MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Documento de Arquitetura

Ester Adaianne Oliveira Ferreira
Gabriel Formigoni dos Santos Neto
Layane Grazielle Souza Dias
Marcos Vinícius de Moraes
Pedro Ivo Santana Melo

Goiânia

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Sumário

1. Introdução

1.1 Finalidade

A principal finalidade deste documento é definir os aspectos essenciais da Arquitetura de Software, sendo direcionado aos stakeholders do projeto, possuindo grande foco para os Desenvolvedores e para a Equipe de implantação.

1.2 Escopo

O **“MoviesRec”** tem como objetivo gerar sugestões de filmes ou séries para o usuário. O sistema fará isso através de uma inteligência artificial, treinada pelo próprio usuário, por meio de avaliações de sugestões. O usuário poderá informar quais serviços de streaming ele possui, gêneros de filme, faixa etária, tipo de ocasião (individual, casal, família, amigos, etc.) para obter resultados otimizados.

1.3 Definições, Acrônimos e Abreviações

Id.: Identificador.
Software: Conjunto de documentações, guias, metodologias, processos, códigos e ferramentas para a solução de um problema.
Stakeholder: Indivíduo, grupo ou organização que possua interesse no Sistema.
Visão Arquitetural: Produto resultante da interpretação de um Stakeholder do sistema.
Arquitetura de Software: Forma como os componentes são agrupados com o objetivo de construir um software ou sistema.
Ponto de Vista Arquitetural: Produto resultante da execução de uma Visão Arquitetural.
Javascript: Linguagem de programação de alto nível, de propósito geral, interpretada, de sintaxe concisa e clara.

1.4 Referências

Id.	Nome do Artefato
AAS_1	Elicitação
AAS_2	Project Model Canvas

AAS_3	Requisitos Iniciais

1.5 Visão Geral

De maneira simples, o documento visa descrever a estrutura geral do sistema a ser desenvolvido, incluindo etapas de organização lógica e física, bem como de componentes e suas relações e ainda acerca de possíveis interfaces.

As principais decisões de projeto, restrições e metodologias adotadas para o desenvolvimento do projeto estão descritas e detalhadas. Além disso, vale ressaltar que esse Projeto Arquitetural se trata de um processo contínuo, sendo assim é suscetível a possíveis melhorias futuras.

2. Contexto da Arquitetura

2.1 Funcionalidades e Restrições Arquiteturais

Id.	Tipo	Id. do Documento de Requisitos
RAS_1	Requisito Não Funcional	RNF-01
RAS_2	Requisito Não Funcional	RNF-02
RAS_3	Requisito Não Funcional	RNF-03
RAS_4	Requisito Não Funcional	RNF-04
RAS_5	Requisito Não Funcional	RNF-05
RAS_6	Requisito Não Funcional	RNF-06
RAS_7	Requisito Não Funcional	RNF-07
RAS_8	Requisito Não Funcional	RNF-08
RAS_9	Requisito Não Funcional	RNF-09
RAS_10	Restrição do Sistema	RS-02

RAS_11	Requisito Funcional	RF-01
RAS_12	Requisito Funcional	RF-03
RAS_13	Requisito Funcional	RF-06
RAS_14	Requisito Funcional	RF-10
RAS_15	Requisito Funcional	RF-12
RAS_16	Requisito Funcional	RF-13
RAS_17	Requisito Funcional	RF-14

2.2 Atributos de Qualidades Prioritários

Para evitar sugestões de filmes e séries indesejáveis, usuários terão a possibilidade de fornecer informações pessoais sobre si mesmos, como: eventos traumáticos dos quais preferem não ser lembrados; assuntos delicados que preferem não ver abordados em obras de entretenimento audiovisual; etc. Com isso em vista, o atributo de qualidade de maior prioridade para o projeto é a **segurança** (*RAS_3*, *RAS_4*, *RAS_5*) dos dados fornecidos pelo usuário.

Ainda sob o ponto de vista da qualidade, é necessário incluir na arquitetura elementos que contribuam com a **usabilidade** (*RAS_6*) e **confiabilidade** (*RAS_7*, *RAS_8*) do sistema, os quais são os dois atributos de segunda maior prioridade no projeto, visando uma experiência de uso mais agradável e sugestões mais precisas de filmes e séries.

Em terceiro nível de prioridade estão os atributos de **escalabilidade** (*RAS_7*) e **manutenibilidade** (*RAS_9*) do sistema, de forma a facilitar futuras alterações de código necessárias para implementar novas funcionalidades, modificar funcionalidades já existentes e incrementar o desempenho do sistema em uma escala maior de uso.

2.3 Tecnologias

Linguagem de programação back-end: Java
Linguagens front-end: HTML, CSS e Javascript
Ambiente de execução: Apache Tomcat e navegador web
Frameworks: Java Spring Boot e Vue.js
Banco de dados: MySQL

Hospedagem do site: Github

3. Representação da Arquitetura Candidata

A arquitetura do software a ser desenvolvido é baseada em uma arquitetura de camadas e cliente-servidor. A camada para área de interação com o usuário consiste em uma interface web, que permite a fácil utilização do sistema. A camada de aplicação consiste em um conjunto de serviços web que processam as solicitações dos usuários e interagem com a camada de dados. A camada de dados consiste em um banco de dados relacional, fornecido pelo cliente, que armazena as informações sobre as medições no solo. As camadas se comunicam através de um conjunto de interfaces pré-definidas.

Além disso, a utilização da arquitetura cliente-servidor divide a aplicação/serviço nos componentes clientes e servidores. Os clientes são os dispositivos, nesse caso os computadores, que realizam as requisições e também fornecem aos usuários uma interface. Por outro lado, os servidores recebem tais requisições, para realizar o seu processamento e fornecer as devidas respostas de volta aos clientes.

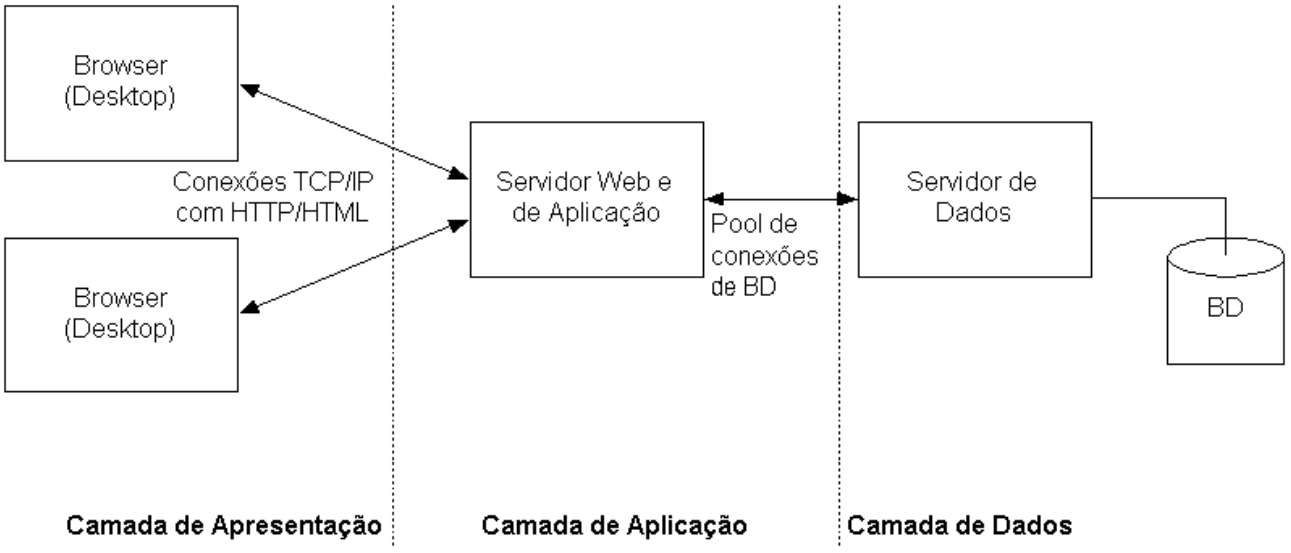


Figura 1: modelo em camadas e cliente-servidor

Com a arquitetura Cliente/Servidor, garantimos que cada instância de um cliente enviará as requisições para um servidor e aguardará a resposta. Através da

decomposição em camadas independentes, atendemos também aos requisitos de manutenibilidade, por ser mais fácil executar manutenções ou substituições de servidores sem afetar o serviço. Além disso, garantimos os critérios de segurança levantados, pois as requisições serão centralizadas apenas no servidor, controlando o acesso dos recursos e permissões.

Com os protocolos de transporte e aplicações de rede dessa arquitetura, garantimos a transferência de dados de forma confiável entre cliente e servidor. Também garantimos uma maior adaptabilidade a diferentes ambientes de clientes. Devemos nos atentar à sobrecarga do servidor, que pode gerar conflito com os requisitos de performance estabelecidos.

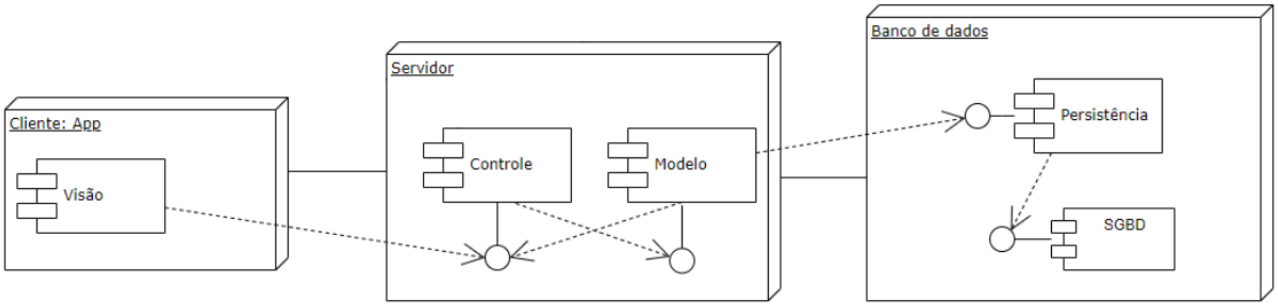


Figura 2: diagrama de componentes

4. Visão Geral

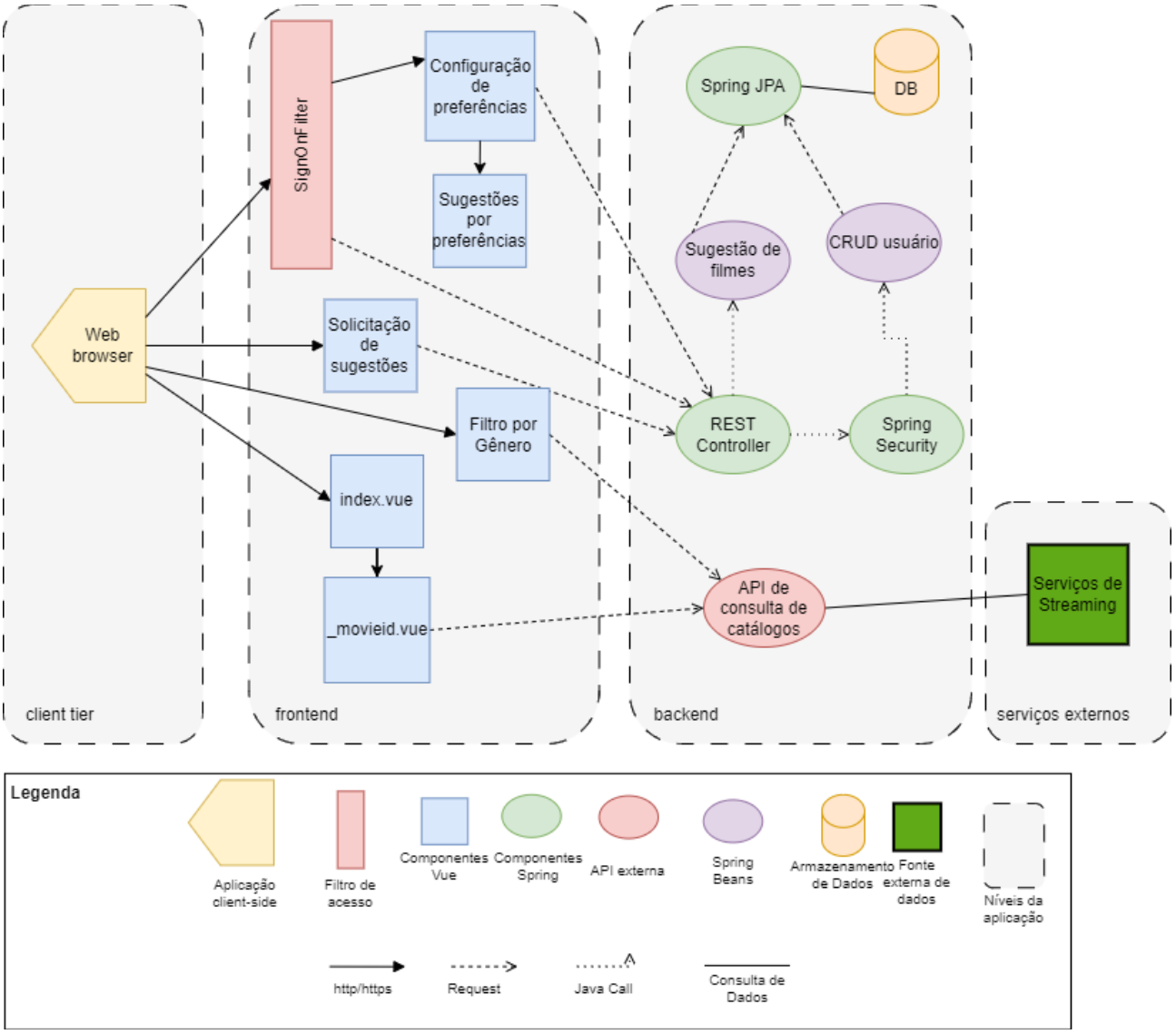
4.1 Descrição

Na visão geral é criada uma representação visual que fornece uma visão geral de um sistema ou processo. Ele mostra as principais entidades envolvidas e suas interações de forma simplificada.

Na visão geral da distribuição do sistema, o lado do cliente vai consistir de uma página web e um aplicativo para acesso ao software. O servidor vai consistir em 3 serviços distintos:

- Código básico que irá ser responsável por login, exibição de páginas, redirecionamentos internos, etc;
- API responsável por gerar sugestões de filmes e séries com base nas preferências e filtros fornecidos pelo usuário;
- API que irá consumir os catálogos de serviços de streaming e exibi-los para o usuário do sistema.

Além da parte do cliente e do servidor, temos a camada de persistência de dados e serviços de streaming que serão utilizados pela API mencionada anteriormente.



5. Decisões arquiteturais

5.1 Requisitos Arquiteturalmente Significativos

ID	RAS	Descrição do Cenário
01	Usabilidade	O sistema deve ser fácil de usar e entender
02	Integração	O sistema deve exibir informações de catálogo dos serviços de streaming, auxiliando no direcionamento
03	Manutenibilidade	O sistema deve seguir boas práticas de desenvolvimento para adequar-se facilmente a novas funcionalidades
04	Segurança	O sistema deve ser protegido contra acesso não autorizado

5.2 Decisões para o Frontend

ID do RAS	01
Tipo de decisão arquitetural	Usabilidade
Decisão	O sistema MoviesRec usará SSR (Server-Side Rendering) para apoiar na usabilidade e desempenho.
Justificativa	Renderização no lado do servidor com renderização no lado do cliente para melhorar o desempenho e a indexação por mecanismos de busca. Trade-offs: + Usabilidade + Desempenho de busca

	+ Modularidade - Requisitos de Hardware
Forma de implementação (tecnologia)	Uma possível implementação seria com o framework Nuxt.js, baseado em Vue para apoiar no desempenho do SEO(Search Engine Optimization), renderização do lado do servidor, melhora a indexação por mecanismos de busca e a usabilidade inicial.

ID do RAS	02
Tipo de decisão arquitetural	Integração
Decisão	O sistema MoviesRec deve se comunicar com diferentes APIs, usando uma biblioteca que faça requisições HTTP
Justificativa	A capacidade de se comunicar facilmente com outras APIs é essencial para o funcionamento das funcionalidades de busca por filmes em cada plataforma de streaming, com uso de uma biblioteca que faça requisições HTTP de maneira simples. Trade-offs: + Flexibilidade +Tratamento de Erros - Personalização Limitada
Forma de implementação (tecnologia)	Uma possível implementação seria com o uso da biblioteca axios, podendo fazer requisições HTTP de forma clara e eficaz.

ID do RAS	03
Tipo de decisão arquitetural	Existencial
Decisão	O sistema MoviesRec deve utilizar o estilo arquitetural de componentes para apoiar na manutenibilidade do Frontend

Justificativa	Permite criar uma arquitetura organizada, onde cada componente representa uma funcionalidade ou elemento visual específico. Trade-offs: + Manutenibilidade + Modularidade + Reutilização de Código - Desempenho - Overhead de Gerenciamento
Forma de implementação (tecnologia)	Uma possível implementação seria com o uso do framework Vue.js, que permite construir interfaces complexas de maneira eficiente, promovendo a reutilização de código, a manutenibilidade e a separação clara de preocupações.

5.3 Decisões para o Backend

ID do RAS	02
Tipo de decisão arquitetural	Integração
Decisão	O sistema MoviesRec deve se comunicar com diferentes APIs, usando uma biblioteca que faça requisições HTTP
Justificativa	O Spring Boot é conhecido por facilitar a integração entre diferentes componentes e sistemas. Ele oferece suporte a várias tecnologias de integração através de requisições e mensageria assíncrona, o que ajuda a conectar diversos sistemas de maneira eficiente. Trade-offs: - Controle - Flexibilidade
Forma de implementação (tecnologia)	Uso de tecnologia REST para comunicação de diversos sistemas através de requisições.

ID do RAS	03
Tipo de decisão arquitetural	Manutenibilidade
Decisão	O sistema MoviesRec deve possuir código bem estruturado e documentado, de forma a facilitar futuras correções ou modificações no sistema
Justificativa	O Spring Boot incentiva a modularização do código por meio da criação de componentes independentes e reutilizáveis. O uso de injeção de dependência, configuração externa e separação de preocupações ajuda a tornar o código mais limpo e manutenível. Além disso, as ferramentas de testes oferecidas pelo Spring Boot (como JUnit e TestNG) facilitam a criação de testes automatizados, o que contribui para a manutenibilidade do código. Trade-offs: + Complexidade de integração + Curva de aprendizado
Forma de implementação (tecnologia)	Uso de Dependency Injection e YAML para reduzir acoplamento entre componentes e facilita a alteração de configurações quando necessárias. Uso de JUnit para testes unitários, testes de integração e testes de aceitação.

ID do RAS	04
Tipo de decisão arquitetural	Segurança
Decisão	O sistema MoviesRec deve possuir estruturas robustas de credenciamento e métodos bem definidos de registro e consulta de dados, os quais não podem ser contornados.
Justificativa	Projetos Spring lidam de forma sólida com aspectos de segurança em aplicações. São fornecidos recursos para autenticação, autorização e proteção contra vulnerabilidades conhecidas, como ataques de injeção e cross-site scripting (XSS).

	Trade-offs: + Complexidade
Forma de implementação (tecnologia)	Com o uso de Spring Security, é possível implementar camadas de segurança na aplicação, protegendo os endpoints e recursos mais vulneráveis.

6. Ponto de vista dos Casos de Uso

6.1 Descrição

Na análise de requisitos, é criada uma visão de casos de uso para fornecer uma base para o planejamento da arquitetura e outros artefatos. Essa visão representa os casos de uso e cenários que o usuário terá, além de classes e riscos técnicos relevantes para a arquitetura do sistema. A visão de casos de uso é considerada em cada iteração do ciclo de vida do software.

6.2 Visão de Casos de Uso

Os requisitos funcionais foram utilizados para compor os casos de uso, resumindo as principais funcionalidades do sistema nos diagramas abaixo:

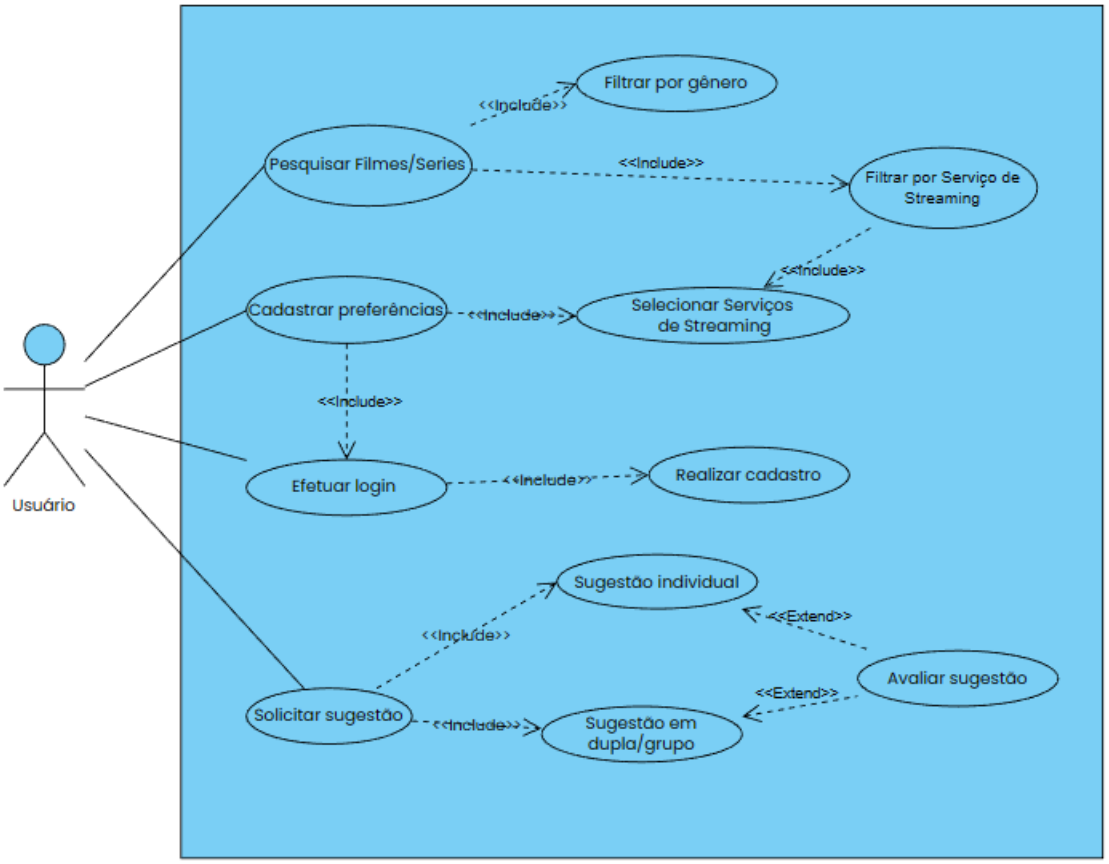


Figura 3: Casos de uso

7. Ponto de vista do Projetista

7.1 Descrição

O ponto de vista de projetista é um dos pontos de vista arquiteturais que se concentra nas decisões de projeto e nas preocupações técnicas específicas relacionadas à implementação do sistema. Ele fornece informações detalhadas e orientações para os projetistas e desenvolvedores envolvidos na construção do sistema.

7.2 Visão em Módulos

- **Módulo de Autenticação/Registro:**
  - Responsável pelo cadastro e autenticação dos usuários no sistema.
  - Funcionalidades: registro de novos usuários, login, recuperação de senha, gerenciamento de sessões.
- **Módulo de Gerenciamento de Usuários:**
  - Responsável por permitir que os usuários gerenciem suas informações pessoais e preferências de recomendação.
  - Funcionalidades: visualização e edição do perfil do usuário, configuração de preferências de recomendação, adição de serviços de streaming, histórico de visualizações.
- **Módulo de Busca de Títulos:**
  - Responsável por permitir que os usuários pesquisem por títulos de séries e filmes com base em critérios como nome, gênero, serviço de streaming, humor e idade.
  - Funcionalidades: pesquisa de títulos por diferentes critérios, exibição dos resultados da pesquisa, filtragem avançada.
- **Módulo de Recomendações:**
  - Responsável por gerar recomendações personalizadas com base nas preferências do usuário e histórico de visualizações.
  - Funcionalidades: geração de recomendações, exibição das recomendações ao usuário, atualização das recomendações com base nas interações do usuário.

- **Módulo de Gerenciamento de Avaliações:**
  - Responsável por permitir que os usuários avaliem títulos e visualizem avaliações e críticas de outros usuários.
  - Funcionalidades: registro de avaliações, exibição de avaliações e críticas de outros usuários, classificação de títulos com base nas avaliações.
- **Módulo de Integração com Serviços de Streaming:**
  - Responsável por integrar o sistema com os serviços de streaming cadastrados pelos usuários, permitindo a busca e o redirecionamento para os títulos disponíveis nessas plataformas.

- Funcionalidades: integração com APIs dos serviços de streaming, busca de informações e disponibilidade de títulos, redirecionamento para o título selecionado na plataforma correspondente.

Estrutura Hierárquica:

1. Módulo de Autenticação/Registro
2. Módulo de Gerenciamento de Usuários
3. Módulo de Busca de Títulos
4. Módulo de Recomendações
5. Módulo de Gerenciamento de Avaliações
6. Módulo de Integração com Serviços de Streaming

Dependências:

- O módulo de Autenticação/Registro pode depender do módulo de Gerenciamento de Usuários para acessar as informações do usuário.
- O módulo de Busca de Títulos pode depender do módulo de Gerenciamento de Usuários para obter as preferências de recomendação do usuário.
- O módulo de Recomendações pode depender do módulo de Gerenciamento de Usuários para obter as preferências de recomendação e histórico de visualizações do usuário.
- O módulo de Gerenciamento de Avaliações pode depender do módulo de Gerenciamento de Usuários para obter informações sobre as avaliações e preferências do usuário.

- O módulo de Integração com Serviços de Streaming pode depender do módulo de Busca de Títulos para realizar pesquisas nos serviços de streaming e do módulo de Gerenciamento de Usuários para acessar as informações dos serviços de streaming cadastrados pelo usuário.

8. Ponto de vista de Segurança

8.1 Descrição

É uma perspectiva específica dentro da arquitetura de um sistema que foca nos aspectos relacionados à proteção, privacidade e segurança das informações e dos recursos do sistema. Essa visão tem como objetivo identificar os requisitos de segurança, estabelecer as estratégias e controles de segurança adequados, e definir as medidas e práticas que devem ser implementadas para garantir a integridade, disponibilidade e confidencialidade do sistema.

8.2 Visão de Segurança

- **Autenticação e Controle de Acesso:**
  - Implementar um sistema de autenticação robusto para garantir que apenas usuários autenticados tenham acesso ao sistema.
  - Utilizar técnicas de controle de acesso para definir permissões e restrições de acordo com o papel ou perfil do usuário.

- **Proteção de Dados:**
  - Utilizar técnicas de criptografia para proteger dados confidenciais, como senhas de usuários e informações pessoais.
  - Garantir que as informações sensíveis sejam armazenadas de forma segura e acessíveis apenas para usuários autorizados.

- **Prevenção de Injeção de Código:**
  - Implementar mecanismos de validação e sanitização de dados para evitar ataques de injeção de código, como SQL injection e XSS (Cross-Site Scripting).

- **Proteção contra Ataques de Força Bruta:**

- Implementar mecanismos de segurança que limitem o número de tentativas de login, como bloqueio de contas após várias tentativas falhas.

- **Proteção de APIs e Integrações:**
  - Utilizar autenticação e autorização adequadas para proteger as APIs utilizadas na integração com os serviços de streaming.
  - Validar e filtrar dados recebidos e enviados pelas APIs, evitando a exposição de informações sensíveis.

9. Ponto de vista Frontend

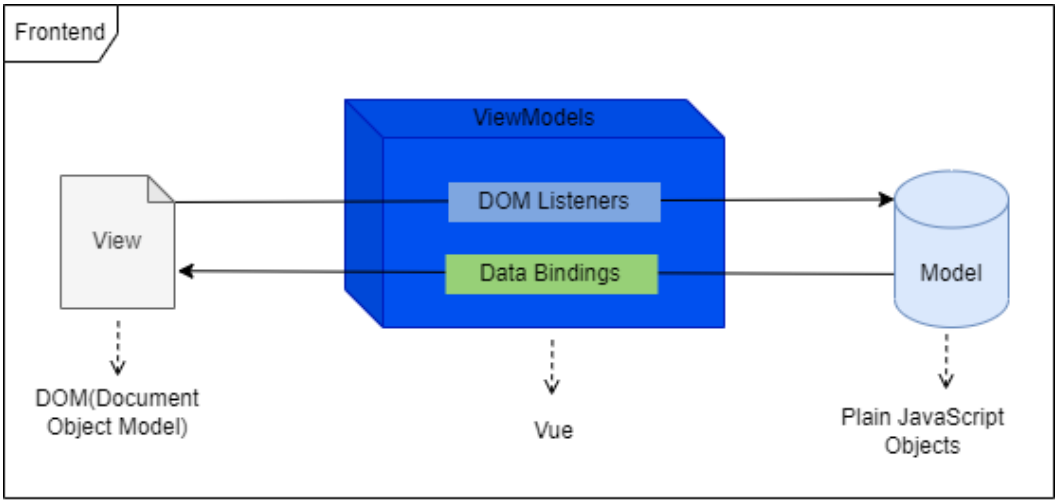
9.1 Descrição

Na análise de requisitos, é definido alguns atributos de qualidade, que serão priorizados de forma a atender os indispensáveis. No desenvolvimento frontend os atributos priorizados são de usabilidade e manutenibilidade, para atender esses atributos foi escolhida uma arquitetura utilizando o framework Vue.js para implementar em componentes.

9.2 Visão Frontend

Vue é focado apenas na camada de visualização, possui uma implementação específica conhecida como Modelo-Visão-ViewModel (MVVM). ajuda a separar preocupações, facilitando o desenvolvimento de interfaces de usuário reativas e componentes reutilizáveis.

- **Model:** objeto JavaScript que contém os dados que a interface de usuário exibe e manipula.
- **ViewModels:** conecta o Model e View, controla a lógica de interação entre eles. Responsável por tornar os dados reativos, ou seja, automaticamente atualizados quando o modelo é alterado.
- **View:** interfaces que os usuários vêem e interagem.



10. Ponto de vista Backend

10.1 Descrição

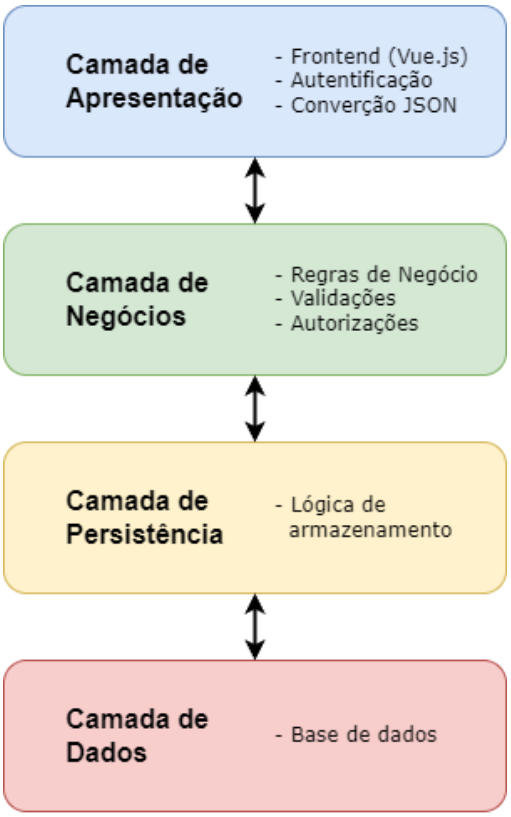
Já para no desenvolvimento do backend, os atributos de qualidade priorizados foram os de integração, manutenibilidade e segurança. Para atender esses atributos foi escolhida uma arquitetura clássica Spring Boot Flow baseada em camadas.

10.2 Visão Backend

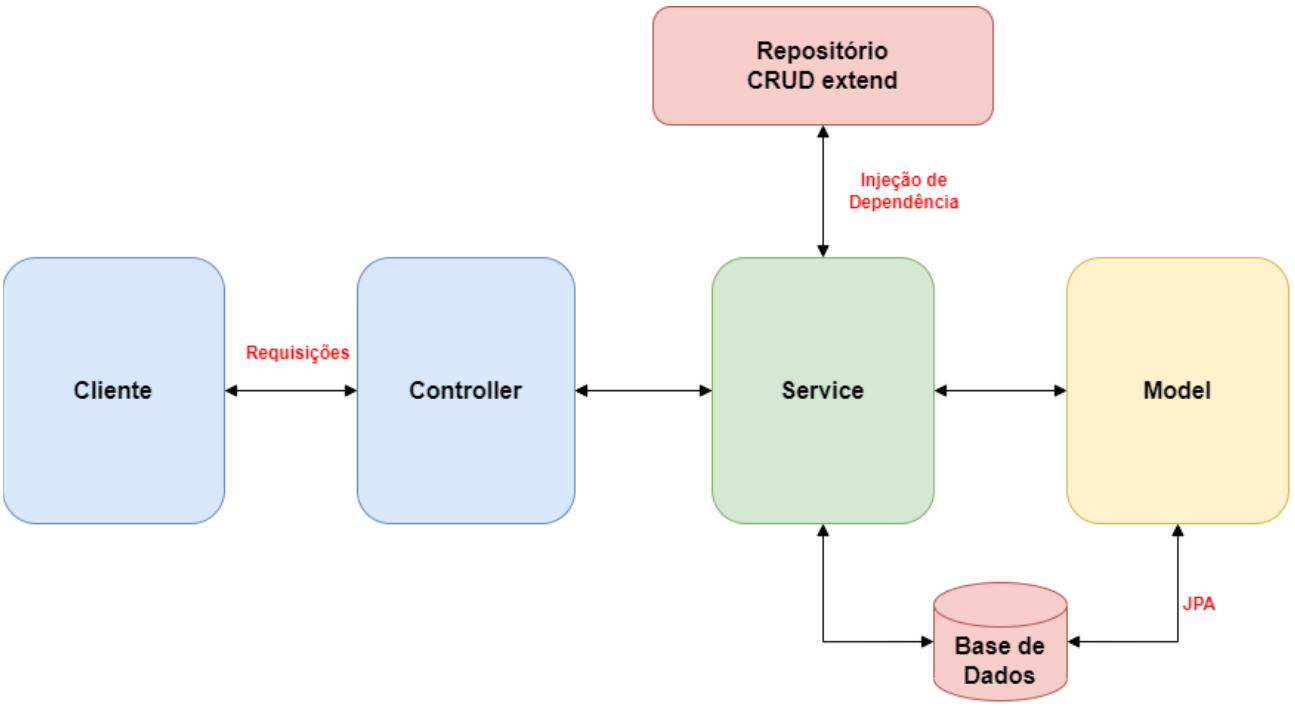
Sendo uma simplificação e automatização do framework Spring, o Spring Boot segue uma arquitetura de camadas onde cada camada se comunica com as camadas imediatamente de cima e de baixo em uma ordem hierárquica. A arquitetura é composta pelas quatro seguintes camadas:

- **Camada de Apresentação:** basicamente o frontend da aplicação. Responsável por gerenciar requisições HTTP e HTTPS e autenticações. Além de converter campos JSON em parâmetros Java Object e vice-versa.
- **Camada de Negócios:** contém e trata as regras de negócio da aplicação através de validações e autorizações feitas por classes de serviço.

- **Camada de Persistência:** contém a lógica de armazenamento e consulta de dados responsáveis por converter objetos de negócio em linhas do banco de dados e vice-versa.
- **Camada de Dados:** contém os dados em si. Pode conter diversos tipos de banco de dados e é a camada responsável por operações de CRUD.



Para garantir controle e segurança, a comunicação entre camadas segue um fluxo estabelecido pela arquitetura Spring Boot Flow.


Repositório
CRUD extend
Injeção de
Dependência
Cliente
Requisições
Controller
Service
Model
Base de
Dados
JPA

11. Referências

VUEJS. Vue.js - Guia de Visão Geral. Disponível em: https://v1.vuejs.org/guide/overview.html. Acesso em: 13 de agosto de 2023.

Spring Boot - Architecture. Disponível em: https://www.geeksforgeeks.org/spring-boot-architecture/ Acesso em 13 de agosto de 2023.

Understanding Spring Boot Architecture. Disponível em: https://levelup.gitconnected.com/understanding-spring-boot-architecture-6083e2631bc6 Acesso em 13 de agosto de 2023.