Universidade Federal de Goiás
Instituto de Informática
Padrões de Arquitetura de Software

Alunos:

Cauã dos Santos Rebelo - 201703743
Eric de Jesus Machado - 201703749
Fabiano Gomes Pires - 201709004
Raul Victor Dias - 201703769
Virgínia de Fernandes - 201700281

1. Escopo do projeto

O projeto UFG Impacto tem a intenção de disponibilizar para a população dados sobre o impacto social e econômico que os estudantes promovem na sociedade através de projetos (de pesquisa, ensino e extensão) e criação de empresas, isso é demonstrado através de indicadores obtidos através dos dados da UFG.

2. Requisitos Funcionais

Requisitos	Descrição
REQ01 - Visualizar dados estatísticos de ingressos da UFG	O usuário poderá visualizar os dados estatísticos de ingressos para os campus Goiânia, Goiás e Aparecida de Goiânia.
REQ02 - Visualizar dados estatísticos de egressos da UFG	O usuário poderá visualizar os dados estatísticos de egressos para os campus Goiânia, Goiás e Aparecida de Goiânia.
REQ03 - Visualizar dados estatísticos dos cursos da UFG	O usuário poderá visualizar os dados estatísticos dos cursos.
REQ04 - Visualizar dados sobre empresas criadas pelos egressos da UFG	O usuário poderá visualizar os dados das empresas criadas pelos egressos da UFG.
REQ05 - Visualizar dados sobre os projetos da UFG	O usuário poderá visualizar os dados dos projetos realizados na UFG.
REQ06 - Visualizar a WordCloud de projetos	O usuário poderá visualizar a WordCloud dos projetos realizados na UFG.

3. Requisitos Não Funcionais

Requisitos	Descrição
RNF01 - Usabilidade	Usabilidade tem como objetivo construir o sistema de forma que seja fácil de aprender e fácil de ser utilizado. Tendo como base o tempo e o esforço mínimo para alcançar um nível de desempenho no sistema e a velocidade relacionada a execução de tarefas juntamente com a redução de erros.
RNF02 - Modificabilidade	Modificabilidade tem como aspecto a capacidade do software de ser fácil de ser modificabilidade a fim de atender a futuras mudanças.

4. Padrões Arquiteturais

Para a construção do sistema foram utilizados 2 padrões arquiteturais: cliente-servidor e MTV.

Padrão Arquitetural	Descrição
Cliente-Servidor	O padrão cliente-servidor é organizado como um conjunto de serviços: serviços de servidores e serviços de clientes. Funciona basicamente utilizando o protocolo *request-reply*, ou seja, um cliente faz um pedido ao servidor e espera pela resposta, o servidor executa o serviço e responde ao cliente. Algumas observações: 1. Requer uma estrutura de rede para clientes acessarem os serviços. 2. Clientes sabem quais os serviços e servidores estão disponíveis. 3. Os servidores não sabem quem são os clientes.
MTV	O padrão MTV é uma derivação do MVC e é utilizado no framework Django. A nomenclatura significa respectivamente, model, template e view. Model contém a estrutura lógica

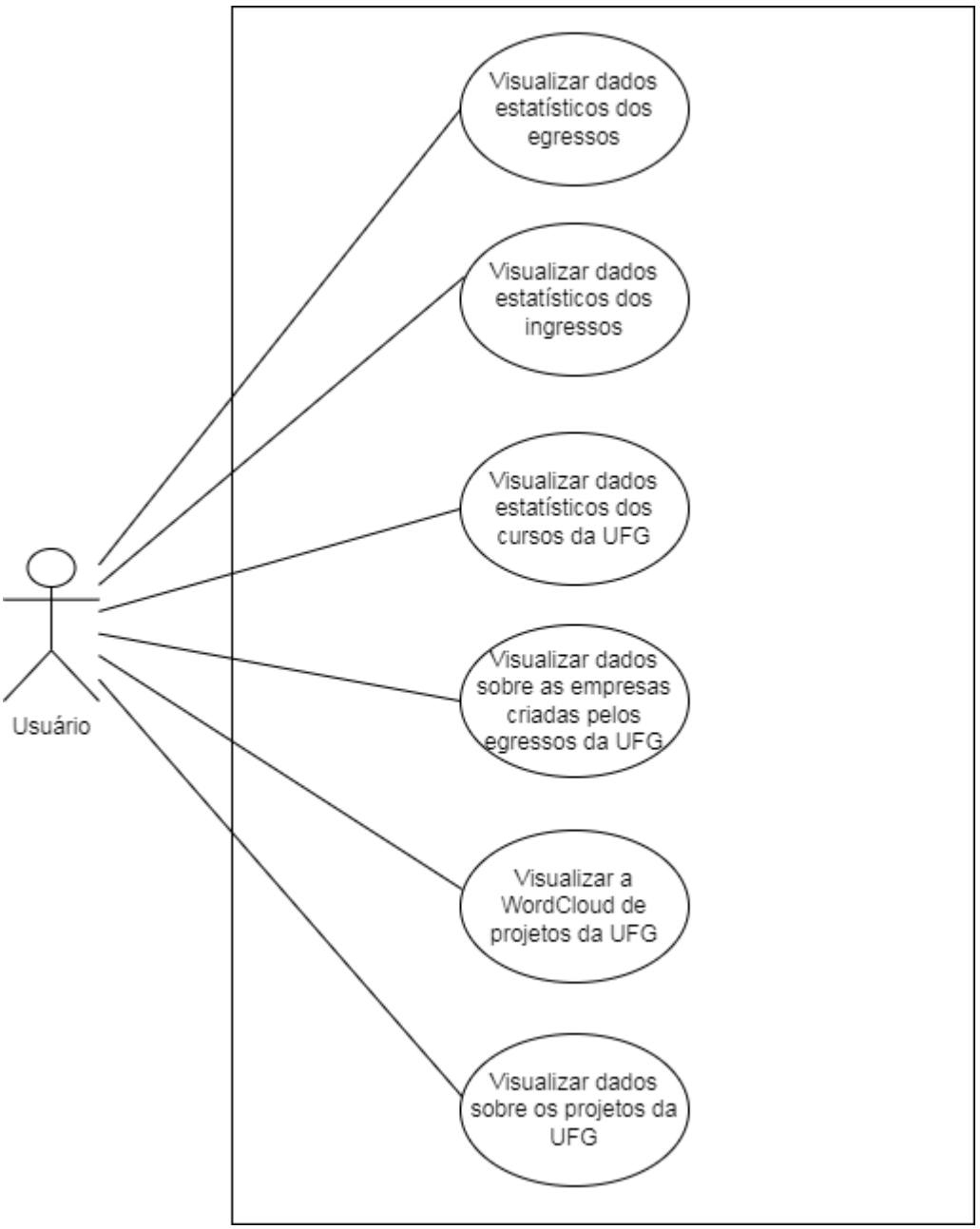
	do projeto e funciona como um intermediário para manipular dados entre o banco de dados e a View. View formata os dados que são vindos do banco através da Model para visualização. Template cuida da parte da visualização para o usuário final, é como o front-end de sua aplicação

5. The 4+1 View Model of Architecture

4.1. Visão de cenários (casos de uso)

Descreve o conjunto de funcionalidades de um sistema, baseando-se nas interações entre os objetos e os processos. As funcionalidades são mostradas do ponto de vista dos atores que fazem parte do sistema. Esta visão é utilizada para identificar elementos de arquitetura e mapeia o relacionamento entre as demais visões, justamente por mostrar a interação entre esses elementos.

Diagrama de casos de uso

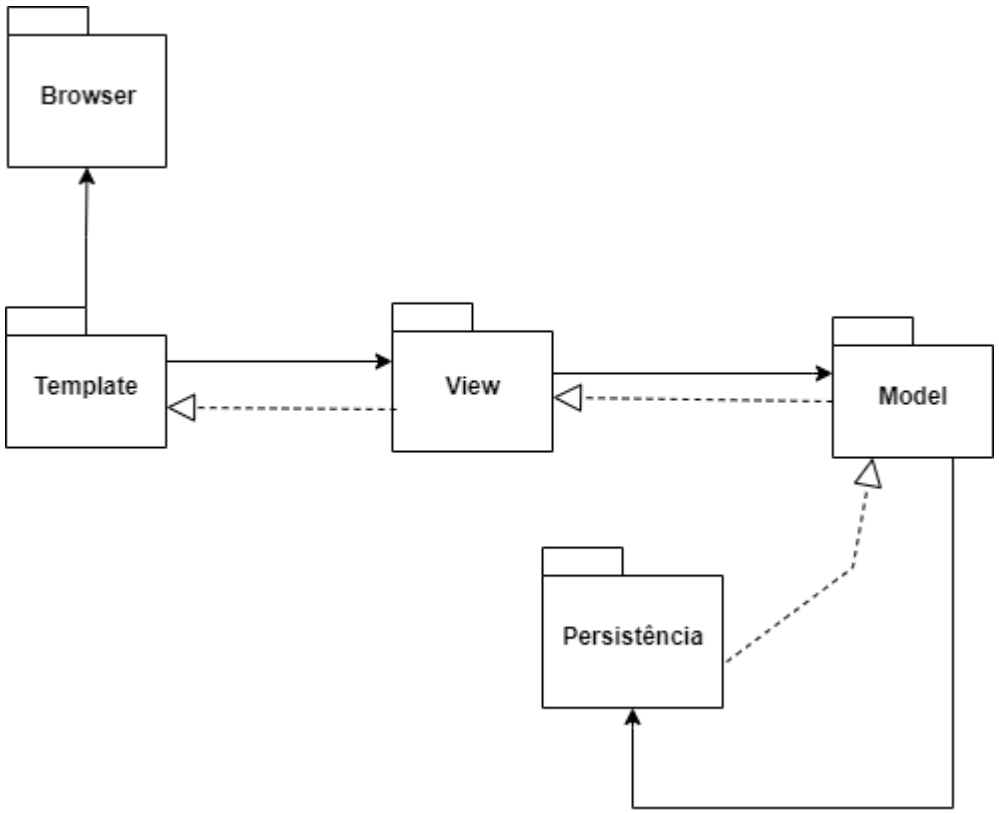


O diagrama de casos de uso representado acima descreve quem são os atores sistema (neste caso, existe apenas 1 (um) ator) e quais são suas principais interações com o sistema: visualizar dados estatísticos de egressos, visualizar dados estatísticos de ingressos, visualizar a wordcloud dos projetos da UFG, visualizar dados sobre os projetos da UFG, visualizar dados sobre os cursos da UFG e visualizar dados sobre as empresas criadas pelos egressos da UFG.

4.2. Visão lógica

Descreve como o sistema é estruturado, em termos de unidades de implementação. Os elementos são pacotes, classes e interfaces. O relacionamento entre os elementos mostra as dependências, as realizações de interface, os relacionamentos parte-todo e assim por diante.

Diagrama de pacotes (alto nível)

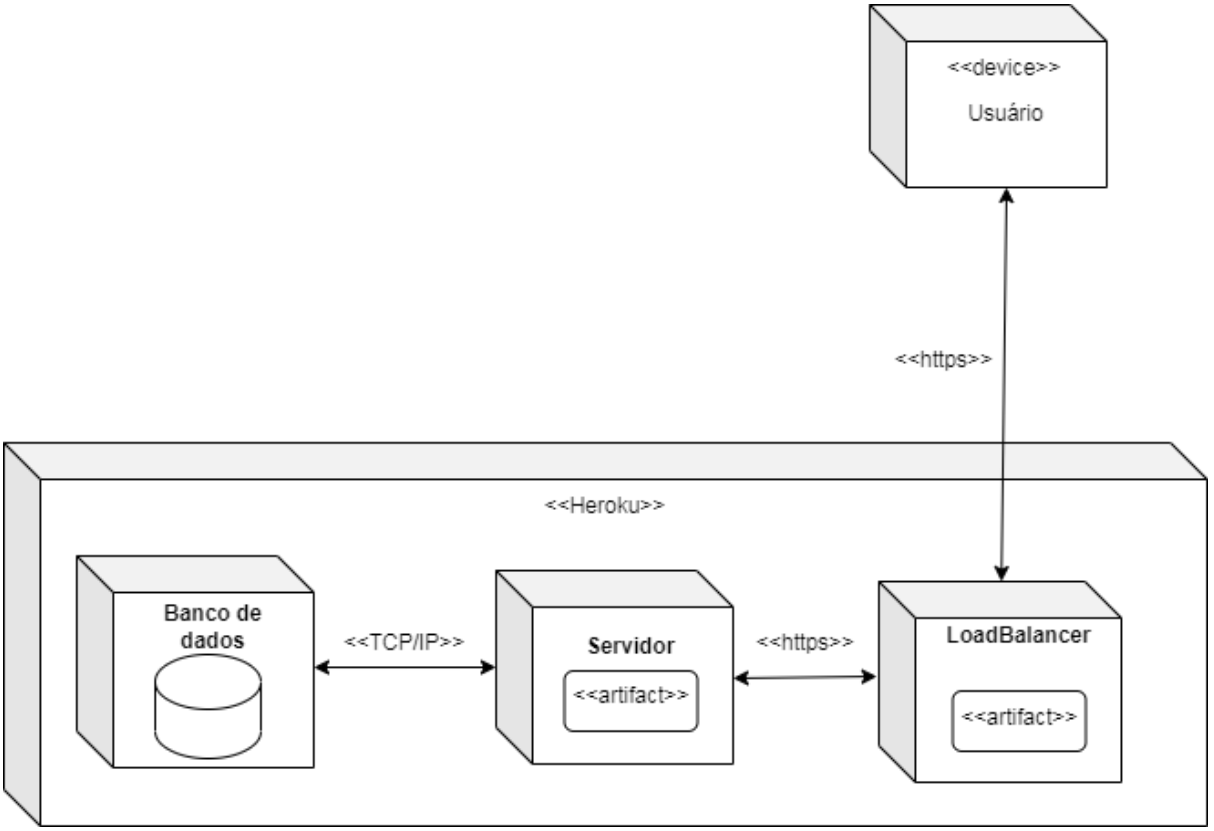


Browser: onde o usuário irá receber a informação passada pelo template.
Template: cuida da parte desta visualização para o usuário final, é como o front-end da aplicação.
View: formata os dados que são vindos do banco através da Model para visualização no template.
Model: contém a estrutura lógica do projeto e funciona como um intermediário para manipular dados entre o banco de dados e a View.
Persistência: contém o banco de dados.

4.3. Visão física

A visão física é responsável por apresentar o mapeamento entre unidades de software e hardware. Traz uma visão estática do aparato necessário de software e hardware subjacente para permitir que a arquitetura seja funcional.

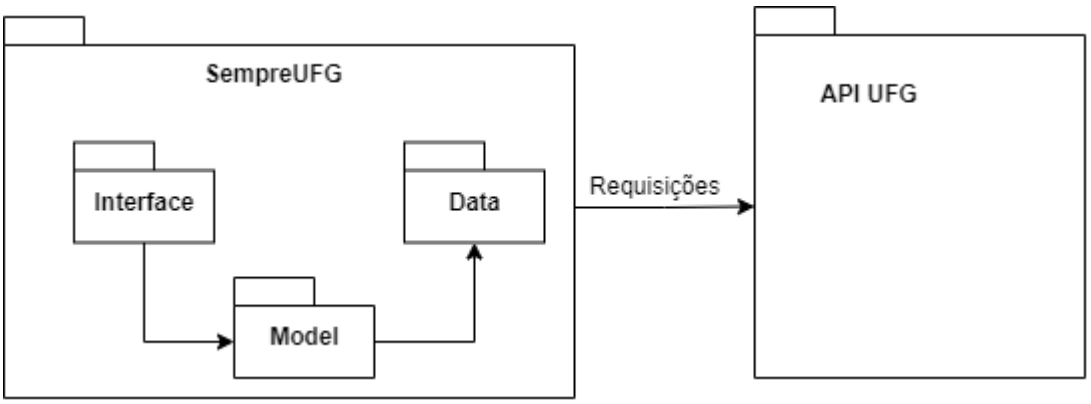
Diagrama de Implantação



Usuário: representa o usuário que acessa o sistema de algum dispositivo com acesso a internet. O acesso se dá através do protocolo HTTPS:
Banco de dados: representa a camada de persistência do sistema, onde os dados serão mantidos. O acesso do banco de dados com o servidor se dá através do protocolo TCP/IP.
Servidor: servidor que executa a aplicação.
LoadBalancer: tem como função manter a estabilidade de um servidor quando o tráfego e volume de dados é muito grande.

4.4. Visão de desenvolvimento

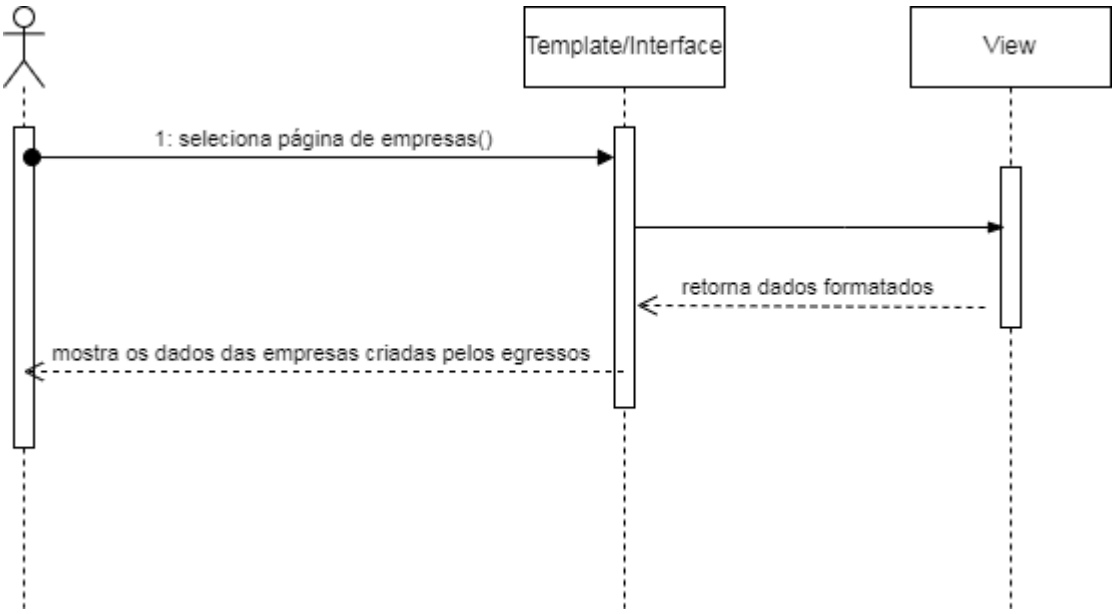
Descreve a organização do software em seu ambiente de desenvolvimento, ou seja, mostra como o software é decomposto para o desenvolvimento do mesmo. Ilustra o sistema do ponto de vista do programador e se preocupa com o gerenciamento do projeto. Esta visão também é conhecida como visão de implementação. Usa o Diagrama de pacotes para representá-la.



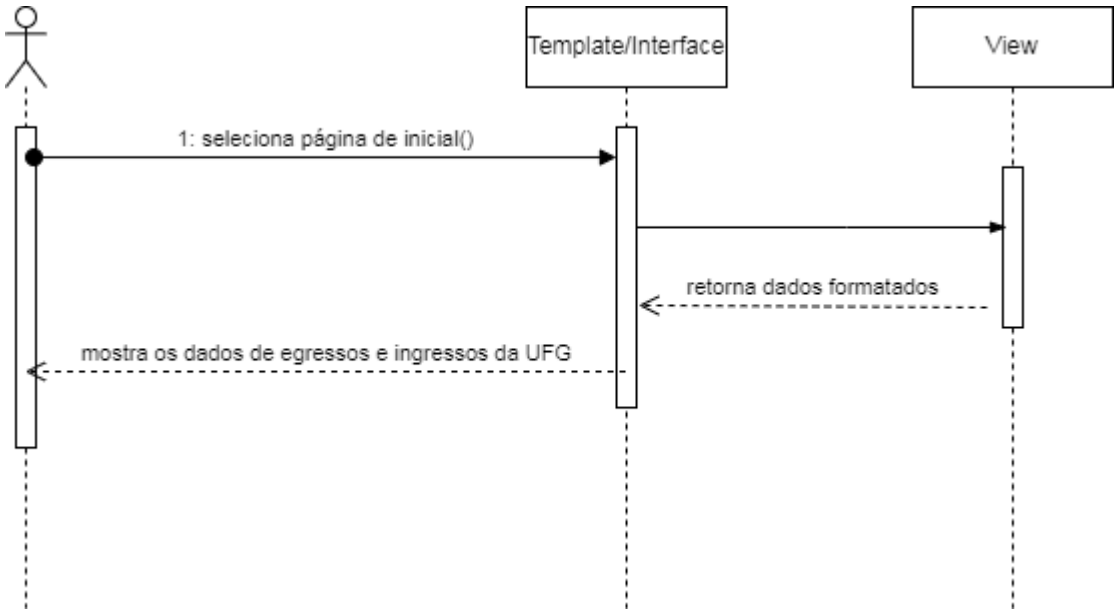
4.5. Visão de processos

A visão de processos tem como objetivo mostrar como será o fluxo de comunicação entre os processos, passagem de mensagens, atividades entre componentes e sequência de mensagens. Assim, serão descritos os principais casos de uso em seu fluxo principal, sem exceções e fluxos alternativos.

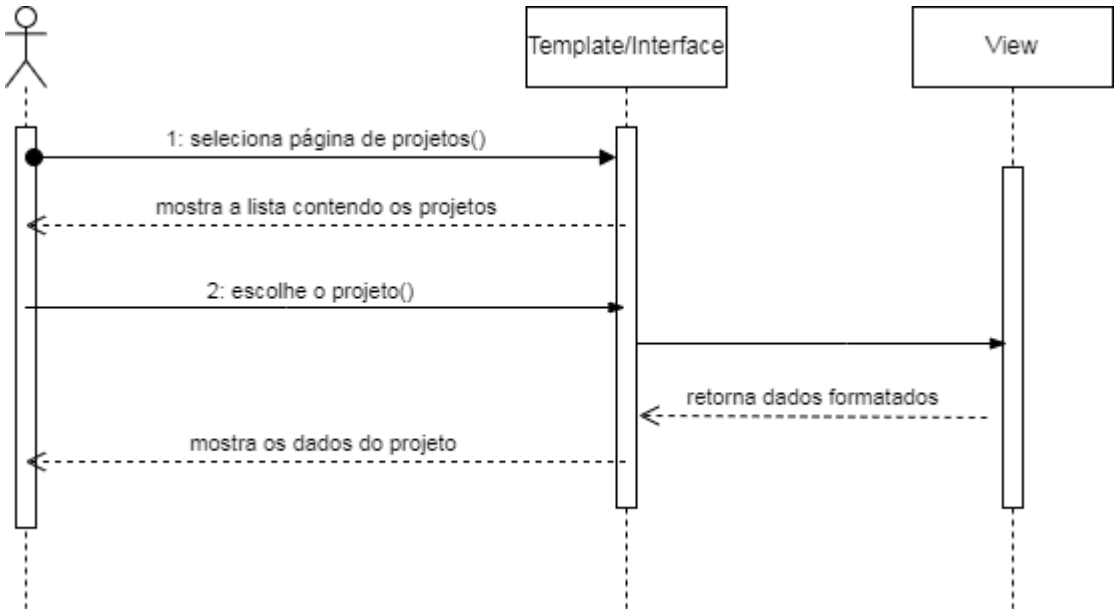
REQ: Visualizar empresas criados por egressos



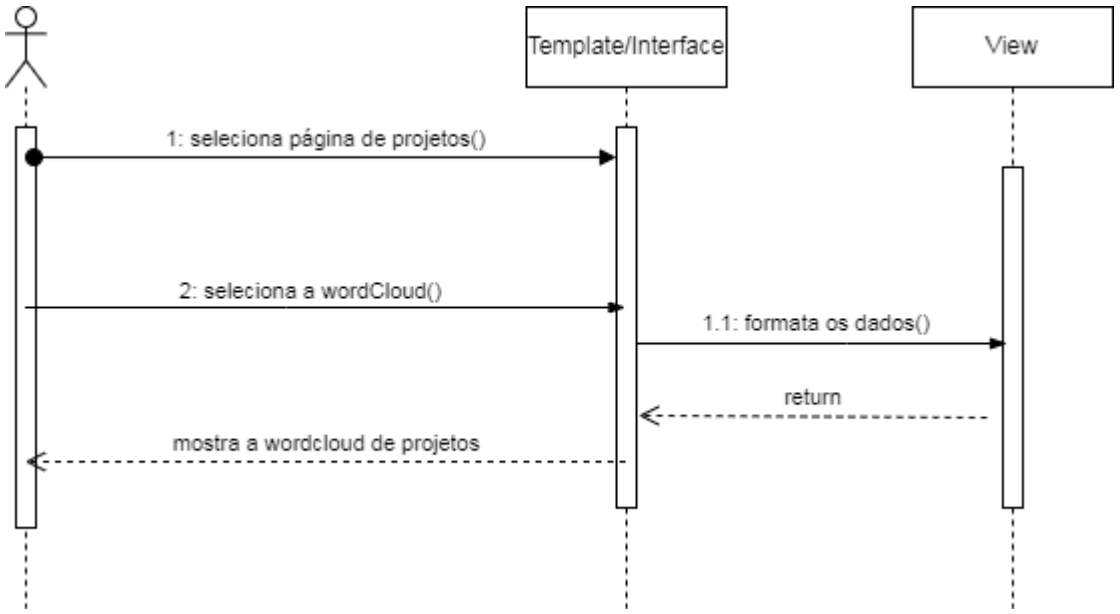
REQ: Visualizar dados estatísticos de egressos da UFG
REQ: Visualizar dados estatísticos de ingressos da UFG



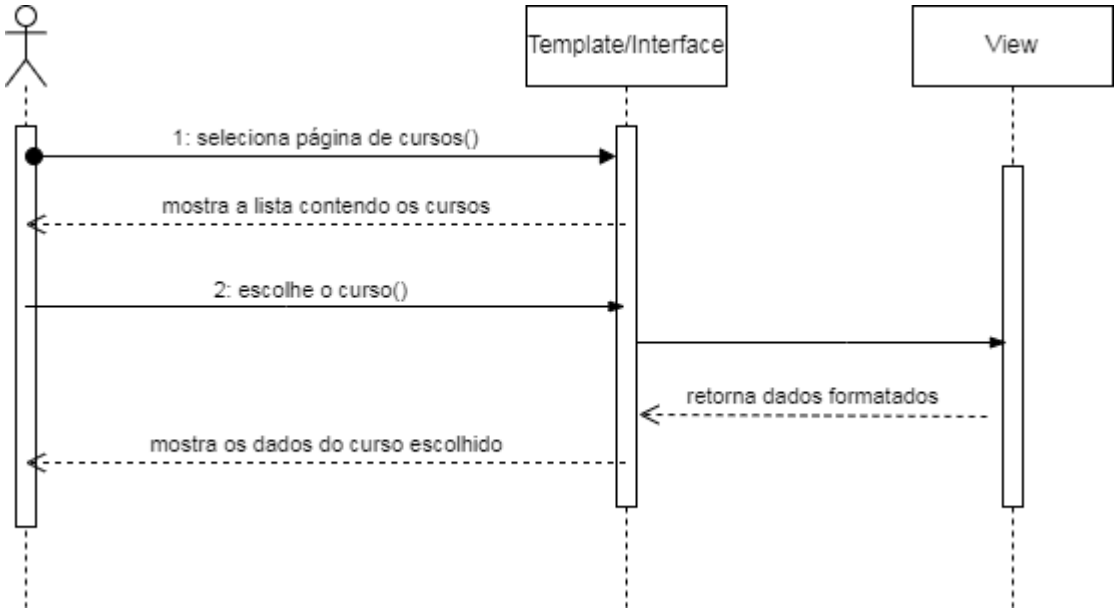
REQ: Visualizar dados sobre os projetos da UFG



REQ: Visualizar a WordCloud de projetos da UFG



REQ: Visualizar dados sobre os cursos de graduação da UFG



6. Decisões Arquiteturais

ID do RAS	01
Tipo de decisão arquitetural	Existenciais
Decisão	Padrão de projeto MTV
Justificativa	O padrão de projeto MTV será utilizado pela facilidade da utilização do mesmo, tendo a separação de interesses (estruturação da aplicação) e diminuição do acoplamento do sistema.
Descrição	O padrão de projeto MTV (Model, Template, View) utilizado no framework Django, é baseado em separação em três camadas, os quais estão conectadas entre si e desempenham papéis para que sua aplicação funcione. Model: Mapeamento do banco de dados para o projeto. Template: Páginas para visualização de dados. Normalmente, é aqui que fica o HTML que será renderizado nos navegadores. View: Lógica de negócio. É aqui que determinamos o que irá acontecer em nosso projeto.
TradeOff	+Modificabilidade +Desempenho +Escalabilidade +Usabilidade -Portabilidade

ID do RAS	02
Tipo de decisão arquitetural	Existenciais
Decisão	Framework Django
Justificativa	Com a utilização do framework Django teremos uma facilidade maior no desenvolvimento da aplicação web, já que o framework disponibiliza soluções para o desenvolvimento rápido com um design limpo e pragmático.
Descrição	O Django é um framework de aplicativos

	web gratuito e de código aberto escrito em Python, é descrito como uma estrutura da Web Python de alto nível que incentiva o desenvolvimento rápido e o design limpo e pragmático.
TradeOff	+Usabilidade +Modificabilidade +Manutenibilidade -Portabilidade

ID do RAS	03
Tipo de decisão arquitetural	Existenciais
Decisão	Cliente-Servidor
Justificativa	Com a utilização do padrão de arquitetura cliente-servidor teremos os seguintes benefícios: recursos centralizados, já que o servidor está no centro da rede, ele pode gerenciar recursos comuns a todos os usuários como, por exemplo, um banco de dados centralizado, a fim de evitar os problemas de redundância e contradição. Esse modelo oferece também maior segurança pois o número de pontos de entrada que permitem o acesso aos dados é menor, o gerenciamento do servidor e uma rede evolutiva. Graças a esta arquitetura, é possível remover ou adicionar clientes sem perturbar o funcionamento da rede e sem modificar o essencial.
Descrição	Arquitetura cliente-servidor ou modelo cliente-servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Existe um processo que é responsável pela manutenção da informação (servidores) e outro responsável pela obtenção dos dados (os clientes).
TradeOff	+Modificabilidade +Segurança +Escalabilidade -Desempenho -Disponibilidade

ID do RAS	04
Tipo de decisão arquitetural	Existenciais
Decisão	Arquitetura REST
Justificativa	É uma arquitetura bastante utilizada na web para desenvolvimento de API 's, fornece um guideline para construir um serviço coeso, escalável e performático, e além disso, é bastante simples de se aprender.
Descrição	O estilo arquitetural REST fornece padrões para comunicação entre sistemas. Os clientes enviam solicitações para o servidor e o servidor responde a essas solicitações.
TradeOff	+Interoperabilidade +Modificabilidade -Segurança -Desempenho

5. Referências

[1] LIVIA. Arquitetura REST: Uma alternativa para construção de Serviços Web, Devmedia. Disponível em: https://www.devmedia.com.br/arquitetura-rest-uma-alternativa-para-construcao-de-servicos-web/15150. Acesso em: 07/04/2022.
[2] O Modelo Cliente Servidor, UFRJ. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2016-1/16_1/p2p/modelo.html. Acesso em: 07/04/2022.
[3] O modelo MTV no Django, Caderno de Laboratorio. Disponível em: https://cadernodelaboratorio.com.br/o-modelo-mtv-no-django/. Acesso em: 07/04/2022.
[4] GOUR, Rino. Working Structure of Django MTV Architecture, Towards Data Science. Disponível em: https://towardsdatascience.com/working-structure-of-django-mtv-architecture-a741c8c64082/. Acesso em: 07/04/2022.