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

SEMPREUFG
PROCESSO DE DESIGN ARQUITETURAL DE HOFMEISTER

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

O presente documento tem como objetivo executar e documentar a arquitetura do software *SempreUFG*, cujo requisitos podem ser acessados no link a seguir: http://bit.ly/SUFG-Requisitos.

É discriminado 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. Análise Arquitetural

Na primeira parte do processo foi realizada uma análise arquitetural do *software*, atividade que teve como fim levantar os requisitos RAS do conjunto de requisitos do SempreUFG.

2.1. Requisitos Arquiteturalmente Significativos

- RNF-SegTrans: A transação autenticada é segura;
- RNF-SegInfo: Há Garantia de segurança da informação;
- RNF-AutentUsu: Há autenticação de usuário;
- RNF-ContrAcesPapel: Recursos com controle de acesso baseado em papel podem ser configurados;
- RNF-IntgrCercom: Há integração com sistemas do CERCOMP para importação de dados de egressos;
- RNF-AmbOpServ: O componente servidor é executado no ambiente operacional lógico do servidor;
- RNF-AplicWeb: O software é uma aplicação Web;
- RNF-FuncIndep: As funções são mutuamente Independentes;

2.2. Requisitos de Qualidade

Com base nos requisitos RAS levantados podemos tirar os requisitos de qualidade que mais são relevantes para o software SempreUFG, que são:

- Segurança - *RNF-SegTrans, RNF-SegInfo, RNF-AutentUsu e RNF-ContrAcesPapel;*
- Portabilidade - *RNF-IntgrCercom, RNF-AplicWeb;*
- Confiabilidade - *RNF-FuncIndep;*
- Manutenibilidade;

3. Síntese Arquitetural

Com a análise arquitetural feita fica claro que é preciso ter uma separação entre cliente e servidor, visto que se trata de software para rodar na plataforma web. Além disso uma das maiores preocupações que precisamos de ter é com a segurança, então independente do estilo arquitetural escolhido é de suma importância a priorização desse atributo de qualidade.

Para atender aos requisitos de qualidade salientamos alguns estilos abaixo:

3.1. **Multicamadas**

Faz uma separação de interesses, cada camada tem uma responsabilidade bem definida. Isso facilita a separação entre cliente e servidor. Permite que servidor e cliente trabalhem e evoluam de

forma independente. O fato de ser multi-camadas permite que haja maior liberdade de comunicação entre elas. Assim uma camada pode fazer uma solicitação à uma camada e receber desta uma resposta.

Com esse estilo podemos ter uma camada que cuide da segurança, ou seja uma camada entre a UI e os serviços que irá cuidar da autenticação do usuário e garantir que cada requisição à api seja feita de forma segura.

3.2. **Cliente-Servidor**

Como trata-se de uma aplicação web em que a parte cliente irá rodar no computador do cliente e a parte server estará em alguma máquina do CERCOMP, é mister que haja a separação entre cliente e servidor, a comunicação se dará então através de redes de computadores. É importante salientar que essa divisão será feita por camadas.

3.3. **Microserviço**

Esse estilo arquitetural nos atenderia bem o atributo da indisponibilidade. Um dos requisitos do software diz que se uma funcionalidade parar de funcionar as outras não devem ser afetadas. Mas ao termos uma aplicação monolítica teremos um único ponto de falha, assim não conseguindo atender esse requisito.

Para isso ao separarmos o sistema em micro serviços teremos a separação das funções, protegendo as outras caso alguma falhe. Cada serviço é independente, ele apenas recebe informações de outros serviços e decide por si só o que fazer com essas informações.

3.4. **REST**

O Rest é um conector, ou um intermediário, entre os micro serviços e camadas, ou seja a comunicação entre os componentes será intermediada por interfaces bem definidas feitas em REST.

Ele é bom porque fornece interoperabilidade entre sistemas de computadores na internet. O REST pode ser considerado como uma série de princípios que beneficia arquiteturas e padrões WEB.

Com o REST podemos utilizar o HTTP de forma mais eficaz, ter um trânsito de informações mais eficiente e assim mais rápido. Dessa forma conseguimos mais desempenho e confiabilidade.

5. Avaliação Arquitetural

De certa forma foi identificado que o que mais precisamos priorizar em nossas decisões arquiteturais seria a segurança. Ao escolher o modelo de camadas e adicionar uma camada específica para segurança estamos tentando atender esse conceito e nosso sistema.

Além disso o modelo de camadas e cliente-servidor atendem bem ao requisito de portabilidade e interoperabilidade, já que quando temos uma separação clara entre servidor fica fácil evoluir esses dois conceitos de forma separada.

Com o REST conseguimos um uso mais eficiente da comunicação entre o cliente e o servidor, podendo prover, assim, segurança e desempenho entre nossas aplicações.

Para corroborar a arquitetura proposta, documentos representantes das viewpoints e views foram criados com seus respectivos modelos.

Outros documentos pertinentes à essa avaliação como ponto de vista e visões estão incluídos em anexo.