Visão Geral do Projeto

Objetivo: Desenvolver um aplicativo de gestão acadêmica para facilitar a organização de aulas, turmas, matrículas, presenças e comunicação entre professores e alunos.

Tecnologias Principais:

- Backend: Spring Boot (com Spring Security, JWT, REST API).
- Frontend: React Native, Componentes reutilizáveis, design responsivo.
- Ferramentas: Trello, GitHub, Postman.

Histórias de Usuário

ID	História	Critérios de Aceitação
1	Visualização de Horário de Aulas	Exibir aulas por dia, diferenciar modalidades (presencial/remota), mostrar detalhes da disciplina.
2	Cadastro de Turmas	Permitir criação de turmas com validação de conflitos de horários.
3	Matrícula em Turmas	Listar turmas disponíveis, bloquear matrículas conflitantes.
4	Consulta de Faltas e Presenças	Exibir porcentagem de presença por disciplina.
5	Lançamento de Presenças	Registrar presenças em massa com data automática.

6	Envio de Avisos para a Turma	Notificar alunos via texto e anexos.
7	Responsividade da Interface	Funcionar em dispositivos móveis e desktops.
8	Cadastro e Login Seguro	Autenticação segura com validação de email e senha.
9	Dashboard de Desempenho	Exibir notas, frequência e status em tempo real.
10	Consulta à Documentação	Permitir download de ementas e cronogramas.

Cerimônias de scrum

Sprint 1 (06/12 - 20/12)

- **Cerimônias**:
  - *Sprint Planning (06/12)*:
    - Estruturação inicial do backend com Spring Boot.
    - Definição do modelo de dados e criação de wireframes.
  - *Daily Scrums*:
    - Discussões sobre estrutura de pacotes no Spring e padrão REST.
  - *Sprint Review (20/12)*:
    - Backend com rotas básicas funcionando.
    - Wireframes aprovados.
  - *Retrospective (21/12)*:
    - **Aprendizado**: A comunicação entre front e back precisa melhorar.
    - **Ação**: Padronizar rotas e autenticação.

Sprint 2 (21/12 - 03/01)

- **Cerimônias**:
  - *Sprint Planning (21/12)*:
    - Foco em autenticação (Spring Security + JWT) e CRUD de turmas.
  - *Daily Scrums*:
    - Ritmo reduzido devido às festas.
    - Primeira integração frontend/backend.
  - *Sprint Review (03/01)*:
    - Login funcional e CRUD básico de turmas.
  - *Retrospective (04/01)*:
    - **Problema**: Conflitos de CORS na integração.
    - **Ação**: Revisar configurações do Spring Security.

Sprint 3 (04/01 - 17/01)

- **Cerimônias**:
  - *Sprint Planning (04/01)*:
    - Desenvolvimento do dashboard e APIs de disciplinas/professores.
  - *Daily Scrums*:
    - Implementação de novas rotas REST.
  - *Sprint Review (17/01)*:
    - Dashboard funcional com dados em tempo real.
  - *Retrospective (18/01)*:
    - **Aprendizado**: Falta de testes automatizados.
    - **Ação**: Adicionar testes unitários na próxima sprint.

Sprint 4 (18/01 - 31/01)

- **Cerimônias**:
  - *Sprint Planning (18/01)*:
    - Preparação para versão de testes (telas completas e integração final).
  - *Daily Scrums*:
    - Testes intensivos com Postman e Spring Test.
  - *Sprint Review (31/01)*:
    - Todas as telas integradas e funcionais.
  - *Retrospective (01/02)*:
    - **Problema**: Revisão de código insuficiente.
    - **Ação**: Implementar fluxo de pull requests.

Sprint 5 (01/02 - 14/02)

- **Cerimônias**:
    - *Sprint Planning (01/02)*:
        - Finalização do frontend e documentação da API.
    - *Daily Scrums*:
        - Ajustes de UI/UX e melhorias nas respostas REST.
    - *Sprint Review (14/02)*:
        - Frontend 100% completo.
        - Documentação da API em 90%.
    - *Retrospective (15/02)*:
        - **Aprendizado**: Retrabalho evitável devido a falhas no planning.
        - **Ação**: Reforçar validação durante planejamento.

Sprint 6 (15/02 - 28/02)

- **Cerimônias**:
    - *Sprint Planning (15/02)*:
        - Foco em estabilidade e documentação final.
    - *Daily Scrums*:
        - Testes finais e revisão de endpoints.
    - *Sprint Review (28/02)*:
        - Projeto finalizado e apresentado à UFPE.
    - *Retrospective (01/03)*:
        - **Aprendizado**: Cerimônias precisam de mais disciplina.
        - **Ação**: Criar calendário fixo para reuniões.

Sprint 7 (02/03 - 17/03) - Pós-Entrega

- **Cerimônias**:
    - *Sprint Planning (02/03)*:
        - Otimização de performance e correção de bugs críticos.
    - *Daily Scrums*:
        - Resolução de problemas de latência e CORS em produção.
    - *Sprint Review (17/03)*:
        - Sistema 15% mais rápido após ajustes.
    - *Retrospective (18/03)*:
        - **Ação**: Implementar monitoramento com New Relic.

Sprint 8 (18/03 - 31/03) - Nova Feature

- **Cerimônias**:
    - *Sprint Planning (18/03)*:
        - Implementação do módulo de Faltas/Presenças (User Story #4).
    - *Daily Scrums*:
        - Desenvolvimento de gráficos de frequência.

    - *Sprint Review (31/03)*:
        - Módulo integrado ao dashboard.
    - *Retrospective (01/04)*:
        - **Ação**: Melhorar responsividade para mobile.

Documento de Postmortem

Reunião 1

Projeto: GenCIN
Data: 15/03/2024
Participantes: Equipe de Desenvolvimento, PO, Scrum Master

1. Objetivo da Reunião
Analisar os desafios e aprendizados da **Sprint 7 (Pós-Entrega)**, focada em otimização de performance e correção de bugs críticos.

2. O que Deu Certo
- **Redução de Latência**:
    - Implementação de cache com Redis reduziu o tempo de resposta em 15%.
- **Correção de Bugs**:
    - Problemas de CORS em produção resolvidos com ajustes no Spring Security.
- **Monitoramento**:
    - New Relic integrado para tracking de performance em tempo real.

3. Problemas Encontrados
- **Conflitos de Autenticação**:
    - Tokens JWT expirando prematuramente em ambientes de teste.
    - *Causa*: Configuração incorreta do tempo de vida do token.
- **Comunicação com Frontend**:
    - Atrasos na sincronização de dados devido a endpoints mal documentados.

4. Lições Aprendidas
- **Validação de Configurações**:
    - Testar exaustivamente ambientes antes do deploy (ex: tempo de tokens).
- **Documentação**:
    - Manter um changelog detalhado das alterações no backend.

5. Ações para o Futuro

- Criar um ambiente de staging idêntico à produção.
- Padronizar templates de documentação de APIs.

Pull Requests

Os pull requests podem ser encontrados neste link.

Documentação de Arquitetura de Software da API - C4 Model

1. Nível 1: Contexto

Sistema: GenCin (API para gestão acadêmica)
Principais Usuários:

- **Alunos**: Consultam atividades e sessões.
- **Professores**: Gerenciam aulas e atividades.
- **Administradores**: Configuram segurança e usuários.

Integrações:

- Autenticação via JWT (Spring Security).
- Documentação da API via Swagger.

2. Nível 2: Contêineres

Contêiner	Tecnologia	Descrição
API GenCin	Spring Boot (Java)	Backend com módulos de usuários, aulas, segurança.
Banco de Dados	PostgreSQL	Armazena entidades como Aluno, Professor, Aula.
Swagger UI	OpenAPI 3.0	Documentação interativa da API (em /swagger-ui).

3. Nível 3: Componentes

Principais Módulos:

1. **Módulo de Usuários**:
    - Entidades: Aluno, Professor (herdam de Usuario).
    - Endpoints: CRUD de usuários e autenticação.

2. **Módulo de Aulas/Atividades**:
    - Entidades: Aula, Atividade, Sessao.
    - Endpoints: Gerenciamento de cronogramas e registros.
3. **Módulo de Segurança**:
    - Configuração: Security (Spring Security + JWT).
    - Funcionalidade: Autenticação e autorização.
4. **Módulo de Documentação**:
    - Configuração: SwaggerConfig.
    - Funcionalidade: Geração automática da API docs.

4. Nível 4: Código

```
src/
├── main/
│   ├── java/com/GenCin/GenCin/
│   │   ├── Aluno/            # Entidade e repository de alunos
│   │   ├── Config/           # SwaggerConfig e Security
│   │   ├── Usuario/          # Classe base Usuario
│   │   ├── Aula/             # Lógica de aulas
│   │   └── Security/         # Configurações de JWT
│   └── resources/
│       ├── application.yml   # Configurações do banco de dados
│       └── static/docs/      # Swagger UI
```

5. Decisões de Arquitetura

- **Padrão RESTful**: Endpoints seguem convenções REST.
- **Segurança**:
    - JWT para autenticação stateless.
- **Documentação**:
    - Swagger para auto-documentação dos endpoints.

6. Dependências Principais

- **Spring Boot**: Web, Security, Data JPA.
- **Lombok**: Simplificação de getters/setters.
- **PostgreSQL Driver**: Conexão com o banco.
- **Swagger**: springdoc-openapi-starter-webmvc-ui.

Documentação de Arquitetura de Software da APP - C4 Model

1. Contexto

Objetivo: Mostrar o sistema como um todo e suas interações com os usuários e sistemas externos.

O **GenCin-APP** é um aplicativo mobile desenvolvido em **React Native** para gerenciar aulas, informações acadêmicas e usuários. Ele se comunica com um sistema de backend (provavelmente uma **API** que gerencia os dados).

Usuários:

- **Aluno**: Pode buscar aulas e se matricular nelas.
- **Professor**: Pode cadastrar e gerenciar aulas.

Sistema Externo:

- **GenCin-API**: Backend que fornece os dados necessários para o funcionamento do aplicativo (usuários, aulas, matriculas, etc.).

2. Contêineres

Objetivo: Detalhar os contêineres dentro do sistema e como eles se comunicam.

- **GenCin-APP (Mobile App)**: A aplicação React Native que interage com a GenCin-API para obter dados.
    - Responsável por:
        - Exibir informações de aulas.
        - Permitir que o usuário se matricule.
        - Permitir que o professor cadastre novas aulas.
        - Exibir informações de login.
- **GenCin-API (Backend)**: API RESTful que lida com requisições do aplicativo.
    - Responsável por:
        - Gerenciar os dados de aulas, matrículas e usuários.
        - Fornecer endpoints REST para interação com o app.

- **Banco de Dados**: Banco de dados que armazena os dados do sistema (aulas, usuários, matrículas, etc.).

3. Componentes

Objetivo: Detalhar os principais componentes dentro de cada contêiner.

GenCin-APP (React Native):

- **Tela de Login**: Responsável por autenticar os usuários (aluno ou professor).
- **Tela de Cadastro**: Permite a criação de novos usuários.
- **Tela Home**: Página inicial que apresenta as opções para os alunos e professores, como "Procurar Aulas" ou "Cadastrar Aulas".
- **Tela de Aulas**: Permite aos professores cadastrarem novas aulas e aos alunos visualizarem as aulas disponíveis.

GenCin-API (Backend):

- **Autenticação**: Módulo responsável por realizar o login e validar credenciais.
- **Gerenciamento de Aulas**: Módulo que gerencia as aulas (criação, listagem, edição e remoção).
- **Gerenciamento de Matrículas**: Módulo que gerencia as matrículas dos alunos nas aulas.

4. Código

Objetivo: Mostrar como o código está organizado no repositório.

O repositório tem a seguinte estrutura:

- **/assets**: Contém arquivos estáticos, como imagens e ícones que são usados no aplicativo.
- **/functions**: Funções auxiliares para validar formulários, manipular dados ou interagir com o sistema de backend.

- **/src**: A pasta principal que contém o código da aplicação e Contém as telas principais da aplicação, como Login, Cadastro e Aulas.

- **App.js**: Arquivo principal que inicializa o aplicativo.

- **index.js**: Ponto de entrada do aplicativo.

- **README.md**: Documento de informações sobre o projeto.