
Pontifícia Universidade Católica do Rio Grande do Sul

Hackatona AGES
Documento de Arquitetura de Software

**Alessandro Medeiros, Chiara Paskulin, Edson da Costa,
Gabriel Correa, João Nascimento**

Versão <1.2>

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

Histórico de Revisão

Data	Versão	Descrição	Autor
<10/Mai/20>	<1.0>	Iniciada a estruturação do documento e seus componentes.	João Nascimento
<15/Mai/20>	<1.1>	Modificação de tópicos e adição dos diagramas de caso de uso e diagrama de pacotes.	João Nascimento
<24/Mai/20>	<1.2>	Adição dos diagramas para o <i>front-end</i> e ajustes na escrita do documento de arquitetura.	João Nascimento

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

Sumário

1.	Introdução	4
1.1	Propósito	4
1.2	Escopo	4
1.3	Definições, Acrônimos e Abreviações	4
2.	Representação Arquitetural – <i>Back-End</i>	5
3.	Representação Arquitetural – <i>Front-End</i>	5
4.	Diagrama de Casos de Uso	6
4.1	Visão Geral	8
4.2	Pacotes importantes para a arquitetura	8
4.3	Realizações de Caso de Uso	9
5.	Tamanho e Performance	9
6.	Qualidade	9

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

Documento de Arquitetura de Software

1. Introdução

1.1 Propósito

Esse documento fornece uma visão compreensível e amigável sobre a arquitetura de software do sistema proposto, sendo este um sistema desenvolvido para administrar as *hackatons* promovidas pela Agência Experimental de Engenharia de Software. Apresentaremos neste documento a arquitetura adotada, as decisões sobre a arquitetura e framework utilizados, seguido da explicação sobre os módulos e componentes da aplicação. Apresentamos nas próximas seções o escopo do documento e seu conteúdo, algumas definições formais sobre arquitetura de software diagramas de pacote, de caso de uso, diagrama de classes e diagrama de implantação.

1.2 Escopo

Este documento tem como objetivo apresentar brevemente os diagramas de caso de uso, as *user stories* (US) elencadas para melhorar a visão sobre as atividades possibilitadas pelos diferentes usuários do sistema, explicar como os pacotes foram estruturados e a motivação por trás da tomada de decisão. Apresentaremos uma breve introdução sobre as tecnologias utilizadas e as arquiteturas implementadas por baixo de cada *framework* tanto para o módulo de interface com o usuário quanto a API construída para que o cliente pudesse consumir os dados de maneira otimizada e organizada do ponto de vista de arquitetura de software, mantendo as responsabilidades de lógica de negócio, manipulação de dados, acesso ao banco de dados e controle de requisições distribuídas entre os diferentes módulos da aplicação.

1.3 Definições, Acrônimos e Abreviações

API: *Application Programming Interface*. Abstração de métodos e rotinas desenvolvidas e disponíveis para uso de diferentes aplicações ou bibliotecas.

Back-End: A parte lógica onde o processamento sobre os dados da aplicação e as regras de negócio (cálculos, operações de armazenamento) são executados.

Front-End: A parte visual apresentada ao usuário final. Toda a interface gráfica utilizada para interagir, como componentes de botões, caixas de texto ou imagens de alerta sobre o comportamento do sistema. Envia e recebe informações ao back-end.

MVC: *Model-View-Controller*. Padrão de projeto de arquitetura de software que visa o reaproveitamento de código, separando a aplicação em três camadas lógicas distintas de entidade, visualização e controle.

US: *User Stories*. Especificação de requisitos de software (funcionalidades, comportamentos) apresentadas através de texto cotidiano e ações de trabalho para o usuário final.

SQL: *Structured Query Language*. Linguagem estruturada para banco de dados relacionais, definida para operar sobre seus respectivos dados.

Persona: uma representação fictícia de uma pessoa enquanto usuário final do sistema, utilizada para melhor visualização de um requisito funcional antes de sua implementação.

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

2. Representação Arquitetural – Back-End

O padrão arquitetural aplicado pelo software atual é o MVC. As entidades e seus controladores foram desenvolvidas em linguagem Java, utilizando *Spring Boot*, um popular *framework* para aplicações web. O lado de modelagem de dados e controle de entrada e saída (as camadas M e C) são construídas a partir deste *framework*. A camada de visualização dos dados (a camada V), muitas vezes sendo a parte visível ao usuário final, é desenvolvida utilizando o *framework* VueJs. Como a interface com o usuário apresenta diversos botões, caixas de texto e tabelas, componentes semelhantes estão presentes em diferentes páginas da aplicação. O *framework* VueJs permite criar componentes reutilizáveis e adaptáveis através da sua arquitetura de componentes genéricos. Apresentaremos a arquitetura adotada para o lado do cliente (View) com VueJs e a arquitetura adotada para o lado do servidor (Model e Controller) com *Spring Boot*.

A Figura 1 apresenta uma breve visão e explicação sobre os módulos presentes no lado do cliente e a Figura 2 apresenta uma visão sobre o lado do servidor e como as duas partes estabelecem comunicação.

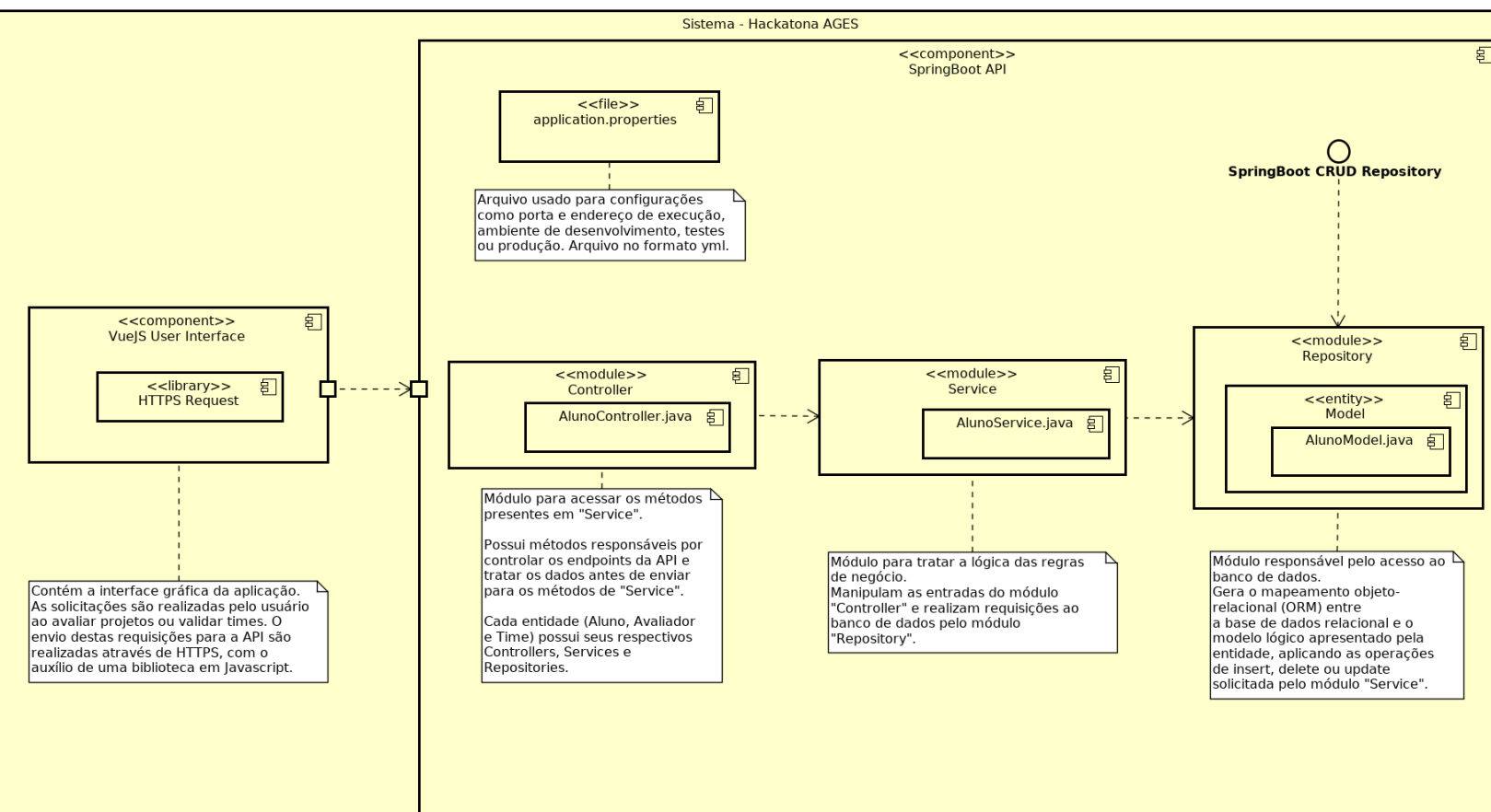


Figura 1: Diagrama de Componentes do Back-end

3. Representação Arquitetural – Front-End

Uma vez que os componentes de *front-end* possuem semelhanças visuais e apenas ajustes em suas funcionalidades, nós optamos por uma arquitetura de componentes para reutilização de código. Dessa forma, o módulo de componentes guarda abstrações sobre itens do *front-end* como botões, caixas de texto ou caixa modal para

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

informativos ao usuário. O módulo de *services* possui os métodos de acesso ao *back-end*.

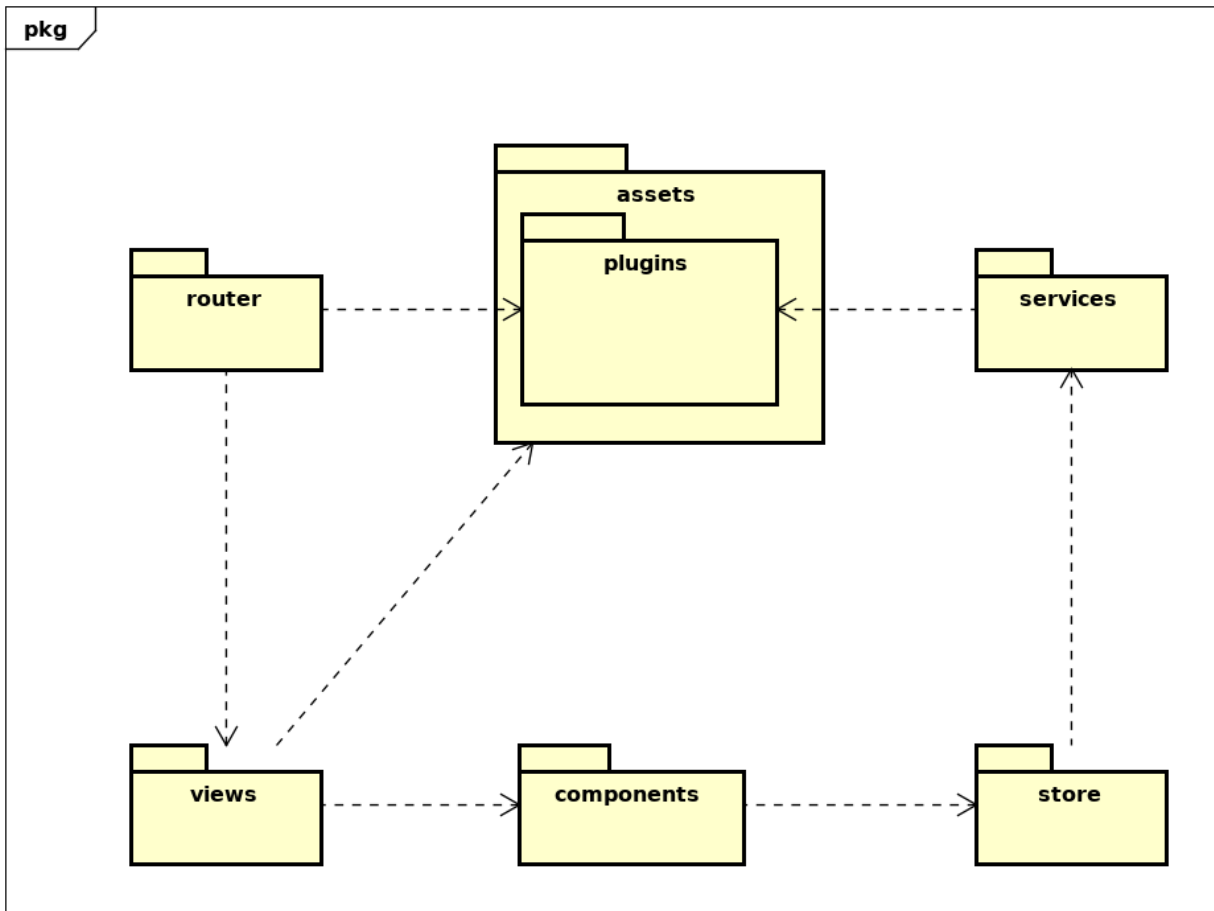


Figura 2 - Diagrama de pacotes do lado View

4. Diagrama de Casos de Uso

Apresentaremos nessa seção a lista dos principais casos de uso elencados através do enunciado do trabalho da disciplina e o diagrama gerado a partir dessas definições. A Figura 1 ilustra o diagrama de caso de uso com duas personas definidas pelo grupo. Anexado ao documento encontram-se os documentos de *Use-Case* para cada uma das funcionalidades apresentadas aqui.

- Moderador:

- Eu, como moderador da Hackatona da AGES, gostaria de cadastrar integrantes a partir dos alunos inscritos, para conseguir formar equipes mistas dos cursos de computação;
- Eu, como moderador da Hackatona da AGES, gostaria de cadastrar equipes com os integrantes do evento, para finalizar o processo de definição de equipes;
- Eu, como moderador da Hackatona da AGES, gostaria de remover integrantes do evento e da equipe, em situações de desistência ou não concordância com as regras;
- Eu, como moderador da Hackatona da AGES, gostaria de remover uma equipe do evento, em situações de desistência ou não conformidade com as regras;

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

- Eu, como moderador da Hackatona da AGES, gostaria de validar os times cadastrados para verificar se os mesmos se encontram em conformidade com as regras do evento;
- Avaliador:
 - Eu, como avaliador da Hackatona da AGES, gostaria de avaliar os projetos apresentados com notas de 1 a 5, para que o vencedor seja eleito de maneira parcial.
 - Eu, como avaliador da Hackatona da AGES, gostaria de avaliar os pontos individuais por projeto, como pitch, inovação, processo de desenvolvimento e funcionamento do software para que todas as equipes tenham chances iguais de vencer independente do seu nível técnico e maturidade profissional;

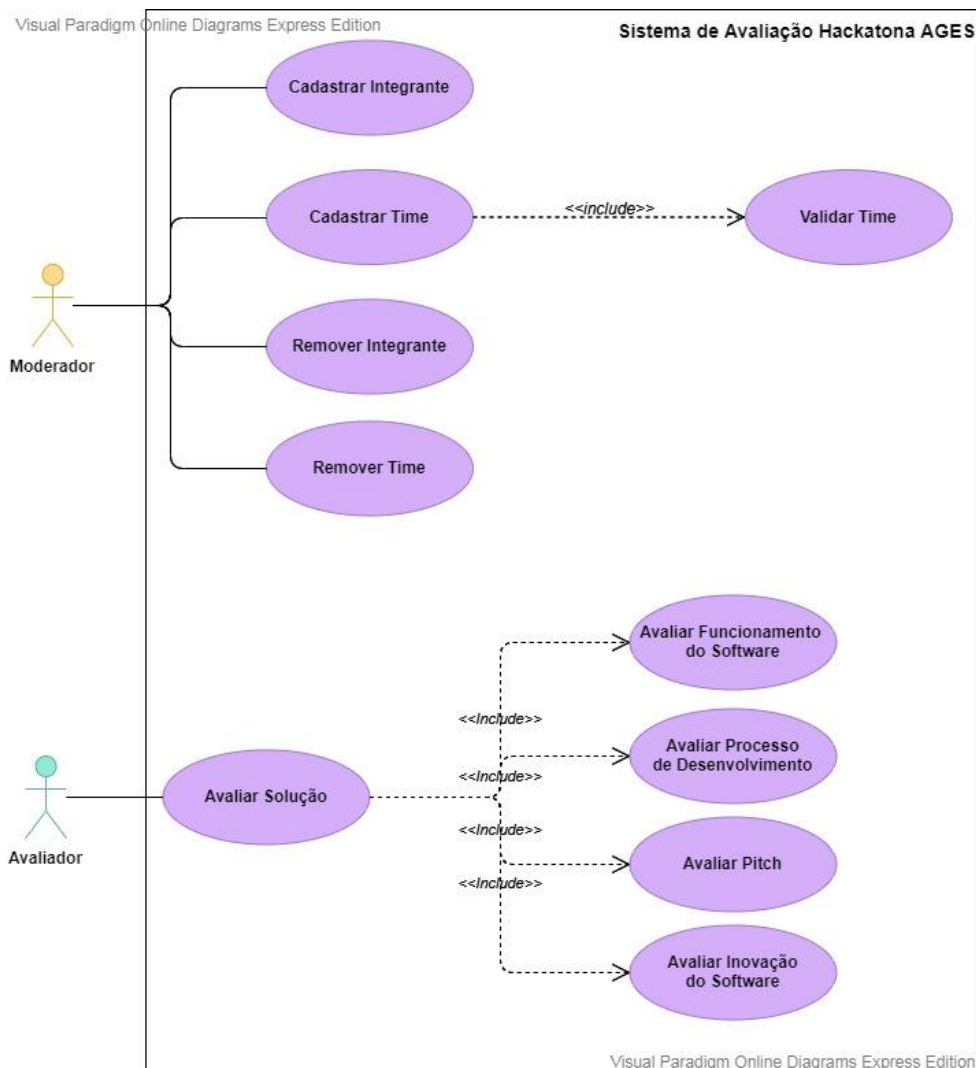


Figura 3: Diagrama de Caso de Uso

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

4.1 Visão Geral

4.2 Pacotes importantes para a arquitetura

A arquitetura da API conta com quatro principais pacotes, sendo estes: *controller*, *model*, *repository* e *service*. Cada um dos pacotes é responsável por armazenar as classes que possuem responsabilidades específicas sobre a API.

Controller: contém os métodos disponíveis para o cliente realizar suas requisições de consumo de dados. Cada *endpoint* é mapeado para uma função específica em Java, que recebe os parâmetros necessários e os converte para objetos e envia então para a camada de serviços.

Service: contém os métodos de lógica de negócio. Possui métodos que manipulam os dados enviados pelas controladoras, seja para operar sobre eles ou requisitar informações ao banco de dados. Cada tipo de entidade de banco de dados possui sua própria classe de serviço. Por exemplo, a entidade “Avaliador” possui sua classe de serviço que é responsável por operar atividades realizadas por um avaliador, como avaliar um time. As consultas ao banco de dados são realizadas através dos seus respectivos repositórios.

Repository: contém interfaces Java que tem como funcionalidade operar sobre os dados do banco de dados, abstraindo as consultas clássicas em SQL (*insert*, *delete*, *update* e *select*), através de métodos criados nessas interfaces. Por exemplo, para evitar a construção de uma *query* de seleção, a interface Java pode conter apenas um método chamado *findById(Integer id)* e o *framework* ficará encarregado de construir uma busca pela tabela de sua respectiva entidade, filtrando pelo identificador passado por parâmetro. Estão diretamente acopladas ao pacote de entidades.

Model: contém as abstrações lógicas de cada entidade do banco de dados. As tabelas e relações entre colunas são mapeadas de maneira automática através de anotações Java, fornecendo a abstração necessária para o repositório conseguir realizar as operações de banco de dados apenas com essas informações. Cada classe Java indica uma tabela no banco de dados e seus atributos indicam suas respectivas colunas.

A Figura 3 representa o diagrama de pacotes da API:

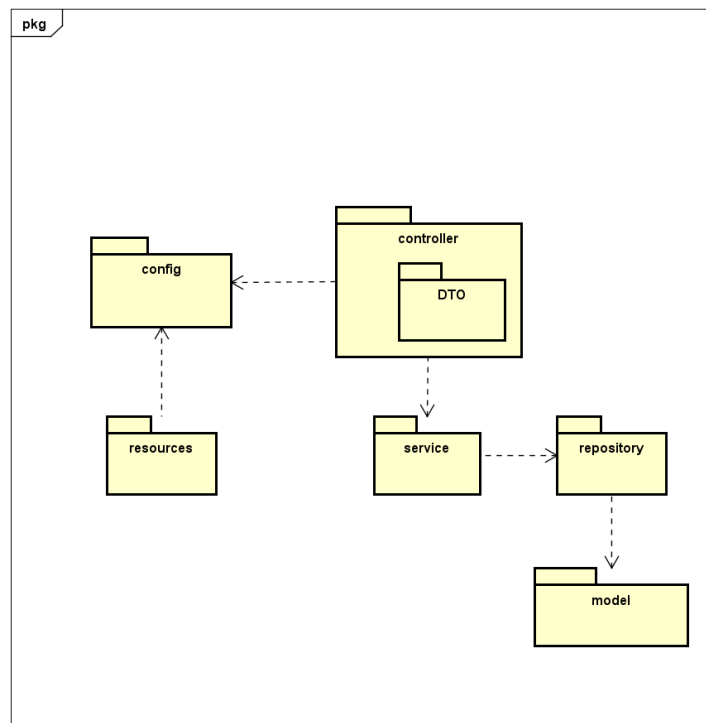


Figura 4 - Diagrama de pacotes da API

Hackatona da AGES	Versão: <1.2>
Documento de Arquitetura de Software	Data: <24/Mai/20>
Pontifícia Universidade Católica do Rio Grande do Sul	

4.3 Realizações de Caso de Uso

Elencamos alguns principais casos de uso através da especificação fornecida no enunciado do trabalho. Para tanto, as principais personas definidas foram “aluno”, “avaliador” e “moderador”. O tópico 4 apresenta detalhadamente algumas das US elaboradas pelo grupo iniciar o desenvolvimento desta solução.

5. Tamanho e Performance

Considerando que a aplicação deve rodar em diferentes dispositivos (notebooks ou smartphones), a performance quanto ao tempo de resposta e envio das informações e o tratamento das mesmas deve ser considerado. Dessa forma,

6. Qualidade

A qualidade do desenvolvimento desta solução foi notável durante a execução do projeto, uma vez que tínhamos uma boa definição dos requisitos e de como ficaria a divisão de entidades e suas responsabilidades dentro da API. Por exemplo, uma vez desenvolvida a funcionalidade de cadastro de alunos, cadastrar um avaliador ou time apresentava um comportamento semelhante necessitando apenas de algumas alterações no código. A manutenção do código também apresentou grande facilidade uma vez que as camadas estavam desacopladas, permitindo descobrir se um método não funcionava por dados não tratados, ausentes ou por lógica de implementação.