Relatório Técnico - Sistema de Agenda de Estudos

Integrantes: Layon Walker Tobias, Marcos Vinícius Matias e Ruan Victor Henrique
Professor: Jesimar da Silva Arantes
Turma: 14B
Trabalho Prático 02 - Programação Web
Link do Projeto

INTRODUÇÃO

O seguinte relatório descreve o desenvolvimento da segunda etapa do projeto da disciplina de Programação Web, que consiste na reimplementação do sistema "Agenda de Estudos" utilizando tecnologias livres. O objetivo principal deste trabalho foi migrar a aplicação, originalmente desenvolvida em uma arquitetura monolítica com o framework Django (Python), para uma arquitetura distribuída (Cliente-Servidor). Para tanto, foram utilizadas as tecnologias Node.js para o backend, react.js para o frontend e SQLite para o banco de dados. O escopo funcional do sistema foi mantido e expandido, englobando o gerenciamento de vida acadêmica (CRUD de matérias, tarefas e provas), além da implementação de novas funcionalidades de produtividade e integração com serviços externos.

TECNOLOGIAS UTILIZADAS

Para a execução desta etapa, foram selecionadas um conjunto de tecnologias modernas baseadas no ecossistema JavaScript, visando alta performance e desacoplamento entre interface e regras de negócio.

Backend (Servidor e API)

Node.js & Express (v5.1): Utilizados para a construção da API REST, gerenciamento de rotas e lógica de negócios.

Sequelize: ORM (Object-Relational Mapping) escolhido para abstrair a comunicação com o banco de dados.

SQLite: Sistema gerenciador de banco de dados relacional, mantido pela portabilidade e simplicidade de configuração.

JWT (JSON Web Token) & Bcrypt: Utilizados para implementação manual de autenticação segura e criptografia de senhas.

Frontend (Interface do Usuário)

React.js (via Vite): Biblioteca para construção de interfaces baseadas em componentes.

Bootstrap 5: Framework CSS utilizado para estilização e responsividade, garantindo consistência visual com a versão anterior do projeto.

Axios: Cliente HTTP baseado em *promises* para consumo da API.

Bibliotecas Auxiliares: *React Router Dom* (v7) para roteamento no lado do cliente, *FullCalendar* para gestão de eventos e *Canvas Confetti* para gamificação.

MODELAGEM E IMPLEMENTAÇÃO

A implementação do sistema exigiu uma reestruturação completa da arquitetura, abandonando o padrão MVT (*Model-View-Template*) do Django em favor de uma arquitetura API RESTful.

Arquitetura e Comunicação Diferentemente da versão anterior, onde o HTML era renderizado no servidor (*Server-Side Rendering*), o projeto atual opera com separação total de responsabilidades. O backend expõe endpoints JSON que são consumidos pelo frontend via requisições assíncronas (AJAX) gerenciadas pelo Axios.

Persistência de Dados (ORM) A modelagem de dados foi recriada utilizando o **Sequelize**. As entidades principais (*Usuários, Matérias, Tarefas, Agenda*) foram definidas através de *Schemas* em JavaScript. Esta abordagem permitiu manter a integridade relacional do banco SQLite, definindo chaves estrangeiras e validações de tipos de dados antes da persistência, de forma análoga aos *Models* do Django.

Autenticação e Segurança Uma mudança significativa na implementação foi o sistema de autenticação. Enquanto o Django provê um sistema de sessões nativo, no Node.js foi implementada uma estratégia *stateless* com **JWT**. O fluxo consiste em:

1. O usuário envia credenciais (login/senha);
2. O servidor válida e retorna um token assinado;
3. O frontend armazena o token e o anexa ao cabeçalho (*Authorization*) de todas as requisições subsequentes.

Integrações Externas e Interface A implementação no frontend permitiu enriquecer a experiência do usuário com recursos dinâmicos não presentes na versão anterior:

Integração de APIs: O sistema consome a *Open-Meteo API* para exibir dados climáticos e a *NewsAPI* para notícias de tecnologia no Dashboard.

Interatividade: Foi implementado um "Modo Foco" (Pomodoro) e feedback visual com *confetti* ao concluir tarefas, demonstrando a capacidade do React de manipular o DOM de forma eficiente.

Breve análise comparativa (Django vs NodeJs+React)

Aspecto	Projeto Anterior (Django)	Projeto Atual (Node.js + React)
Arquitetura	Monolítica (Acoplada). O servidor processa a lógica e entrega o HTML pronto.	Distribuída (Desacoplada). Backend fornece dados (JSON) e Frontend monta a interface.
Linguagem	Python (Tipagem dinâmica forte).	JavaScript (Tipagem dinâmica fraca) em ambas as pontas (*Fullstack*).

Banco de Dados	Gerenciado pelo Django ORM (implícito e integrado).	Gerenciado pelo Sequelize (biblioteca externa configurável).
Roteamento	Centralizado no arquivo urls.py	Híbrido: Rotas de API no Express e Rotas de Navegação no React Router.
Autenticação	Nativa (Session/Cookies). Abstraída pelo framework.	Manual (JWT). Exigiu implementação de geração, validação e armazenamento de tokens.
Desenvolvimento	Alta produtividade inicial devido aos recursos nativos ("baterias inclusas"), como o Admin.	Maior flexibilidade e controle sobre a arquitetura, porém com maior necessidade de configuração inicial.

Desafios durante a implementação

Durante o desenvolvimento, o principal desafio enfrentado foi a gestão de estados assíncronos no Frontend. Diferentemente do Django, onde os dados chegam prontos no template, no React foi necessário implementar Hooks (useEffect e useState) para gerenciar o ciclo de vida dos componentes enquanto as requisições à API eram processadas. Outro ponto de atenção foi a configuração do CORS (Cross-Origin Resource Sharing) no Node.js, necessário para permitir que o Frontend (porta 5173) se comunicasse com o Backend (porta 3000), uma configuração que não era exigida na arquitetura monolítica anterior.

Trabalhos Futuros

Como propostas para a evolução contínua do sistema, o grupo vislumbra a implementação de:

- **Visualização Kanban:** Implementação de uma interface *drag-and-drop* para gestão visual do status das tarefas.
- **Relatórios de Desempenho:** Geração de gráficos de atividade (heatmap) para que o estudante visualize sua constância nos estudos ao longo do semestre.
- **Deploy em Nuvem:** Hospedagem da aplicação em serviços como Vercel (Frontend) e Render (Backend) para acesso remoto.

Conclusões

Diante da implementação da aplicação, tivemos um desenvolvimento essencial para captar as tecnologias e sua maneira de utilização. `Podemos concluir também que a stack JavaScript oferece maior liberdade para a criação de interfaces ricas e interativas (*Single Page Applications*), permitindo funcionalidades como atualizações em tempo real e gamificação com maior facilidade. Por outro lado, o Django mostrou-se mais eficiente para o desenvolvimento rápido de funcionalidades padrão (CRUD e Admin) devido ao seu alto nível de abstração. A migração foi bem-sucedida, resultando em um sistema robusto, moderno e com melhor experiência de usuário.