**Hackatona dus Guri**

**Software Architecture Document**

**Version 3.0**

**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 |

**Table of Contents**

[**Introduction**](#_heading=h.30j0zll)4

[Purpose](#_heading=h.1fob9te) 4

[Scope](#_heading=h.2et92p0) 4

[Definitions, Acronyms, and Abbreviations](#_heading=h.3dy6vkm) 4

[References](#_heading=h.1t3h5sf) 4

[Overview](#_heading=h.4d34og8) 4

[**Architectural Representation**](#_heading=h.2s8eyo1)4

[**Architectural Goals and Constraints**](#_heading=h.17dp8vu) **7**

[**Logical View**](#_heading=h.26in1rg) **7**

[Overview](#_heading=h.lnxbz9) 7

[Architecturally Significant Design Packages](#_heading=h.35nkun2) 8

[Client](#_heading=h.bqisasj1ygr7) 8

[Server](#_heading=h.kbxyr7hkimms) 8

[Use-Case Realizations](#_heading=h.1ksv4uv) 9

[**Process View**](#_heading=h.44sinio) 9

[**Deployment View**](#_heading=h.2jxsxqh)9

[**Data View**](#_heading=h.4i7ojhp)9

[**Size and Performance**](#_heading=h.2xcytpi)10

[**Quality**](#_heading=h.1ci93xb)10

**Software Architecture Document**

# Introduction

## 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.

## 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.

## 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 Java Script 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

## 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.

## 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.

# 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.

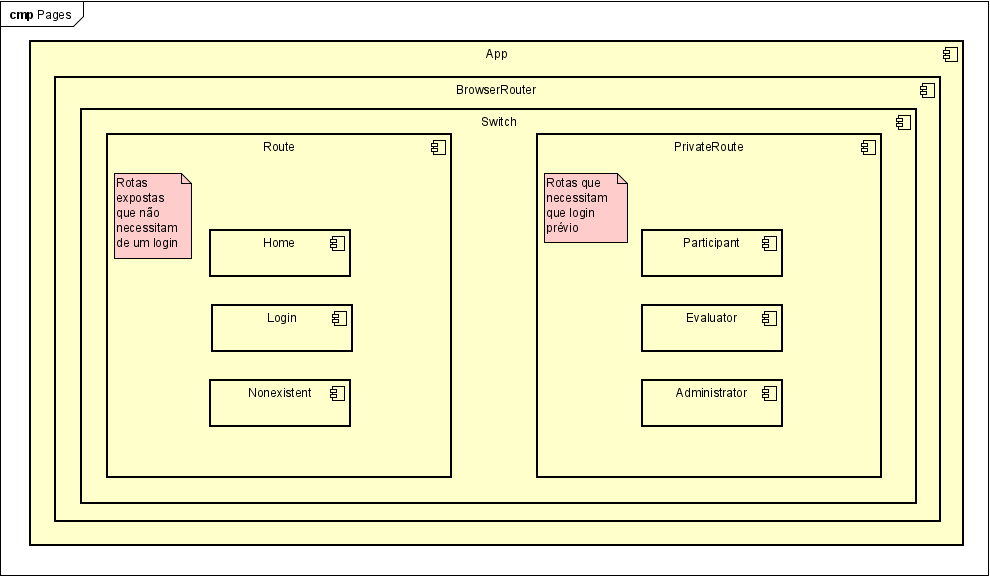
**

Figura 1: Representa a visão do App

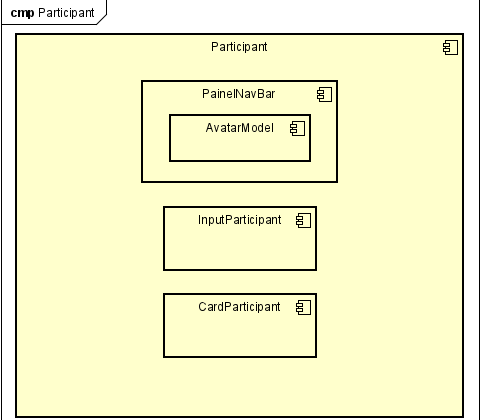


Figura 2. Diagrama do Componente do Participante

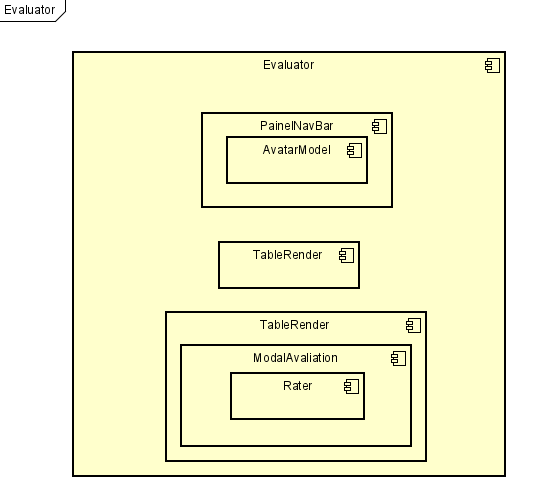


Figura 3. Diagrama de Componente do Avaliador

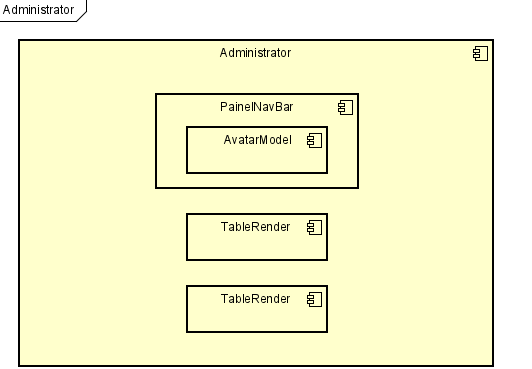


Figura 4. Diagrama de Componente de Administrador

# 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. |
| 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. |

# Logical View

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

## Overview

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

## 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.

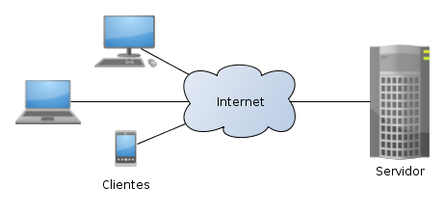
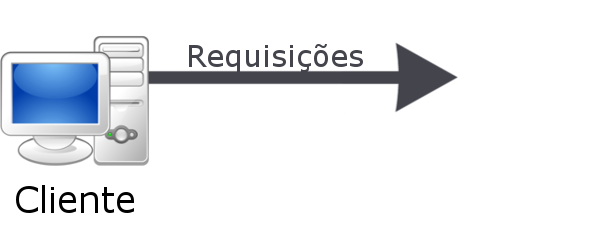
****

Figura 2: Representa o padrão de projeto utilizado

## Client

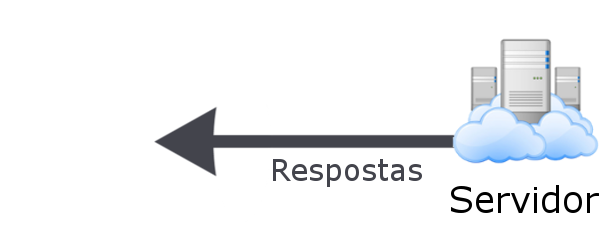
Cliente - Solicita um determinado serviço, através do envio de uma mensagem ao servidor. Enquanto o servidor está resolvendo a solicitação, o cliente fica livre para realizar outras tarefas.



## 

## 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.



## 

## 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. |

# 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.

# 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.

# 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*.

# 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*.

# 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.