**CENTRO UNIVERSITÁRIO CATÓLICA DE SANTA CATARINA**

**ENGENHARIA DE SOFTWARE**

**RICHARD SCHMITZ RIEDO**

**Sistema Web para Organização e Gerenciamento de Cenários de Teste QA**

**Joinville**

**2025**

**Resumo**

Este documento apresenta os fundamentos técnicos, conceituais e especificações para o desenvolvimento de uma aplicação web voltada ao gerenciamento de cenários de teste no contexto de Quality Assurance (QA). O objetivo do sistema é centralizar e organizar o planejamento, a execução e o acompanhamento de testes manuais de software, promovendo rastreabilidade, visibilidade e padronização das atividades de teste em projetos de desenvolvimento de software.

A plataforma ou aplicação permitirá o registro de cenários de teste vinculados a requisitos e funcionalidades, a atribuição de responsáveis, o upload de evidências e a geração de relatórios gerenciais. O sistema será desenvolvido com uma arquitetura moderna baseada em Vue.js (Quasar) no frontend, Node.js (Express) no backend, e banco de dados PostgreSQL, garantindo segurança, desempenho e facilidade de manutenção. Com isso, busca-se aprimorar a eficiência das equipes de QA, reduzir falhas em produção com métricas para apoiar a tomada de decisão por parte dos gestores de produto.

**Introdução**

A garantia da qualidade (Quality Assurance – QA) desempenha um papel fundamental no ciclo de vida do desenvolvimento de software, assegurando que os sistemas entregues atendam aos requisitos funcionais e não funcionais esperados. No entanto, o gerenciamento de cenários e casos de teste frequentemente é realizado de maneira informal ou manual, espalhada entre planilhas, documentos e anotações manuais. Essa prática pode comprometer a rastreabilidade dos testes, dificultar a colaboração entre equipes e reduzir a eficiência na identificação de falhas.

Para solucionar esses problemas, este projeto de portfólio propõe o desenvolvimento de uma aplicação web voltada à **organização, execução e acompanhamento de cenários de teste de QA**. A ferramenta permitirá que equipes de qualidade planejem testes, associem-nos a requisitos ou funcionalidades, registrem evidências, atribuam responsáveis, monitorem o status de execução e documentem os resultados de forma centralizada.

A proposta tem como objetivo facilitar a **gestão de qualidade nos projetos de software**, promovendo a visibilidade dos testes realizados, a padronização da documentação e a redução de retrabalho causado por falhas não detectadas precocemente. Com uma interface intuitiva, integração entre frontend e backend via API RESTful, e banco de dados relacional para armazenamento seguro das informações, o sistema visa atender tanto times técnicos quanto gestores de produto.

**Descrição do Projeto**

O projeto propõe o desenvolvimento de uma **aplicação web para o gerenciamento completo de cenários e casos de teste**, voltada para equipes de QA, desenvolvedores e gestores de produto. A aplicação visa centralizar todas as informações relacionadas ao planejamento, execução, rastreamento e evidência de testes realizados durante o ciclo de desenvolvimento de software.

A aplicação permitirá que os usuários cadastrem **cenários de teste** detalhados, vinculando-os a requisitos funcionais, módulos do sistema e versões de release. Cada cenário poderá conter informações como **pré-condições, passos a executar, dados esperados, responsável, status, evidências (arquivos ou links)** e observações. Será possível também acompanhar o histórico de execuções e gerar relatórios com base nos resultados obtidos.

Além disso, a plataforma contará com **controle de usuários e permissões**, permitindo o acesso segmentado para testadores, desenvolvedores e gestores. Com isso, será possível registrar quem executou determinado teste, em qual data, e qual foi o resultado, promovendo **rastreabilidade e responsabilidade no processo de QA**.

O sistema será desenvolvido em uma arquitetura cliente-servidor moderna, com frontend em Vue.js (Quasar Framework), backend em Node.js (Express) e banco de dados relacional PostgreSQL. A comunicação entre os módulos será realizada via RESTful API, utilizando autenticação segura com JWT.

**Problemas a Resolver**

Este projeto visa solucionar desafios recorrentes no processo de planejamento, execução e documentação de testes de qualidade, incluindo:

* **Falta de rastreabilidade de testes:** Muitas equipes utilizam planilhas ou documentos dispersos, o que dificulta o rastreamento do histórico e da autoria dos testes.
* **Desorganização e duplicidade de cenários:** A ausência de um repositório centralizado leva à repetição de testes e perda de tempo na reestruturação dos mesmos.
* **Baixa visibilidade do status dos testes:** Gestores e desenvolvedores muitas vezes não têm acesso em tempo real à situação dos testes executados, dificultando a tomada de decisão.
* **Ausência de padronização e evidências:** A documentação dos testes varia entre equipes, dificultando auditorias e a análise de falhas, além da falta de armazenamento organizado de evidências (prints, vídeos, logs).
* **Dificuldade de colaboração entre QA, devs e produto:** Sem uma plataforma única, a comunicação e atualização entre diferentes perfis do time torna-se ineficiente e propensa a ruídos.

**Limitações**

Algumas limitações iniciais serão definidas para manter o escopo do projeto viável na fase atual:

* **Foco exclusivo em cenários manuais de teste:** A ferramenta será voltada à documentação e execução manual de testes, não incluindo automação nesta fase.
* **Ausência de integração com ferramentas externas de testes (ex: JIRA, TestLink):** A versão inicial não prevê integração com outras plataformas de gestão.
* **Não haverá execução automatizada de scripts:** O sistema não rodará testes automatizados nem coletará métricas de forma ativa.
* **Edição limitada de artefatos:** A edição de cenários e casos será simples, sem suporte inicial a versionamento avançado ou workflows customizados.

**Especificação Técnica**

A especificação técnica é uma parte essencial no desenvolvimento de sistemas, pois fornece uma descrição detalhada dos aspectos operacionais e estruturais de um projeto. O principal objetivo é assegurar que todos os envolvidos no processo de desenvolvimento tenham uma compreensão clara dos requisitos, das tecnologias que serão utilizadas, dos fluxos de trabalho e das decisões de design que guiarão a implementação da solução proposta.

**Requisitos do Tema Proposto**

O sistema deverá contemplar os seguintes requisitos:

**Protocolos:**

* **HTTP/HTTPS**: Comunicação entre cliente e servidor.
* **RESTful API**: Estrutura de requisições entre frontend e backend.
* **JWT (JSON Web Token)**: Autenticação segura para acesso de usuários autenticados.

**Funcionalidades Principais**

* Criação e organização de cenários de teste.
* Associação de cenários a requisitos ou módulos do sistema.
* Atribuição de responsáveis e status (ex: Pendente, Em Execução, Aprovado, Reprovado).
* Upload de evidências (print, vídeo, documentos).
* Geração de relatórios por status, por testador ou por funcionalidade.
* Dashboard com métricas e indicadores de qualidade.

**Procedimentos:**

1. Realiza cadastro e login no sistema.
2. Cria um projeto ou seleciona um existente.
3. Adiciona cenários de teste com título, descrição, pré-condições, passos e dados esperados.
4. Atribui testadores responsáveis.
5. Registra a execução do teste com status e evidências.
6. Visualiza resultados, métricas e relatórios.
7. Edita, exclui ou duplica cenários conforme necessidade.

**Formatos de Dados:**

* Cenários de teste: JSON (frontend/backend) com estrutura contendo campos como título, descrição, requisitos vinculados, status e evidências.
* Evidências: Arquivos .jpg, .png, .pdf.
* Exportação: Relatórios gerados em PDF.
* Banco de dados relacional (PostgreSQL): Armazenamento seguro de usuários, projetos, cenários, execuções e evidências.

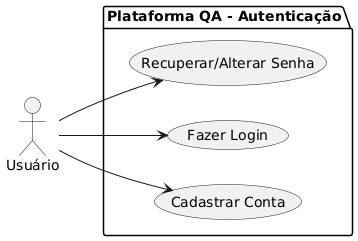
**Módulos do Sistema**

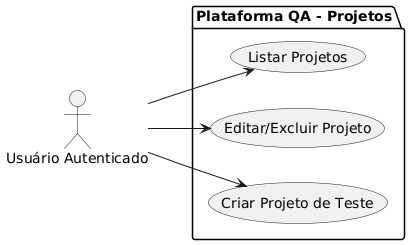
* **Frontend (Vue.js + Quasar):** Interface responsiva com foco em usabilidade e performance.
* **Backend (Node.js + Express):** API RESTful com endpoints para autenticação, gerenciamento de projetos e cenários.
* **Armazenamento de Arquivos:** Evidências serão armazenadas em serviço externo.
* **Banco de Dados (PostgreSQL):** Estrutura relacional para dados persistentes.

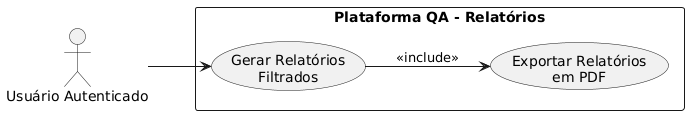
**Requisitos Funcionais (RF)**

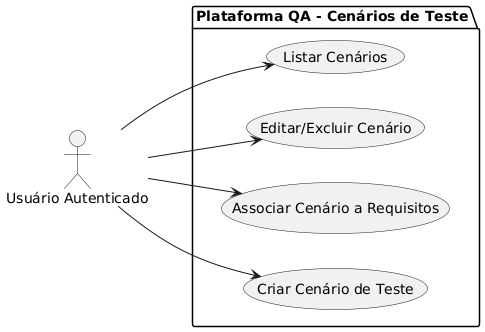
|  |  |  |
| --- | --- | --- |
| **Código** | **Prioridade** | **Requisito Funcional** |
| RF01 | Must | O sistema deve permitir que o usuário se cadastre com e-mail e senha, validando duplicidade e formato. |
| RF02 | Must | |  | | --- | | O sistema deve permitir login com e-mail e senha, retornando token JWT em caso de sucesso. |  |  | | --- | |  | |
| RF03 | Must | O sistema deve permitir recuperação e alteração de senha com envio de link por e-mail (válido por 15 minutos). |
| RF04 | Must | |  | | --- | |  |  |  | | --- | | O sistema deve permitir criação e gerenciamento de projetos de teste com nome, descrição e data de criação. | |
| RF05 | Must | O sistema deve permitir a criação de cenários com título, descrição, pré-condições, passos, dados esperados e observações. |
| RF06 | Should | |  | | --- | | O sistema deve permitir vincular cenários a requisitos ou funcionalidades específicas do sistema testado. |  |  | | --- | |  | |
| RF07 | Must | |  | | --- | | O sistema deve permitir upload de evidências (imagens JPG/PNG ou PDF), com limite de 5MB por arquivo. |  |  | | --- | |  | |
| RF08 | Should | O sistema deve permitir a atribuição de responsáveis pelos testes. |
| RF09 | Must | O sistema deve registrar o resultado da execução de cada cenário (aprovado, reprovado, pendente). |
| RF10 | Should | O sistema deve manter um histórico de execuções por cenário. |
| RF11 | Could | O sistema deve permitir comentários ou anotações em cada cenário ou execução. |
| RF12 | Must | O sistema deve gerar relatórios com filtros por status (Concluído, em teste, não começado) responsável ou projeto. |
| RF13 | Could | O sistema deve permitir a exportação dos relatórios em formato PDF. |
| RF14 | Should | O sistema deve permitir a exclusão e edição de cenários e execuções. |
| RF15 | Must | O sistema deve listar os projetos e cenários criados pelo usuário. |

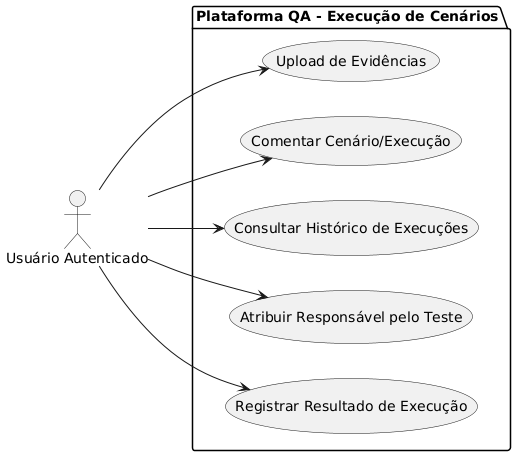
**Diagramas de casos de uso**











**Requisitos Não-Funcionais (RNF)**

|  |  |  |
| --- | --- | --- |
| **Código** | **Prioridade** | **Requisito Não-Funcional** |
| RNF01 | Must | O sistema deve ter uma interface web responsiva. |
| RNF02 | Must | As informações do usuário e senhas devem ser armazenadas de forma criptografada. |
| RNF03 | Must | |  | | --- | | As evidências devem ser armazenadas com integridade, incluindo checksum para validação e vinculação ao cenário via ID único. |  |  | | --- | |  | |
| RNF04 | Must | |  | | --- | | A API deve utilizar autenticação via JWT, validar dados com express-validator e proteger rotas com middleware. |  |  | | --- | |  | |
| RNF05 | Should | |  | | --- | | 90% das requisições da API devem ter tempo de resposta menor que 500ms em condições normais de carga. |  |  | | --- | |  | |
| RNF06 | Should | |  | | --- | | O sistema deve suportar pelo menos 100 usuários simultâneos sem degradação significativa de performance. |  |  | | --- | |  | |
| RNF07 | Must | |  | | --- | | A disponibilidade da aplicação deve ser de no mínimo **99% uptime mensal**, considerando infraestrutura em nuvem. |  |  | | --- | |  | |
| RNF08 | Could | |  | | --- | | O sistema deve registrar logs de erros e atividades críticas com timestamp, tipo e origem, para auditoria. |  |  | | --- | |  | |
| RNF09 | Should | A aplicação deve seguir boas práticas de acessibilidade (WCAG 2.1 nível AA), com contraste e navegação por teclado. |

**Discussão sobre as Escolhas de Design**

O sistema foi concebido com base em uma **arquitetura modular** e na aplicação do padrão **MVC (Model-View-Controller)**, tanto no frontend quanto no backend, visando organização, escalabilidade e manutenibilidade.

A separação entre **frontend (Vue.js + Quasar)**, **backend (Node.js + Express)** e o **banco de dados relacional (PostgreSQL)** permite que cada módulo evolua de forma independente, além de facilitar testes, manutenções e futuras integrações com outras ferramentas de QA, como JIRA ou ferramentas de CI/CD.

**Visão Inicial da Arquitetura**

A arquitetura do sistema está organizada em **cinco componentes principais**, que interagem de forma integrada para garantir uma experiência fluida ao usuário, processamento eficiente e armazenamento seguro das informações.

**Interface Web (Frontend)**

Responsável por toda a interação com o usuário, construída com Vue.js e Quasar Framework. Esta interface permitirá:

* **Cadastro e login de usuários.**
* **Criação e gerenciamento de projetos e cenários de teste.**
* **Execução de testes e registro de resultados.**
* **Upload e visualização de evidências.**
* **Acompanhamento de indicadores e relatórios.**

**Mockups e Wireframes**

Antes da implementação, serão criados wireframes (esboços estruturais) das principais telas, como:

* **Tela de login/cadastro**
* **Dashboard com métricas**
* **Tela de criação de cenários**
* **Tela de execução e registro de testes**
* **Tela de geração de relatórios**

Esses wireframes servirão como base para o design visual (mockups de alta fidelidade, se o cronograma permitir) e auxiliarão nas validações com usuários e stakeholders.

**API Backend (Node.js + Express)**

Camada intermediária que gerencia a lógica de negócio e o acesso aos dados, estruturada em controladores e serviços:

* **Validação e autenticação de usuários (JWT).**
* **Gerenciamento de projetos, cenários e execuções.**
* **Processamento de uploads de evidências.**
* **Geração de relatórios filtráveis e exportáveis.**

**Módulo de Upload e Armazenamento de Evidências**

Componente responsável por tratar os arquivos enviados como evidência, como imagens, vídeos ou documentos:

* **Validação de formatos e tamanhos.**
* **Armazenamento seguro no banco de dados.**
* **Associação direta com cenários e execuções específicas.**

**Módulo de Relatórios e Indicadores**

Responsável pela análise dos dados e geração de relatórios gerenciais:

* **Rumo por projeto, por testador, por funcionalidade ou status.**
* **Gráficos de cobertura de testes, taxa de sucesso e evolução por sprint.**

**Banco de Dados (PostgreSQL)**

Base relacional estruturada para garantir persistência e integridade dos dados, incluindo:

* **Informações de usuários, autenticação e permissões.**
* **Projetos, cenários, execuções, evidências e comentários.**
* **Logs de alterações e histórico de execução.**

A comunicação entre os módulos ocorrerá via requisições REST em formato JSON, garantindo interoperabilidade entre tecnologias distintas e fácil manutenção futura.

**Padrões de Arquitetura**

Para garantir a organização, escalabilidade e manutenibilidade do sistema, foram adotados dois principais padrões arquiteturais: o padrão **MVC (Model-View-Controller)** e a **separação lógica em microsserviços**.

**Padrão MVC (Model-View-Controller)**

Aplicado tanto no **frontend (Vue.js + Quasar)** quanto no **backend (Node.js + Express)**, o padrão MVC assegura a separação de responsabilidades e a clareza no desenvolvimento de funcionalidades.

* **Model (Modelo):** Representa as entidades do sistema como usuários, projetos, cenários, execuções e evidências. Esses modelos são manipulados por meio de um ORM (Prisma), permitindo consultas seguras e performáticas no PostgreSQL.
* **View (Visão):** Refere-se à interface de usuário desenvolvida com componentes Vue.js, seguindo os princípios de design responsivo e acessibilidade, permitindo interações intuitivas para todas as ações do sistema.
* **Controller (Controlador):** Responsável por receber requisições, aplicar regras de negócio, validar dados e acionar os serviços adequados. Cada funcionalidade possui seu controlador específico (ex: AuthController, TestCaseController, ExecutionController).

**Arquitetura modular (separação lógica de responsabilidade)**

Embora o sistema seja implementado em uma única aplicação, os módulos são organizados de forma lógica e desacoplada, permitindo maior independência entre os componentes internos. Isso facilita futuras expansões, como:

* **Módulo de Upload de Evidências:** Responsável por gerenciar o upload, validação e armazenamento de arquivos, como imagens e PDFs, seja no sistema de arquivos local ou em serviços externos (ex.: AWS S3).
* **Módulo de Relatórios:** Encapsula a geração e exportação de relatórios, permitindo filtrar e consolidar dados de projetos, cenários e execuções.

Essa abordagem promove:

* **Manutenção facilitada**, já que cada módulo possui uma responsabilidade bem definida dentro da aplicação.
* **Evolução contínua**, permitindo introduzir novas funcionalidades ou até mesmo transformar módulos em microsserviços no futuro, sem impacto nas partes críticas do sistema.
* **Reutilização de código**, como middlewares de autenticação, validações genéricas, serviços utilitários e integração com banco de dados ou armazenamento de evidências.

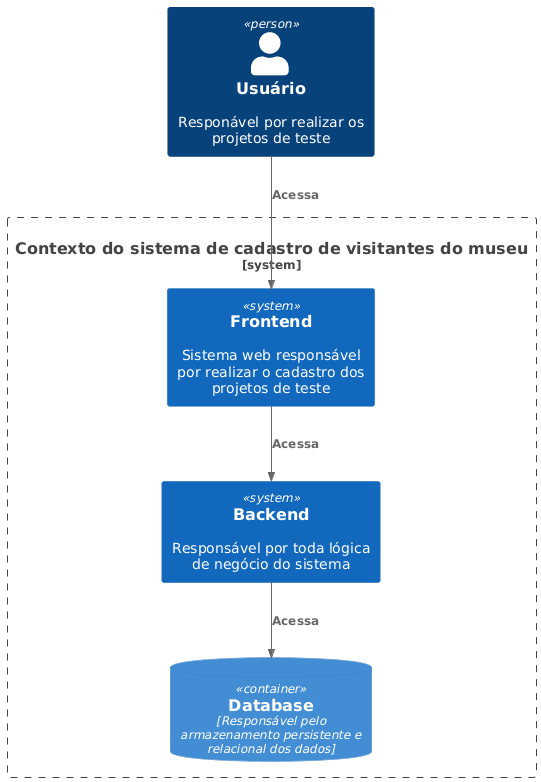
**Modelo C4 na Arquitetura**

O sistema foi modelado utilizando o C4 Model, que permite visualizar sua arquitetura em diferentes níveis de abstração:

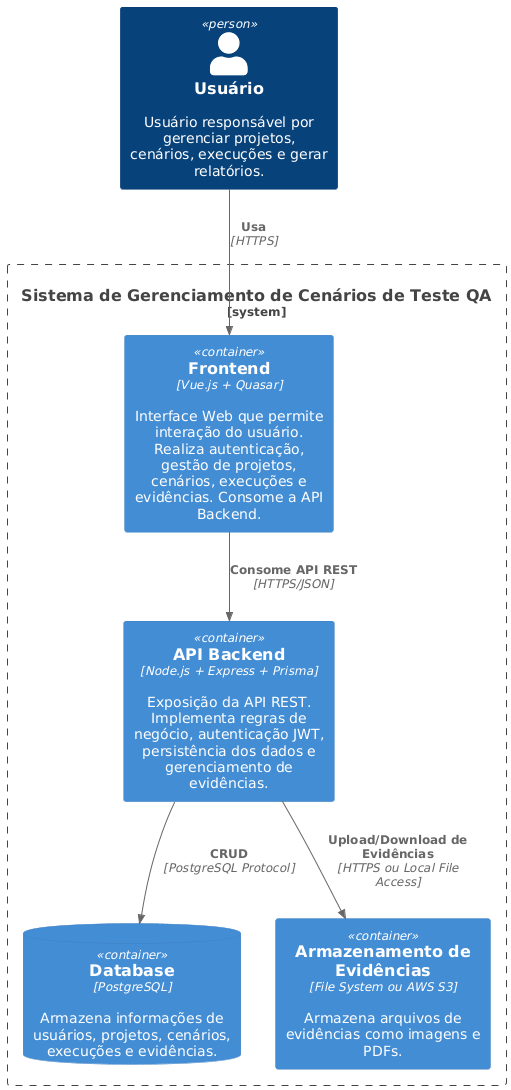
* Nível 1 – Contexto: Define a visão geral da plataforma QA.
* Nível 2 – Contêineres: Demonstra como o sistema é dividido em contêineres, como o frontend (Vue.js + Quasar), backend (Node.js + Express), banco de dados (PostgreSQL) e serviços auxiliares, como o serviço de relatórios e o serviço de upload de evidências.
* Nível 3 – Componentes: Detalha os componentes internos de cada contêiner, como os controllers (AuthController, ProjectController, TestCaseController, ExecutionController), os serviços e os modelos ORM (Prisma).

**Modelo C4:**

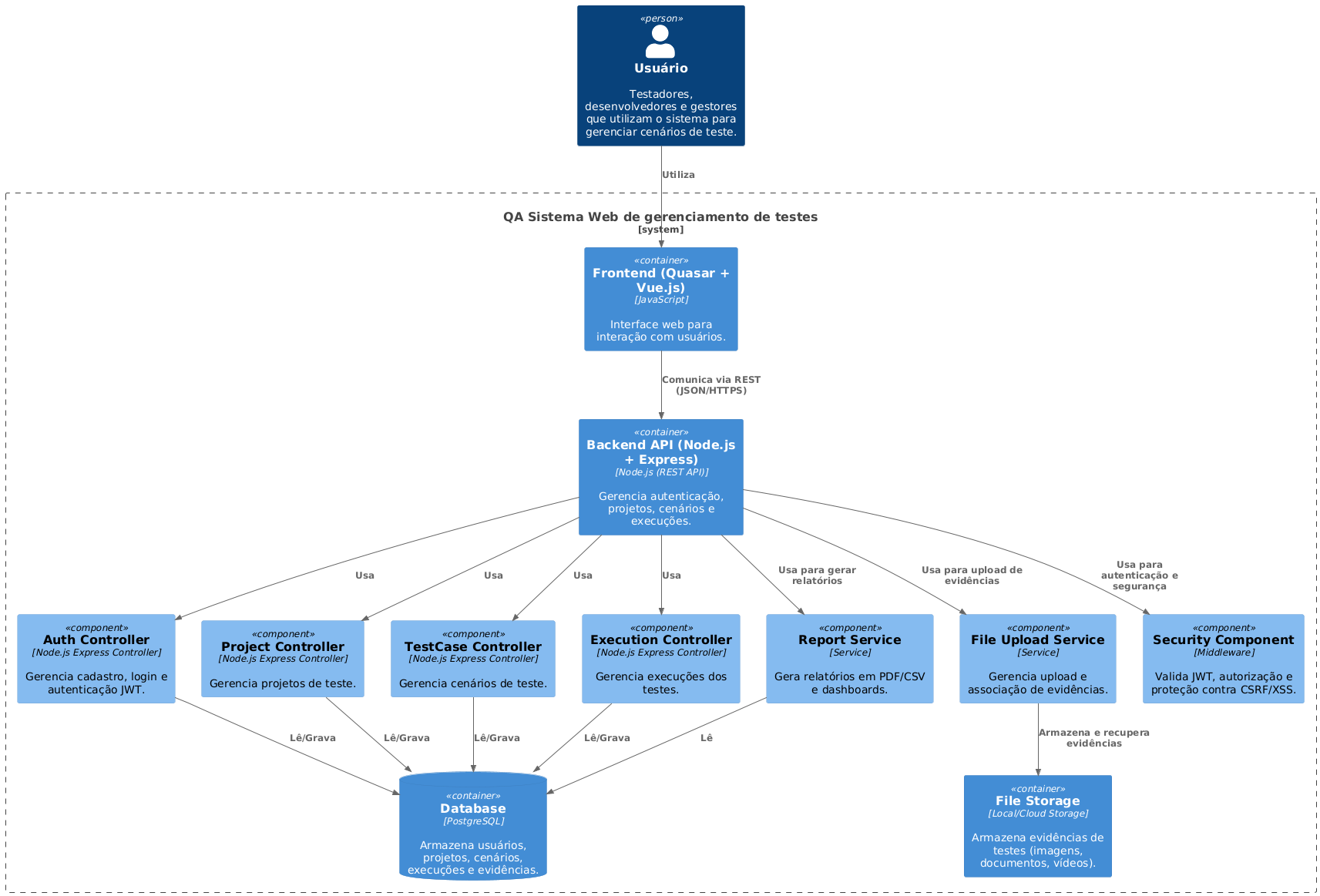
**Contexto:**



**Contêineres:**



**Componentes:**



**Stack Tecnológica**

**Linguagem de programação:**

* **JavaScript / TypeScript:** Utilizadas no frontend (Vue.js) e backend (Node.js). TypeScript garante tipagem estática e maior segurança no código, evitando erros em tempo de execução.
* **SQL (PostgreSQL):** Utilizada para manipulação de dados relacionais com alto desempenho, integridade e suporte a relacionamentos complexos.

**Frameworks e Bibliotecas:**

**Frontend (interface web):**

* **Vue.js:** Framework progressivo, com curva de aprendizado acessível e alta produtividade no desenvolvimento de interfaces dinâmicas.
* **Quasar Framework:** Baseado em Vue.js, permite criação rápida de interfaces responsivas com componentes prontos e compatibilidade mobile/desktop.

**Backend (lógica de negócio):**

* **Node.js:** Ambiente leve e escalável para execução de código JavaScript no servidor.
* **Express.js:** Framework web minimalista e flexível para construção de APIs RESTful.
* **Prisma ORM:** Mapeamento objeto-relacional (ORM) moderno e seguro, facilita a integração com PostgreSQL.

**Autenticação e Segurança:**

* **JWT (jsonwebtoken):** Geração de tokens seguros para autenticação de sessões.
* **Bcrypt.js:** Hash de senhas de forma segura e resistente a ataques.

**Upload e Armazenamento:**

* **Multer:** Middleware para upload de arquivos no backend.

**Relatórios:**

* **pdfkit / json2csv:** Bibliotecas para geração de relatórios em PDF e CSV.

**Ferramentas de Desenvolvimento e Gestão de Projeto**

* **Visual Studio Code:** IDE leve, com suporte a extensões de Vue, Node e integração com Git.
* **Git e Github:** Controle de versão, documentação e colaboração entre integrantes.
* **Postman:** Testes de APIs RESTful durante o desenvolvimento backend.
* **GitHub Projects:** Gestão de tarefas, sprints e acompanhamento do progresso do projeto.

**Banco de Dados:**

* **PostgreSQL:** Banco de dados relacional robusto, ideal para manter integridade entre entidades como cenários, execuções, usuários e evidências. Suporta relacionamentos complexos, controle de transações e segurança avançada.

**Considerações sobre testes**

Embora o sistema proposto seja voltado principalmente para o gerenciamento de testes manuais de QA, é essencial assegurar que a própria plataforma também seja testada de forma robusta, garantindo sua qualidade e confiabilidade como produto de software. Para isso, propõe-se uma estratégia de testes combinando abordagens manuais e automatizadas.

**Testes Automatizados**

Para garantir a integridade do sistema ao longo das iterações de desenvolvimento, serão implementados diferentes níveis de testes automatizados:

* **Testes Unitários:** Serão utilizados para validar funções e serviços isolados no backend com Jest, e nos componentes do frontend com Vue Test Utils. O objetivo é detectar falhas em unidades específicas do código logo nas fases iniciais.
* **Testes de Integração:** Serão aplicados principalmente para validar os endpoints da API e a comunicação entre os módulos do backend. Ferramentas como Supertest serão empregadas para simular requisições HTTP e verificar respostas, autorizações e persistência de dados.
* **Cobertura de Testes:** Será estipulada uma meta mínima de 80% de cobertura para rotas críticas e funções principais do sistema. Relatórios de cobertura serão gerados automaticamente durante o processo de integração contínua.

**Testes End-to-End (E2E)**

A simulação de fluxos completos de usuário será realizada por meio de testes end-to-end utilizando Cypress. Esses testes permitirão verificar, por exemplo:

* **Criação e login de usuários.**
* **Criação e execução de cenários de teste.**
* **Upload de evidências.**
* **Geração de relatórios.**

Além disso, os testes E2E serão integrados ao pipeline de CI/CD, garantindo que funcionalidades principais estejam operacionais após cada deploy.

**Testes Manuais**

Como objetivo principal da plataforma é justamente organizar cenários de testes manuais, essa camada também será aplicada ao próprio sistema. A equipe de QA utilizará o próprio sistema em desenvolvimento para registrar, executar e validar cenários de teste da plataforma

**Testes de Usabilidade**

Estão previstos testes de usabilidade com usuários reais (ex: colegas da área de QA e desenvolvimento) durante a fase de testes manuais do sistema. O objetivo é identificar:

* **Dificuldades de navegação ou entendimento**
* **Etapas confusas ou redundantes**
* **Melhorias de layout e acessibilidade**

Esses testes ajudarão a validar se a interface cumpre sua função de forma eficiente, minimizando o tempo de aprendizado e evitando erros operacionais.

**Considerações de Segurança**

A segurança é um aspecto crítico em qualquer sistema que lide com dados sensíveis, como credenciais de acesso, informações de usuários e arquivos enviados. No projeto de gerenciamento de cenários de teste QA, as considerações de segurança são fundamentais para garantir a integridade dos dados, a proteção contra acessos não autorizados e a confidencialidade das informações tratadas.

**Autenticação e Controle de Acesso**

Uma das primeiras camadas de segurança será o **controle de autenticação**, utilizando **JSON Web Tokens (JWT)** para garantir que apenas usuários autenticados tenham acesso às funcionalidades da plataforma. Os tokens JWT serão gerados após o login e utilizados para validar a identidade do usuário em cada requisição subsequente, com um tempo de expiração para reduzir os riscos de uso indevido. Além disso, o sistema adotará um controle de permissões baseado em papéis (por exemplo, Testador, Desenvolvedor, Gestor de QA) para garantir que os usuários acessem apenas as funcionalidades que lhes são permitidas.

**Criptografia de Dados Sensíveis**

As **senhas dos usuários** serão armazenadas de forma **criptografada** utilizando a biblioteca **bcrypt.js**. Isso assegura que mesmo em caso de violação de segurança no banco de dados, as senhas não poderão ser recuperadas ou utilizadas de forma indevida. Para garantir a confidencialidade dos dados durante a comunicação entre cliente e servidor, todo o tráfego será protegido com **HTTPS**, utilizando SSL/TLS para a encriptação dos dados em trânsito.

**Proteção Contra Vulnerabilidades Comuns**

Diversas vulnerabilidades comuns em aplicações web serão mitigadas:

* **Proteção contra Cross-Site Scripting (XSS):** A entrada de dados será cuidadosamente validada e sanitizada para evitar que scripts maliciosos sejam injetados nas páginas, protegendo tanto o sistema quanto os usuários finais.
* **Proteção contra Cross-Site Request Forgery (CSRF):** O sistema utilizará tokens anti-CSRF em todas as requisições que modificam dados no servidor, garantindo que as solicitações só possam ser realizadas a partir da própria interface de usuário e não por ataques externos.
* **Prevenção contra SQL Injection:** Embora o uso do ORM **Prisma** reduza significativamente o risco de injeção SQL, validações adicionais e práticas seguras serão adotadas para garantir que todas as entradas de dados sejam corretamente tratadas, impedindo a execução de comandos maliciosos no banco de dados.
* **Validação e Sanitização de Arquivos:** Os arquivos enviados pelos usuários, como evidências de teste, serão validados quanto ao tipo e tamanho. Apenas formatos permitidos (imagens, vídeos e documentos) serão aceitos, evitando o upload de arquivos executáveis ou maliciosos. Além disso, os nomes dos arquivos serão sanitizados para prevenir ataques por injeção ou manipulação de caminho.

**Gerenciamento de Sessões**

Para garantir a segurança durante as sessões de usuário, o sistema implementará **controle de sessões** robusto, com tokens JWT com tempo de expiração configurável. Caso o usuário permaneça inativo por um período prolongado, a sessão será automaticamente invalidada, exigindo novo login. Além disso, o **logout** será completamente gerenciado, garantindo que a sessão seja encerrada corretamente.

**Monitoramento e Auditoria**

A plataforma também será projetada para realizar o **monitoramento constante** de atividades suspeitas. Serão registrados logs de eventos críticos, como falhas de autenticação, tentativas de acesso não autorizado e alterações de dados. Esses logs permitirão a **auditoria** do sistema e poderão ser utilizados em investigações, caso necessário.

**Backup e Recuperação de Dados**

A segurança também envolve garantir que os dados possam ser recuperados em caso de falha. O sistema realizará **backups regulares** do banco de dados e das evidências de testes armazenadas, protegendo esses dados contra perdas acidentais ou ataques de ransomware.

**Próximos Passos**

Com a conclusão deste documento RFC e a definição clara dos objetivos, escopo e arquitetura da aplicação, os próximos passos para o desenvolvimento do projeto serão organizados em duas grandes etapas:

**Desenvolvimento Inicial**

* **Configuração do Ambiente de Desenvolvimento:** Instalação de ferramentas como Node.js, PostgreSQL, VSCode, Vue.js com Quasar, e bibliotecas auxiliares (Prisma, Express, JWT, etc.).
* **Criação do Repositório GitHub:** Estruturação inicial do repositório, documentação README e definição dos papéis da equipe (Product Owner, Dev, QA, etc.).
* **Implementação do Backend:**
  + Cadastro e autenticação de usuários com JWT.
  + Estrutura básica de gerenciamento de projetos e cenários de teste.
* **Desenvolvimento do Frontend:**
* Telas de login, cadastro, dashboard e criação de cenários.
* **Teste Manual e Correções:** Validação das rotas principais, persistência no banco e segurança básica.

**Expansão e Validação Final**

* **Refinamento da Interface e Usabilidade:** Melhorias visuais, acessibilidade, responsividade e testes com usuários.
* **Módulos Avançados:**
  + Upload e visualização de evidências de teste.
  + Histórico de execuções e exportação de relatórios em PDF.
  + Painel de métricas e indicadores em tempo real.
* **Segurança e Performance:**
  + Implementação de controle de permissões por perfil de usuário.
  + Adoção de práticas de segurança avançadas (CSRF, XSS, validação de arquivos).
* **Documentação Final:**
  + Elaboração do manual do usuário.
  + Atualização da documentação técnica e do repositório.
* **Teste de Aceitação com Feedback:** Sessão de validação com professores e colegas para ajustes finais.
* **Publicação e Demonstração:** Apresentação pública do sistema completo e funcional como entrega do Portfólio II.

**Referências**

**Linguagens e Banco de Dados**

* JavaScript / TypeScript – Linguagens principais para desenvolvimento frontend e backend.
* SQL (PostgreSQL) – Banco de dados relacional utilizado para armazenamento persistente.

**Frameworks e Bibliotecas Frontend**

* [Vue.js](https://vuejs.org/) – Framework progressivo para criação da interface do usuário.
* [Quasar Framework](https://quasar.dev/) – Framework baseado em Vue.js para desenvolvimento de interfaces responsivas.

**Frameworks e Bibliotecas Backend**

* [Node.js](https://nodejs.org/) – Ambiente de execução para JavaScript no lado do servidor.
* [Express.js](https://expressjs.com/) – Framework web minimalista para APIs RESTful.
* [Prisma ORM](https://www.prisma.io/) – ORM para integração com PostgreSQL de forma tipada e segura.

**Autenticação e Segurança**

* [JWT (jsonwebtoken)](https://www.npmjs.com/package/jsonwebtoken) – Geração e validação de tokens de autenticação.
* [bcrypt.js](https://www.npmjs.com/package/bcryptjs) – Criptografia de senhas.

**Upload de Arquivos e Evidências**

* [Multer](https://www.npmjs.com/package/multer) – Middleware para upload de arquivos no Node.js.

**Geração de Relatórios**

* [pdfkit](https://www.npmjs.com/package/pdfkit) – Biblioteca para geração de relatórios em PDF.
* [json2csv](https://www.npmjs.com/package/json2csv) – Conversão de dados JSON para formato CSV.

**Desenvolvimento**

* [Visual Studio Code](https://code.visualstudio.com/) – Ambiente de desenvolvimento integrado (IDE).
* [Postman](https://www.postman.com/) – Plataforma para testes de APIs.
* [Git](https://git-scm.com/) e [GitHub](https://github.com/) – Controle de versão e colaboração em equipe.
* GitHub Projects – Ferramenta de gerenciamento de tarefas e sprints ágeis.

**Testes:**

* [Jest](https://jestjs.io/) - Framework de testes unitários para JavaScript/TypeScript, utilizado principalmente no backend para testar funções e serviços isolados.
* [Vue Test Utils](https://v1.test-utils.vuejs.org/) **-** Biblioteca oficial de testes unitários para componentes Vue.js. Permite montar, renderizar e simular interações em componentes de interface.
* [Supertest](https://github.com/ladjs/supertest) - Biblioteca de testes de integração voltada para APIs Node.js (Express), permitindo simular requisições HTTP e validar respostas.
* [Cypress](https://www.cypress.io/) **-** Framework de testes end-to-end que executa testes no navegador, simulando a experiência real do usuário.
* [Playwright](https://playwright.dev/) - Biblioteca moderna para testes E2E e automação de browsers, compatível com múltiplos navegadores.

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**