UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
Bacharelado em Engenharia de Software
Arquitetura de Software
Discentes: Gustavo Batista, Murillo Nunes, Saulo Calixto

PROCESSO DE DESIGN ARQUITETURAL DE HOFMEISTER

1. Introdução

O presente documento tem como objetivo executar e documentar o processo de Design Arquitetural de Hofmeister para o *software* SempreUFG, cujo requisitos podem ser acessados no link a seguir: http://bit.ly/SUFG-Requisitos.

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;
- RNF-MaxDadosTrans: O limite máximo de dados que uma transação transporta pode ser configurado;
- RNF-MaxTransSimul: O limite máximo para transações simultâneas pode ser configurado;

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;*
- Interoperabilidade - *RNF-IntgrCercom, RNF-AplicWeb;*
- Confiabilidade - *RNF-FuncIndep;*
- Desempenho: *RNF-MaxDadosTrans, RNF-MaxTransSimul;*
- Funcionalidade;

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.

4. 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 microserviços, 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 client 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 à dados que são exclusivos para àqueles cadastrados no sistema.
Motivação: Como o requisito deu muita prioridade à segurança escolhemos implementar esse protocolo para poder garantir a segurança dos dados do servidor, fazendo com que nossa aplicação atinga o fim que foi pedido no requisito.

4. 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.