ProjArq - Projeto e Arquitetura de Software

Hackatona dus Guri
Software Architecture Document

Version 4.0

Revision History

Date	Version	Description	Author
09/05/2020	1.0	First informations about the project	Bernardo de Cesaro, Guilherme Deconto e Gustavo Possebon
16/05/2020	2.0	Adding more details	Bernardo de Cesaro, Guilherme Deconto e Gustavo Possebon
23/05/2020	3.0	Finalizing the document	Bernardo de Cesaro, Guilherme Deconto e Gustavo Possebon
01/07/2020	4.0	Finalizing the document v2	Bernardo de Cesaro, Guilherme Deconto e Gustavo Possebon

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

Este documento apresenta uma visão geral da arquitetura do sistema e utiliza uma série de visões arquiteturais diferentes para ilustrar os diversos aspectos do sistema. Sua intenção é capturar e transmitir as decisões significativas do ponto de vista da arquitetura que foram tomadas em relação ao sistema.

1.2 Scope

Este Documento de Arquitetura de Software se aplica no sistema da Hackatona desenvolvido para o trabalho 1 da disciplina de Projeto e Arquitetura de Software. O sistema *backend* foi desenvolvido em Node js e o *front* foi desenvolvido utilizando o *framework* React.
O sistema tem como objetivo auxiliar no gerenciamento dos das inscrições e da montagem de grupos da hackatona de engenharia de software. Algumas das ações do sistema, para os participantes são: Registrar, escolher time, sugerir times. O sistema também deve ter uma visão para avaliadores e administradores do sistema.

1.3 Definitions, Acronyms, and Abbreviations

CSS - Linguagem de folhas de estilo
Bootstrap - *Framework* web para definição de interface
Postman - *Software* utilizado apenas para testar os endpoints do backend
Node js - Plataforma para construir aplicações Web escaláveis, que utiliza o JavaScript como sintaxe
React - *Framework* Javascript
Hapi - *Framework* utilizado para construir aplicações e serviços.
Joi - *Framework* para verificação de *response* JSON.
JSON - Formato de documento mundialmente utilizado *Javascript Document Notation*
Api - *Software* que irá ficar hospedado em um servidor que irá gerenciar as requisições feitas pela aplicação web, além disso ela tem a funcionalidade de fazer a comunicação com o banco de dados.
App - Aplicativo
PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul
RF - Requisito Funcional;
RNF - Requisito Não-Funcional;
UC - Use Case
RTDESENV - Restrição de Desenvolvimento
RTSEG - Restrição de Segurança
RTIHC - Restrição de Interface

1.4 References

SILVA, V. Cayo; ALVES, C.E. Gabriel; FERNANDES, S.B. Lucas. **Redes Par-a-Par.** UFRJ, 2016. Retirado de: <https://www.gta.ufrj.br/ensino/eel878/redes1-2016-1/16_1/p2p/modelo.html>. Acesso em 10/05/2020.
Link para o repositório do projeto: <Repositorio - GitHub>

1.5 Overview

No restante deste documento iremos descrever a arquitetura escolhida que melhor soluciona os problemas e necessidade encontrados neste projeto. Além desta descrição, também será apresentado as diferentes visões aplicadas ao projeto em si, assim como o Desempenho e a Qualidade encontrados e esperados.

- Avaliador: Logar, ver os times, avaliar os times seguindo as normas de avaliação.
- Administrador: Logar, editar participantes, deletar participantes.
- Participantes: Cadastrar, escolher time e sugerir times.
- Resultados: Exibir o resultado final do evento hacktona.

2. Architectural Representation

Este documento apresenta a arquitetura como uma série de visões: visão de casos de uso, visão lógica, visão de processos, visão de implantação, visão de implementação e visão de dados.

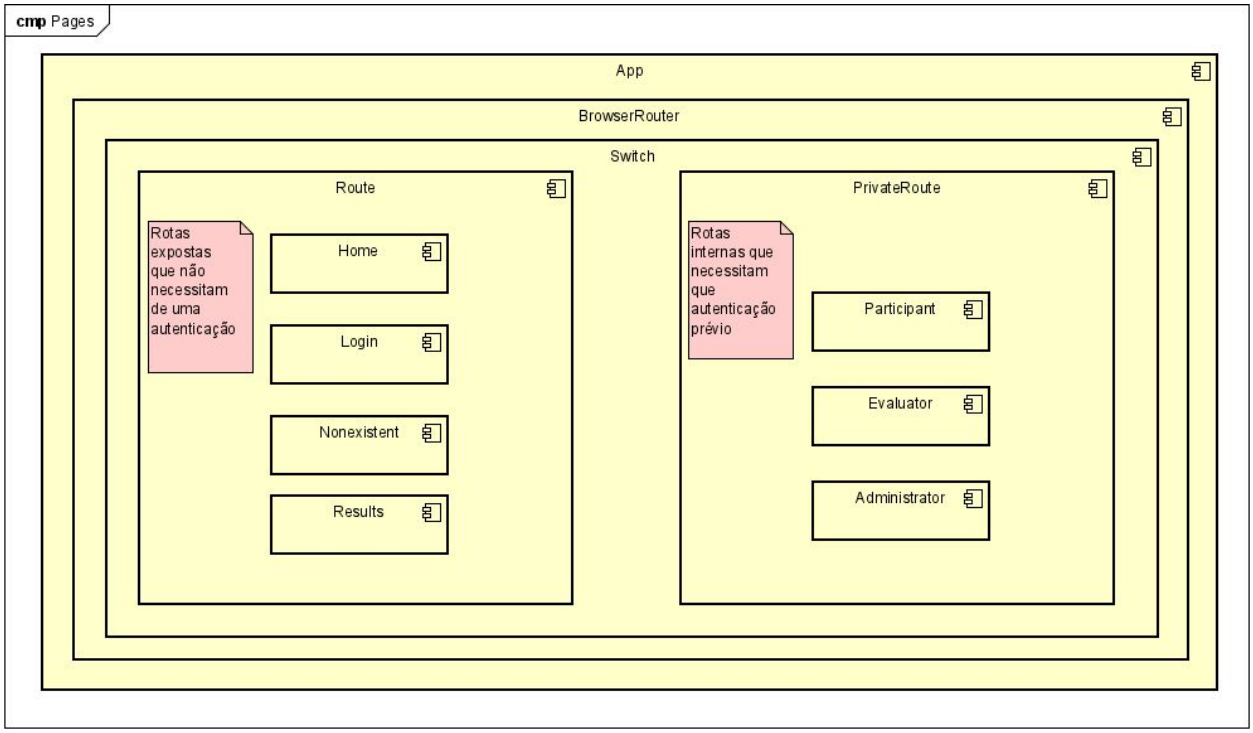

Figura 1: Representa a visão do App

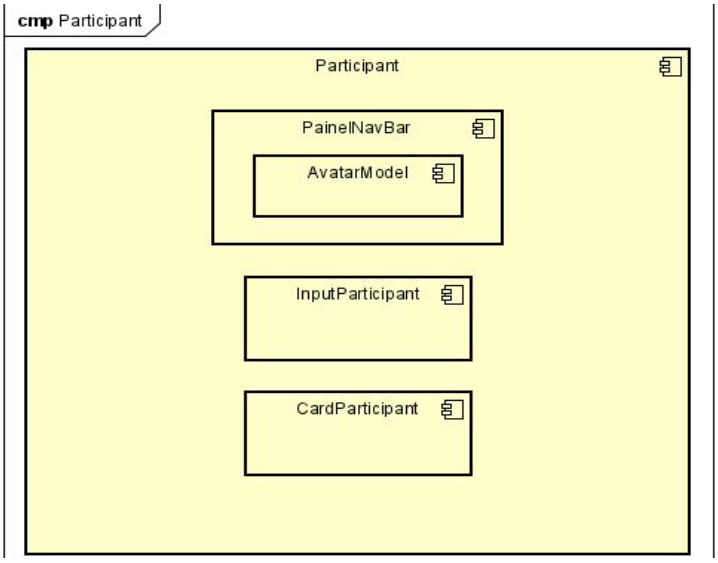

Figura 2. Diagrama do Componente do Participante

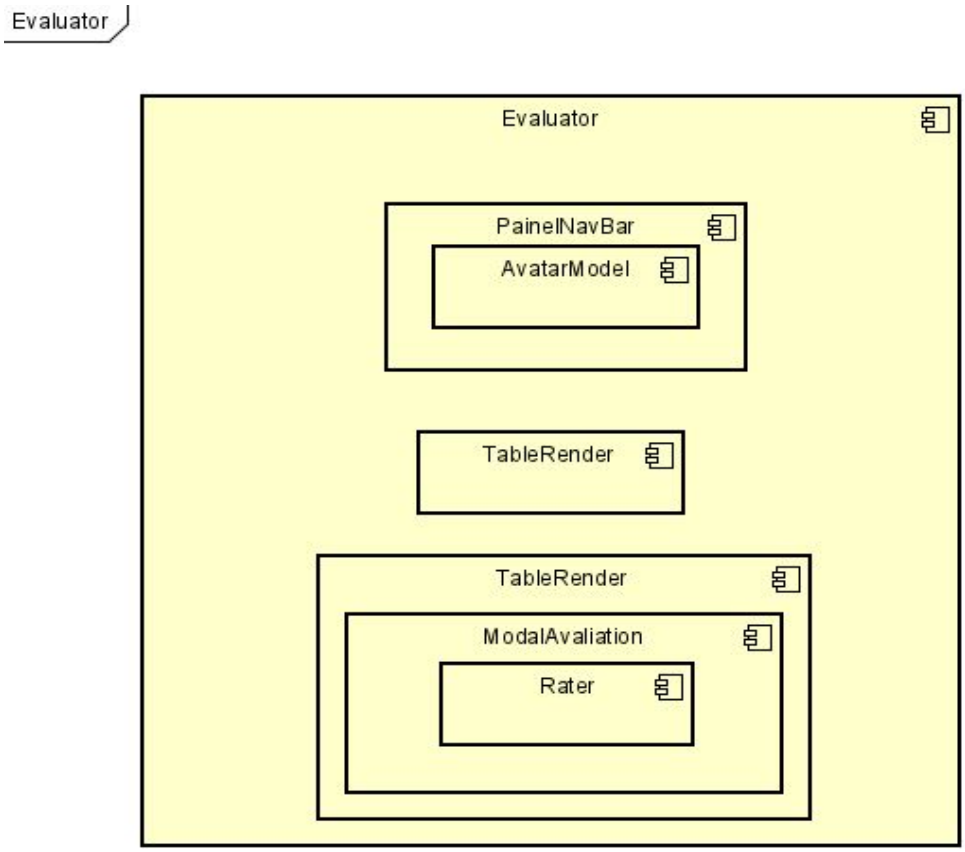

Figura 3. Diagrama de Componente do Avaliador

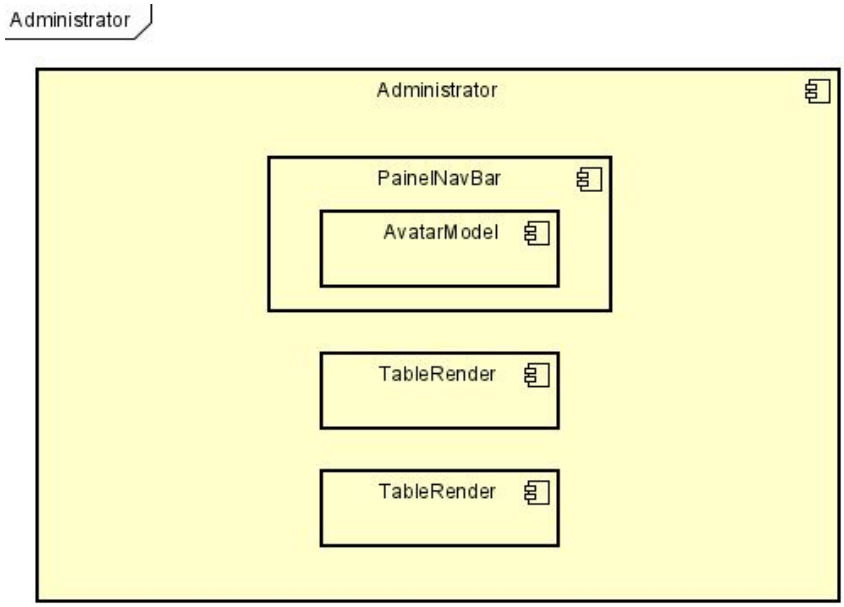

Figura 4. Diagrama de Componente de Administrador

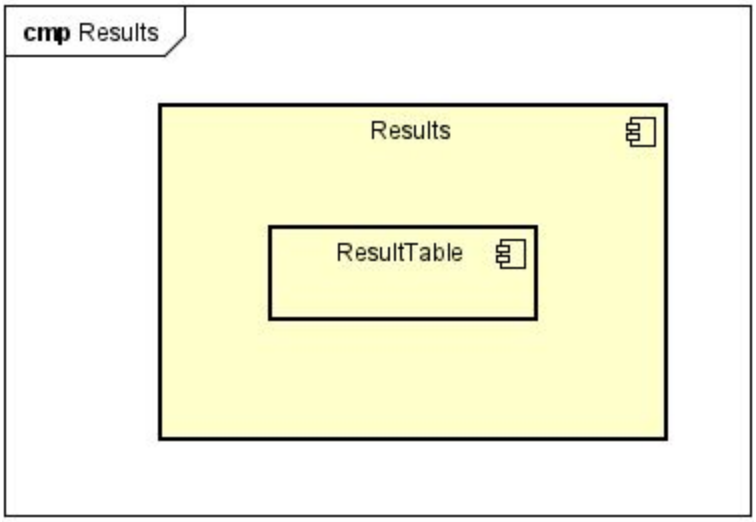

Figura 5. Diagrama de Componente de Resultado

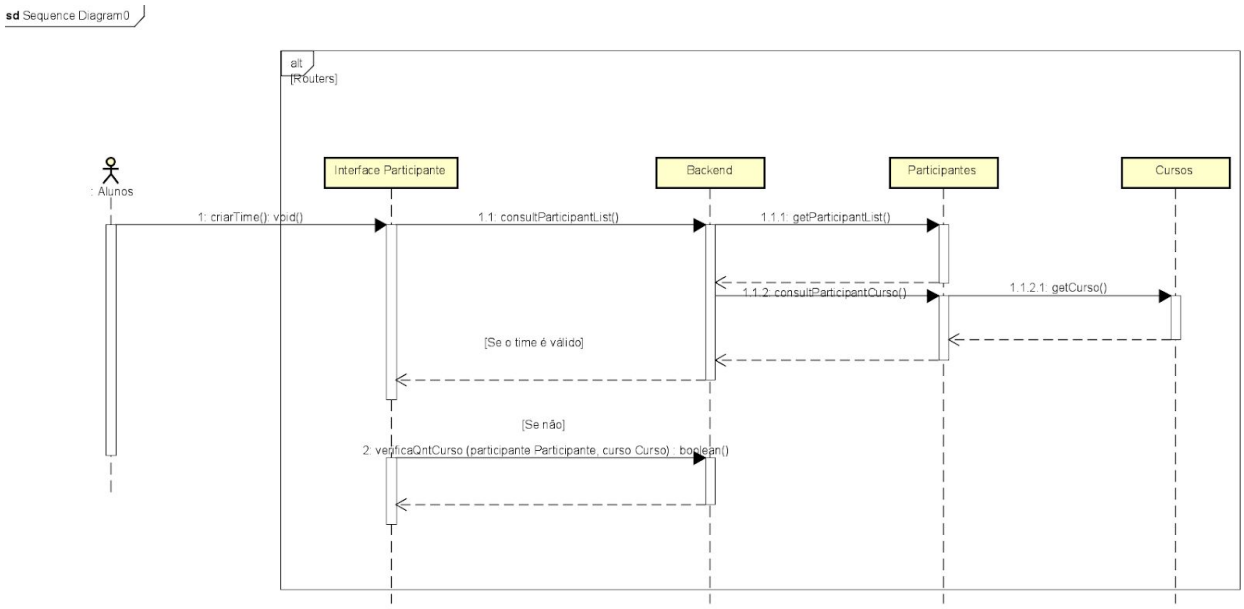

Figura 6. Diagrama de Sequência - Criação de Time

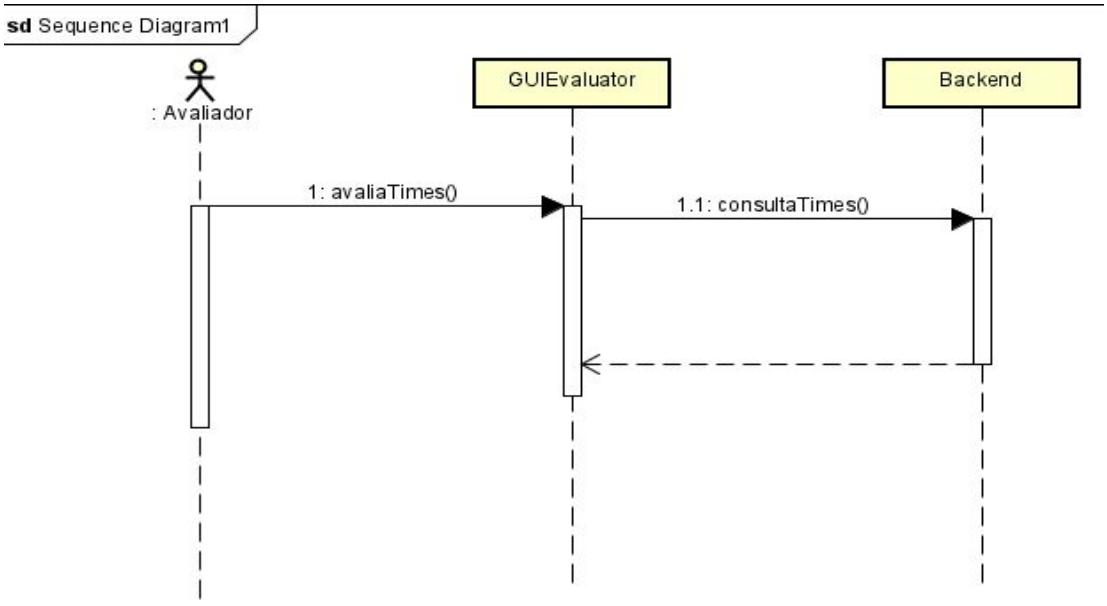

Figura 7. Diagrama de Sequência - Avaliação de Time

3. Architectural Goals and Constraints

Existem alguns requisitos chaves e algumas restrições de sistema que possuem uma relevância em relação a arquitetura. Elas estão descritas abaixo:

Referência	Descrição
RTSEG1	O usuário precisa estar logado para acessar os requisitos do sistema.
RTSEG2	O usuário deve estar autenticado através de um *token* de validação.
RTDESENV1	O sistema não pode trancar.
RTSEG3	Os dados cadastrados pela utilização da interface *web* devem persistir apenas no banco de dados.
RTSEG4	Os dados modificados pela interface *web* devem persistir apenas no banco de dados.
RTIHC1	A tela do avaliador deve conter os critérios de avaliação para cada time.
RTIHC2	A tela do administrador deve conter as opções para editar, adicionar e remover participantes.
RTIHC3	A tela do avaliador deve conter as opções para adicionar ou remover avaliações.
RTDESENV2	Um time deve possuir pelo menos dois cursos diferentes entre os participantes.
RTDESENV3	O sistema deve conferir se um time é válido ou não.
RTDESENV4	Deve ser possível criar e excluir times, assim como os integrantes.
RTDESENV5	Avaliadores da hackatona utilizam o sistema para avaliar os times.
RTDESENV6	O registro de um aluno deve ser feito apenas por um administrador.
RTDESENV7	O sistema deve informar o resultado final.
RTDESENV8	O resultado final deve ser apresentado em ordem crescente de notas.
RTDESENV9	As notas dos times devem ser a média das notas atribuídas pelos avaliadores.
RTDESENV10	Caberá apenas aos avaliadores fazer a avaliação dos times.
RTDESENV11	Cada time deverá ter 3 avaliações de avaliadores distintos.
RTDESENV12	Importar os participantes e os times de um banco simulado.
RTDESENV13	É possível sugerir os integrantes de um time.
RTDESENV14	Um avaliador deve avaliar os seguintes critérios: software funcionando, processo, pitch, inovação, formação do time.
RNFDESENV1	O sistema deve utilizar Mongodb como banco de dados.
RNFDESENV2	O sistema deve utilizar Javascript como linguagem e Node js como plataforma no back-end.

RNFDESENV3	O sistema deve utilizar Javascript como linguagem e React como framework no front-end.
RNFDESENV4	O sistema deve utilizar Javascript como linguagem e React como framework no front-end.

4. Logical View

Descreve as classes mais importantes do projeto, sua organização em componentes e subsistemas de serviços.

4.1 Overview

Esta subseção descreve a decomposição geral do modelo de design em termos de hierarquia e camadas de pacotes.

4.2 Architecturally Significant Design Packages

A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes.

O cliente não compartilha nenhum de seus recursos com o servidor, mas no entanto ele solicita alguma função do servidor, sendo ele, o cliente, responsável por iniciar a comunicação com o servidor, enquanto o mesmo aguarda requisições de entrada.

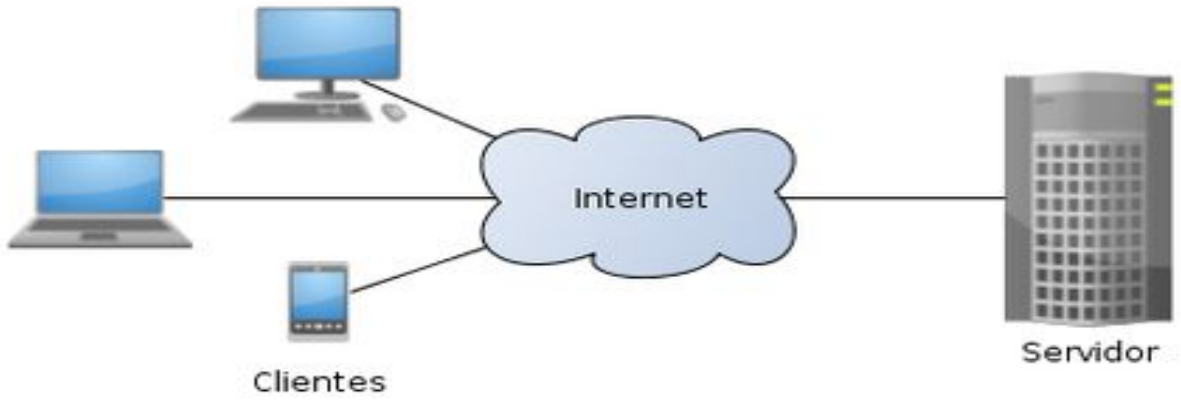

Figura 8: Representa o padrão de projeto utilizado

4.2.1 Client

Cliente - Solicita um determinado serviço, através do envio de uma mensagem ao servidor. Enquanto o servidor está resolvendo a solicitação, o cliente fica livre para realizar outras tarefas.

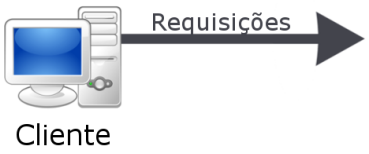

Figura 9: Representa o padrão de projeto utilizado

4.2.2 Server

Oferece serviços a processos usuários (clientes), ou seja, executam a tarefa solicitada e enviam uma resposta ao cliente que se traduz nos dados solicitados.

Figura 10: Representa o padrão de projeto utilizado

4.2.3 Patterns

- Observable - Função natural do React onde a função *setState* fica esperando acontecer algo para alterar o estado da aplicação ou página atual.
- Singleton - Componentes funcionais executam apenas uma instância quando invocados ou somem quando chamados.
- Decorator - Componentes não funcionais (ou de classes) que extendem do *React.Component* (biblioteca nativa do *React*).
- Façade - Componente *App* onde se tem os *Routes* para as páginas.
- Prototype - Ex.: No componente *ResultTable* tem *prototypes* implementados para *Row*.

4.3 Use-Case Realizations

Caso de Uso	Descrição
UC1	Neste caso de uso, o usuário acessa aplicação Hackatona dus Guri.
UC2	Neste caso de uso, considerando que o participante já acessou o sistema, ele deseja sugerir um time.
UC3	Neste caso de uso, após selecionar os integrantes do time, o participante desejar enviar sugestão.
UC4	Neste caso de uso, considerando que o avaliador já acessou o sistema, ele seleciona um time que deseja avaliar.
UC5	Neste caso de uso, considerando que o avaliador já avaliou um time, ele deseja salvar a avaliação.
UC6	Neste caso de uso, considerando que o administrador já acessou o sistema, ele gostaria de editar a descrição de um time.

UC7	Neste caso de uso, o administrador gostaria de selecionar o time e inserir os dados desejados.
UC8	Neste caso de uso, um avaliador gostaria de selecionar um time e avaliar o seu projeto.
UC9	Neste caso de uso, qualquer usuário logado pode visualizar o resultado da hackatona.
UC10	Neste caso de uso, apenas os administradores possuem permissões para gerenciar os times e os participantes.
UC11	Neste caso de uso, avaliador deseja excluir a sua avaliação de um time.
UC12	Neste caso de uso, avaliador deseja adicionar a sua avaliação de um time.

5. Process View

O sistema é gerenciado por meio de serviços, esses serviços podem ser divididos em sua capacidade de influência no sistema em geral. Existem dois tipos:

- Serviço leve: Serviço de baixa importância para o sistema, como por exemplo listagem de participantes.
- Serviço pesado: Processo de alto impacto dentro do sistema em que, caso ocorra o mal gerenciamento, pode comprometer outras áreas do sistema. Um exemplo é a atualização de dados de um usuário e time.

6. Deployment View

O sistema é construído utilizando a linguagem JavaScript com o foco voltado para dispositivos que possuem acesso a *web*. O sistema *backend* foi construído usando a plataforma Node js com integração ao banco de dados não-relacional conhecido como MongoDb. O *backend* implementado para a execução do projeto tem como funcionalidade prover os dados em tempo real para a interface *web* construída utilizando o *framework* React.

7. Data View

No projeto selecionado, as ferramentas de persistência de dados escolhida foram a utilização do sistema de banco de dados MongoDB, o qual é baseado em um banco de dados não relacional, e a outra ferramenta escolhida pela equipe de desenvolvimento foi a utilização localstorage no *front-end* para armazenamento de *token* de autenticação de acesso à API.

Foi escolhido fazer a utilização de um banco de dados não relacional pela necessidade de ter uma melhor performance e alta escalabilidade, para dessa forma ter um maior e melhor gerenciamento de dados e também pela maior facilidade de uso e integração com serviços *RestFul*.

8. Size and Performance

1. O sistema deve suportar N usuários simultâneos utilizando o banco de dados MongoDb.
2. A aplicação deve possuir uma usabilidade boa.

3. A aplicação deve ser disponibilizada via interface *web*.

9. Quality

O padrão de arquitetura Cliente-Servidor foi a solução mais satisfatória para atender a qualidade esperada do sistema, que deverá ser desenvolvido em linguagem JavaScript, visando uma interface que seja interativa e fácil de se usar.

Em relação ao atributo escolhido para ser analisado, manutenibilidade foi o que mais se encaixou na abordagem deste projeto. O motivo da escolha foi pelo tipo de *framework* utilizado, que fornece em sua grande parte propriedades desejáveis como modularidade, reusabilidade e modificabilidade.