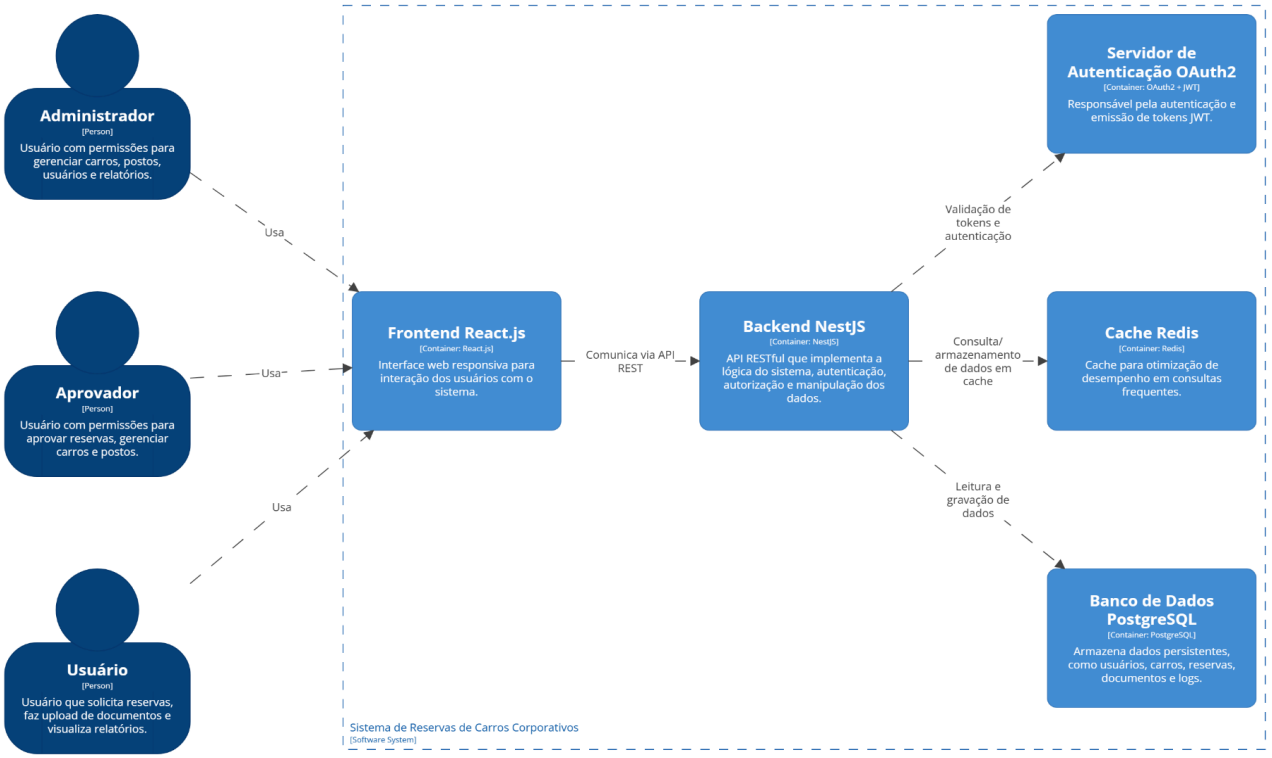
Este diagrama fornece uma visão geral do sistema no seu ecossistema. Ele mostra como o sistema se relaciona com os principais usuários (atores) e define o propósito da aplicação.



[System Context] Sistema de Reservas de Carros Corporativos

3.2.4.2 Diagrama de Contêineres - Nível 2

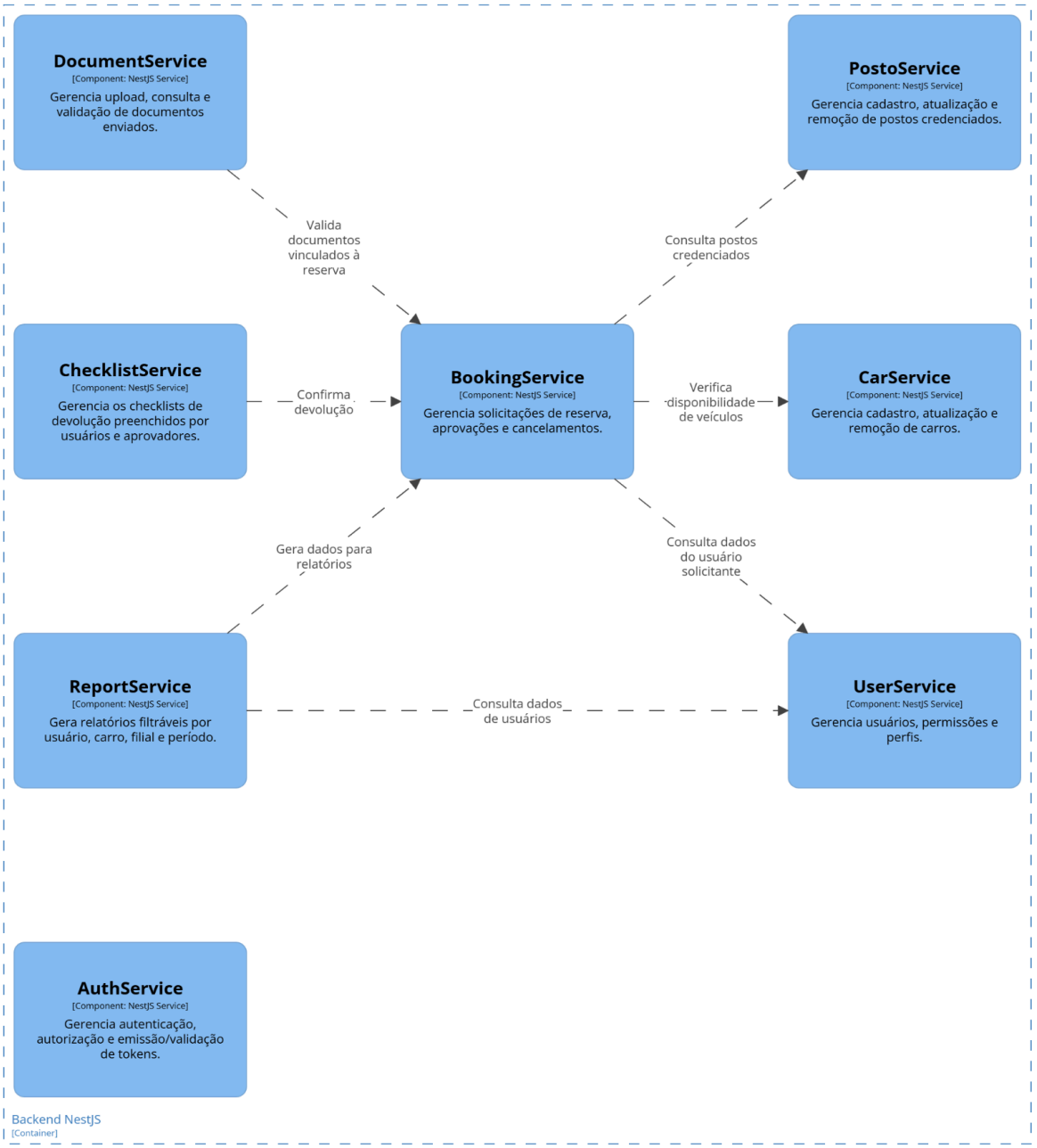
Neste nível, o sistema é decomposto em seus principais contêineres. O diagrama mostra como esses contêineres se comunicam e quais tecnologias são utilizadas, como React.js, NestJS, PostgreSQL, Redis e OAuth2.



[Container] Sistema de Reservas de Carros Corporativos

3.2.4.3 Diagrama de Componentes - Nível 3

Por fim, os componentes internos (módulos ou serviços) são detalhados. Cada componente representa uma responsabilidade clara no sistema, e o diagrama evidencia as interações entre eles dentro do escopo da API do backend.



[Component] Sistema de Reservas de Carros Corporativos - Backend NestJS

3.3. Stack Tecnológica

Linguagens de Programação

- **TypeScript:** Utilizado tanto no frontend quanto no backend pela sua tipagem estática, facilitando a manutenção e reduzindo erros em tempo de execução.
- **SQL (PostgreSQL):** Escolhido por ser um banco de dados relacional robusto, com suporte a ACID, além de contar com ampla documentação e comunidade ativa.

Frameworks e Bibliotecas

- **Frontend**
    - **React.js:** Permite desenvolvimento baseado em componentes reutilizáveis.
    - **Tailwind CSS**: Framework utilitário que acelera o desenvolvimento visual e garante consistência na estilização.
    - **Axios:** Biblioteca eficiente para requisições HTTP, com fácil integração ao React.
    - **React Router**: Gerência navegação entre páginas de forma declarativa.
- **Backend**
    - **NestJS:** Framework baseado em TypeScript, ideal para aplicações em micro serviços.
    - **TypeORM/Prisma:** Proporcionam abstração sobre o banco de dados relacional.
    - **Passport.js + JWT:** Conjunto seguro para autenticação baseada em tokens.
    - **Class Validator:** Validação automática de dados nas classes DTOs, com integração nativa ao NestJS.
    - **Swagger:** Ferramenta para documentação automática de APIs.

Ferramentas de Desenvolvimento e Gestão de Projeto

- **Git + GitHub:** Controle de versão distribuído e hospedagem com suporte a pull requests, revisão de código e integração com GitHub Actions.
- **CI/CD (GitHub Actions):** Automatização de build, testes e deploy contínuo.
- **Figma:** Ferramenta utilizada para criação dos protótipos navegáveis do frontend.

- **Monitoramento (Prometheus + Grafana):** Ferramentas para coleta e visualização de métricas, permitindo análises proativas de desempenho.
- **Documentação Técnica (Wiki):** Organização de documentos técnicos, decisões e guias.
- **Gerenciamento de Projeto (GitHub Projects):** Utilizado para planejamento, acompanhamento e priorização de tarefas de forma ágil e integrada ao repositório.

Ambiente de Execução

- **Docker:** Todos os serviços (frontend, backend, banco de dados, cache) são isolados em containers, garantindo consistência e independência de ambiente.
- **Nuvem (Azure):** Escolhida por sua escalabilidade, integração nativa com serviços de segurança corporativa e confiabilidade no provisionamento de infraestrutura.

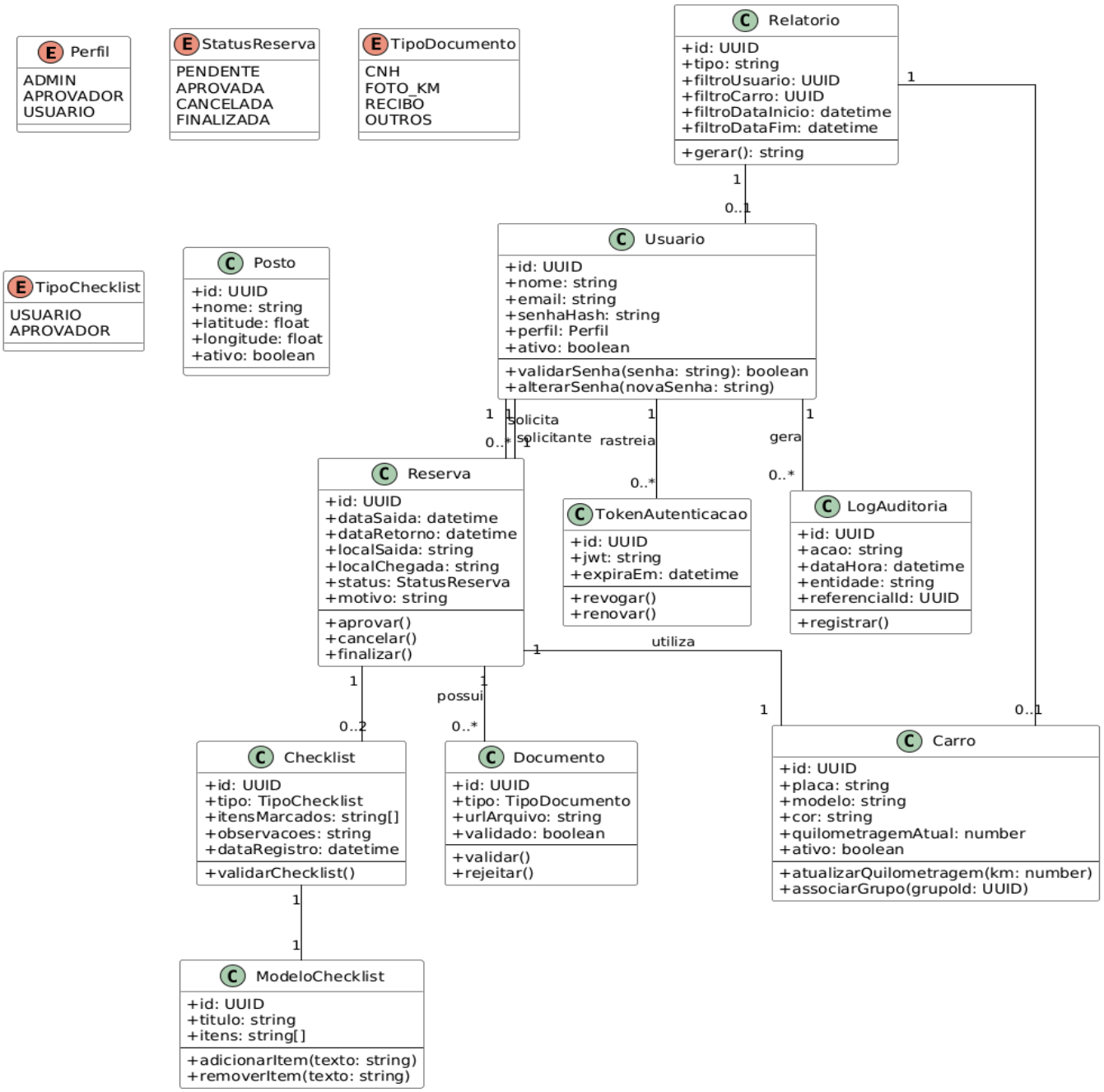
3.4. Considerações de Segurança

- **Autenticação e Autorização:** Utilização de JWT com suporte a OAuth2 para login seguro. O controle de acesso segue o modelo RBAC (Role-Based Access Control), com proteção por guards e policies em cada endpoint.
- **Proteção contra Ataques Comuns:** O sistema aplica proteção contra SQL Injection, XSS e CSRF por meio de ORMs seguros (TypeORM/Prisma), validações com class-validator e políticas de segurança via headers HTTP (CSP, CORS, HSTS).
- **Upload Seguro de Arquivos:** Arquivos enviados pelos usuários passarão por validação de tipo, tamanho e nome. Itens inválidos serão descartados, e os arquivos válidos serão armazenados de forma segura.
- **Logs de Auditoria:** Todas as ações críticas serão registradas com Winston, estruturadas para integração com OpenTelemetry e visualização em tempo real via Prometheus e Grafana.
- **Gerenciamento de Sessão:** Sessões inativas serão expiradas automaticamente. Tokens inválidos serão rejeitados, e as sessões removidas do Redis no logout.

- **Deploy Seguro em Nuvem:** A infraestrutura na Azure será configurada com variáveis de ambiente protegidas, redes privadas, HTTPS obrigatório e princípios de menor privilégio.

3.5 Diagrama de Classes (UML)

Abaixo, apresenta-se o diagrama de classes do sistema, representando entidades, atributos e seus relacionamentos no domínio do sistema.



4. Próximos Passos

Portfólio I

- Modelagem de interfaces e fluxos no Figma.
- Definição do banco de dados no PostgreSQL.
- Setup de ambiente com Docker.
- Estruturação do repositório e CI/CD.
- Implementação inicial do backend (NestJS) e autenticação.
- Primeiras telas do frontend (React.js).
- Defesa do Portfólio I.
- Elaboração do plano de produto.

Portfólio II

- Implementação completa das funcionalidades.
- Integração entre frontend e backend.
- Upload de documentos e geração de relatórios.
- Otimização de performance e sessões com Redis.
- Testes automatizados (unit, integração, e2e).
- Deploy com monitoramento.
- Documentação técnica e manual do usuário.
- Apresentação final e defesa.

5. Referências

ATLASSIAN. Jira Software. Disponível em: https://www.atlassian.com/software/jira.

C4 MODEL. Structurizr – C4 model for visualising software architecture. Disponível em: https://c4model.com/.

DOCKER. Docker Documentation. Disponível em: https://docs.docker.com/.

FACEBOOK. React – A JavaScript library for building user interfaces. Disponível em: https://reactjs.org/.

FIGMA. Figma – Interface design tool. Disponível em: https://www.figma.com/.

FOWLER, Martin. MVC – Model View Controller. Disponível em: https://martinfowler.com/eaaDev/uiArchs.html.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

GITHUB. GitHub Actions Documentation. Disponível em: https://docs.github.com/en/actions.

GITHUB PROJECTS. GitHub Project Management. Disponível em: https://github.com/features/project-management/.

GRAFANA. Grafana Labs – Open Source Analytics & Monitoring. Disponível em: https://grafana.com/.

JWT.IO. JSON Web Tokens – Introduction. Disponível em: https://jwt.io/introduction.

MICROSOFT. Microsoft Azure Documentation. Disponível em: https://learn.microsoft.com/en-us/azure/.

NESTJS. NestJS: A progressive Node.js framework. Disponível em: https://nestjs.com/

OAUTH. OAuth 2.0 Authorization Framework. Disponível em: https://oauth.net/2/.

OPENTELEMETRY. OpenTelemetry Documentation. Disponível em: https://opentelemetry.io/.

POSTGRESQL. PostgreSQL: The World's Most Advanced Open Source Relational Database. Disponível em: https://www.postgresql.org/.

PROMETHEUS. Prometheus Monitoring System. Disponível em: https://prometheus.io/.

REDIS. Redis Documentation. Disponível em: https://redis.io/.

WINSTON. Winston – A logger for Node.js. Disponível em: https://github.com/winstonjs/winston.

6. Avaliações de Professores

- **Considerações** **Professor/a:** _____

- **Considerações** **Professor/a:**

  _____

- **Considerações** **Professor/a:**

  _____