UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
BACHARELADO EM ENGENHARIA DE SOFTWARE
ARQUITETURA DE SOFTWARE

Trabalho final da disciplina:

ARQUITETURA DO SOFTWARE SEMPREUFG DE ACORDO COM A NORMA ISO/IEC/IEEE 42010

GOIÂNIA
2018-1

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
BACHARELADO EM ENGENHARIA DE SOFTWARE

ARQUITETURA DE SOFTWARE

Discentes:
Gustavo Batista
Murillo Nunes
Saulo Calixto

GOIÂNIA
2018-1

1 INTRODUÇÃO **4**

2 DESCRIÇÕES DA ARQUITETURA **4**

2.1 Introdução 4

2.2 Identificação, descrição da arquitetura e visão geral 4

2.3 Identificação dos stakeholders 5

2.4 Pontos de vista arquiteturais 6

2.4.1 Ponto de vista de Informação 6

Resumo do ponto de vista 6

Preocupações e não-preocupações 6

Stakeholders típicos 7

Tipos de modelos 7

Representação 7

Fontes 8

2.4.2 Ponto de vista do Desenvolvedor 8

Resumo do ponto de vista 8

Preocupações e não-preocupações 8

Stakeholders típicos 9

Tipos de modelos 9

Representação 9

Fontes 10

2.4.3 Ponto de vista do Implantador 10

Resumo do ponto de vista 10

Preocupações e não-preocupações 10

Stakeholders típicos 10

Tipos de modelos 10

Representação 10

Fontes 11

2.4.4 Ponto de vista do Projetista 12

Resumo do ponto de vista 12

Preocupações e não-preocupações 12

Stakeholders típicos 12

Tipos de modelos 12

Representação 12

2.4.5 Ponto de vista de Segurança 12

Resumo do ponto de vista 12

Preocupações e não-preocupações 12

Stakeholders típicos 13

Tipos de modelos 13

Fontes 13

2.4.6 Ponto de vista de UI 13

Resumo do ponto de vista 13

Preocupações e não-preocupações 13

Stakeholders típicos 13
Tipos de modelo 13
Fontes 13
2.4.7 Ponto de vista do Usuário 14
Resumo do ponto de vista 14
Preocupações e não-preocupações 14
Stakeholders típicos 14
Tipos de modelo 14
Representação 15
Fontes 15
2.4.8 Ponto de vista do Usuário Final 15
Resumo do ponto de vista 15
Preocupações e não-preocupações 15
Stakeholders típicos 15
Tipos de modelo 15
Representação 16
Fontes 16
2.5 Visões arquiteturais 16
2.5.1 Visão de Dados 16
Resumo da visão 16
Modelo 16
Embasamento 20
2.5.2 Visão de Dados 20
Resumo da visão 20
Modelo 20
Embasamento 30
2.5.3 Visão Física 31
Resumo da visão 31
Modelo 31
Elementos do modelo 32
Embasamento 32
2.5.4 Visão de Desenvolvimento 32
Resumo da visão 32
Modelo 32
Embasamento 34
2.5.5 Visão de Segurança 34
Resumo da visão 34
Modelo 34
2.5.6 Visão de Front-end 36
Resumo da visão 36
Modelo 36
Embasamento 37
2.5.7 Visão do Usuário 37
Resumo da visão 37

Modelo 37
Embasamento 39
2.5.8 Visão de Processos 39
Resumo da visão 39
Modelo 39
Embasamento 40
2.6 Decisões arquiteturais 40
[DSUFG01] 40
[DSUFG02] 41
[DSUFG03] 41
[DSUFG04] 41
[DSUFG05] 41
[DSUFG06] 42
[DSUFG07] 42
[DSUFG08] 42
[DSUFG09] 43
[DSUFG10] 43
[DSUFG11] 43
ANEXO A **43**
Matriz de Rastreabilidade - Atributos x Requisitos 44
ANEXO B **44**
Matriz de Rastreabilidade - Estilos Arquiteturais x Atributos 45

---

1 INTRODUÇÃO

O presente documento tem como objetivo documentar, de acordo com a norma ISO/IEC/IEEE 42010, a arquitetura do software SempreUFG, cujo requisitos podem ser encontrados em [1].

São discriminados aqui os requisitos mais relevantes para a arquitetura, os estilos arquiteturais escolhidos para atender o software como um todo além das decisões em geral que compõe o modelo arquitetural do *SempreUFG*.

2 DESCRIÇÕES DA ARQUITETURA

2.1 Introdução

Essa seção especifica as características da arquitetura e suas premissas, assim como a descrição da mesma, a identificação dos *stakeholders* do sistema e suas preocupações, uma definição de cada ponto de vista arquitetural usados na descrição da arquitetura, uma visão e um modelo arquitetural para cada ponto de vista definido, as decisões arquiteturais e as justificativas para cada decisão tomada ao longo do desenvolvimento e da definição da arquitetura do software SempreUFG.

2.2 Identificação, descrição da arquitetura e visão geral

A arquitetura definida pelo presente documento tem como interesse atender as especificações do software SempreUFG.

O SempreUFG é um software que tem como objetivo apoiar a Universidade Federal de Goiás (UFG) na gestão de seus egressos, gerenciando informações que incluem dados pessoais dos egressos, curso e informações sobre a área de conhecimento, participações do egresso como bolsista ou voluntário em programas acadêmicos, atuação profissional do egresso, informações sobre a continuidade do egresso em programas de pós graduação, além de opiniões dos egressos sobre o seu curso.

Informações mais detalhadas em relação ao software SempreUFG estão descritas em [1].

De acordo com uma intensa análise arquitetural realizada, foi explicitado que é preciso ter uma separação entre cliente e servidor, visto que se trata de software que deverá rodar na plataforma Web. Além disso, uma das maiores preocupações que ficaram evidentes foi com a segurança. Sendo assim, independente do estilo arquitetural escolhido, é de suma importância a priorização desse atributo de qualidade.

A seguir, na **Figura 1**, é possível visualizar, de uma forma geral, a arquitetura proposta para o software SempreUFG.

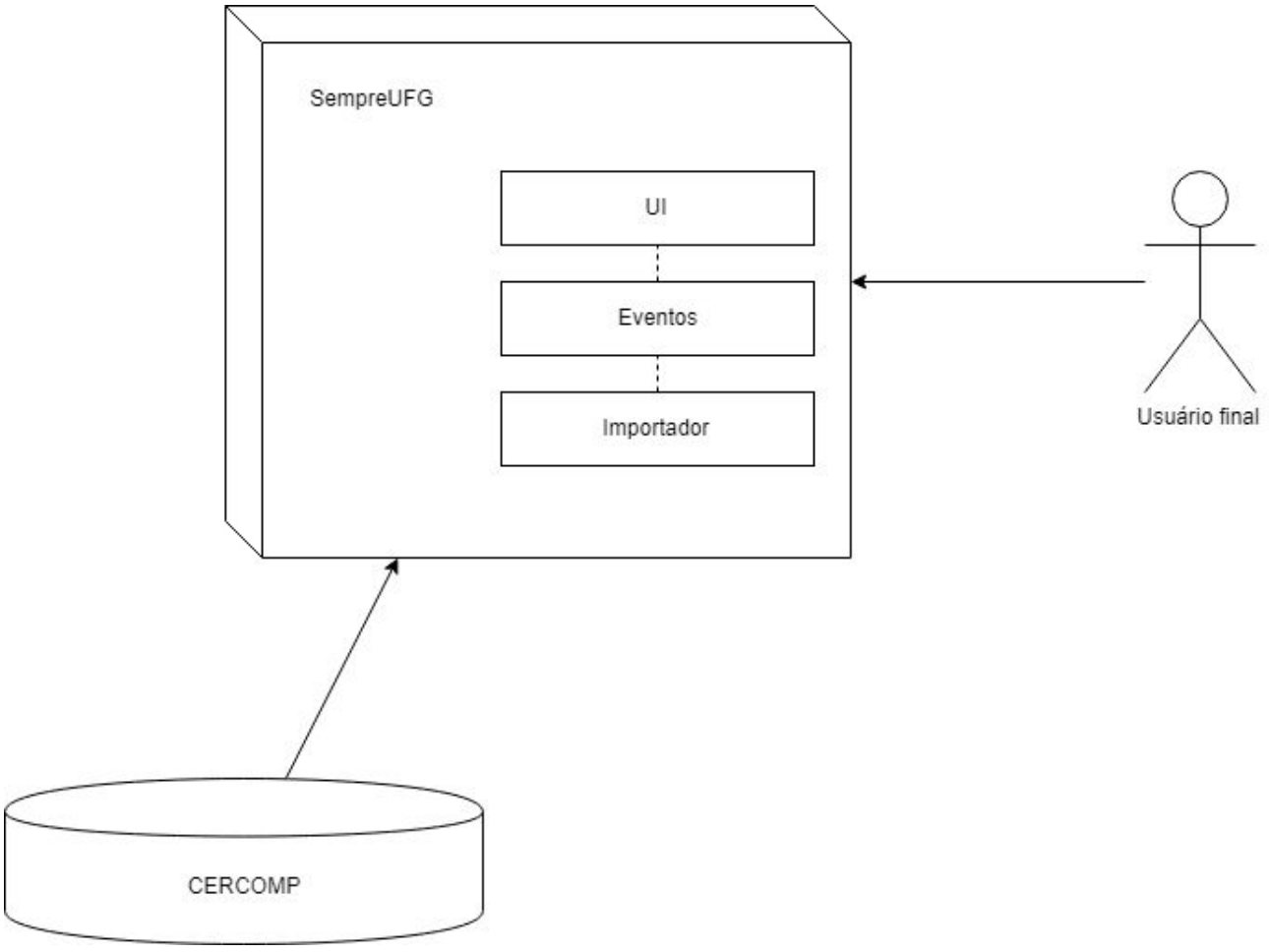

Figura 1 - Visão geral da arquitetura

2.3 Identificação dos stakeholders

Os seguintes stakeholders foram considerados durante a especificação da arquitetura para o software SempreUFG.

- Usuários do software;
- Operadores do software;
- Fábrica de Software do Instituto de Informática;
- Desenvolvedores do software;
- Mantenedores do software.

2.4 Pontos de vista arquiteturais

Sendo que os documentos que especificam os pontos de vista arquiteturais possuem características estruturais muito específicas, os mesmos serão disponibilizados em forma de anexos do presente documento. Este formato visa prover uma melhor organização e legibilidade da arquitetura.

2.4.1 Ponto de vista de Informação

Resumo do ponto de vista

O Ponto de vista de Informação descreve a maneira com que o sistema guarda, manipula, gerencia e distribui os dados e informações relacionados a ele.

Preocupações e não-preocupações

Preocupações:

- Estrutura e conteúdo de dados;
- Uso e propósito de dados;
- Proprietário dos dados;
- Dados confidenciais;
- Identificadores e mapeamento;
- Volatilidade de dados;
- Modelos de armazenamento de dados;
- Fluxo de dados;
- Consistência de dados;
- Qualidade de dados;
- Ciclo de vida dos dados;
- Retenção e arquivamento de dados.

Não-preocupações:

- Representação de incompatibilidades;
- Deficiência de correspondência de chaves;
- Complexidade de interfaces;
- Banco de dados sobrecarregado;
- Banco de dados distribuído inconsistentemente;
- Baixa qualidade de dados;
- Latência de dados excessiva;

- Métricas de volume inconsistentes.

Stakeholders típicos

Os stakeholders típicos relacionados a este ponto de vista são, principalmente, usuários, adquirentes, desenvolvedores, testadores e mantenedores. Porém, no geral, todos os stakeholders possuem um nível de interesse.

Tipos de modelos

- Modelo de estrutura de dados estático:

  A representação de como os dados serão armazenados em sua estrutura lógica.

- Modelo de fluxo de dados:

  A representação do fluxo de dados, por onde passam desde que são criados até serem utilizados ou arquivados.

- Modelo de ciclo de vida de dados:

  A representação do ciclo de vida dos dados, por onde são criados e quando são arquivados pelo sistema.

- Modelo de proprietário de dados:

  A representação do proprietário de dados.

- Modelo de metadados:

  A forma como serão representados os metadados.

Representação

O Modelo de Entidade-Relacionamento será utilizado para representar a estrutura estática dos dados e metadados. A seguir, na **Figura 2**, um exemplo da representação.

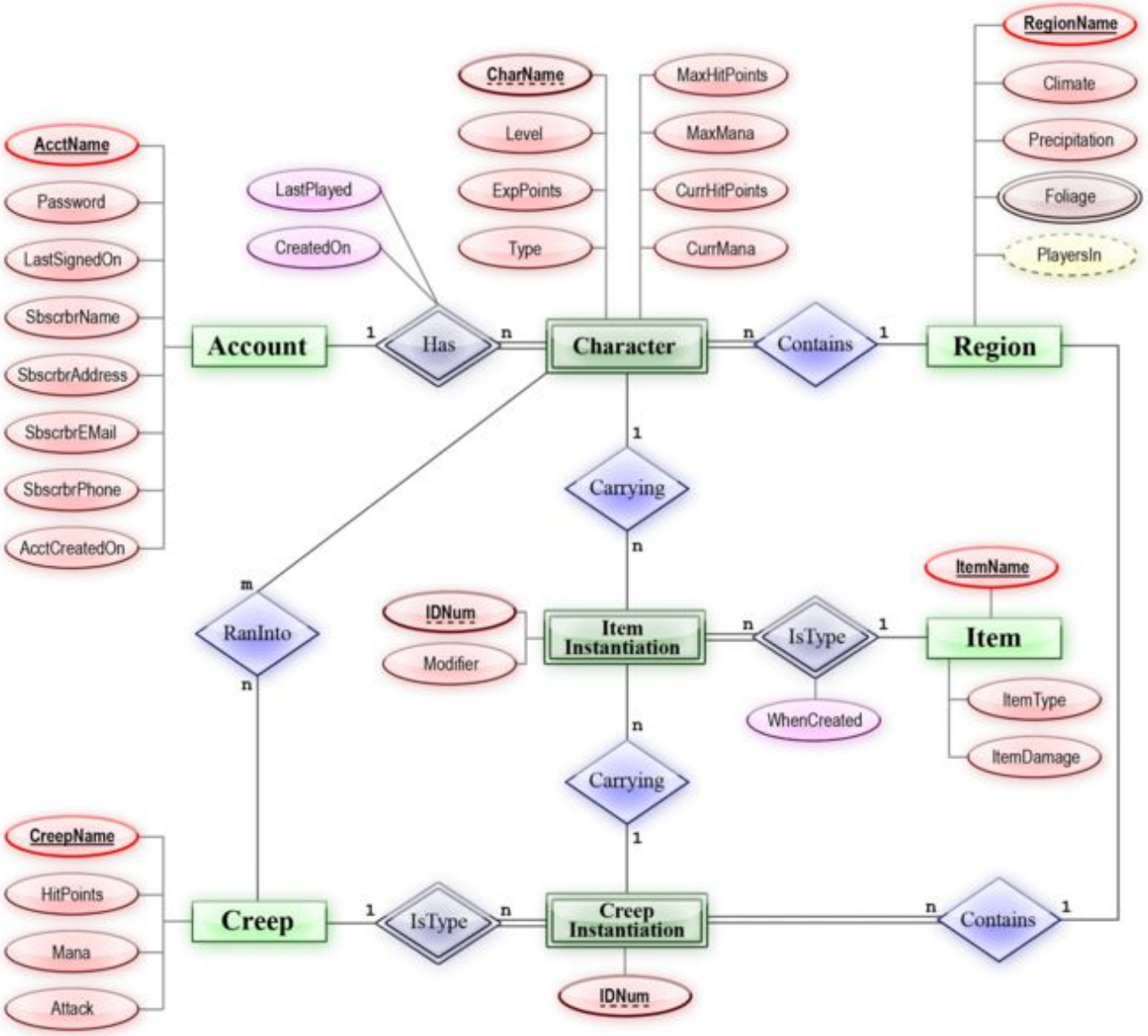

Figura 2 - Exemplo do Modelo Entidade-Relacionamento

Fontes

[2] N. Rozanski and E. Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.

[3] Chen, Peter, The Entity Relationship Model: Toward a Unified View of Data.

2.4.2 Ponto de vista do Desenvolvedor

Resumo do ponto de vista

O Ponto de vista do Desenvolvedor descreve a maneira com que o sistema deve ser implementado e organizado.

Preocupações e não-preocupações

Preocupações:

- Organização de métodos;

- Padronização de projeto;
- Padronização de testes;
- Organização de códigos.

Não-preocupações:

- Visão alto nível;
- Falta de foco.

Stakeholders típicos

Os stakeholders típicos relacionados a este ponto de vista são engenheiros de produção, desenvolvedores de software e testadores.

Tipos de modelos

- Modelo de estrutura de métodos:

  A representação como os métodos devem ser estruturados e organizados.

- Diagrama de classes

  A representação de como as classes e seus atributos devem ser organizados, e as relações entre si.

Representação

Para representar a solução e a organização lógica do sistema, será utilizado o Diagrama de Classes. A seguir, na **Figura 3**, um exemplo da representação.

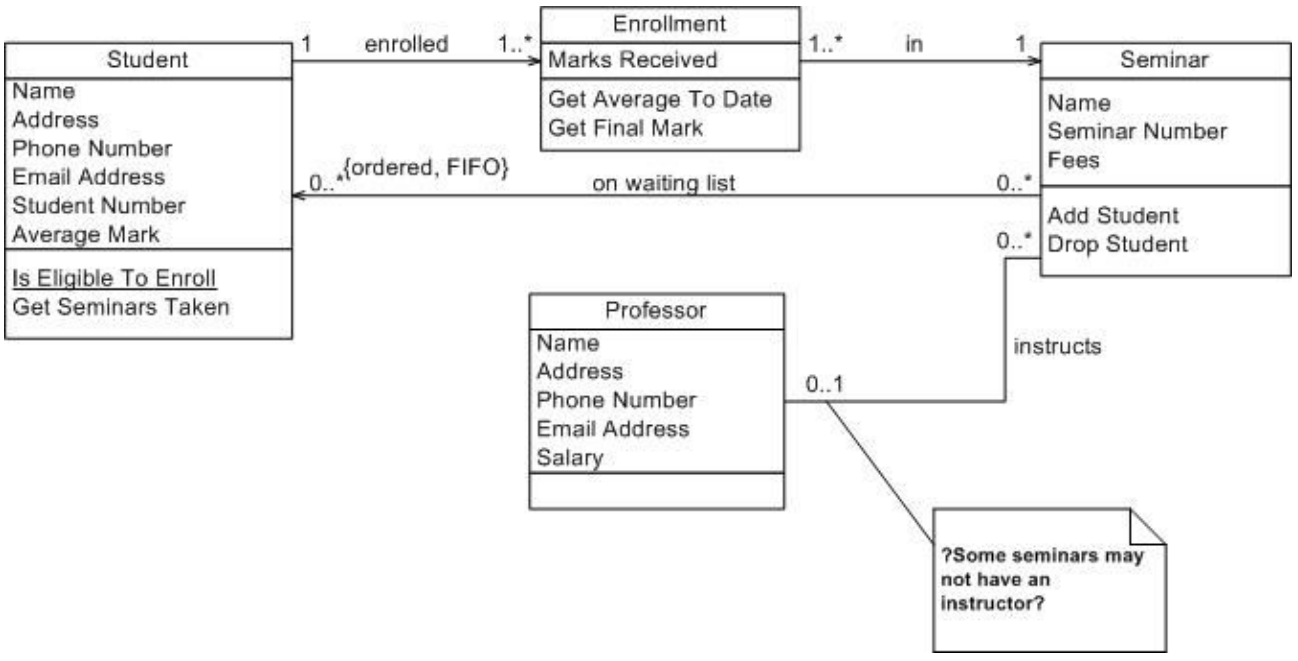

Figura 3 - Diagrama de Classes

Fontes

[4] N. Rozanski and E. Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.

[5] UML 2 Class Diagrams http://www.agilemodeling.com/artifacts/classDiagram.htm.

2.4.3 Ponto de vista do Implantador

Resumo do ponto de vista

O Ponto de vista do Implantador descreve a forma como ocorre a passagem do software do desenvolvimento para a produção, isto é, ao ambiente para o qual ele foi desenvolvido e deverá operar.

Preocupações e não-preocupações

Preocupações:

- Versão estável e executável do software;
- Disponibilidade de recursos no ambiente de produção;
- Coleta de informações em cada fase da implantação;
- Configuração do sistema no ambiente de produção;
- Adaptação do sistema após sua ativação;
- Atualização do sistema após sua ativação;
- Documentação da implantação.

Não-preocupações:

- Instabilidades relacionadas ao fornecimento de recursos computacionais;
- Treinamento de usuários.

Stakeholders típicos

Os stakeholders típicos relacionados a este ponto de vista são, inicialmente, implantadores, usuários do sistema, mantenedores e desenvolvedores. De uma forma geral, todos os stakeholders.

Tipos de modelos

- Diagrama de Implantação:

  A representação de como o sistema deverá ser estruturado em sua implantação.

Representação

Para representar a solução de implantação do sistema, será utilizado o Diagrama de Implantação. A seguir, na **Figura 4**, um exemplo da representação.

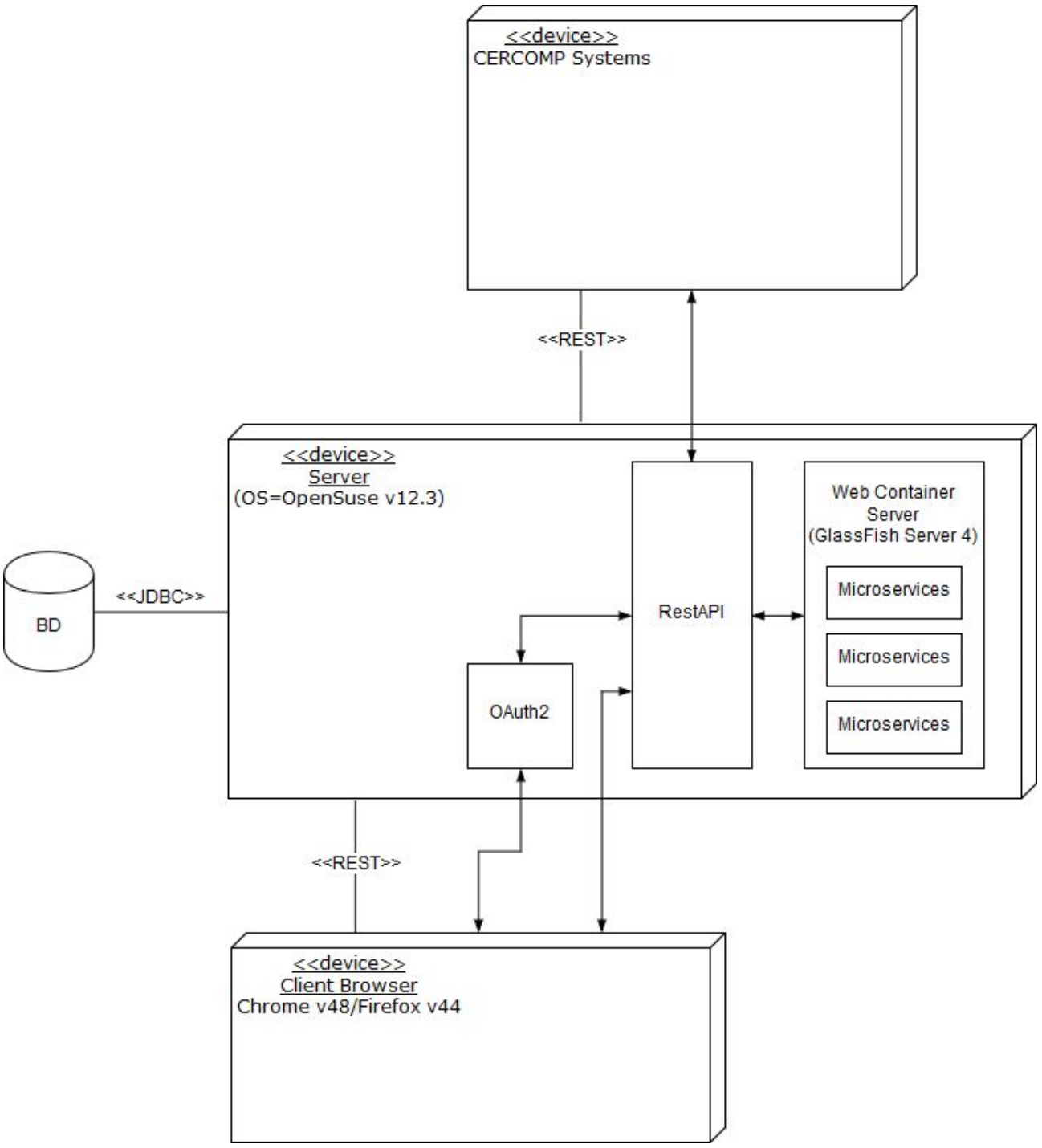

Figura 4 - Diagrama de Implantação

Fontes

[6] Object Management Group (OMG). Deployment and Configuration of component-based Distributed Applications-DEPL. OMG specification (2006).

[7] Carzaniga, A., et al. A Characterization Framework for Software Deployment Technologies — Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado (1998).

2.4.4 Ponto de vista do Projetista

Resumo do ponto de vista

O Ponto de vista do Projetista descreve os componentes principais do software e como ele se comunicam entre si e com sistemas externos.

Preocupações e não-preocupações

Preocupações:

- Responsabilidade dos módulos do sistema;
- Como as camadas vão se comunicar;
- Como os diferentes módulos se integram;
- Como é feita a comunicação com sistemas externos;
- Apresentar as partes principais do sistema.

Não-preocupações:

- Interface com o Usuário;
- Modelagem de Banco de Dados;
- Detalhamento dos componentes.

Stakeholders típicos

Todos os stakeholders.

Tipos de modelos

- Modelo de Pacotes:

  É um tipo de modelo que identifica os módulos dos sistema.

Representação

O modelo sugerido para esta representação é o Diagrama de Pacotes que tem como objetivo identificar as camadas do sistema bem como suas dependências.

2.4.5 Ponto de vista de Segurança

Resumo do ponto de vista

O Ponto de vista de Segurança descreve como é feita a segurança de dados do usuário, impedindo que informações sensíveis sejam acessadas de forma escusa.

Preocupações e não-preocupações

Preocupações:

- Apresentar os componentes que fazem parte do sistema de segurança;
- Como é feita a autenticação;
- Como os dados são protegidos.

Não-preocupações:

- Como é feita a implementação.

Stakeholders típicos

Todos os stakeholders.

Tipos de modelos

Não há um modelo previamente definido para essa representação. Dessa forma, foi criado um diagrama próprio para representar o sistema de segurança.

Fontes

Como o modelo foi proposto pela própria equipe, não foram encontradas fontes teóricas que o embasam.

2.4.6 Ponto de vista de UI

Resumo do ponto de vista

O Ponto de vista de UI descreve o funcionamento da camada *client side*, mostrando como os dados chegam até ela e como é feito o fluxo desses dados dentro da camada.

Preocupações e não-preocupações

Preocupações:

- Entrada de dados na camada *client*;
- Tratamento de dados na camada *client*;
- Fluxo de dados na camada *client*.

Não-preocupações:

- Com o server-side;
- Conexão com api;
- Origem dos dados.

Stakeholders típicos

Os stakeholders típicos relacionados a este ponto de vista são, a princípio, os usuários e operadores do sistema, além dos desenvolvedores.

Tipos de modelo

Não foi utilizado um modelo previamente definido para essa representação. Sendo assim, foi criado um diagrama próprio para que pudesse ser representado esse ponto de vista e sua respectiva visão.

Fontes

[8] Object Management Group (OMG). Deployment and Configuration of component-based Distributed Applications-DEPL. OMG specification (2006).

[9] Carzaniga, A., et al. A Characterization Framework for Software Deployment Technologies - Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado (1998).

2.4.7 Ponto de vista do Usuário

Resumo do ponto de vista

O Ponto de vista do Usuário diz respeito ao mapeamento dos Requisitos Funcionais (RFs) que, ocasionalmente, se tornarão funcionalidades, em uma forma na qual seja possível uma melhor visualização destas pelo usuário, cliente e/ou operador do software.

Preocupações e não-preocupações

Preocupações:

- Ilustrar Requisitos Funcionais (RFs) especificados no Documento de Requisitos.
- Demonstrar, de forma explícita, as interações entre o usuário e o *software* e/ou demais sistemas e o *software*.
- Estabelecer um modelo padrão para ilustrações desse tipo.
- Validar as funcionalidades descritas no Documento de Requisitos juntamente com o usuário, cliente e/ou operador do *software*.
- Prover uma visualização dos atores, casos de uso e como eles interagem entre si.
- Servir de base, de alto nível, para os desenvolvedores nas fases iniciais do desenvolvimento.

Não-preocupações:

- Ilustrar Requisitos Não-Funcionais (RNFs) especificados no Documento de Requisitos.
- Detalhar as ações e interações entre atores e casos de uso.
- Servir de base, de alto ou baixo nível, para os desenvolvedores nas fases finais do desenvolvimento.

Stakeholders típicos

Os stakeholders típicos relacionados a este ponto de vista são, a princípio, usuários do sistema, operadores do sistema, desenvolvedores e analistas de requisitos.

Tipos de modelo

- Modelo de Casos de Uso (MCU):

  O Modelo de Casos de Uso (MCU) é um modelo das funções pretendidas do sistema e suas vizinhanças, que serve como contrato entre o cliente e os desenvolvedores. Os casos de uso funcionam como um thread de unificação por todo o desenvolvimento do sistema.

Representação

O MCU é composto por um documento que descreve e detalha os casos de uso identificados para um sistema, com base nos seus requisitos funcionais, e uma série de diagramas de casos de uso que ilustram e representam os casos de uso especificados no documento.

Na respectiva visão deste ponto de vista é possível visualizar o Diagrama de Casos de Uso criado.

Fontes

[10] UFPR. Diretrizes: Modelo de Casos de Uso. Disponível online (acesso em maio de 2018): http://www.funpar.ufpr.br:8080/rup/process/modguide/md_ucmod.htm.

[11] HILLIARD, Rich. Architecture viewpoint template for ISO/IEC/IEEE 42010. 2012.

[12] ISO/IEC/IEEE. Norma de padronização ISO/IEC/IEEE 42010. 2007.

2.4.8 Ponto de vista do Usuário Final

Resumo do ponto de vista

Essa Viewpoint descreve a maneira que o usuário final irá interagir com o sistema.

Preocupações e não-preocupações

Preocupações:

- Atividades dos usuários;
- Fluxos alternativos;
- Atores.

Não-preocupações:

- Interfaces;
- Construção do sistema.

Stakeholders típicos

Usuário final.

Tipos de modelo

- Business Process Model Notation (BPMN):

  O modelo serve para apoiar a gestão de processos de negócios tanto para usuários técnicos e usuários de negócios, fornecendo uma notação que é intuitiva para os usuários corporativos ainda capaz de representar a semântica complexa do processo.

Representação

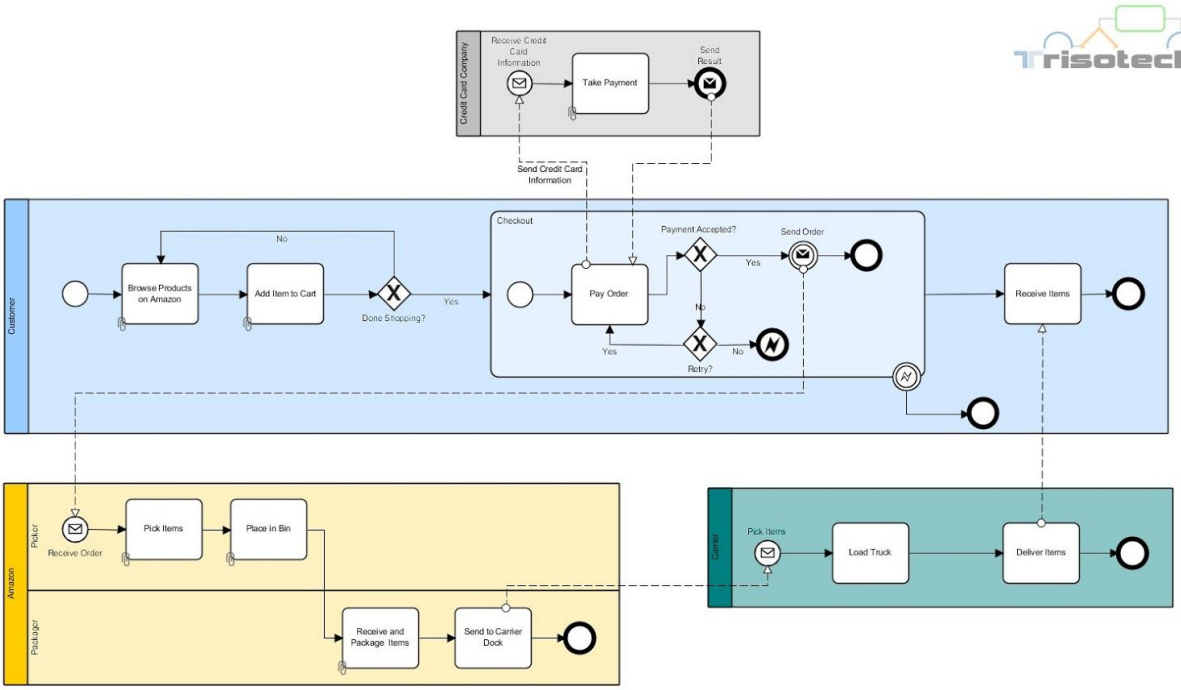

Figura 5 - BPMN

Fontes

[10] UFPR. Diretrizes: Modelo de Casos de Uso. Disponível online (acesso em maio de 2018): http://www.funpar.ufpr.br:8080/rup/process/modguide/md_ucmod.htm.
[11] HILLIARD, Rich. Architecture viewpoint template for ISO/IEC/IEEE 42010. 2012.
[12] ISO/IEC/IEEE. Norma de padronização ISO/IEC/IEEE 42010. 2007.

2.5 Visões arquiteturais

2.5.1 Visão de Dados

Resumo da visão

A visão de dados do sistema do SempreUFG é representada pelo Modelos de Entidade-Relacionamento (MER) que se referem aos requisitos de dados e a forma como devem ser organizados.

A seguir, na Figura 4, Figura 5, Figura 6, Figura 7, Figura 8, Figura 9 e Figura 10, está representada a Visão de Dados através do modelo escolhido.

Modelo

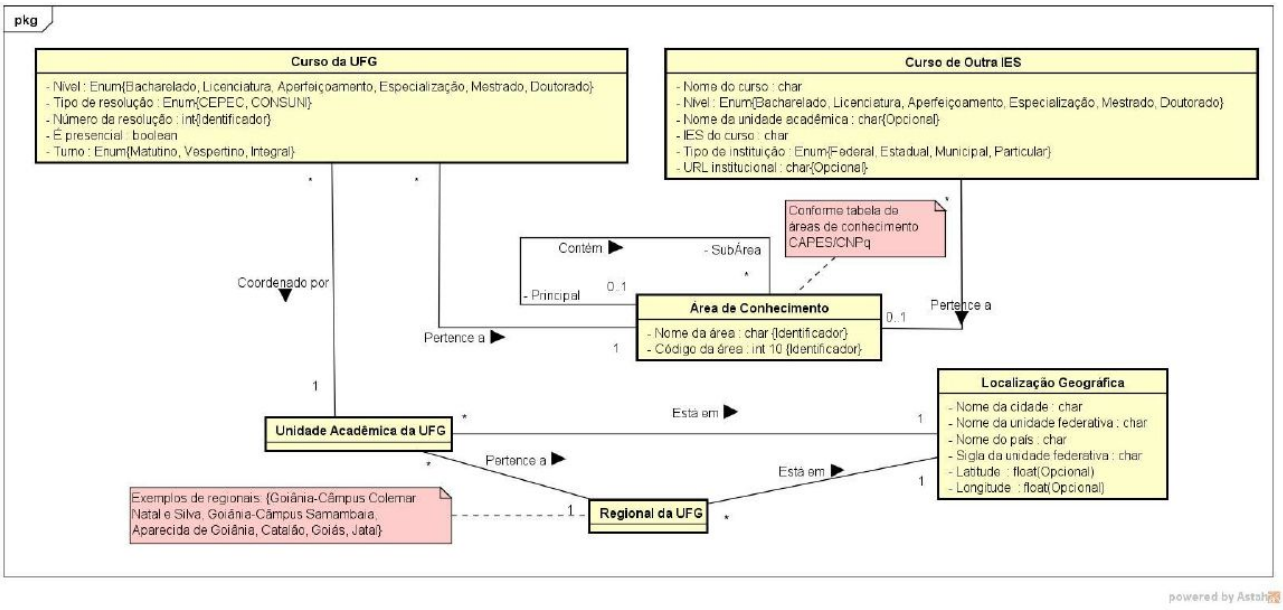

Figura 6 - Primeira parte do MER

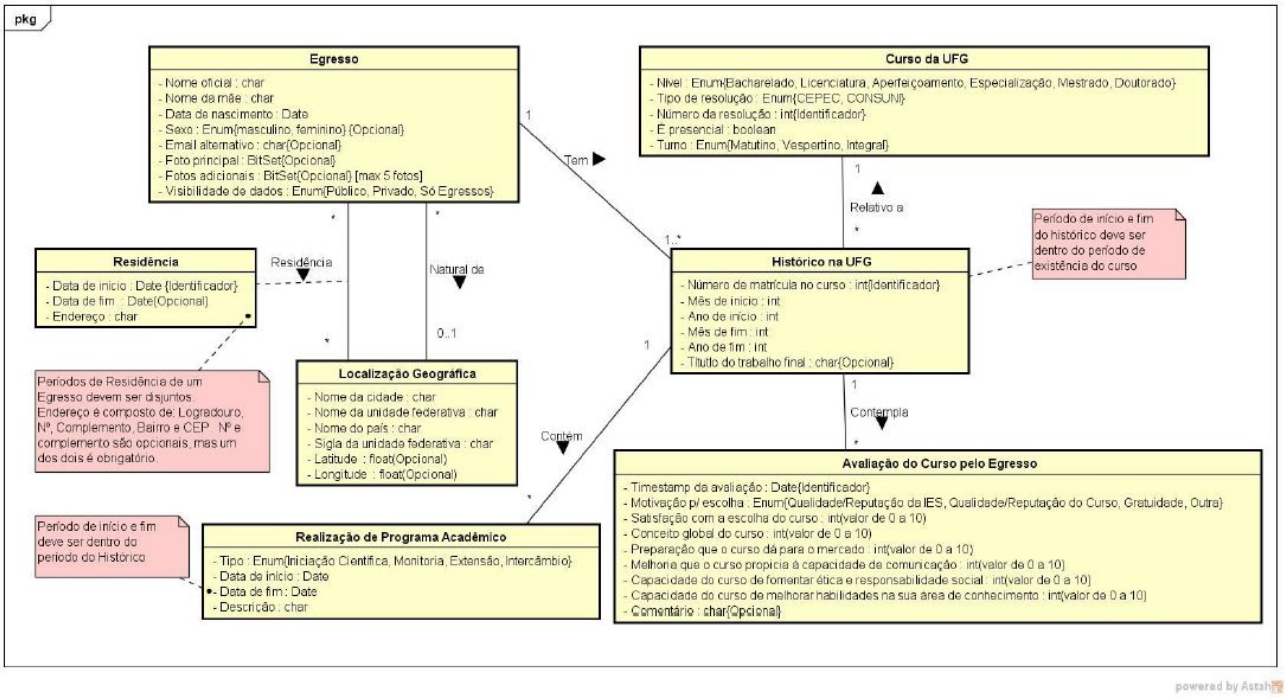

Figura 7 - Segunda parte do MER

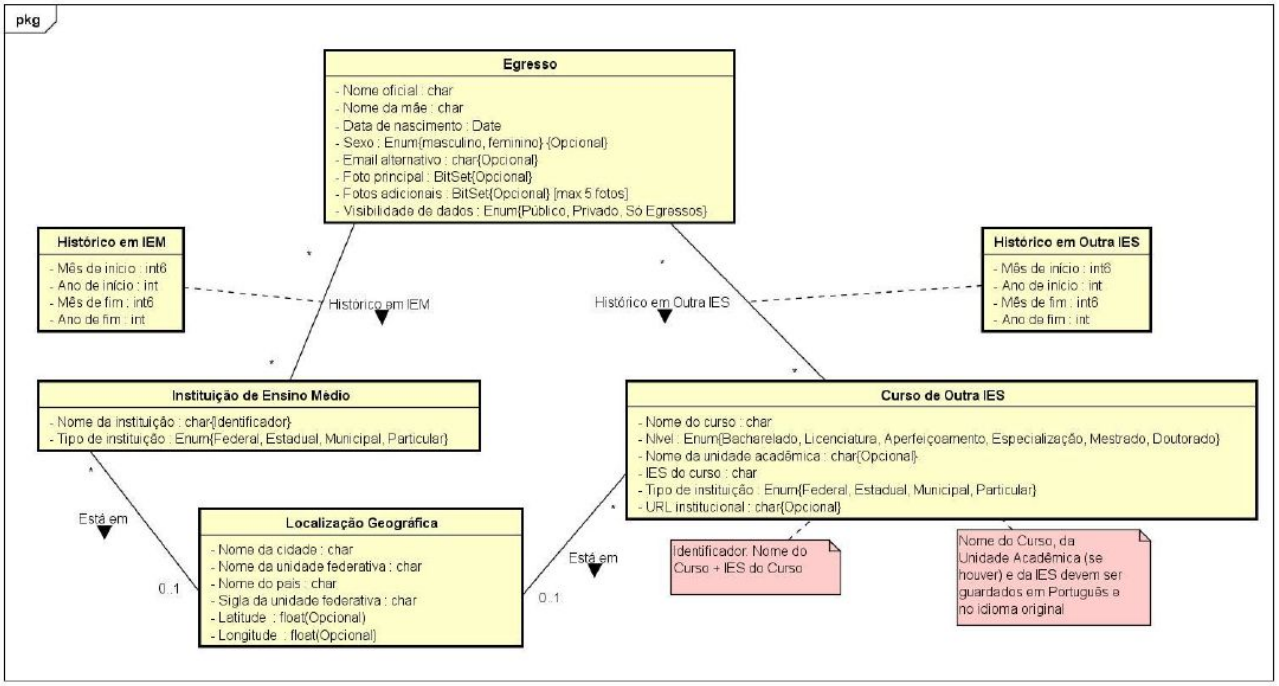

Figura 8 - Terceira parte do MER

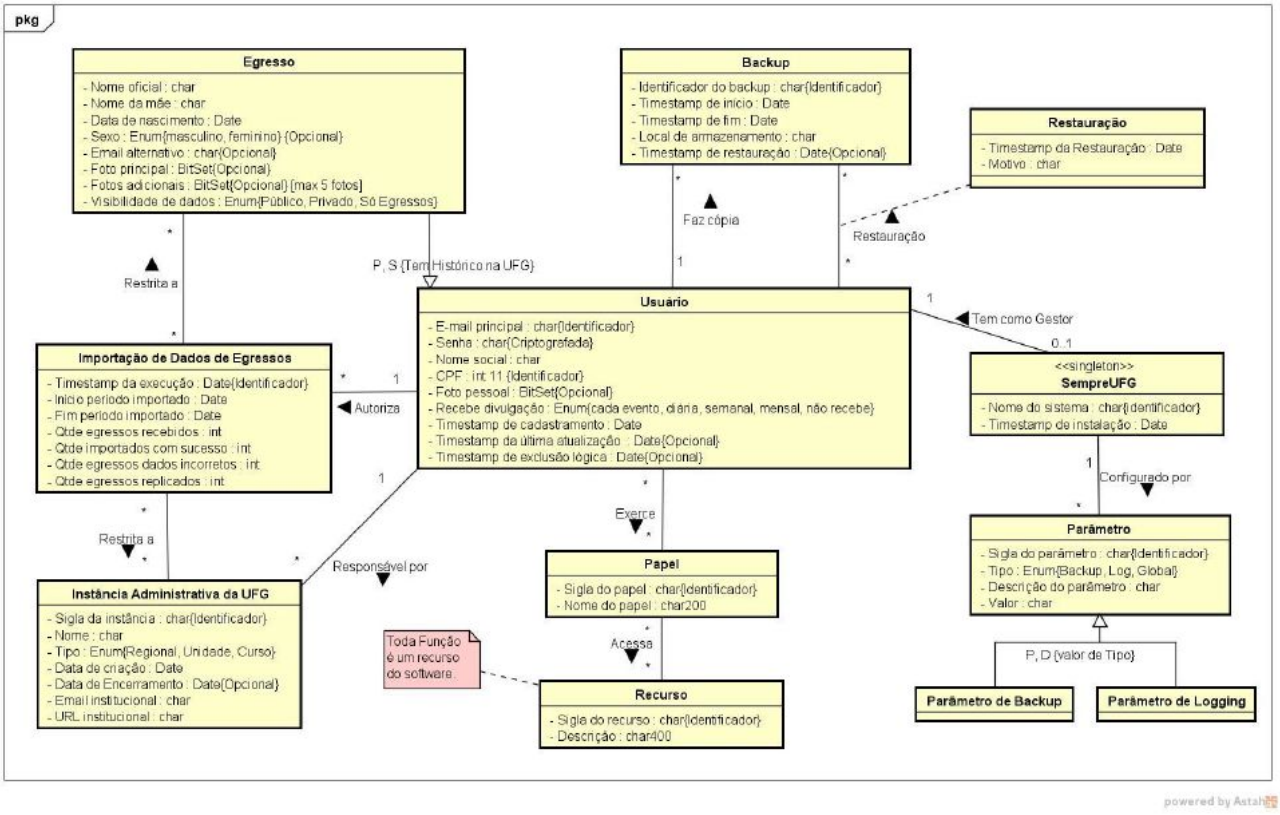

Figura 9 - Quarta parte do MER

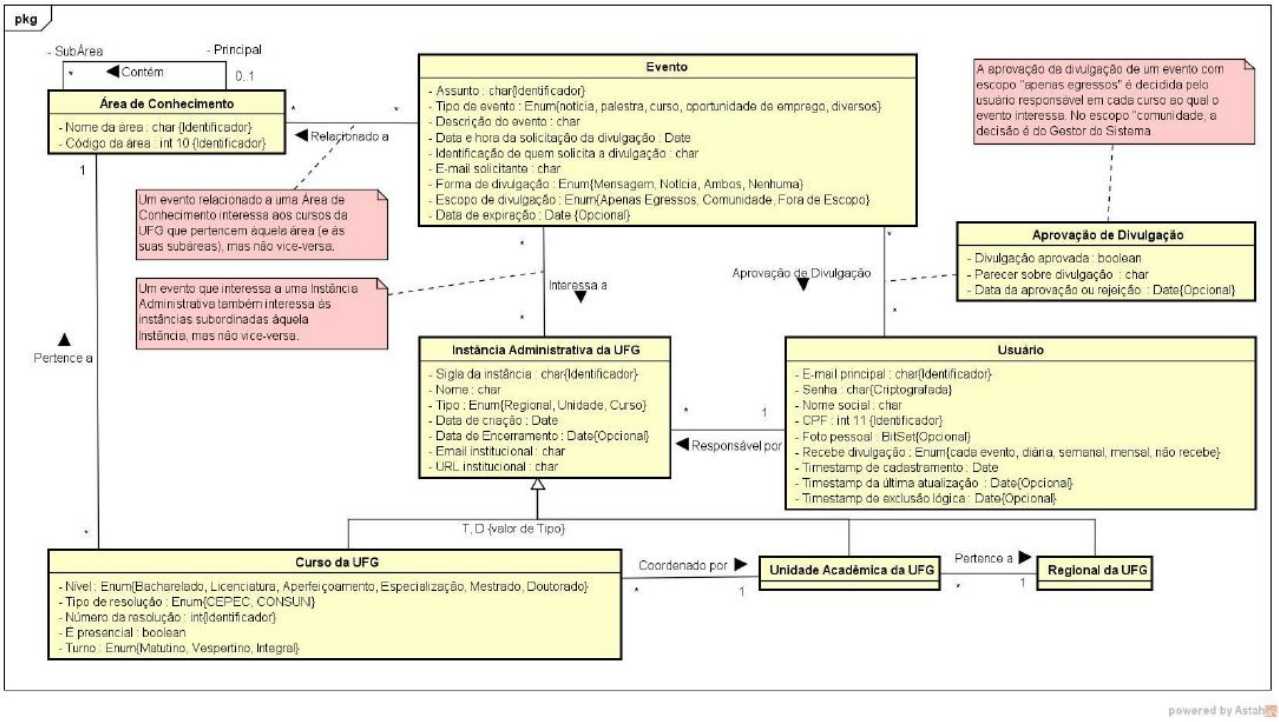

Figura 10 - Quinta parte do MER

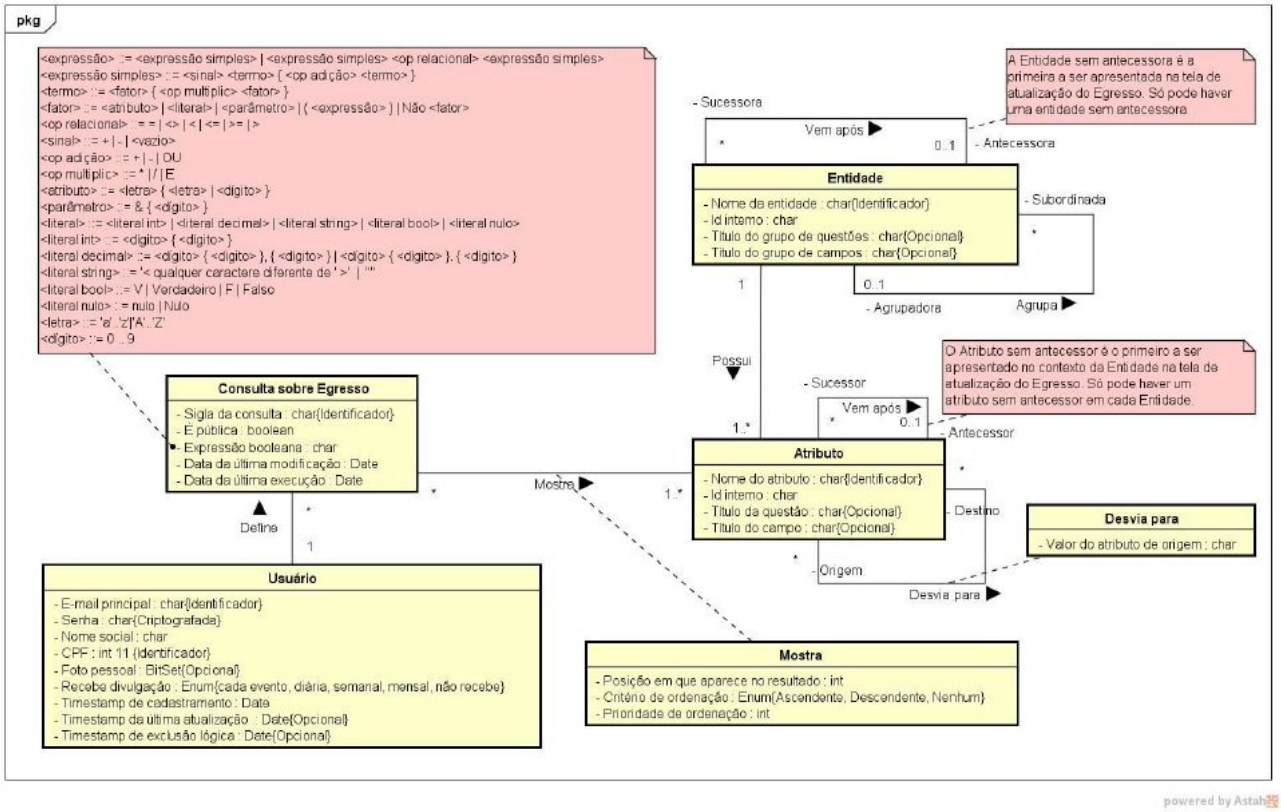

Figura 11 - Sexta parte do MER

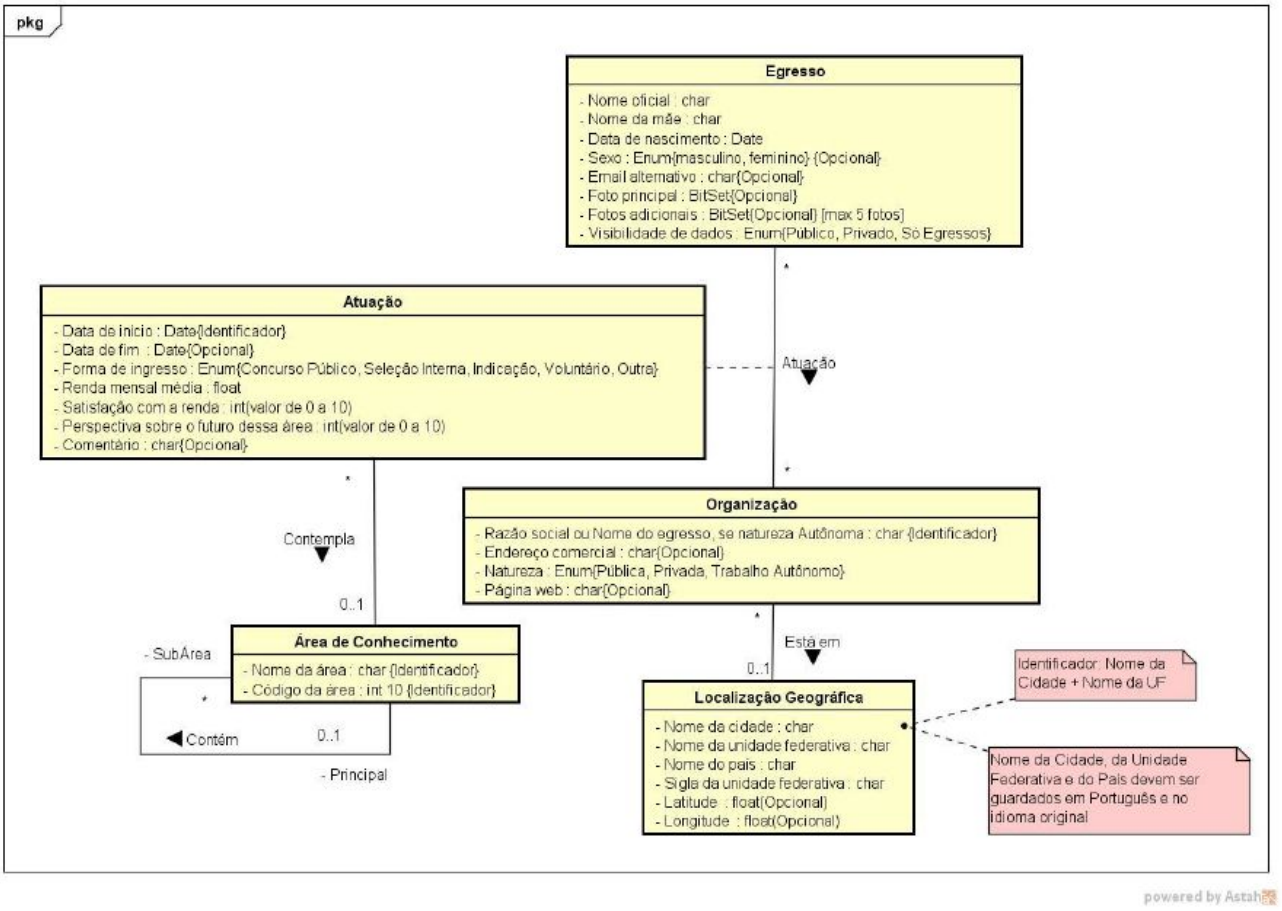

Figura 12 - Sétima parte do MER

Os modelos representam as diversas visões de dados especificadas no documento de requisitos da SempreUFG, são elas: Visão do egresso na ufg; Visão do egresso em outras instituições de ensino; Visão da atuação profissional do egresso; Visão da definição de cursos superiores; Visão da divulgação de informações; Visão de metadados do egresso; e Visão da gestão do sistema.

Embasamento

Essa visão foi escolhida por representar a forma como o sistema deve organizar seus dados e metadados, atendendo os requisitos especificados. É necessário para que as informações que o software precisa armazenar e manter atendam as expectativas e necessidades das partes interessadas.

2.5.2 Visão de Dados

Resumo da visão

A visão lógica do sistema do SempreUFG é representado por diagramas de classe que referem-se aos requisitos funcionais e não funcionais e a forma como devem ser implementados.

Modelo

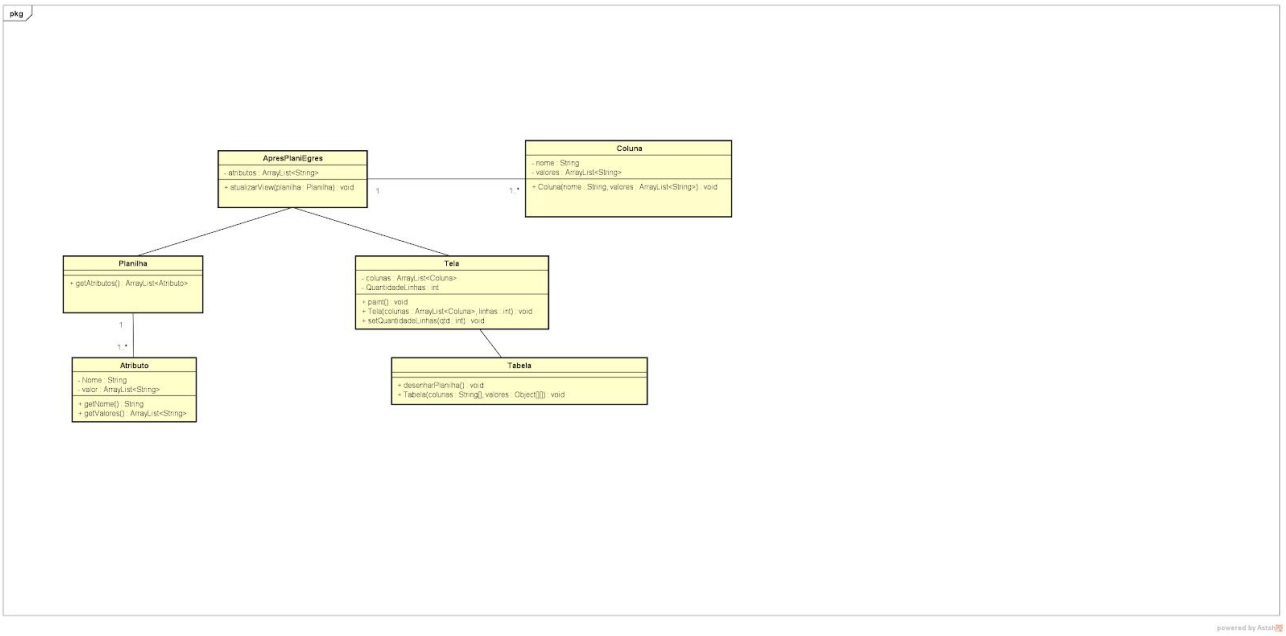

Figura 13 - Primeira parte do DC

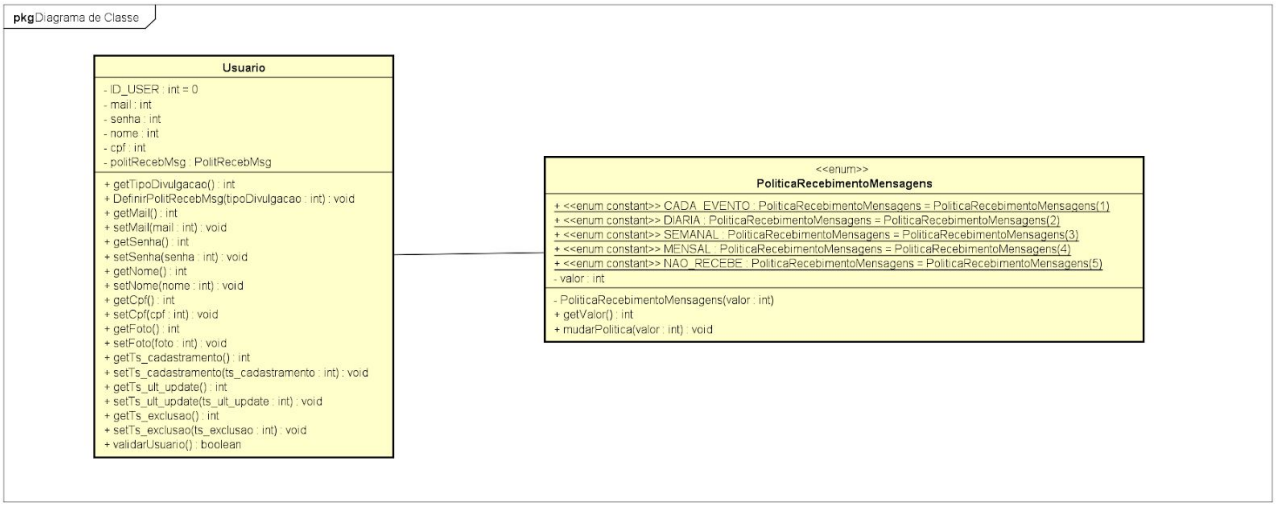

Figura 14 - Segunda parte do DC

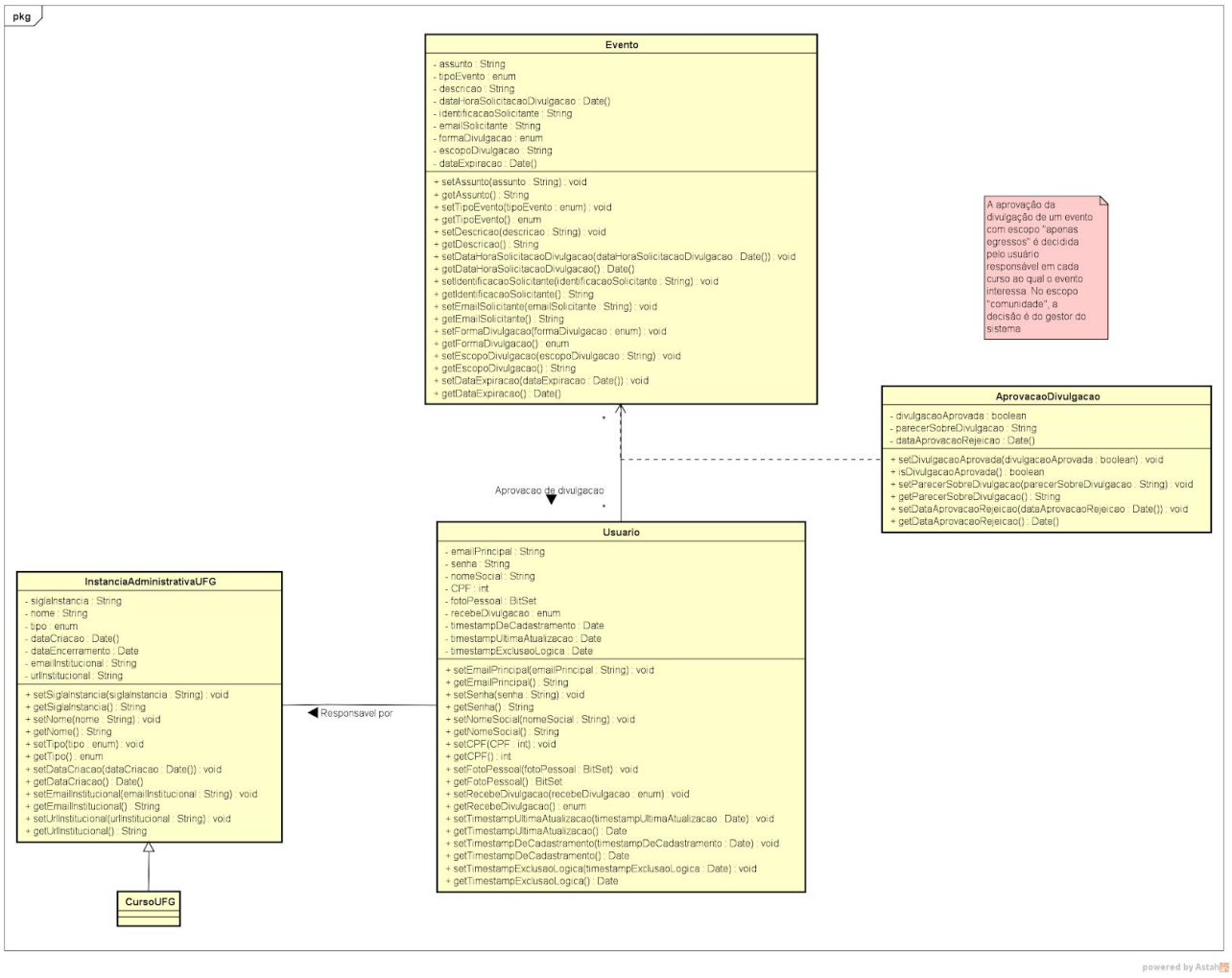

Figura 15 - Terceira parte do DC

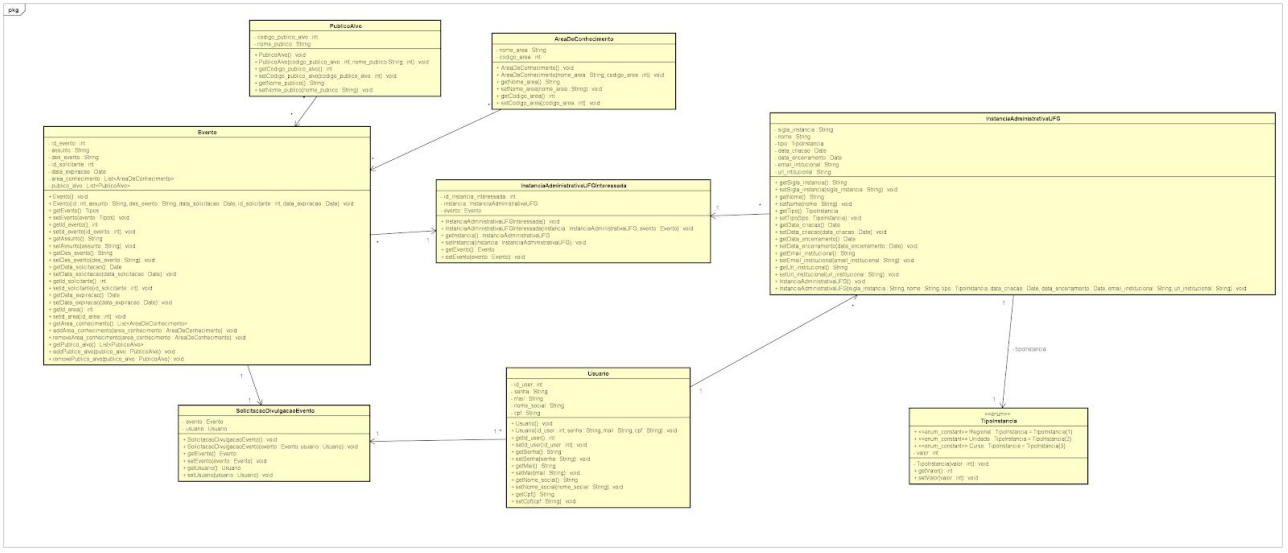

Figura 16 - Quarta parte do DC

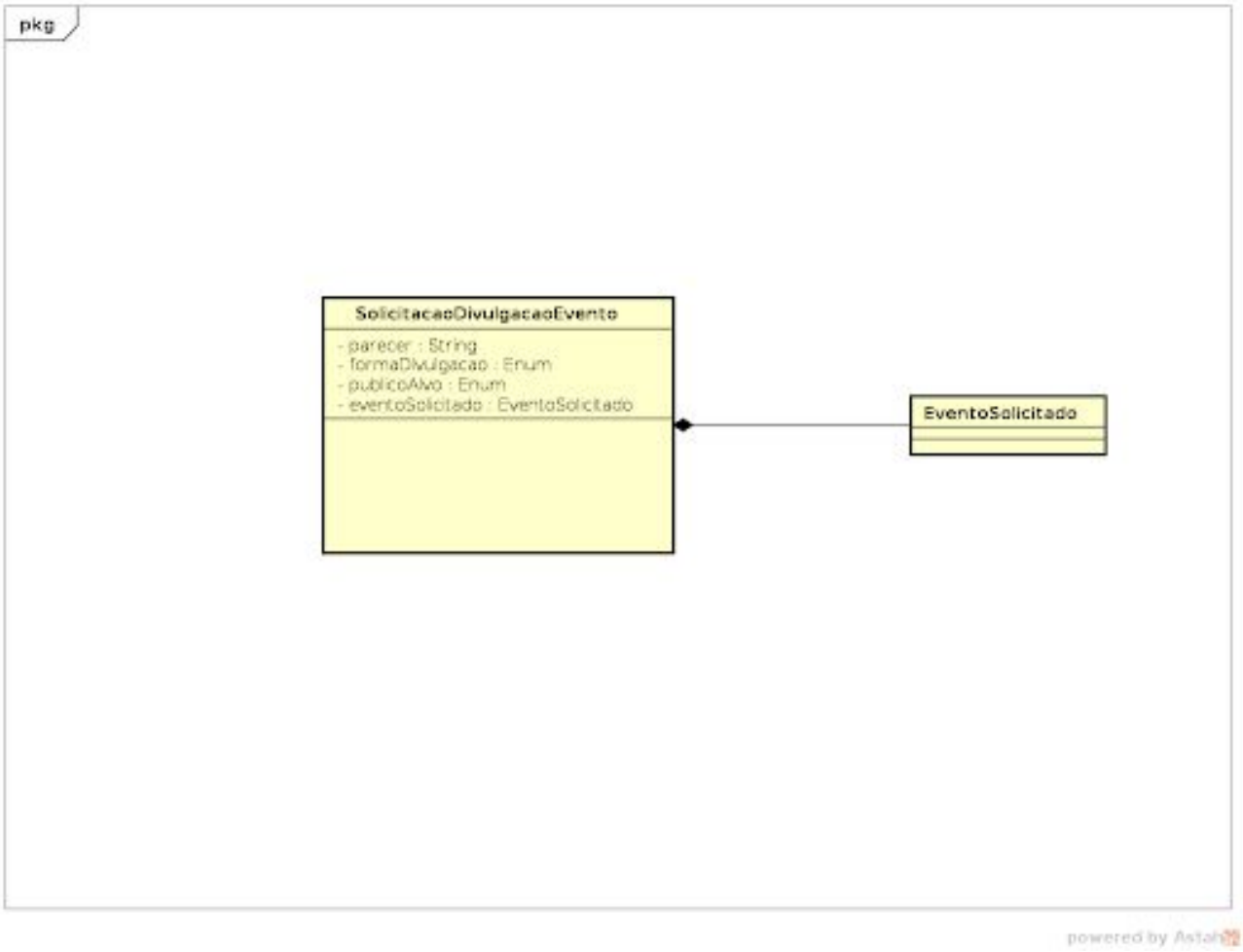

Figura 17 - Quinta parte do DC

Figura 18 - Sexta parte do DC

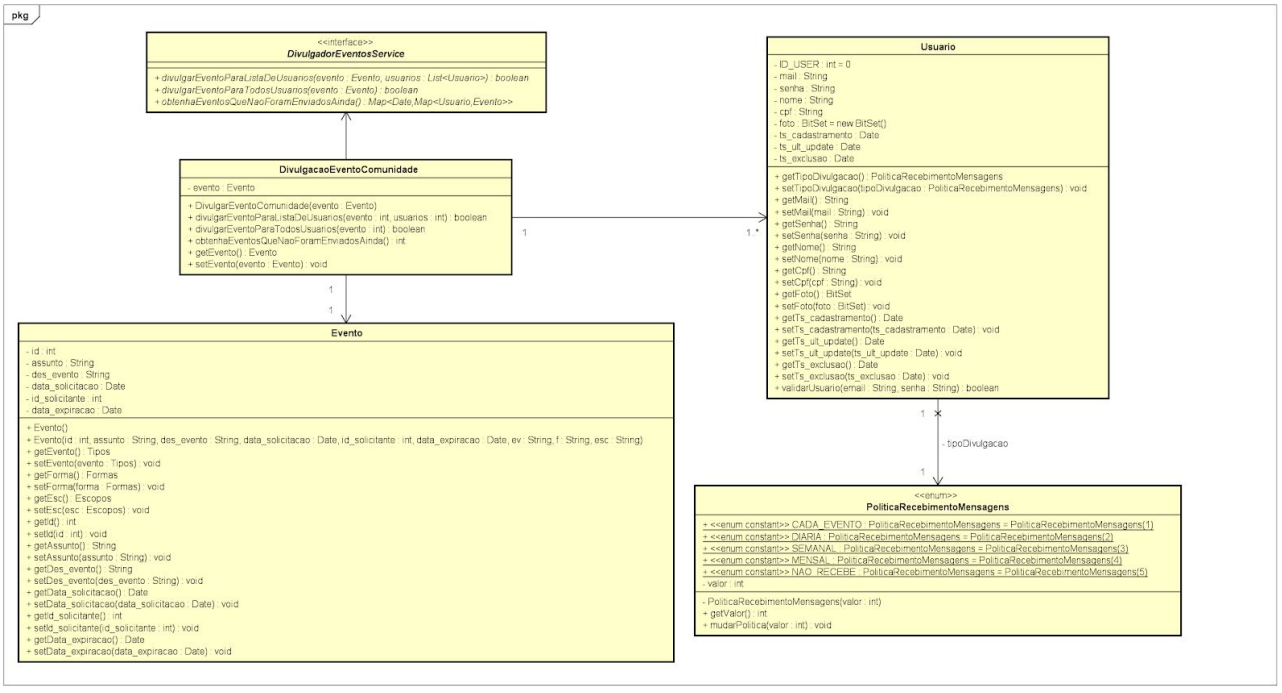

Figura 19 - Sétima parte do DC

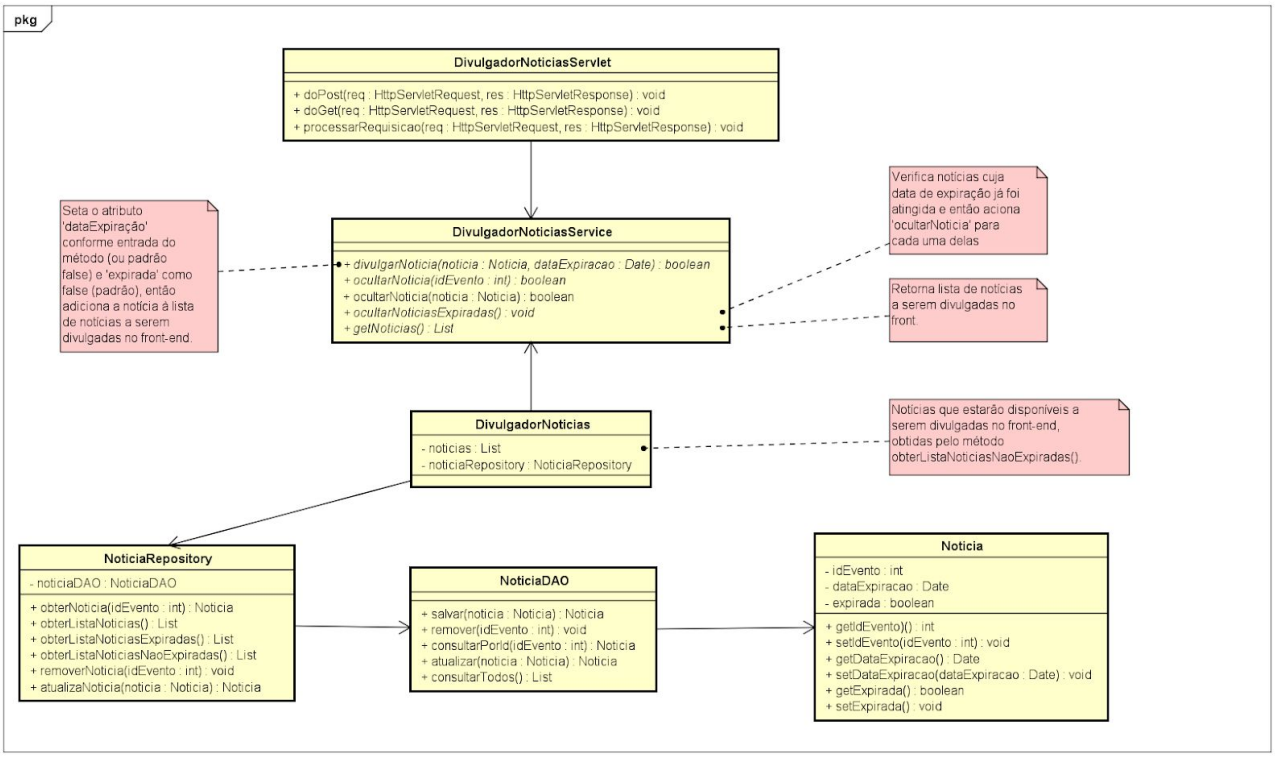

Figura 20 - Oitava parte do DC

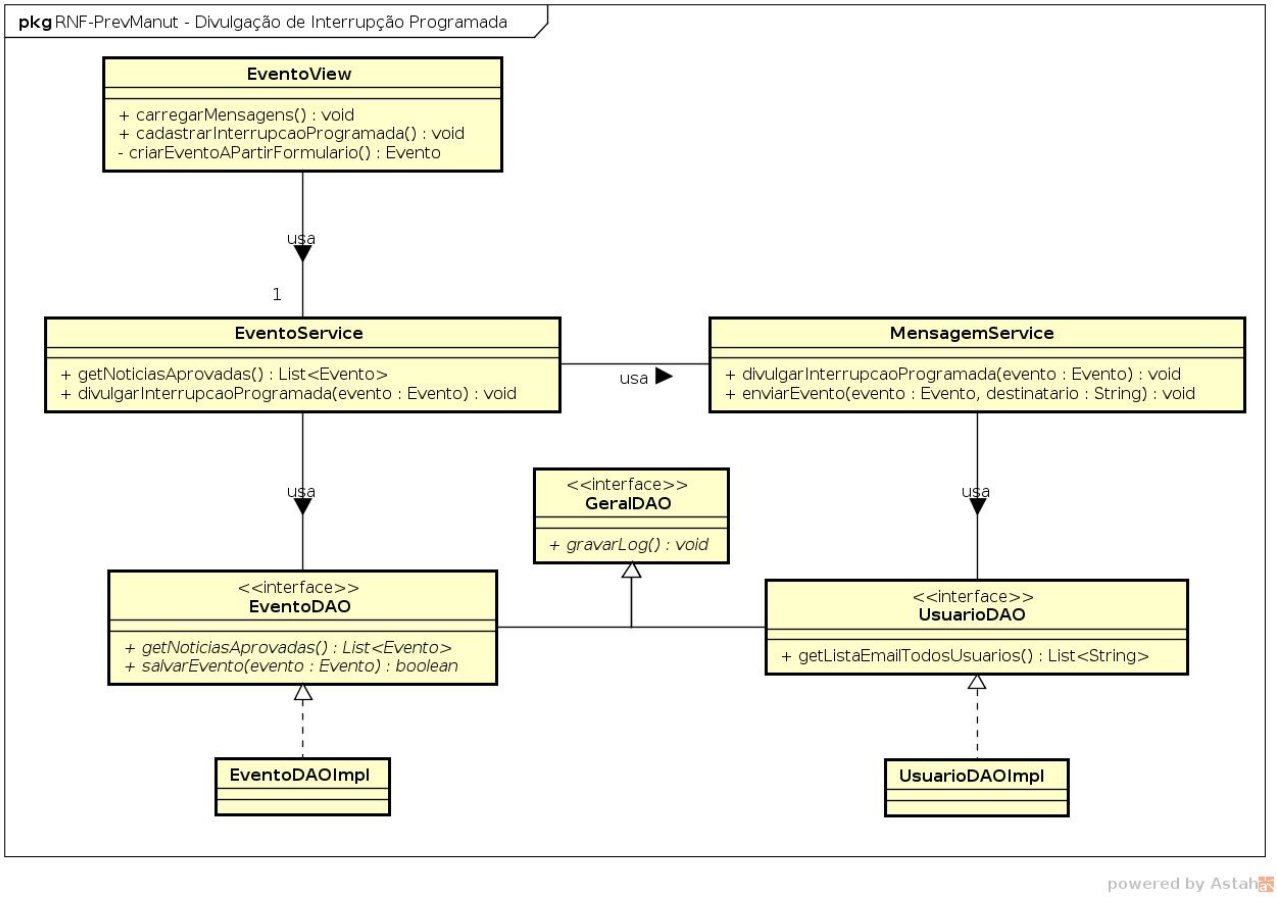

Figura 21 - Nona parte do DC

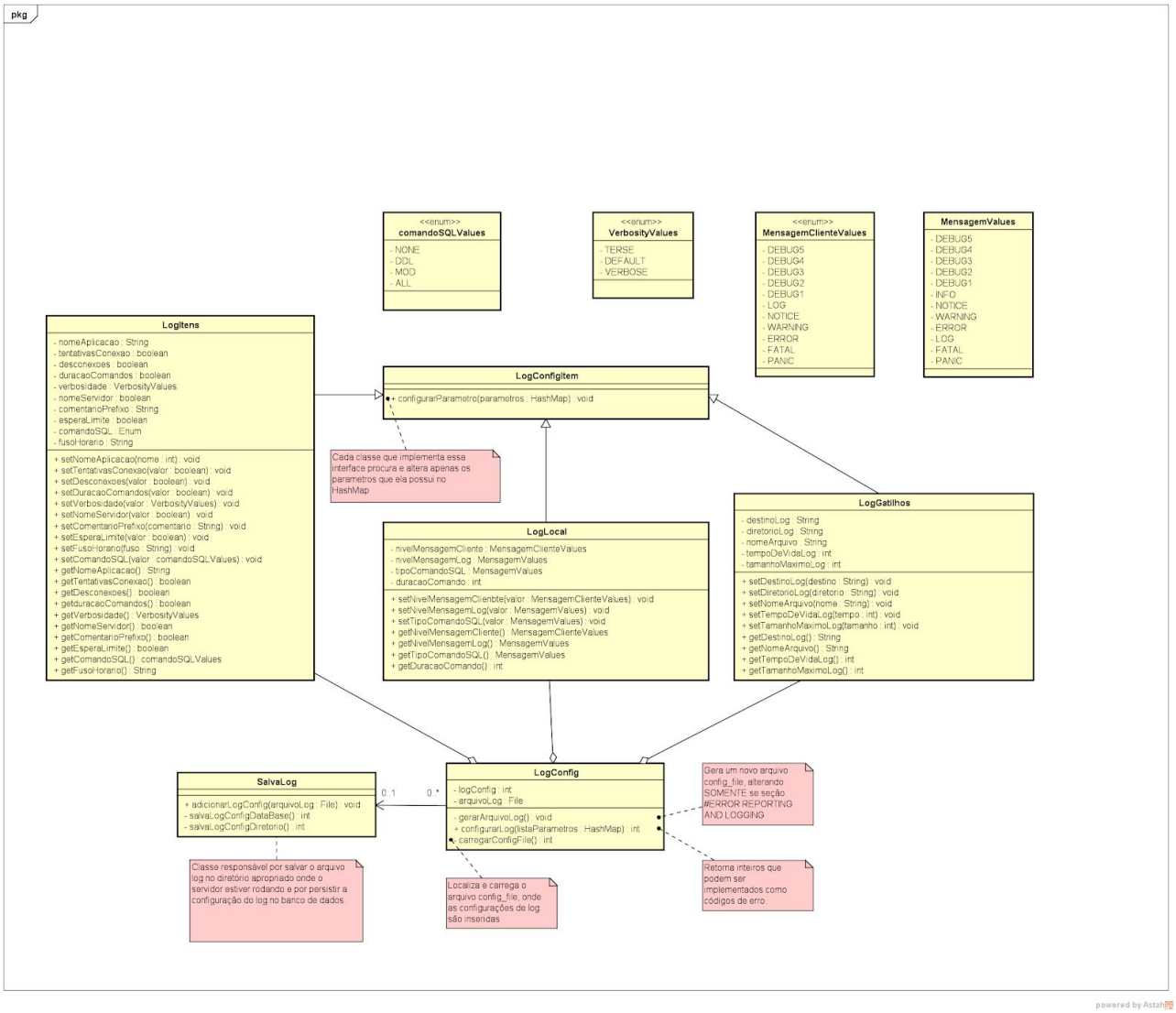

Figura 22 - Décima parte do DC

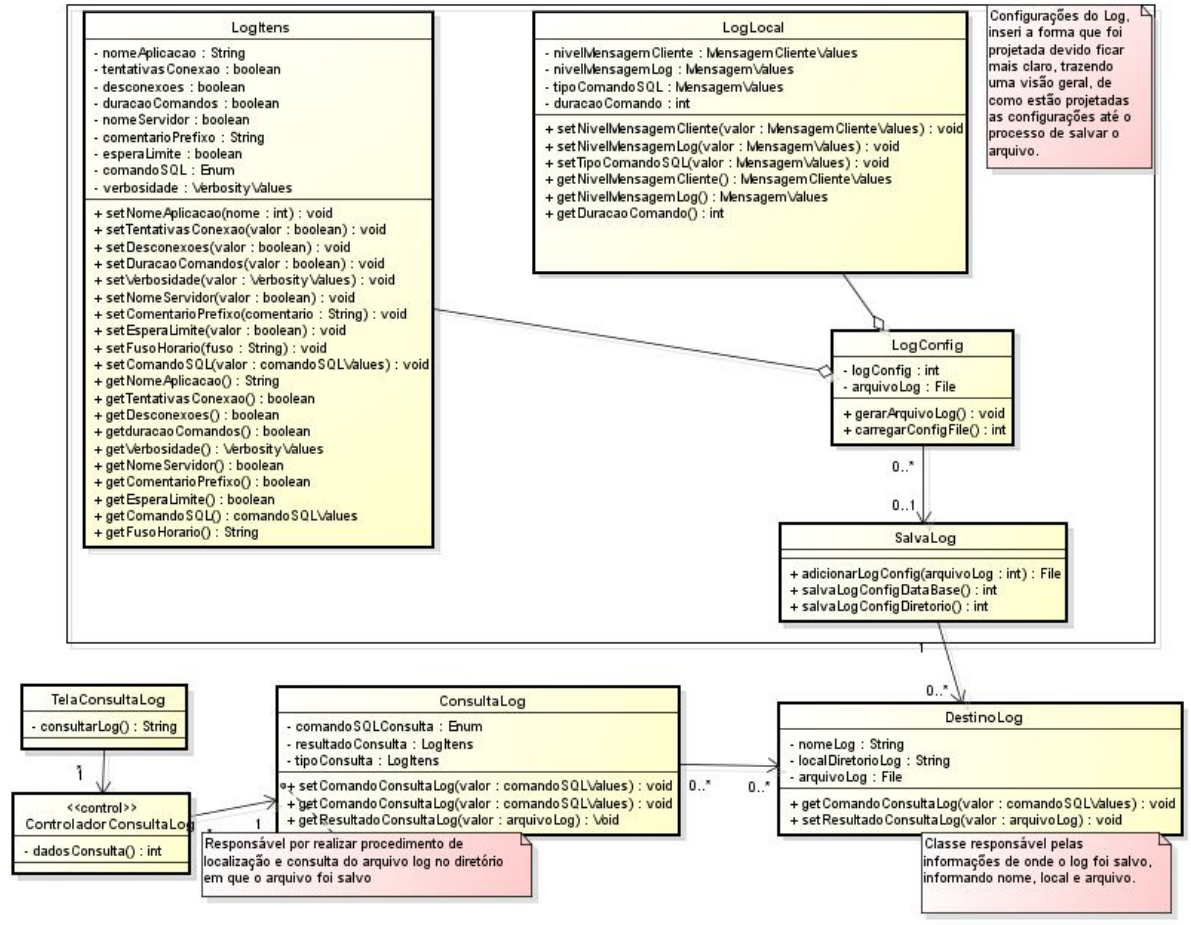

Figura 23 - Décima primeira parte do DC

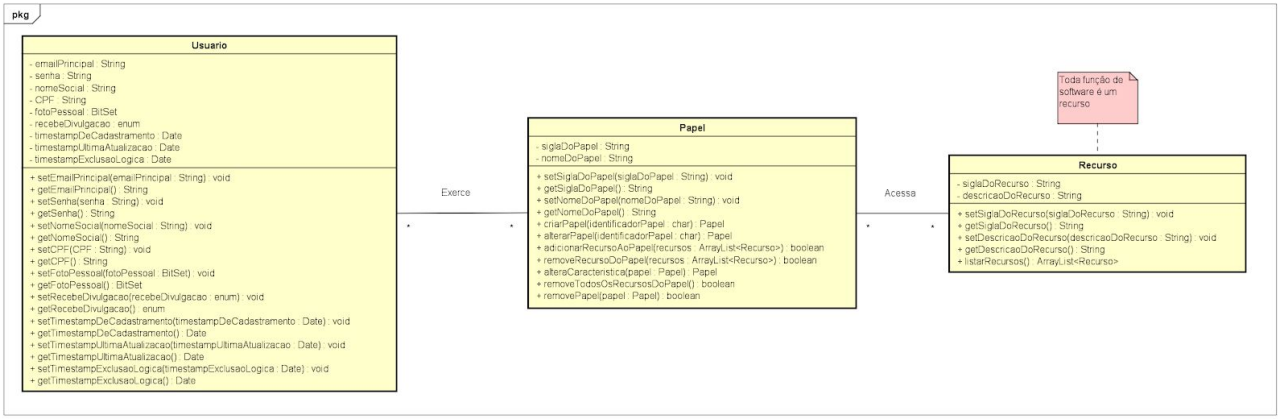

Figura 24 - Décima segunda parte do DC

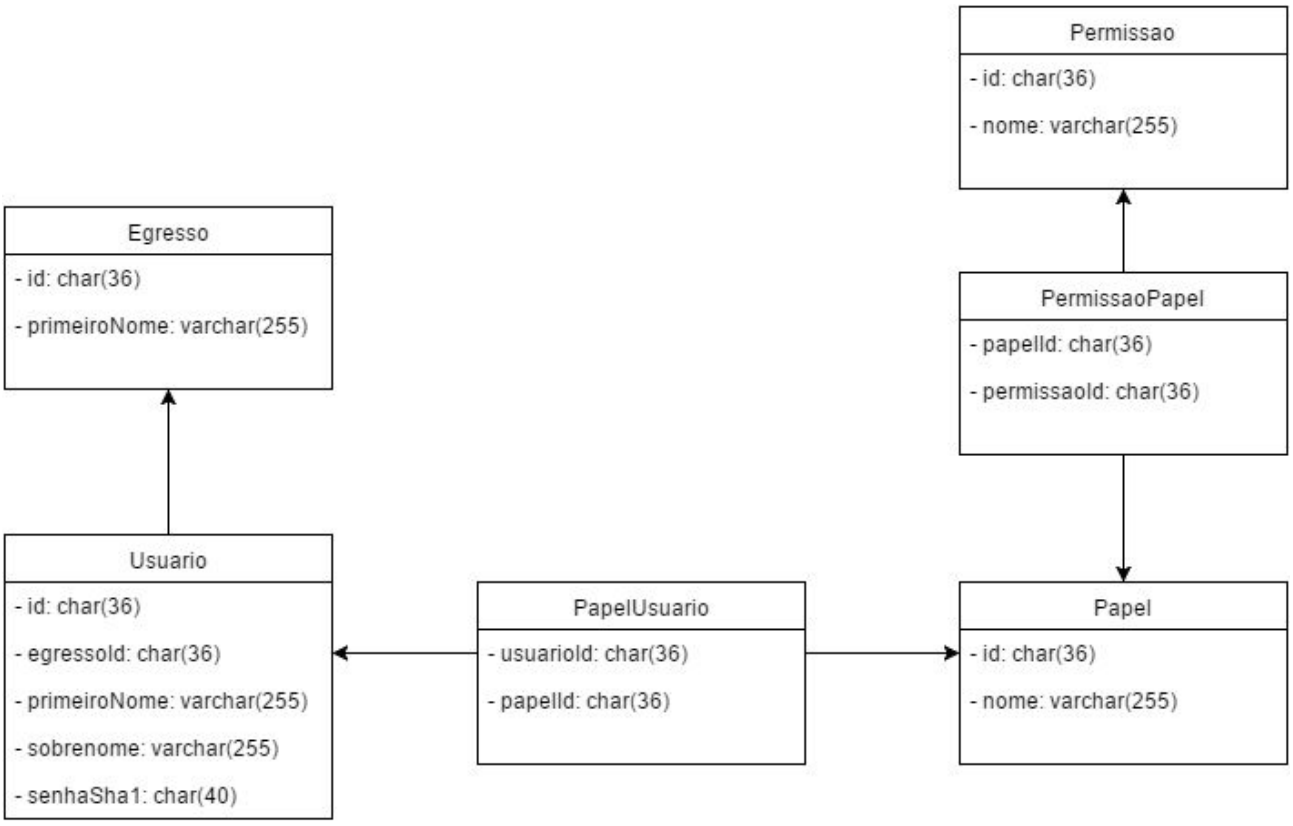

Figura 25 - Décima terceira parte do DC

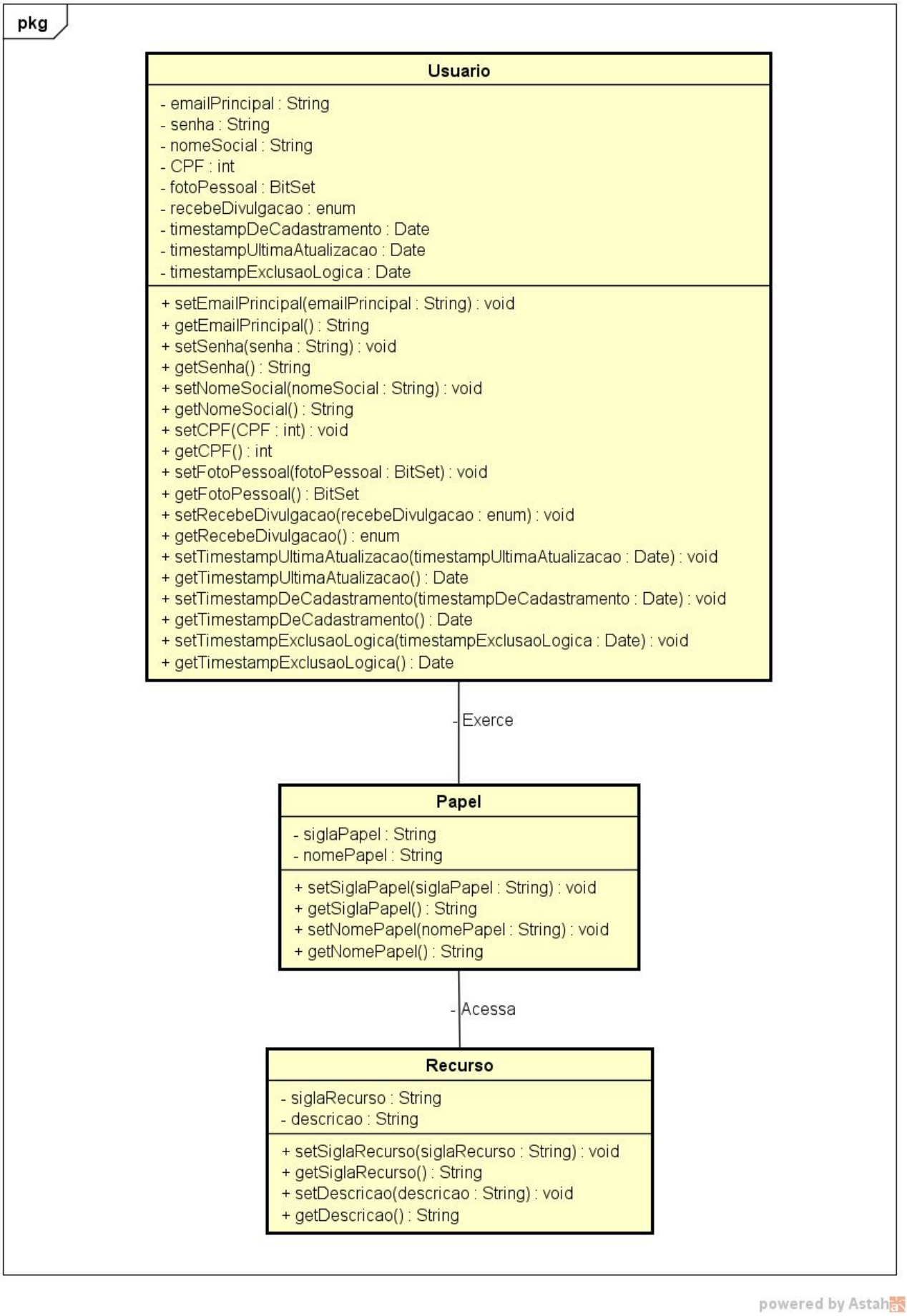

Figura 26 - Décima quarta parte do DC

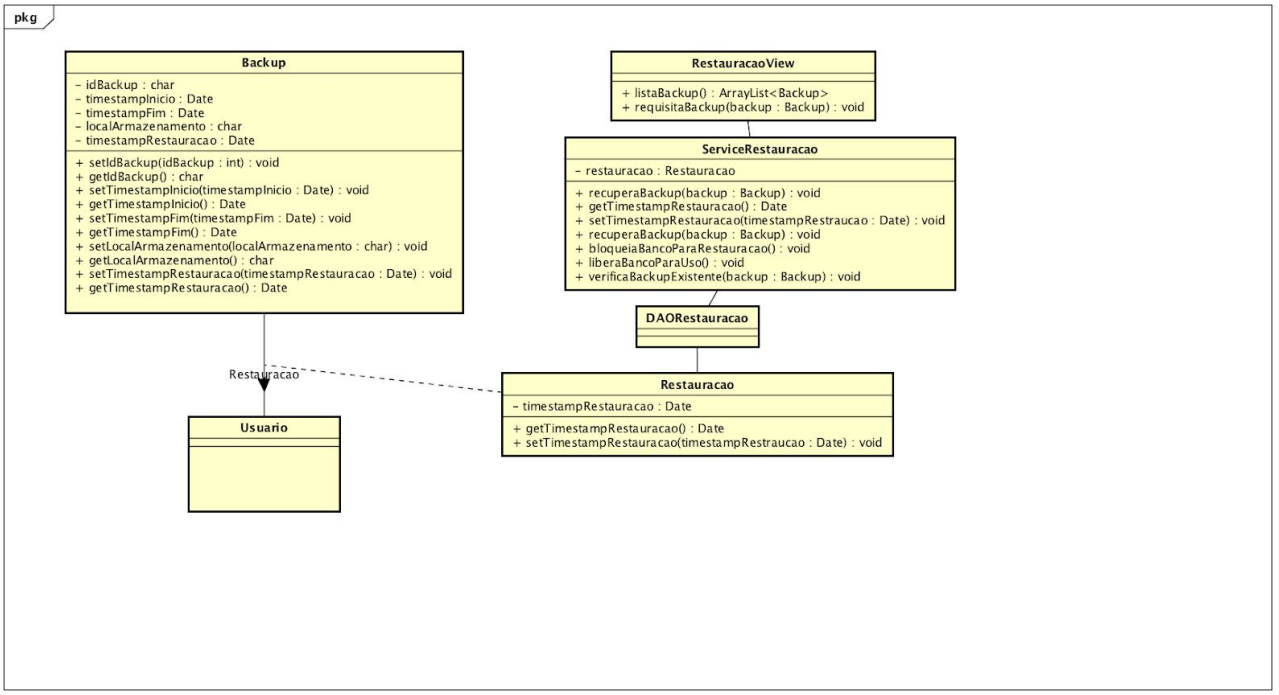

Figura 27 - Décima quinta parte do DC

Os diagramas representam os seguintes requisitos funcionais e não-funcionais:

- RF-ApresPlaniEgres;
- RF-AprovDivulgEvent;
- RF-AvalSolicDivulgEvent;
- RF-DivulgEventComun;
- RF-DivulgNotic;
- RF-PolitRecebMsg;
- RF-RespAprovDivulgEvent;
- RF-SolicDivulgEvent;
- RNF-ConfigLog;
- RNF-ConsulLog;
- RNF-ContrAcesPapel;
- RNF-MantPapel;
- RNF-MantUsu;
- RNF-PrevManut;
- RNF-RestauBD.

Os requisitos funcionais focados dão importância às consultas e notificações providas pelo software, e a forma como devem ser organizadas logicamente e solucionadas está definida nos modelos.

Os requisitos não-funcionais contemplados pelos modelos focam no quesito de segurança,que deve utilizar uma API de terceiro para prover esse serviço, não contemplado aqui, porém, são definidos as partes de responsabilidade do sistema.

Embasamento

Essa visão foi escolhida por representar a forma como o sistema deve ser efetivamente desenvolvido, definindo a solução para os requisitos especificados.

2.5.3 Visão Física

Resumo da visão

O Diagrama UML de Implantação fornece uma visão estática do aparato necessário, de *software* e *hardware* subjacente, para permitir que a arquitetura seja funcional.

Modelo

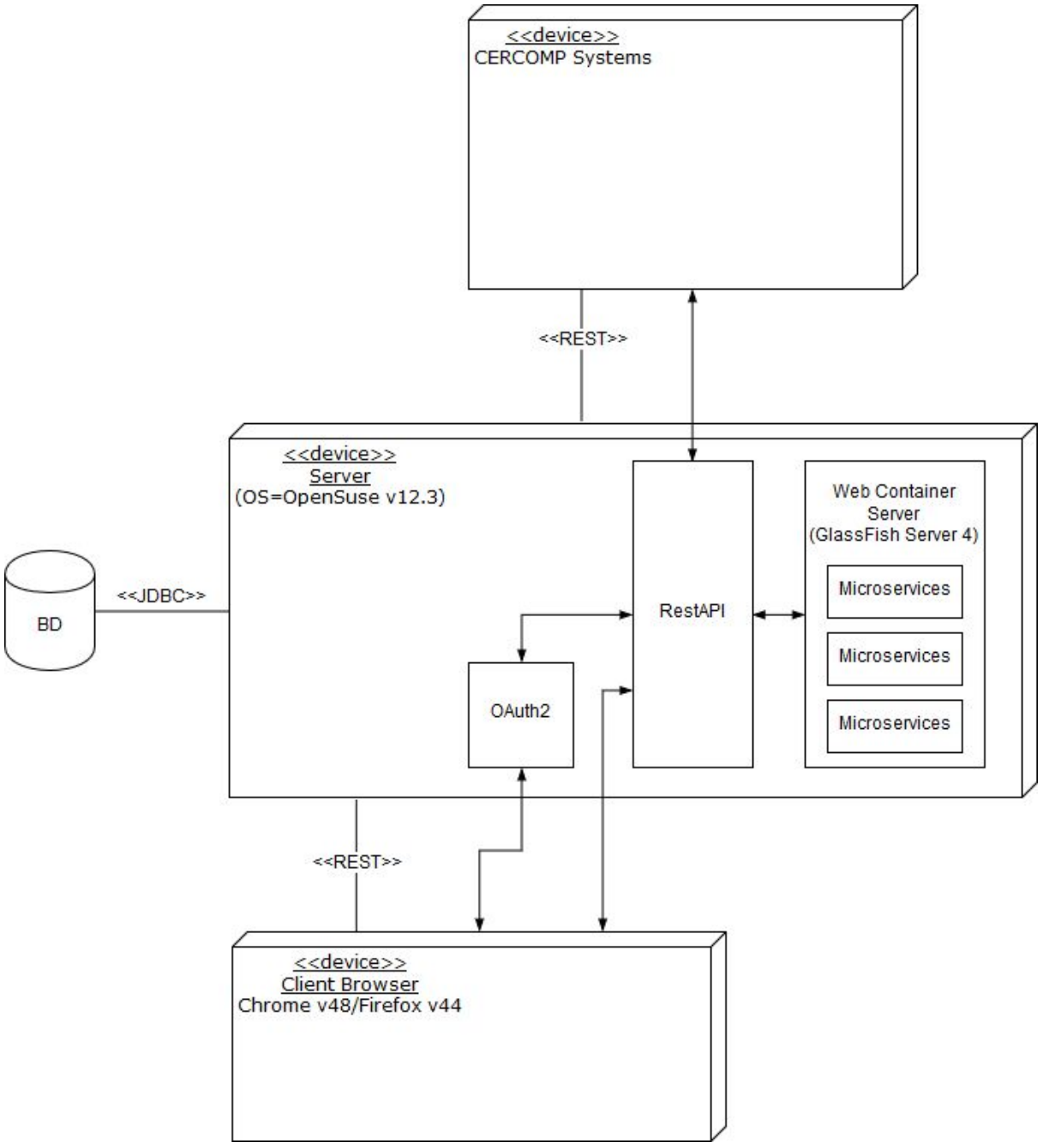

Figura 28 - Diagrama de Implantação

Elementos do modelo

- ***Client***

O *Client* (cliente) é o componente que representa o dispositivo do usuário que utiliza a aplicação. Esse dispositivo, como padrão, é definido como um computador com um navegador (*browser*) instalado e com acesso à Internet. O navegadores recomendados são: Google Chrome versão 48, ou Mozilla Firefox versão 44.

- ***Server***

O *Server* (servidor) é o dispositivo que representa o núcleo da aplicação. Nele estão os componentes responsáveis pela autenticação, microsserviços e recursos físicos (processamento, armazenamento, etc.).
O *OAuth2* é responsável por prover a autenticação de usuários e controlar o acesso devido à aplicação.
A *RestAPI* é responsável por todas as requisições de acesso ao servidor.
O *Web Container Server* executará e controlará os microsserviços que proverão acesso à aplicação.

- ***Data Base* (*DB*)**

A *Data Base* (*DB*) é o componente que armazena todas os dados e informações registrados e produzidos pela aplicação.

- ***CERCOMP Systems***

O componente *CERCOMP Systems* é integrado ao *Server* para realizar a importação de dados de egressos.

Embasamento

A Visão Física foi escolhida para fornecer apoio ao processo de Implantação e Instalação. Esta visão apoiará o Gestor do Sistema e a equipe de desenvolvimento nas atividades relacionadas à disponibilização do *software* aos usuários, quando o mesmo estiver pronto.

2.5.4 Visão de Desenvolvimento

Resumo da visão

O diagrama abaixo é organizado em camadas e micro serviços. Cada camada tem uma responsabilidade bem definida sendo que na camada de server temos subcamadas de serviços, onde estão os micro serviços e banco de dados.
Cada micro serviço provê uma função bem definida do SempreUFG, de forma que cada um desses serviços trabalham de forma independente se comunicando através Api's REST.
O objetivo do diagrama então é mostrar como as camadas e micro serviços irão se comunicar entre si.

Modelo

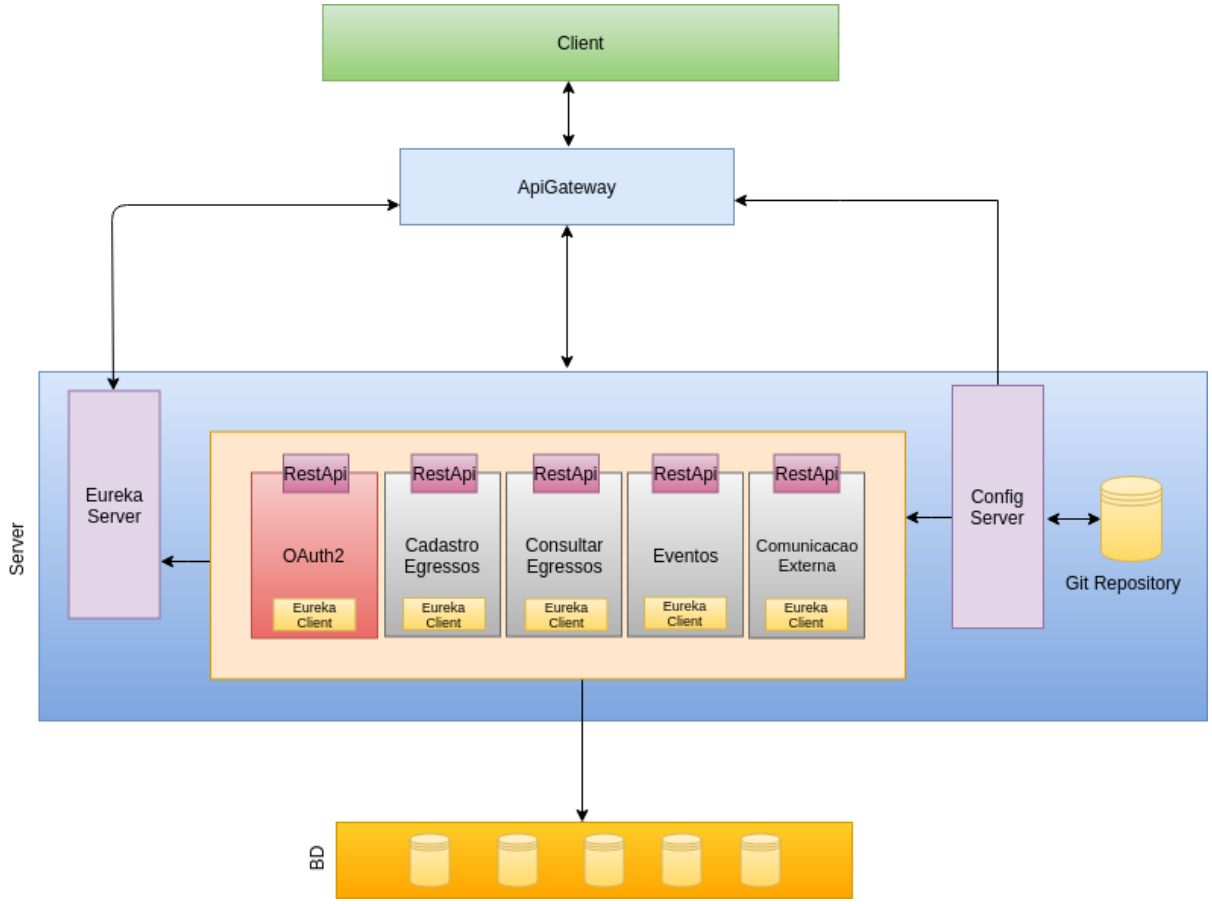

Figura 29 - Diagrama de Desenvolvimento

Elementos do modelo

- ***Cliente***

A camada Cliente é responsável pela interface com o usuário. Ela que faz o intermédio entre o sistema e o cliente.

A responsabilidade dessa camada é unicamente fazer a apresentação do conteúdo ao usuário que rodará a aplicação através de um browser.

- ***API GateWay***

A ApiGateWay fará o papel de orquestrador das apis. Cada micro serviço terá sua própria api REST pela qual o client irá se conectar, contudo deixar com que o client gerencie várias apis rests é contra produtivo, por isso a api gateway é importante para concentrar em um único ponto as apis, fazendo com que o client se preocupe apenas com se conectar à api gateway, deixando para ela fazer o gerenciamento das requisições.

- ***API REST***

API é uma interface de programação de aplicações, ou seja, é um conjunto de rotinas e padrões documentados por uma aplicação para que outras aplicações consigam utilizar funcionalidades desta aplicação sem precisar conhecer os detalhes da implementação do software.

Dessa forma o cliente pode ter acesso ao server através da API de forma independente, ou seja, sem conhecer em detalhes sua implementação. Essa camada então faz uma ponte entre os componentes do sistema.
O REST consiste em princípios que ajudam na criação de interfaces bem definidas para que as aplicações se comuniquem. Dessa forma o REST é apenas o conjunto de princípios e regras que utilizamos para construir nossas API's.
Para que a aplicação faça uso do REST de forma plena ele precisa usar o protocolo HTTP para se comunicar, não possuir estado entre as comunicações e deve ser clara a definição do que faz parte do cliente e o que faz parte do servidor.

- ***Server***

Nessa camada é onde estarão os serviços, os quais se comunicarão entre si através de Apis Restfull. Haverá também um serviço específico que fará a comunicação com os sistemas externos do CERCOMP.
Cada micro serviço é independente e funciona separadamente, assim se um falhar o outro continuará funcionando.
Os micro serviços contidos nesta camada também são responsáveis por fazer a intermediação entre a camada de apresentação e o banco de dados e é nessa camada que as regras de serviço são implementadas.
O micro serviço Eureka, é na verdade um Service Registry , ou seja ela faz o gerenciamento de todos os outros micro serviços, é usado para localizar serviços com o objetivo de balanceamento de carga e failover de servidores.
O micro serviço de autenticação oauth2 é responsável pela segurança dos dados. Antes de qualquer requisição aos outros micro serviços é preciso conseguir um token desse serviço.
O micro serviço de configuração é responsável pela centralização de todos os arquivos de configuração dos demais micro serviços. Ele também está contido na camada de serviço.

- ***BD***

Essa camada é responsável pela persistência de dados, pode haver mais de um banco de dados, pois eles serão usados por micro serviços independentes.

Embasamento

Essa visão foi escolhida para atender ao ponto de vista do projetista e é importante pois ela dá uma visão geral do sistema, dos módulos, camadas e como eles se relacionam entre si.
Com essa visão fica mais claro como o software será construído, qual responsabilidade de cada parte dele e assim facilitando a implementação de cada uma dessas partes já respeitando essas responsabilidades bem definidas.

2.5.5 Visão de Segurança

Resumo da visão

O diagrama abaixo oferece detalhes sobre como é feita a segurança de dados da aplicação. Apresenta como é feita a autenticação do usuário e o acesso mediante a posse do token aos dados sensíveis presentes nos bancos de dados.

Modelo

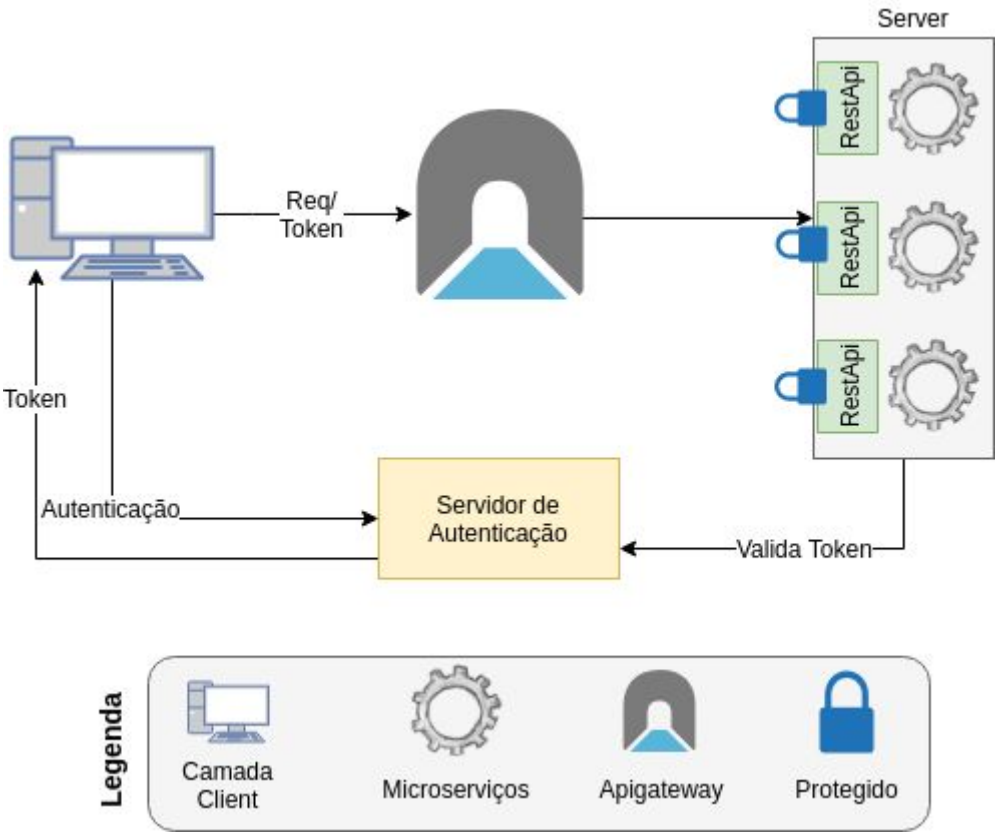

Figura 30 - Diagrama de Segurança

Elementos do modelo

- ***Cliente***

É o usuário que está tentando acessar os dados da aplicação. Como o sistema é web a autenticação é requerida através de um browser. Se estiver tudo certo com a autenticação o usuário consegue um token com o qual consegue acesso aos dados.

- ***Servidor de Acesso***

Servidor responsável por guardar os dados de autenticação do usuário de forma criptografada. Ao receber um pedido de autenticação ele verifica a veracidade dos dados e fornece um token de acesso ao usuário, este token dá acesso às funcionalidades do sistema de acordo com nível de privilégio do usuário.

- ***Apigateway***

Responsável pela orquestração dos microservices. O client não precisa gerir todas as apis dos microservices, ele irá interagir apenas com a apigateway e ela se responsabilizará em enviar as requisições para o destino correto já com o token informado.

- ***Rest Api***

Todas as requisições feitas com a api REST necessita em seu cabeçalho do token de acesso para que o servidor possa validar essas requisições. Dessa forma ao receber o token no cabeçalho a primeira

coisa a se verificar no servidor de acesso é se este token é válido. Caso seja o processo de requisição segue normalmente. Caso o token seja inválido ou tenha inspirado é retornado uma exceção de autorização e assim será necessário autenticar-se novamente.

- ***Micro Serviços***

Os micro serviços só serão acessados caso o token seja válido, para isso é imprescindível que seja feita a validação do mesmo com o server de autenticação. Uma vez validado o serviço segue seu fluxo.

Embasamento

Essa visão foi escolhida para atender ao ponto de vista de segurança e é importante, pois ela dá detalhes de como é feita a autenticação do usuário e mostra como é feita a segurança dos dados presentes no servidor.

2.5.6 Visão de Front-end

Resumo da visão

O diagrama abaixo representa a camada cliente de nossa arquitetura. Essa camada utiliza React para construção dos componentes de tela e Redux, que contribui para o gerenciamento de estado na aplicação.

Modelo

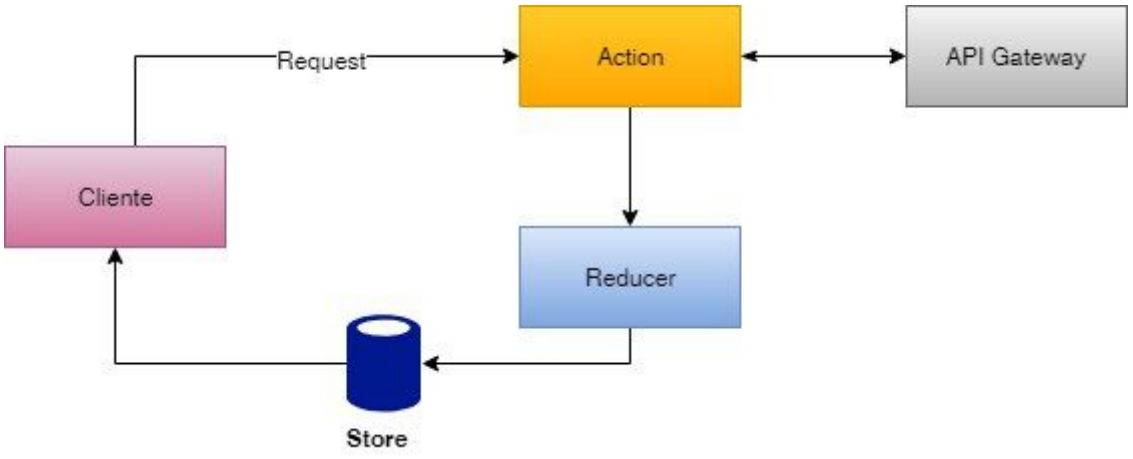

Figura 31 - Diagrama Front-end

Elementos do modelo

- ***Cliente***

Pode representar o browser que renderiza os componentes feitos em React. É a parte com a qual o usuário interage. É partindo dela que os pedidos de requisição são feitos.

- ***Action***

A action é a única maneira de mudar os dados da aplicação. Toda vez que o usuário precisar mudar o estado da aplicação uma ação é disparada. Essa ação precisa especificar o tipo e os parâmetros necessários para a mudança. Ela que se conecta com a Apigateway para pegar os novos dados necessários para mudar o estado da aplicação.

- ***Reducer***

No reducer é feito o tratamento da mudança que foi disparada pela action. O reducer cria um novo estado de acordo com a ação disparada. Toda mudança feita no estado da aplicação é centralizada no reducer. Ele cria um novo estado, não muda o estado anterior, pois são funções puras. Isso evita efeitos colaterais e torna a aplicação mais previsível e menos suscetível a erros.

- ***Store***

A store é o único lugar onde o estado da aplicação está armazenado. É basicamente um objeto com todos os dados da aplicação. A store ao ser criada precisa de um reducer para poder fazer as alterações do estado da aplicação.

Embasamento

Essa visão foi escolhida para que fique claro como funcionará o fluxo de dados na camada cliente.Por ser uma tecnologia não muito comum de ser utilizada fez-se necessário especificá-la mais para os stakeholders.

2.5.7 Visão do Usuário

Resumo da visão

A Visão de Requisitos foi baseada no Diagrama de Casos de Uso (DCU). A presente visão, gerada pelo Ponto de Vista do Usuário (usuário, cliente, egresso, gestor do sistema), tem como objetivo estabelecer um relacionamento entre os requisitos especificados no Documento de Requisitos do SempreUFG e sua ilustração materializada no DCU.

Modelo

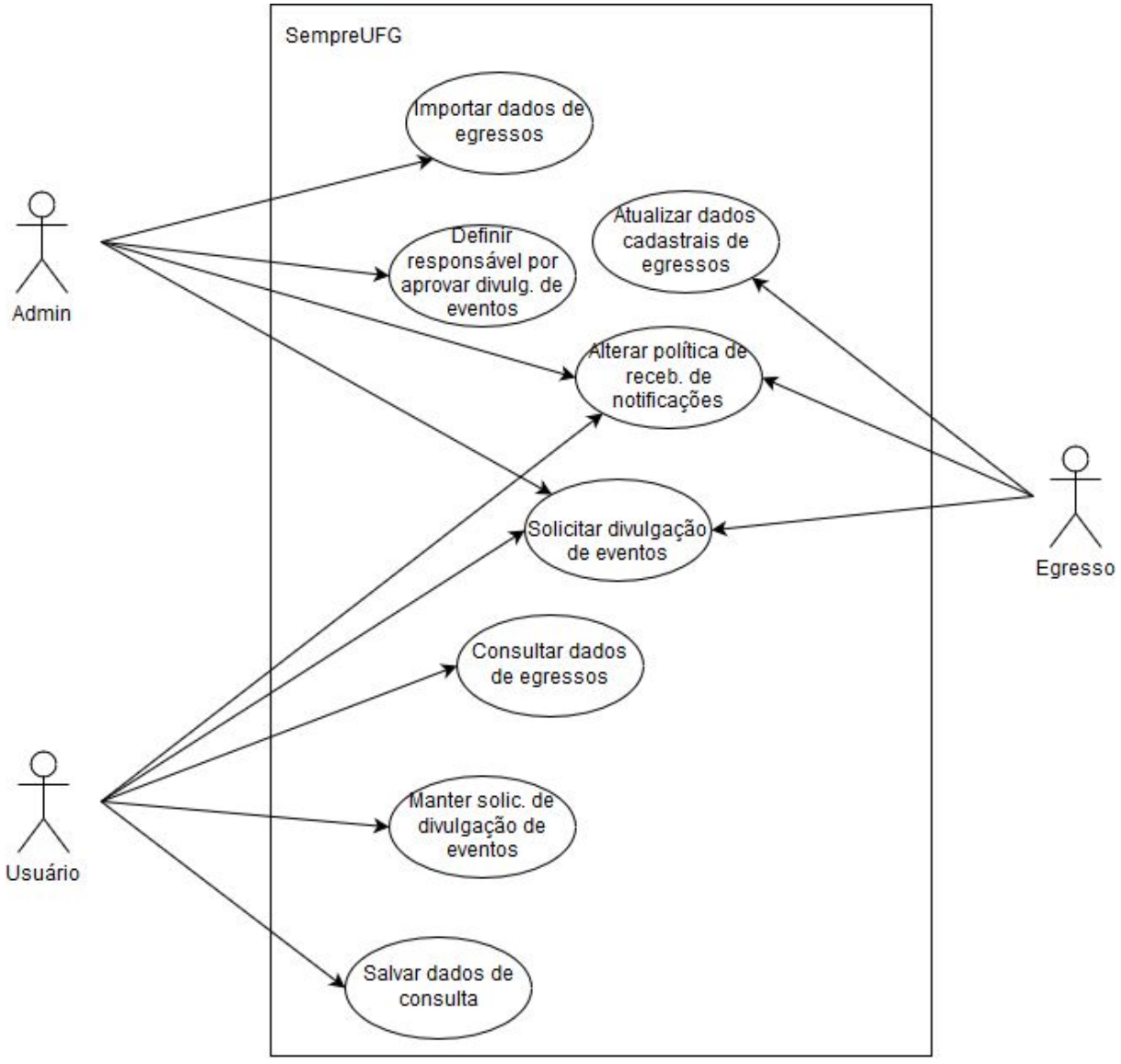

Figura 32 - Diagrama de Casos de Uso

Elementos do modelo

- ***Admin***

O ator Admin é a representação do Gestor do Sistema do SempreUFG. Ele poderá, por exemplo, importar dados de egressos a cada semestre, definir o/a responsável pela aprovação/reprovação de divulgação de eventos, etc.

- ***Usuário***

O ator Usuário é a representação dos interessados institucionais (UFG) no SempreUFG, como por exemplo, CAVI, PROGRAD e PRPG. Este ator poderá, por exemplo, consultar dados de egressos e salvar dados destas consultas.

- ***Egresso***

O ator Egresso é a representação dos alunos que entraram ou que já estavam no quadro de discentes da universidade. Ele poderá, por exemplo, atualizar seus próprios dados cadastrais do SempreUFG, alterar a política de recebimento de mensagens e notificações e solicitar a divulgação de eventos.

- ***Casos de uso***

Os casos de uso estão dispostos no interior do ambiente do SempreUFG e, no geral, são representações de funcionalidades em que os atores (que podem ser pessoas ou sistemas/subsistemas) poderão interagir diretamente e/ou indiretamente. São representados, no diagrama, por elipses.

Embasamento

A Visão de Requisitos foi escolhida para fornecer apoio ao processo de engenharia de requisitos e desenvolvimento, nas fases iniciais. É uma forma de prover ao cliente (stakeholders do projeto) uma ilustração de como as funcionalidades do programa poderão ser utilizadas. Além disso, auxilia no processo de validação dos requisitos.

2.5.8 Visão de Processos

Resumo da visão

O objetivo dessa visão é ilustrar os fluxos de atividades dos egressos, diretores e solicitantes de evento no sistema.

Modelo

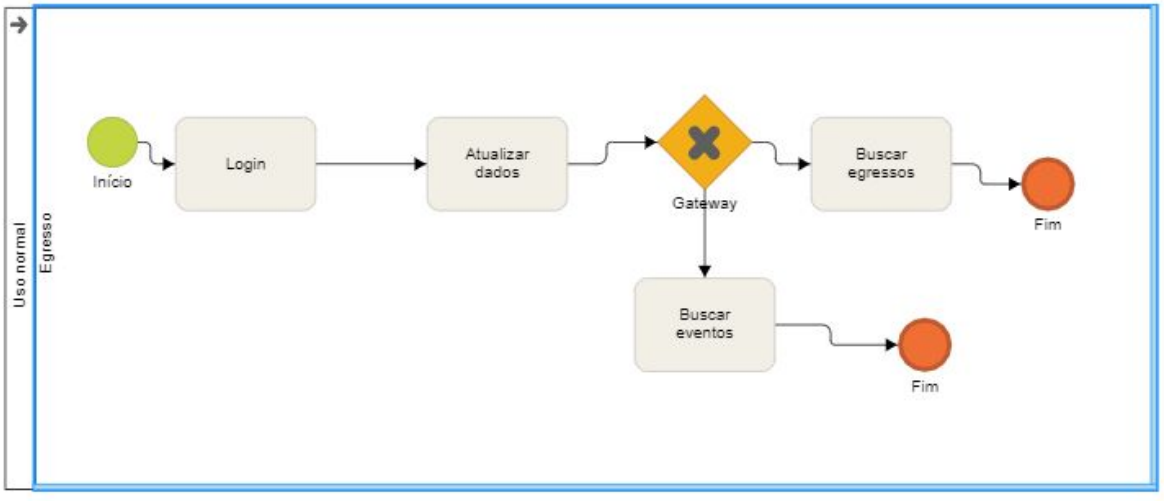

Figura 33 - Fluxo de atividades do usuário no sistema.

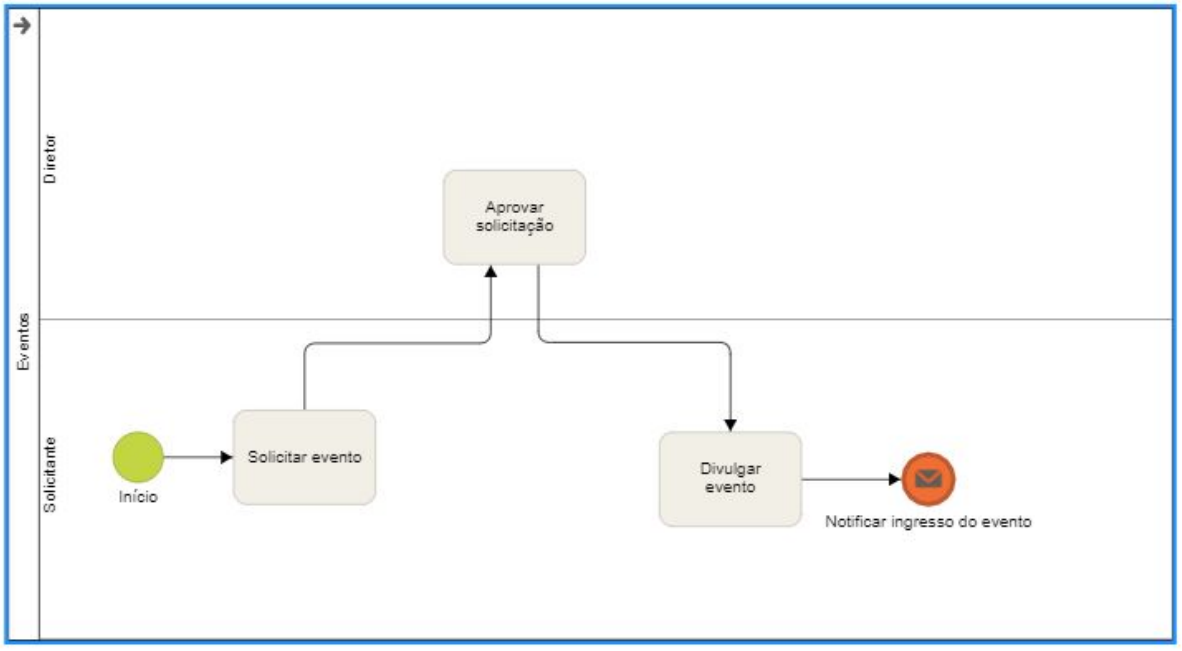

Figura 34 - Processo de solicitação e divulgação de eventos

Elementos do modelo

- ***Egresso***

O egresso pode logar no sistema, atualizar seus dados e após isso, realizar buscas por outros egressos ou eventos.

- ***Eventos***

Os eventos podem ser solicitados por membros da comunidade. Os eventos serão então aprovados pelos diretores e posteriormente divulgados pelo sistema e o solicitante.

Embasamento

Essa visão foi escolhida para atender ao ponto de vista do usuário final e é importante pois ela dá uma visão das atividades de cada ator.

Com essa visão fica mais claro para os usuários as atividades que podem executar assim como a sequência que devem seguir.

2.6 Decisões arquiteturais

[DSUFG01]

Descrição: A arquitetura do SempreUFG é divida em três camadas lógicas, cliente, servidor e banco de dados. A camada cliente se comunica com o servidor através de requisições apis rests e a camada servidor não conhece a camada cliente.

Objetivo: Essa divisão tem como objetivo trazer maior modularidade ao software, dividindo bem as responsabilidades e ajudando na manutenção.

Motivação: A maior motivação da divisão em camadas é justamente promover a coesão e o baixo acoplamento do sistema, o que facilitará e muito sua evolução e manutenção, visto que cada parte pode evoluir separadamente, bem como ser mantida de forma mais eficaz.

[DSUFG02]

Descrição: A camada servidor é composta por vários micro serviços, os quais representam alguma funcionalidade crítica do sempreUFG, os serviços se comunicam através de endpoints rest api.
Objetivo: A divisão em micro serviços tem como objetivo atender a um requisito do sempreUFG que diz que a indisponibilidade de uma funcionalidade não deve afetar a outra. Também ajuda na manutenção e na modularidade, por diminuir o acoplamento.
Motivação: Os micro serviços são importantes para promover a modularidade do sistema, contudo o que mais motivou o uso dos micros serviços foi o requisito que pede que as funcionalidades sejam independentes e uma das melhores formas de torná-las assim seria separando cada uma em serviços menores, que são mais fáceis de manter, melhora a coesão, diminui o acoplamento e ainda promove a disponibilidade do sistema.

[DSUFG03]

Descrição: A camada cliente foi feita utilizando react, que é uma biblioteca JavaScript que ajuda a manipular o estado da aplicação e melhora a performance renderizando apenas o que é necessário atualizar. Utiliza-se também redux, que é um framework que ajuda no gerenciamento de estado.
Objetivo: O objetivo de se utilizar essa tecnologia é diminuir a complexidade da parte client-side, fazendo com que a mudança no estado da página seja sempre unidirecional, totalmente controlado pelo próprio client-side, fazendo com que seja apenas necessário consumir as apis que vêm no servidor. O desenvolvimento se torna mais simples, pois é fácil de fazer reutilização de código e testar o software.
Motivação: A motivação de se utilizar react mais redux é dada pelo conhecimento da equipe de desenvolvimento dessa tecnologia, o que traria um maior desempenho para construir o software. Também haverá uma diminuição da complexidade, com com react e redux as informações seguem um fluxo previsível o que facilita a depuração. Outro benefício é o desempenho, pois o react faz uso do dom virtual e com ele é possível atualizar apenas o componente que sofreu alterações.

[DSUFG04]

Descrição: Foi utilizado o protocolo de autorização Oauth2 para fazer a autenticação dos usuários no sistema. É um protocolo bem difundido usado para poder autorizar autenticações apis, garantindo assim a segurança do server. Basicamente o usuário faz o login, consegue um token de acesso caso esteja tudo certo e com esse token ele tem acesso às informações do servidor.
Objetivo: O objetivo ao se utilizar esse protocolo é promover a segurança de informações de nossa aplicação, evitando que qualquer um tenha acesso aos dados que são exclusivos para àqueles cadastrados no sistema.
Motivação: O principal motivo que levou à escolha desse protocolo é o fato dele ser bastante difundido e ter já comprovada sua eficácia quanto à segurança de dados sensíveis dos usuários. Como os requisitos exigiram que todas as transações no software fosse segura esse protocolo se mostrou uma ótima opção para poder atingir esse fim.

[DSUFG05]

Descrição: Os micro serviços irão expor seus serviços através de apis restfull. Api nada mais é do que um conjunto de padrões e rotinas usadas para se acessar determinado serviço. Rest é um conjunto de princípios de arquitetura usado para fazer a conexão entre serviços e clientes. Utiliza para tal conexão o protocolo HTTP e os dados são trafegados utilizando JSON ou XML. Sendo JSON preferível, por ser mais leve. Cada comunicação REST é independente, ou seja, nela é preciso ter toda informação necessária.

Objetivo: O objetivo de se utilizar as apis REST é possibilitar a conexão entre o cliente e os micro serviços de forma simples e leve.

Motivação: Uma vez que os serviços e o cliente são construídos em tecnologias diferentes é preciso que haja um meio eficiente de comunicação entre essas camadas. O REST possibilita então que as rotinas dos serviços sejam expostas sem necessariamente expor detalhes da implementação do serviço, com isso é possível que haja essa conexão entre a camada cliente e serviço, mesmo que tenha sido construídos em tecnologias diferentes.

[DSUFG06]

Descrição: A orquestração dos micro serviços serão feitos pela ApiGateway. Por causa da quantidade de apis que são disponibilizadas pelos micro serviços o cliente ficaria com dificuldade de gerenciar todas essas requisições. Portanto as apis ficariam concentradas na api gateway que faria o papel de orquestrar todos os endpoints dos micro serviços. A api gateway também ajuda a diminuir o fluxo de dados, pois ela permite que o cliente recupere dados de vários micro serviços diferentes com uma única requisição.

Objetivo: O objetivo de se utilizar a api gateway é abstrair a complexidade de se comunicar com várias apis diferentes da camada cliente, deixando para a api gateway toda decisão de tratamento caso um serviço caia, ou fluxo de chamada, etc.

Motivação: Como foi definido que nossa arquitetura seria baseada em micro serviços foi preciso pensar em uma forma de fazer a orquestração das apis de cada um desses serviços, pois ficaria muito complexo tratar isso na camada cliente, além de ficar caro ter que fazer várias requisições para serviços diferentes. Com a api gateway tivemos as soluções desse problemas.

[DSUFG07]

Descrição: Os micro serviços serão construídos em Java e implementarão o paradigma OO, de forma que os conceitos que fazem parte das regras de negócio serão divididos em classes e objetos.

Objetivo: O objetivo em se utilizar a linguagem Java é em usar uma linguagem consolidada, que conta com uma ampla gama de suporte. Ao se utilizar o paradigma OO buscamos uma codificação mais próxima do cenário real com classes e objetos, maior reutilização de código, maior manutenibilidade, etc.

Motivação: Java por ser uma linguagem de programação bem consolidada, em constante evolução e com uma comunidade bem ativa nos dará subsídios para construirmos um sistema sólido, estável e passível de suporte por parte da comunidade. O fato de ser uma linguagem que implementa o paradigma OO é outro motivador, já que esse paradigma, se implementado corretamente, facilita e muito a manutenibilidade do sistema, ajudando na manutenção e evolução do mesmo.

[DSUFG08]

Descrição: A gerência do banco de dados do sempreUFG será feita pelo PostgreSQL, que é um sistema de gerenciamento de banco de dados relacional. Ou seja, os dados serão organizados em tabelas, linhas e colunas.

Objetivo: Com o PostgreSQL desejamos ter um controle maior dos dados de nosso sistema, organizando-os em tabelas, definindo relações, etc.

Motivação: O PostgreSQL por ser um sistema bem consolidado, com uma ótima comunidade de suporte se torna uma boa opção para o gerenciamento de nossos dados. O fato de ser da comunidade open source também é uma motivação, pois com isso teremos mais controle sobre ele. O fato de escolhermos o PostgreSQL também é corroborado pelos requisitos do sistema.

[DSUFG09]

Descrição: Cada micro serviço terá seu próprio banco de dados.

Objetivo: Com isso desejamos melhorar a confiabilidade do sistema. Os micro serviços devem ser independentes, caso um venha a cair os outros não devem ser impactados por isso, então para evitarmos de termos apenas um ponto de falha, que no caso seria o banco de dados, foi decidido que o banco de dados também será específico para cada micro serviço.

Motivação: A maior motivação é atender ao requisito de confiabilidade que está sendo priorizado nesse sistema. Com essa decisão o sistema estará menos sujeito a falhas, pois é eliminada a possibilidade de ter um único ponto de falha, assim os micro serviços podem ser independentes e assim melhorar também a manutenibilidade do sistema.

[DSUFG10]

Descrição: Todos as configurações dos micro serviços são centralizados em um Servidor de Configuração que guardará os arquivos de configuração em um repositório do Git Hub.

Objetivo: Pela grande quantidade de micro serviços que o SempreUFG terá que gerenciar com o fim de facilitar esse gerenciamento é feito a centralização dos arquivos de configuração para ter apenas um ponto de mudança.

Motivação: Otimizar a manutenção dos micro serviços e centralizar todos os arquivos de configuração em um único lugar .

[DSUFG11]

Descrição: É utilizado o Eureka como serviço de descoberta de micro serviços, dessa forma todos os micro serviços serão clientes eureka e irão se registrar no servidor eureka.

Objetivo: Com isso os serviços para serem usados precisam se registrar no servidor eureka. A apigateway irá disponibilizar para a camada cliente apenas àqueles micro serviços que forem registrados. O eureka tem o objetivo de fazer a descoberta de micro serviços e balanceamento de cargas.

Motivação: O objetivo é ter um controle maior dos micro serviços, fazer o balanceamento de cargas e evitar o consumo de micro serviços desconhecidos.

ANEXO A

Matriz de Rastreabilidade - Atributos x Requisitos

DOCUMENTO ANEXADO EXTERNAMENTE

ANEXO B

Matriz de Rastreabilidade - Estilos Arquiteturais x Atributos

	Segurança	Portabilidade	Confiabilidade	Manutenibilidade	Usabilidade	Eficiência	Funcionalidade
Multicamadas				o		x	
Cliente Servidor				o		x	
Rest	o	o					
Micro serviços		o	o	o		x	

Legenda:
o → Beneficia;
x → Prejudica;

Multicamadas e Manutenibilidade: O modelo de multicamadas permite a organização dos componentes em camadas que possuem responsabilidades bem definidas. Com isso a manutenção é facilitada, pois o impacto de uma mudança fica apenas na camada modificada.

Cliente Servidor e Manutenibilidade: Esse modelo faz a separação entre cliente e servidor, fazendo esses dois componentes trabalhar de forma independente, podendo evoluir e ser modificados de forma independente.

Micro serviços e Manutenibilidade: Cada micro serviço representa uma única função bem definida do sistema, fazendo com que eles possam evoluir de forma independente.

Micro serviço e confiabilidade: Como cada micro serviço trabalha de forma independente caso algum caia o sistema ainda continuará funcionando com os micro serviços que ainda estão em funcionamento.

Micro serviço e portabilidade: Cada micro serviço se comunica através de apis rest, a comunicação entre eles não depende de uma tecnologia específica, o que faz eles portáteis.

Rest e portabilidade: Considerando que portabilidade também diz a respeito de um sistema conseguir se comunicar com outro, podemos dizer que o REST promove isso, visto que cada serviço expõe seus dados pelas apis rest, deixando livre para que qualquer outro sistema, independente de tecnologia possa consumir esses dados, desde que eles atinjam certos requisitos.

Rest e segurança: O rest em sim não promove a segurança do sistema, mas caso ele seja implementado de forma a sempre exigir um token de autenticação, então ele pode ajudar na segurança, evitando que requisições não autorizadas sejam feitas.

Cliente Servidor e Portabilidade: O cliente servidor promove a portabilidade pois sistemas cliente servidor geralmente trabalham de forma independente, um cliente pode se comunicar com outro servidor e um servidor pode ser consumido por outro cliente. Ou seja, eles podem ser portados para outros sistemas.

Micro serviço e desempenho: Se mal desenvolvido o micro serviço pode prejudicar o desempenho pelo aumento de requisições feitas aos serviços e ao banco de dados.

Multi camadas e desempenho: O estilo de multicamadas pode prejudicar o desempenho caso haja muitos níveis de camada. Com isso os dados teriam que viajar por muitas camadas até seu destino e isso se não for bem gerenciado pode ser problemático para o desempenho da aplicação.