PUCRS - Projeto de Arquitetura de Software

T2 - Gerenciamento de Inscrições da Hackatona
Documento de Arquitetura de Software

Bruno Guerra, Eduardo Lessa, Guilherme Rizzotto, João de Leão, Pedro Portella

Histórico de Revisão

Data	Versão	Descrição	Autor
18/05/2020	1.0	Referente ao trabalho 1	
23/06/2020	2.0	Referente ao trabalho 2	

Sumário

Software Architecture Document

1. Introdução

1.1 Objetivo

Este documento tem o principal objetivo de apresentar as diferentes visões da arquitetura do sistema desenvolvido como método de avaliação da disciplina de "Projeto e Arquitetura de Software". No presente documento será apresentada a visão e decisões arquiteturais que foram escolhidas pelo grupo e os motivos que levaram a essa escolha.

1.2 Escopo

Este documento de arquitetura de software se aplica ao "Sistema de Avaliação da Hackatona" que será desenvolvido pela equipe composta por Bruno Guerra, Eduardo Lessa, Guilherme Rizzotto, João de Leão e Pedro Portella como um dos métodos de avaliação da disciplina de "Projeto e Arquitetura de Software", ministrada pela professora Ana Paula Bacelo para a obtenção do grau de bacharelado no curso de Engenharia de Software da PUCRS.

1.3 Definições, Acrônimos, e Abreviações

- Biblioteca: conjunto de arquivos e funções disponibilizados a fim de facilitar o desenvolvimento de aplicações e melhorar o reuso de código.

1.4 Referências

[KRU41]: The "4+1" view model of software architecture, Philippe Kruchten, November 1995, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

1.5 Visão Geral

De maneira simples, o software cujas escolhas arquiteturais estão apresentadas nesse documento tem o propósito de gerenciar o método de avaliação dos times da Hackatona de Engenharia de Software da PUCRS. O sistema apresenta três áreas diferentes que podem ser acessadas por grupos distintos. A primeira área é para os participantes que desejam sugerir times, a segunda consta todos os times para que os avaliadores possam escolher as notas e deixá-las salvas no sistema, a terceira serve para os administradores do evento poderem organizar os participantes e os times.

2. Representação da Arquitetura

2.1 Visão Arquitetural

O documento adotará uma estrutura baseada na visão de arquitetura " Kruchten 4+1" apresentada nas aulas da disciplina. Essa visão consiste em separar as diferentes formas de se olhar uma arquitetura em quatro categorias, visão lógica, de implementação, de processos e de implantação. Além disso também são utilizados casos de uso ou histórias de usuário para explorar melhor essas visões.

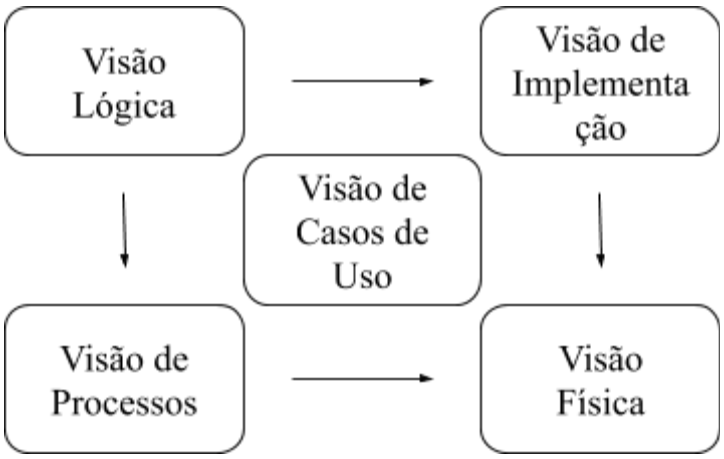

Figura: Representação visual da visão Kruchten"4+1"

Geralmente no desenvolvimento de um sistema, essas visões podem servir como um olhar mais amplo do mesmo e podem ser de responsabilidade de grupos distintos. Pelo fato do trabalho servir, não só como método de avaliação, mas como método de aprendizagem, todas as visões foram cuidadas por todos os membros do grupo. Isso tornou o desenvolvimento do trabalho menos eficiente, mas ajudou a todos a consolidar os conhecimentos.

As visões apresentadas podem ser melhor interpretadas considerando os seguintes aspectos delas:
A visão lógica mostra as abstrações fundamentais do sistema;
A visão de desenvolvimento apresenta a maneira que o software pode ser decomposto;
A visão de processo mostra como é composto o sistema em relação a processos interativos da execução;
A visão física mostra como os componentes do software são divididos e distribuídos na infraestrutura;
A visão de casos de uso mostra como as funcionalidades do sistema estão representadas;

2.2 Padrão Arquitetural

O grupo optou por utilizar o padrão arquitetural cliente-servidor por conta da natureza do sistema e dos requisitos definidos para o trabalho. Uma vez que é necessário criar times a partir de estudantes importados pelo administrador do sistema, bem como contar com a avaliação dos times realizadas por diversos professores, tais dados devem ser persistidos e acessíveis de diferentes locais. Assim, o servidor realiza as operações envolvendo o banco de dados, e envia os dados ao serem requisitados pelo cliente, recebendo como resposta apenas as informações necessárias.

Assim, é possível utilizar diferentes tecnologias para o frontend, uma vez que o backend retorna apenas os dados necessários, e não uma página estática. Além disso, com essa abordagem, é possível desenvolver um SPA (single-page application), melhorando a usabilidade e apresentação para os usuários. Por fim, a arquitetura permite uma melhor manutenção do projeto, além de permitir melhor uso de recursos em cada camada do sistema.

Adotamos três padrões de projeto bem conhecidos para solucionar particularidades do sistema e que são comumente utilizados ao criar aplicações com a biblioteca React, são eles:

State - Padrão muito utilizado no desenvolvimento frontend com a biblioteca React. Ele permite que um objeto altere seu comportamento quando seu estado interno muda. Foi usado ao longo de todo código do frontend da nossa aplicação para alterar os estados dos componentes criados na aplicação.

Facade - Fornece uma interface simplificada para uma biblioteca ou qualquer conjunto complexo de classes. Serve como um encapsulamento de um sistema complexo com um conjunto de muitos objetos de alguma biblioteca ou framework sofisticado com intuito de facilitar o uso dos recursos. Portanto, expõe um método de uma classe que irá acionar os serviços necessários para executar a requisição solicitada. É utilizado na inicialização do server no nosso código de backend.

Singleton - Permite a você garantir que uma classe tenha apenas uma instância provendo um ponto de acesso global para essa instância, isso é muito interessante no controle de acesso à um recurso compartilhado (como o uso de uma base de dados ou de um arquivo).

3. Objetivos e Restrições Arquiteturais

Neste sistema temos apenas um quesito que causa impacto na arquitetura. Esse quesito é o estilo de projeto que o grupo decidiu implementar. O projeto usa de React, uma biblioteca JavaScript com foco na criação de interfaces de usuário. O código em React é composto de entidades chamadas componentes que alteram a visão de implementação e visão lógica do projeto. Um dos objetivos propostos pelo segundo trabalho da disciplina é o uso de três padrões de projeto no código. Assim, conforme a versão 2.0 do presente documento, serão apresentadas, na visão de implementação, os padrões utilizados.

4. Visão de Casos de Uso

Um rascunho do projeto pode ser desenvolvido desde o começo graças aos requisitos funcionais apresentados pelo enunciado. Os casos de uso que foram gerados a partir dos requisitos são os seguintes

Caso de Uso	Descrição
UC01: Sugerir time	Sugestão e criação de times por meio dos participantes
UC02: Editar time	Edição dos integrantes de um time (adição e remoção) por meio dos participantes
UC03: Cadastrar participantes	Cadastro de participantes inscritos no sistema por meio dos administradores
UC04: Deletar times	Remoção de times sugeridos por meio dos administradores
UC05: Avaliar times	Avaliação de times cadastrados no sistema por meio dos avaliadores
UC06: Visualizar ranking	Verificar o ranking de pontuação dos times avaliados

Tabela: Casos de uso do sistema

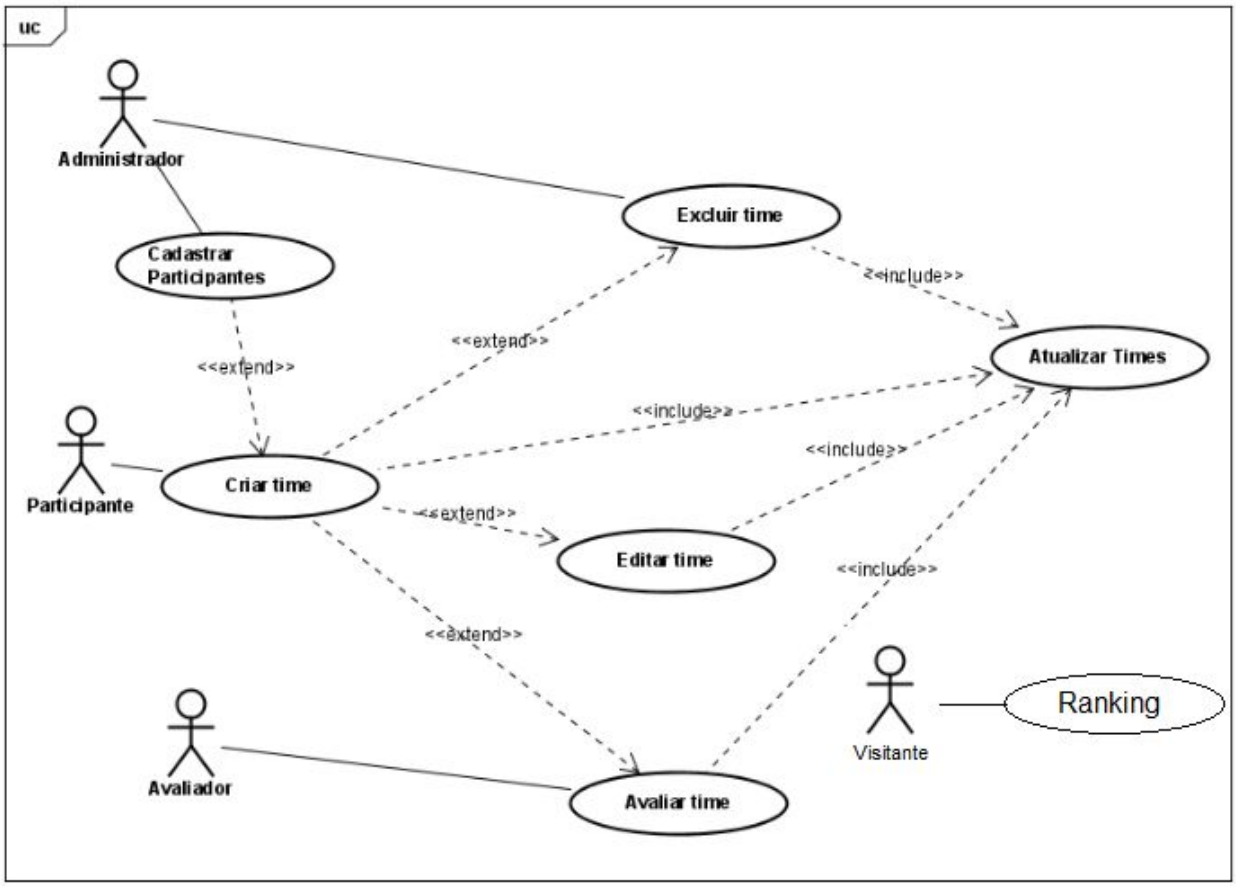

Figura: Casos de uso do sistema

Cada um dos casos de uso é de suma importância para a arquitetura uma vez que os mesmos são organizados em arquivos diferentes, o que os torna essenciais na hora de comentar a visão lógica.

5. Visão Lógica

Com o intuito de fornecer uma maior compreensão sobre e estrutura e a organização do design do sistema, uma visualização arquitetural denominada Visão Lógica é utilizada. Ela ilustra as principais realizações de caso de uso, subsistemas, pacotes e classes que abrangem o projeto em termos de arquitetura. A visão lógica mostra um um subconjunto das classes, subsistemas, pacotes e realizações de caso de uso. A visão lógica pode ser divida em visões com diferentes propósitos e serão contempladas abaixo.

5.1 Visão Geral

A visão geral mostra um subconjunto do modelo de design significativo em termos de arquitetura, ou seja, uma representação da estrutura do sistema. Como demonstrado neste diagrama a seguir:

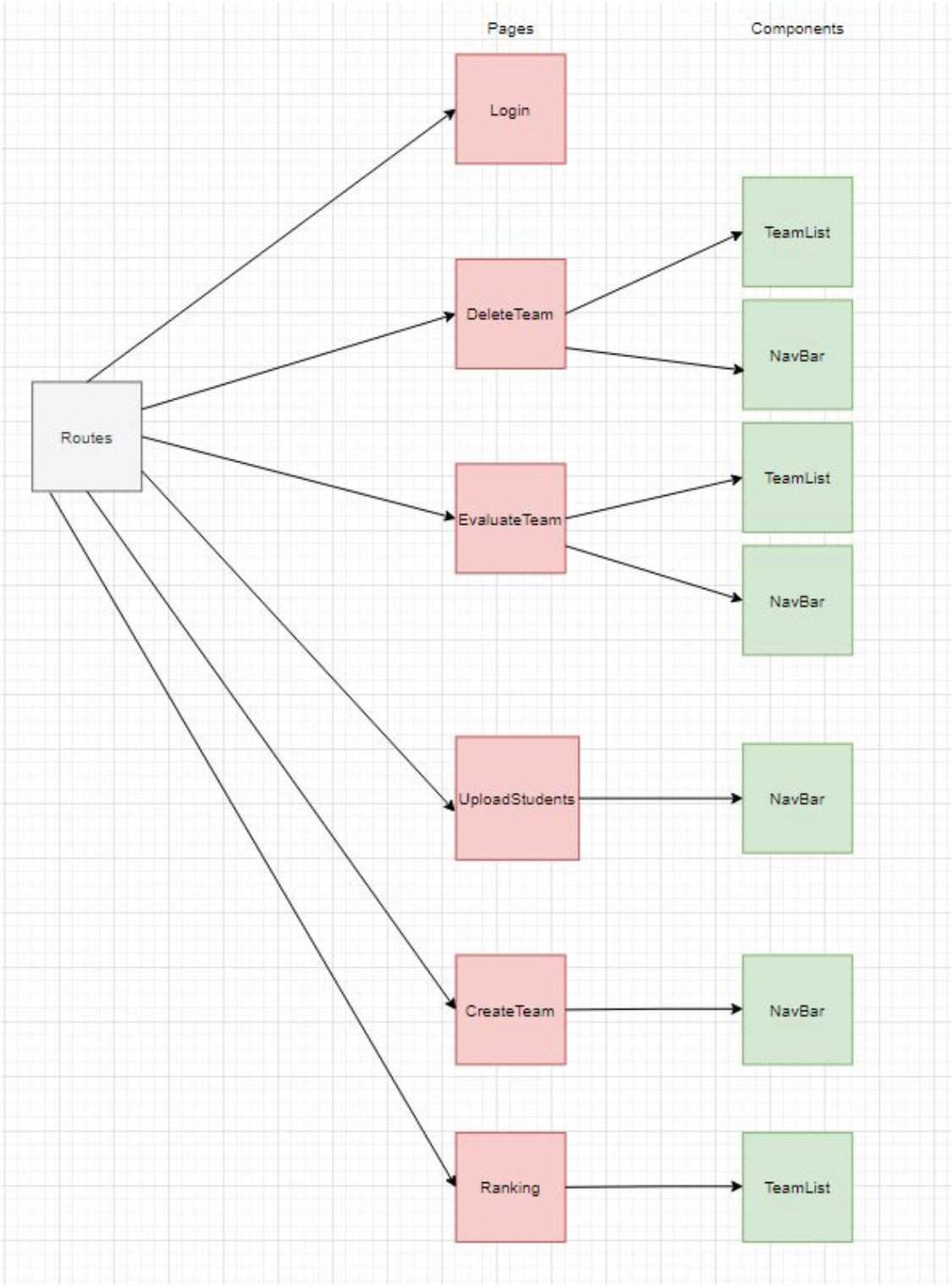

Figura: Visão lógica do cliente.

6. Visão de Processos

Para decomposição dos processos, os mockups criados foram abstraídos ao máximo para criar componentes simples e reutilizáveis que pudessem descrever as funcionalidades esperadas nestes mockups, mas sem perder a característica de reúso, que era um dos principais objetivos do projeto. Estes, quando integrados, se tornaram as pages da aplicação, que nada mais são do que reuniões de componentes, que juntos conseguem obter as funcionalidades esperadas para aquela tela em específico, separando a responsabilidade de cada parte do código, e, auxiliando na manutenção e adição de novas funcionalidades ao sistema.

Neste software, ao tratarmos do *frontend*, a troca de mensagens é realizada através de uma biblioteca disponibilizada pelo React chamada de react-router-dom, que possibilita a comunicação entre diversos componentes. Assim, é possível trocar de tela no sistema, também podendo passar diversos dados entre as telas utilizando *props* e também com funções *callbacks*, assim permitindo trocas de mensagem de componentes filhos para seus respectivos pais.

No *backend* - servidor - também utilizou-se uma biblioteca para construção do mesmo. A biblioteca em questão é o *express*, que permite a criação de rotas em uma API HTTP.

7. Visão de Implantação

Este software se baseia na arquitetura cliente/servidor, sendo assim será necessário um servidor para armazenar o banco de dados e o *backend*. Poderia ser alguma instância na nuvem, utilizando AWS, por exemplo. Porém, como é apenas uma prova de conceito, o sistema ficará rodando em um computador de um dos membros do time. Para o lado do cliente, a porta do servidor seria exposta, para que qualquer cliente consiga acessá-la. Porém, para esta PoC, tudo está rodando em *localhost*.

8. Visão de Implementação

8.1 Visão Geral

A organização das camadas do sistema pode ser visualizada na figura a seguir que apresenta as camadas de acordo com o projeto que foi desenvolvido com React.

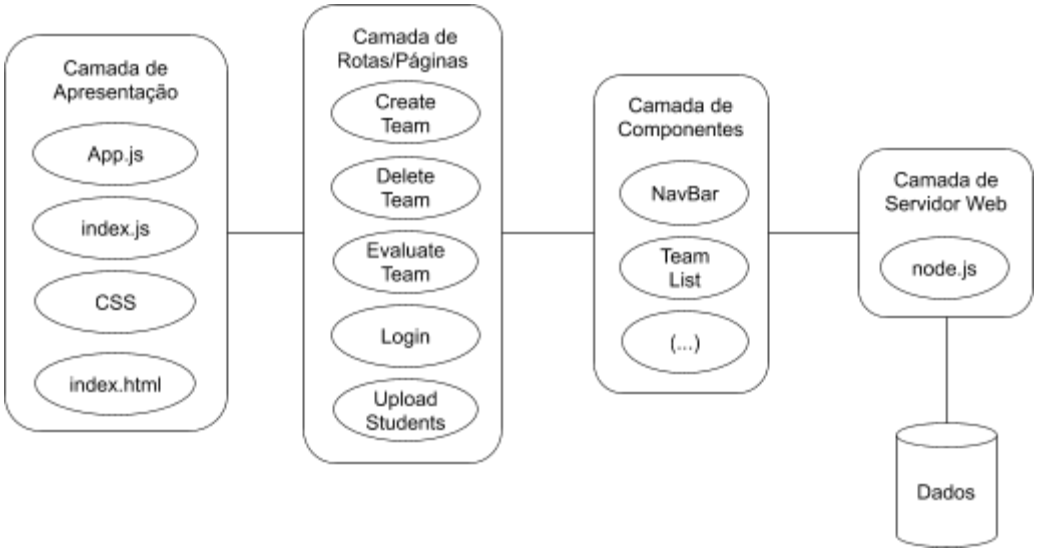

Figura: Diagrama de camadas do sistema

No *frontend*, o grupo além de utilizar a arquitetura padrão utilizada em projetos ReactJS, foi utilizado o padrão de projeto *State*, que por sua vez, é disponibilizado pelo *hook useSatate* do React. Segundo a documentação do React, "Um *Hook* é uma função especial que te permite utilizar recursos do React. Por exemplo, *useState* é um *Hook* que te permite adicionar o state do React a um componente de função." No caso do projeto, os *states* são utilizados quando o valor de uma variável, ao ser alterado, deve refletir em uma atualização do componente.

Já para o *backend* (servidor), o mesmo foi codificado utilizando *Models* e *Controllers* para representar as entidades do sistema e suas respectivas ações. A seguir, segue diagrama do *backend*.

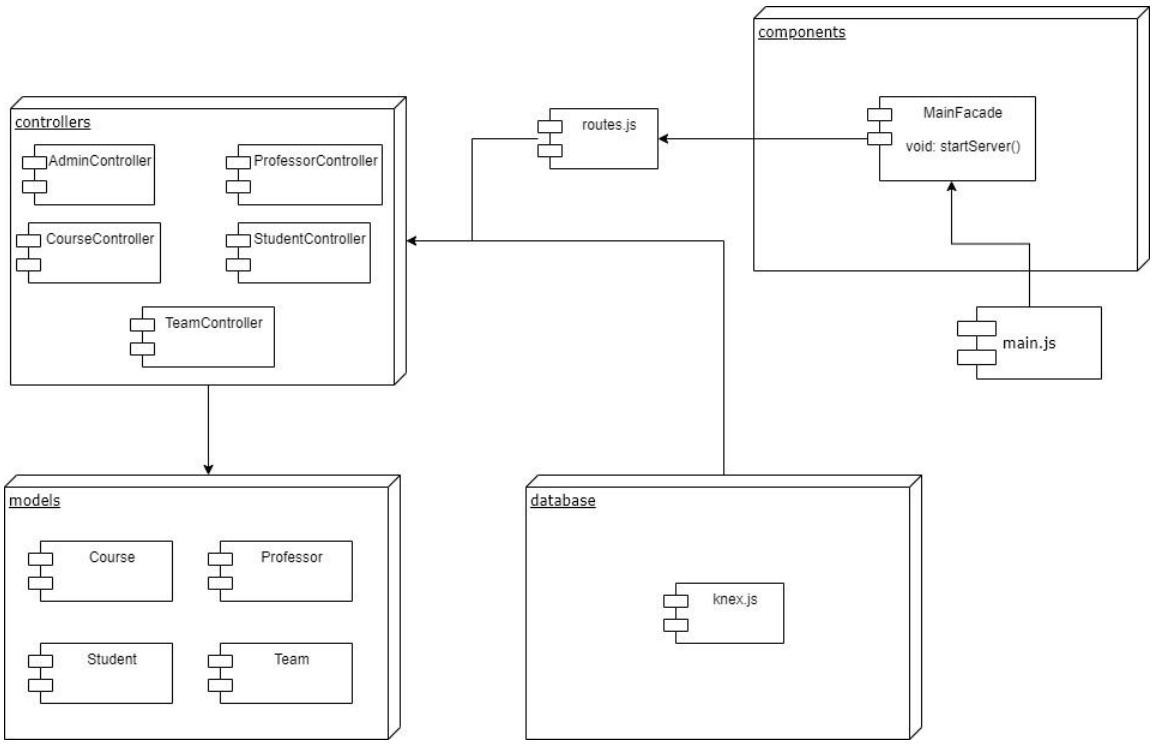

Figura 5: Diagrama dos arquivos do *backend*.

Além disso, conforme solicitado pelo enunciado do segundo trabalho da disciplina, foram implementados alguns padrões de projeto no *backend*. O primeiro implementado foi o padrão Fachada, utilizado ao iniciar a execução do servidor. Chama-se a função ***startServer()*** que inicializa a execução do *backend*. A seguir, segue imagem da implementação da fachada.

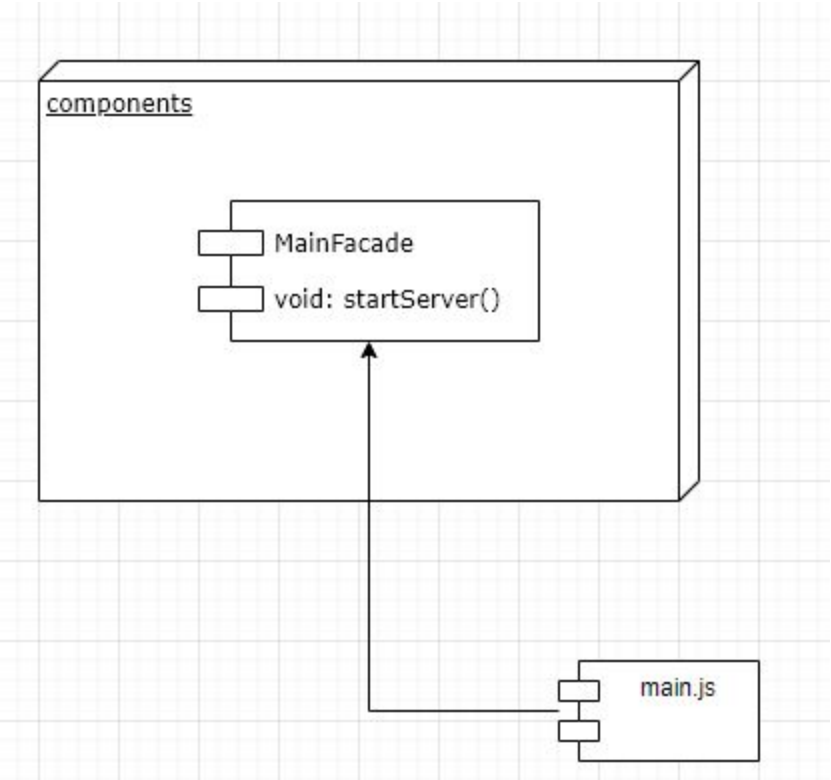

Figura: Diagrama do padrão Fachada utilizado no servidor.

O padrão *Singleton* também foi utilizado, entretanto, de maneira implícita. Por utilizar o *Node* como ambiente de execução, ao utilizar o comando *require* para solicitar algum módulo, uma instância *Singleton* é criada para tal módulo. No caso do trabalho, essa prática foi utilizada para configuração e acesso ao banco de dados, utilizando como biblioteca de ORM *Knex*. O arquivo *knex.js* exporta uma instância do *Knex* com as configurações necessárias para as outras funcionalidades do *backend*.

```
const knex = require('knex')
const config = require('../../knexfile')

module.exports = knex(config.development)
```

Figura: Instância *Singleton* para o banco de dados.

8.2 Camadas

A camada de apresentação contém as interfaces que estarão em contato diretamente com o usuário. As configurações dessa camada são as seguintes:

- App.js - Arquivo javascript que importa as instruções dos componentes do React;
- index.js - Responsável por renderizar o App.js e por importar instruções de outras bibliotecas;
- CSS - Arquivo que consiste da estilização da página;
- index.html - O arquivo front-end responsável por renderizar todos os componentes e

páginas do React;

A camada de Rotas apresenta as telas que são utilizadas no sistema. As rotas são responsáveis por garantir o fluxo entre as mesmas. A camada de Componentes carrega todos os componentes que deverão ser carregados pelas páginas, os componentes podem interagir com o código ou apenas serem apresentados. A camada do servidor Web permite a interação da aplicação com a camada de dados.

8.3 Utilização nos Casos de Uso

Utilizando como exemplo o caso de uso UC05, podemos observar que primeiramente os dados dos times seriam carregados da camada de dados para a camada de componentes por meio da camada de servidor web. Uma vez na camada de componentes os dados seriam carregados no formato de lista que é informado na sua respectiva página da camada de rotas. O usuário então poderia acessar a página por meio da camada de apresentação.

9. Visão de Dados

Como explicado anteriormente, considerando a utilização do padrão arquitetural de cliente-servidor, foi utilizado o servidor para realizar todas as operações do banco de dados. Os dados solicitados são enviados para os clientes evitando informações desnecessárias e trechos de código que poderiam ser implementados no frontend dos clientes.

Ao analisar os requisitos e o enunciado da atividade, concluiu-se que um banco de dados relacional seria uma escolha adequada para armazenar os dados da aplicação. Uma vez que os dados possuem relacionamentos entre si (como times e alunos, por exemplo), a estruturação utilizando linguagem SQL acaba sendo mais natural e enforça as regras de negócio ao desenvolver a base de dados. Por fim, por contar apenas com informações de tipos primitivos, o uso de um banco de dados não-relacional ou ainda orientado a arquivos não fez-se necessário.

Para ilustrar melhor a solução de dados desenvolvida pelo grupo, documentou-se o banco de dados e os processos que o circundam através de diagramas. Estes serão apresentados a seguir.

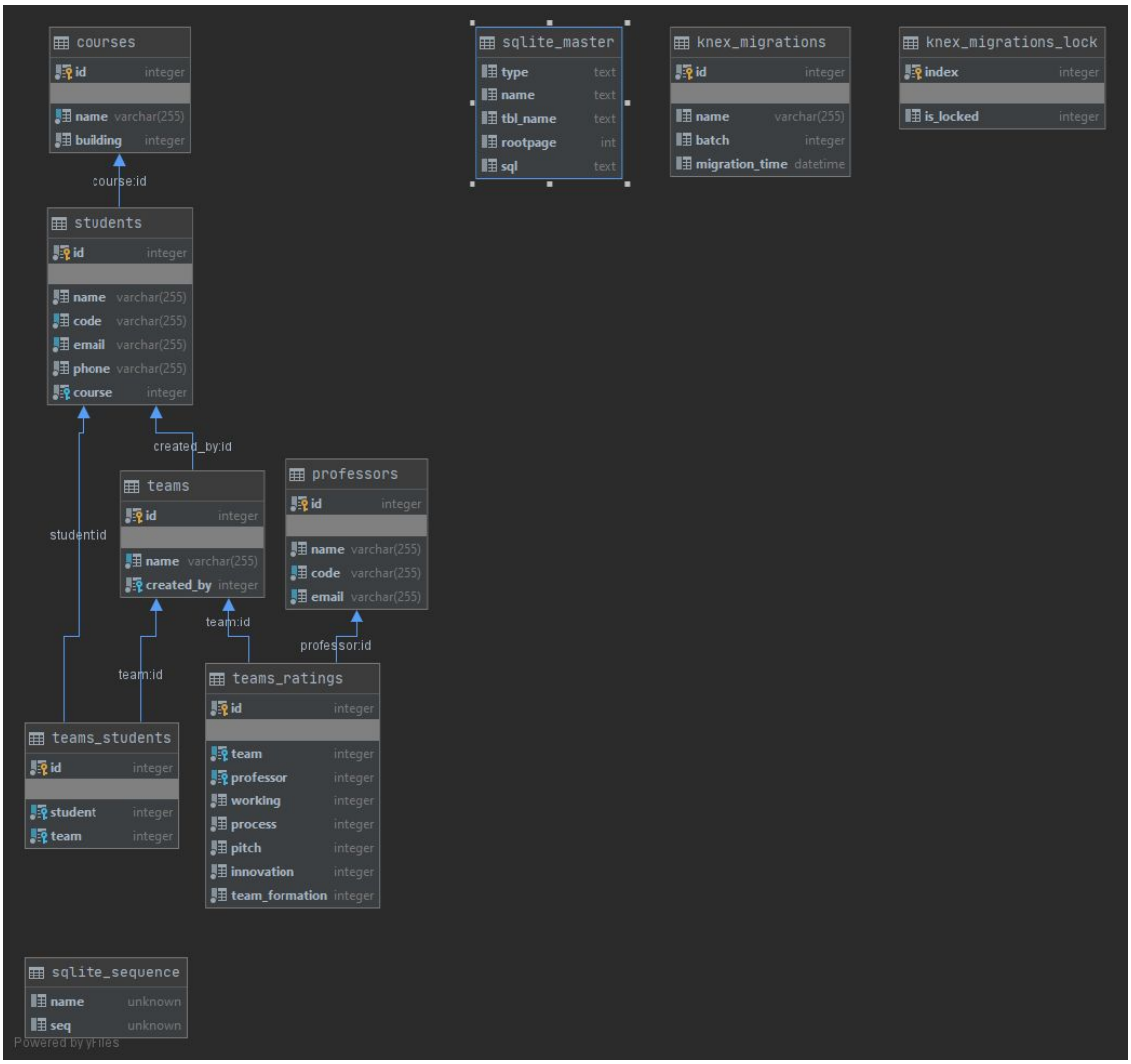

Figura: Diagrama físico do banco de dados

10. Tamanho e Performance

O sistema tem como principal objetivo a organização das Hackathons de Engenharia de software e portando o corpo docente e discente da PUCRS seriam os principais utilizadores do mesmo. É esperado que o mesmo cumpra com as expectativas dos administradores da Hackatona, mas ainda há espaço para adição de novas features e melhorias. O padrão arquitetural já havia sido escolhido de maneira que a performance, organização e tamanho do sistema fossem o melhor possível, portanto o mesmo será de fácil acesso para todos os participantes, avaliadores e administradores.

11. Qualidade

A arquitetura de cliente-servidor permite definir algumas metas de qualidade como facilidade no acréscimo de novas funcionalidades, manutenção e reaproveitamento de código. Algumas dessas metas foram atingidas graças às tecnologias que foram utilizadas pelo grupo para

criação do sistema.

React se provou uma ótima ferramenta para componentização e reaproveitamento de código em diversas funcionalidades. Além disso o conteúdo criado pela comunidade muito ativa do React contribui para correção de bugs e para resolução de dúvidas em relação a componentes ou lógica.

O sistema também foi feito de maneira que seja fácil a criação de novas funcionalidades ao longo do tempo, algo que se prova de valor considerando que o mesmo será utilizado por diversos alunos e professores de Engenharia de Software.