Implementação Parcial (Protótipo Funcional)

Objetivo

Entregar uma primeira versão **funcional e executável** do sistema, cobrindo os casos de uso principais e validando se a arquitetura planejada realmente funciona na prática. Este protótipo deve mostrar a **integração entre módulos, aplicação de POO e uso inicial de padrões de projeto**.

Estrutura do Entregável

Implementação Parcial (Protótipo Funcional)

Objetivo: Entregar uma primeira versão funcional e executável do sistema, cobrindo os casos de uso principais e validando se a arquitetura planejada realmente funciona na prática. Este protótipo deve mostrar a integração entre módulos, aplicação de POO e uso inicial de padrões de projeto.

1. Identificação

- **Celso Vieira:** Desenvolvedor Frontend e Analista técnico
- **Erlano Benevides:** Desenvolvedor Fullstack
- **Guilherme Alves:** Desenvolvedor Frontend e Analista técnico
- **Marcos Barros:** Desenvolvedor Fullstack
- **Rafael Maciel:** Scrum Master e Desenvolvedor Fullstack

2. Casos de Uso Implementados

Dois fluxos principais e essenciais do sistema foram implementados e estão funcionais:

- **UC01 – Gerenciamento de Usuários e Autenticação (Professor e Coordenador)**
    - **Descrição:** O sistema permite o cadastro completo (CRUD) de Professores e Coordenadores. Cada tipo de usuário possui dados específicos e sua persistência é tratada de forma transacional, garantindo a integridade entre a superclasse Usuario e as subclasses. Adicionalmente, foi implementada uma rota de autenticação (/auth/login) que verifica as credenciais (e-mail e senha com hash bcrypt) e retorna um token de acesso (JWT) para sessões seguras.
- **UC02 – Fluxo Completo de Solicitação de Reposição**
    - **Descrição:** Este é o caso de uso central do MVP. O fluxo implementado inclui:

1. **Notificação de Falta:** O Coordenador notifica um Professor sobre sua ausência via e-mail (disparado pela API).
2. **Criação da Solicitação:** O Professor cria uma solicitação de reposição, que é salva no banco de dados com status "Pendente".
3. **Coleta de Assinaturas:** O sistema automaticamente envia e-mails para todos os alunos da turma com um link para um Google Form.
4. **Processamento de Respostas:** As respostas do formulário acionam um webhook (via Google Apps Script) que notifica o backend. Cada resposta é salva e o contador de concordâncias é atualizado.
5. **Verificação de Quórum:** A cada resposta, o sistema verifica se o quórum de 75% foi atingido. Em caso positivo, o status da solicitação muda para "Aguardando Aprovação" e uma notificação é gerada para o Coordenador.
6. **Decisão do Coordenador:** O Coordenador pode aprovar ou negar a solicitação, disparando e-mails de confirmação ou cancelamento para todos os envolvidos (Professor, Alunos e Nutricionista).

3. Estrutura do Código

O projeto foi organizado em uma arquitetura de camadas (Layered Architecture) para garantir a separação de responsabilidades e a manutenibilidade do código.

- **config/:** Contém arquivos de configuração, como a conexão com o banco de dados (db.js).
- **constants/:** Armazena valores constantes, como os status de solicitação (Enum SolicitacaoStatus.js).
- **controller/:** Responsável por receber as requisições HTTP, validar entradas básicas e delegar a lógica de negócio para a camada de serviço.
- **exceptions/:** Define classes de erro customizadas para um tratamento de exceções mais específico. (Não usamos ainda)
- **model/:** Contém as classes que representam as entidades do sistema (ex: Usuario.js, Professor.js).
- **persistence/:** A camada de acesso a dados (Repository Pattern). É a única camada que interage diretamente com o banco de dados, executando queries SQL.
- **routes/:** Define os endpoints da API, mapeando URLs e métodos HTTP para as funções nos controllers.
- **services/:** Contém a lógica de negócio principal da aplicação. Orquestra as chamadas aos repositórios e executa as regras do sistema.

Ajuste na Arquitetura: A principal evolução em relação ao documento de planejamento foi a formalização da **Camada de Serviço (services/)**, que não estava explícita inicialmente. Esta camada foi criada para abrigar a lógica de negócio, desacoplando-a dos controllers e tornando o sistema mais robusto e testável.

4. Persistência de Dados

Os dados são salvos em um banco de dados **PostgreSQL relacional**, hospedado na plataforma **Supabase**. A interação com o banco é feita pela camada de persistência (persistence/), que utiliza a biblioteca pg do Node.js para executar queries SQL puras de forma segura, com o uso de consultas parametrizadas para prevenir SQL Injection.

Exemplo de Código (Salvar um professor):

```
// persistence/ProfessorRepository.js

async salvar(professorData) {
	const { nome, email, matricula, senha, disciplinas } = professorData;
	const client = await db.pool.connect();

	try {
		await client.query('BEGIN');
		// 1. Cria a entrada na tabela 'usuario'
		const usuarioSalvo = await UsuarioRepository.salvar({ nome, email, tipo:
'Professor' }, client);
		 // 2. Insere na tabela 'professor'
		const professorSql = 'INSERT INTO professor (matricula_professor,
id_usuario, senha) VALUES ($1, $2, $3)';
		await client.query(professorSql, [matricula, usuarioSalvo.idUsuario, senha]);
		// ... (lógica para disciplinas)
		await client.query('COMMIT');
		return new Professor(/*...*/);
	} catch (error) {
		await client.query('ROLLBACK');
		throw error;
	} finally {
		client.release();
	}
}
```

5. Aplicação de POO (na prática)

- **Herança:** A principal aplicação de herança está na modelagem dos usuários, onde Professor.js, Coordenador.js e Aluno.js herdam da superclasse Usuario.js.

- **Enums:** Utilizamos um objeto congelado para simular uma enumeração, garantindo um conjunto fixo de valores para o status das solicitações.

6. Padrões de Projeto (Implementação inicial)

- **Repository:** Toda a camada persistence/ implementa o padrão Repository, isolando a lógica de acesso a dados do resto da aplicação.

- **Singleton:** A conexão com o banco de dados é gerenciada como um Singleton para garantir que uma única instância do pool de conexões seja utilizada em toda a aplicação, otimizando recursos.
  JavaScript

7. Interface Inicial

A interface para interagir com este protótipo é a **API RESTful**. Não há interface gráfica (GUI/web) implementada neste estágio. A API pode ser testada e operada através de ferramentas como **Insomnia** ou **Postman**, seguindo a documentação de rotas definida no arquivo README.md do projeto.

8. Tratamento de Exceções

O sistema utiliza blocos try...catch nos Controllers para capturar erros vindos das camadas de serviço e persistência. Erros de negócio (ex: e-mail já cadastrado) e erros de autenticação são tratados com classes de exceção customizadas (ex: AutenticacaoInvalidaException), permitindo o retorno de status HTTP específicos (como 400 ou 401). Outros erros inesperados resultam em um status 500 Internal Server Error, com o erro detalhado sendo logado no console do servidor para depuração.

JavaScript

```
// controller/AuthController.js

async login(req, res) {
	try {
		// …
		} catch (error) {
			if (error.status) { // Verifica se é uma exceção customizada com status
			return res.status(error.status).json({ message: error.message });
		} else {
			console.error(error);
			res.status(500).json({ message: 'Ocorreu um erro inesperado no
servidor.' });
		}
	}

}
```

9. Guia de Execução

- **Requisitos:**

1. Node.js (v18 ou superior)
2. npm (gerenciador de pacotes)
3. Uma ferramenta de teste de API (Insomnia, Postman)

- **Como Executar:**
    1. Abra o Insomnia ou Postman.
    2. Utilize a URL base da aplicação hospedada no Render para todas as requisições: https://sistema-de-reposicao-de-aulas.onrender.com
    3. Siga a documentação de rotas presente no arquivo README.md para criar, listar, atualizar e deletar recursos.

10. Registro no GitHub

- **Link para o Repositório:** https://github.com/MarcosBarros1/Sistema-de-reposicao-de-aulas
- O repositório contém commits consistentes que refletem o progresso do desenvolvimento, com mensagens claras descrevendo cada etapa. A estrutura de pastas está organizada conforme descrito na Seção 3.