Identificador Único de Questões

Documento de Arquitetura de Software

Arthur Moura
Pietro Niero
Marcos Chediak

1. Introdução

1.1 Finalidade

Este documento tem como objetivo definir os aspectos da Arquitetura do software Identificador Único de Questões (IUQ) e é direcionado para as partes interessadas do software a ser desenvolvido como Gerentes do Projeto, clientes e a equipe técnica.

1.2 Escopo

Este documento se baseia nos requisitos do IUQ para definir os atributos de qualidade a serem priorizados, assim como os estilos de arquitetura que favorecem tais atributos e a representações das visões arquiteturais, principalmente do Projetista e do Desenvolvedor.

2. Contexto da Arquitetura

2.1 Funcionalidade e Restrições Arquiteturais

Id.	Tipo	Descrição
RNF01	Requisito Não Funcional	O sistema deve ser possível de ser acessado pela web.
RNF02	Requisito Não Funcional	O sistema deve funcionar 24h por dia. 7 dias por semana.
RNF03	Requisito Não Funcional	O sistema deve possuir infraestrutura em cloud.

RNF04	Requisito Não Funcional	Dados sensíveis devem ser armazenados com segurança de acordo com as normas de LGPD.
RNF05	Requisito Não Funcional	O sistema deve estar sempre atualizado para conseguir fazer a leitura de diferentes tipos de provas.

Com base nos RNFs, conseguimos algumas informações importantes que devem ser consideradas na escolha do estilo arquitetural do software. O RNF01 deixa claro que o contexto da aplicação é Web e o RNF03 mostra a existência de componentes, um cliente e um servidor. Já o RNF05 revela que o software precisa ser de fácil manutenção. Esses requisitos direcionam para um estilo arquitetural **MVVM** (**Model-View-ViewModel**).

2.2 Atributos de Qualidade Prioritários

Conforme definido pelos requisitos funcionais e não funcionais, há alguns atributos que devem ser priorizados. A **Segurança** é um deles. Para favorecer esse atributo, podem ser selecionados estilos arquiteturais de camadas e componentes que, devido a sua modularização, permite que diversos níveis de segurança sejam implementados.

Outro atributo a ser priorizado é a **Manutenibilidade**, que permite que os componentes possam ser substituídos com relativa facilidade caso haja necessidade.

3. Representação da Arquitetura

Conforme o que foi estabelecido pelos tópicos anteriores, o software a ser desenvolvido terá a arquitetura MVC, mais especificamente **MVVM** (**Model-View-ViewModel**).

Para representar as decisões arquiteturais definidas ao findar da análise, serão utilizados os seguintes pontos de vista:

Ponto de Vista	**Visão**	**Diagrama(s)**
Projetista	Desenvolvimento	Componentes
Desenvolvedor	Lógica	Classes
	Segurança	Pontos-fracos

Os próximos tópicos abordam esses pontos de vista arquiteturais com mais detalhes.

4. Ponto de Vista do Projetista

O ponto de vista do projetista é direcionado aos projetistas e desenvolvedores do software e tem como objetivo definir as principais partes que o compõem. Aqui serão tratados os três componentes do software e as responsabilidades de cada um deles.

4.1 Visão dos Componentes

- **Cliente:** É o navegador, a plataforma onde o usuário irá interagir com o sistema. Realiza a comunicação com o servidor.

- **Servidor:** Responsável pela implementação da lógica presente no software. É onde se encontra a **Model, View, ViewModel, Controllers** e **Router** .
    - **Model:** É a classe a ser desenvolvida, com suas propriedades e métodos, o back-end da aplicação. São as entidades do sistema;
    - **View:** É a “tela” usada para interação, o front-end da aplicação. Se trata da interface para as operações e se comunica com as ViewModels;
    - **ViewModel:** É onde ocorre a comunicação entre o front e o back-end da aplicação. Usada para declaração de métodos de validação, cadastro, edição, etc. Chama os métodos da API a partir dos controllers;
    - **Controller:** É onde são declarados os métodos da API Rest (GET, POST, PUT e DELETE). Cada modelo/entidade possui um controller.
    - **Router:** São as rotas da aplicação que cada View deve ser direcionada. Se comunica com a API.

- **Banco de dados:** Local de armazenamento de todas as informações e dados dos usuários.

5. Ponto de Vista do Desenvolvedor

O ponto de vista do desenvolvedor é direcionado aos projetistas e desenvolvedores do software e tem como objetivo definir as principais partes responsáveis por definir as funcionalidades e restrições do software, tal como as classes.

5.1 Visão das Classes

As classes seguirão os padrões da arquitetura escolhida com **Model**, **View**, **ViewModel**, **Controllers** e **Router**. Vão ser usados objetos de transferência de dados como componentes **ViewModel**. Objetos de Transferência de Dados, ou **DTO**’s, são um objeto simples usado para transferir dados de um local a outro na aplicação, sem lógica de negócios em seus objetos e comumente associado à transferência de dados entre uma camada de visão.

Para a classe de validação, foi usado um algoritmo inspirado na ***distância de Levenshtein***, que na teoria da informação, é dada pelo número mínimo de operações necessárias para transformar um string no outro. No sistema, esse algoritmo irá comparar

enunciados de questões e vai chegar a um valor entre 0 e 1 de similaridade. O software não permitirá adição de questões com um índice de similaridade maior do que 0.6.

5.2 Visão de Segurança

A segurança é um dos atributos de qualidade a serem priorizados durante o desenvolvimento da arquitetura do software. A visão de segurança é responsável por definir como a segurança dos usuários e do software será realizada à nível estrutural.

Na classe **App**, haverá uma validação e verificação para garantir que as **Routes** e **Views** foram geradas corretamente antes da interação com o usuário. Em relação ao **Servidor**, deve haver uma verificação para garantir que os dados estão sendo mostrados corretamente, sem nenhuma informação sensível e desnecessária. Por último, na classe **User**, mais especificamente em sua **ViewModel**, haverá uma validação para garantir que o usuário está de fato cadastrado no sistema e pode acessá-lo.

Na parte do back-end, foi feito o uso da **API Swagger**. Swagger é uma linguagem de descrição de interface para descrever APIs RESTful expressas usando JSON. Ela foi escolhida por ser open source e bem documentada, o que diminui a chance de prováveis falhas de segurança.