Relatório 2 - Leitura: Arquitetura e Boas Práticas (I)

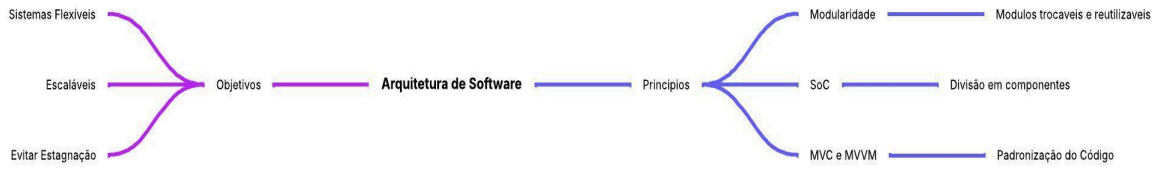
José Carlos Seben de Souza Leite

Descrição da atividade

Começando este card com o artigo “**Princípios de Arquitetura de Software: Design de Sistemas Escaláveis e Flexíveis**” que nos apresenta a base para a estruturação de um sistema, primeiro nos contextualizando com o que é a arquitetura de software e sua importância para a confecção de sistemas flexíveis e escaláveis. Dando continuidade ao artigo temos uma abordagem mais aprofundada sobre a importância da aplicação dos conceitos da arquitetura de software no desenvolvimento de um sistema, pois é quem impede que implementemos um sistema que ficará estagnado no tempo, sem a possibilidade de adaptação a novos requisitos que surgirem ou futuras implementações como funcionalidades novas, ao seguirmos oque nos é imposto pela arquitetura de software nosso produto final será algo componentizado que nos permite alterações ou adições em partes específicas do sistema sem causar problema no restante, ou seja, um sistema escalável e flexível capaz de se adaptar ao mercado e necessidades do cliente, e também evitando retrabalho da equipe de desenvolvimento.

Alguns pontos importantes citados:

- **Separação de Preocupações (SoC)**: Dividir o sistema em componentes independentes
- **Modularidade**: Criar módulos que podem ser trocados ou atualizados sem quebrar o todo.
- **Padrões de Design**: O uso de padrões MVC ou MVVM para organização.



Conteúdo do vídeo “**MVC Descomplicado em 3 Minutos**”.

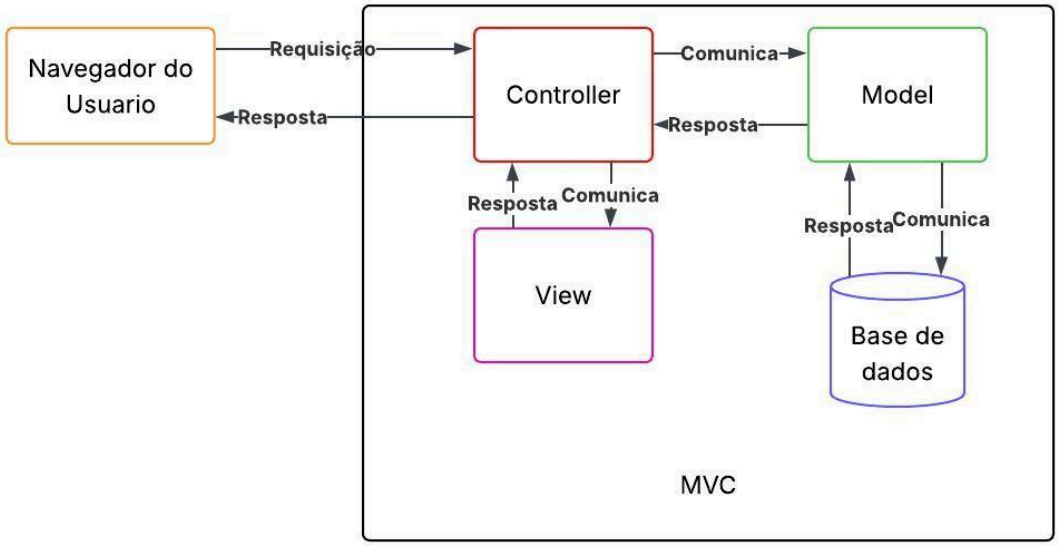
O que é MVC?

É um padrão de Arquitetura de Software que divide sua aplicação em 3 camadas.

Sendo essas:

- **Model** que é a camada responsável pela manipulação dos dados.
- **View** camada de interação do usuário.
- **Controller** camada de controle.

Onde basicamente funciona em um padrão onde quando um usuário acessa um site tem uma requisição que será mandada para o **Controller** que se comunicara com o **Model** (Responsável pela manipulação dos dados) que é quem ira fazer o acesso ao banco de dados para pegar a resposta a requisição do usuário e mandara essa resposta novamente para o **Controller** que agora se continuará com a camada de **View** e com a sua resposta voltando novamente para o **Controller** será dado o retorno para o usuário.



Video “**back end, front end, padrão camadas, MVC, rest**”

Aqui nos é falado um pouco sobre a diferença entre o back e o front-end onde o front é a parte em que o usuário tem acesso, por exemplo, quando acessa um site ou coisa assim, enquanto o back é a parte que estaria no servidor, as funcionalidades, os dados, funções, etc.

Onde nos é apresentado o padrão de organização para o back-end divididos em 3 partes, controladores rest, camada de serviço e camada de acesso a dados, onde cada um tem sua seguinte função:

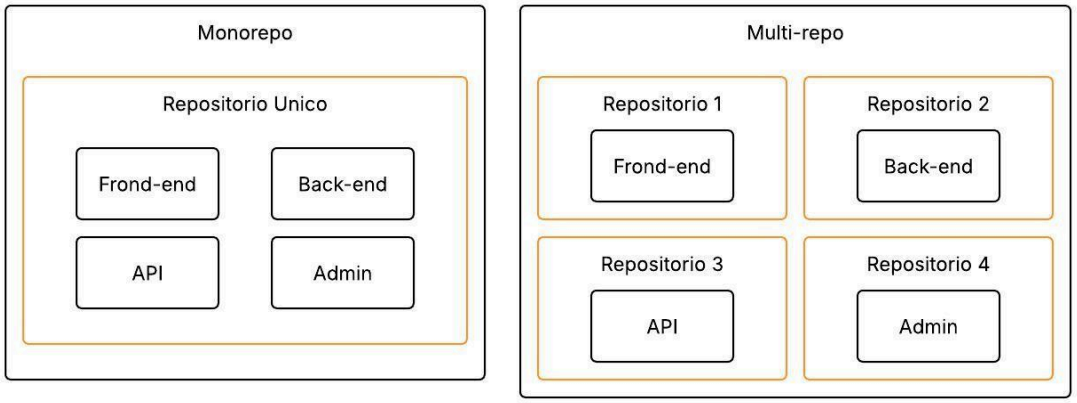
- **Controladores Rest**: Recebem as requisições enviadas.
- **Camada de Serviço:** Onde estão definidas as regras de negócios e validações.
- **Camada de Acesso a Dados:** Responsável por fazer as operações no banco de dados como salvar, apagar ou atualizar algo.

Também nos é apresentado a diferença de um MVC e de uma web service onde o MVC é basicamente oque vimos no vídeo anterior, mas aqui ele tem uma explicação um pouco mais completa já que na representação dele podemos ver com mais clareza a diferença de como a requisição realizada pelo usuário é enviada para o Controll e após fazer o percurso de ir para o model e depois para o view aqui a resposta é envida completa os dados e o front a parte visual como tem na representação uma resposta http e HTML, enquanto no web service fica mais dividido, pois a requisição enviada para o back-end em padrão rest é respondido apenas com os dados enquanto a parte visual fica por conta do front end organizar e mostrar para o usuário.

Vídeo “**Monorepo vs Multi-repo**”

Basicamente o vídeo nos apresenta as diferenças dessas duas abordagens onde uma utilizamos um único repositório para manter todo o projeto enquanto na outra utilizamos diversos repositórios para manter o mesmo projeto o separando por partes, nos é mostrado as vantagens e desvantagens de cada um como no **Monorepo** tem como vantagem a fácil manutenção, reaproveitamento de códigos e mais fácil adaptação para os programadores, e também como foi citado no vídeo os “Atomic Commits” que são você poder alterar uma biblioteca compartilhada e a aplicação que a usa ao mesmo tempo, em um único commit, garantindo que nada quebre, mas tendo como desvantagem por o projeto todo estar sendo mantido em um único repositório acaba que ele pode ficar muito extenso que vai dificultar as manutenções e também ao manter tudo em um repositório se haver um problema nele o

projeto inteiro vai ficar com problema e consequentemente parar de funcionar enquanto no **Multi-repo** temos o projeto dividido em partes então se uma parte der problema as partes que não dependem dessa continuarão funcionando normalmente, por o código ser dividido dificilmente algum dos repositórios vai se tornar muito extenso, mas um problema dessa divisão que quando algo é alterado em algum lugar tera de ser alterado nos outros manualmente com isso podendo gerar problemas ou códigos replicados.



Vídeo “**Por que Monorepos Estão Mudando a Maneira como Desenvolvemos Software**”

Neste vídeo temos uma visão mais aprofundada sobre a adaptação ao uso dos monorepos e como algumas empresas grandes com diferentes projetos estão adotando o uso deles como a Uber e a meta e levanta alguns pontos das vantagens de utilizar monorepos como a facilidade de manutenções, os atomics commits, e como e citado no vídeo mesmo sendo em único repositório os projetos podem e devem ser divididos em pastas então com essa organização consegue-se aproveitar algumas vantagens do multi-repo sem perder as vantagens do monorepo, e um dos pontos mais citados é o reaproveitamento de códigos e bibliotecas, que facilita o desenvolvimento e acaba prevenindo alguns erros, pois com o reaproveitamento você não precisa toda vez estar fazendo as configurações então acaba que evita códigos duplicados, e trabalho desnecessário. Mas também nem tudo são maravilhas, então também e reforçado para tomar cuidado antes de adotar tanto o monorepo quanto o multi-repo o correto a se fazer é analisar os dois modelos para identificar o que se encaixa melhor com suas necessidades.

Dificuldades

Conclusões

Com esse card de leitura e vídeo onde nos é apresentado alguns pontos importantes sobre arquitetura de software, alguns modelos e como funcionam e também modelos como os monorepos e multi-repos conseguimos ter uma boa base teórica para quando formos desenvolver e também compreender como as coisas estão funcionando, em um mundo como hoje que fazer algo um uma IA é muito fácil apenas com alguns promps criamos um sistema básico ter esses conhecimentos sobre as arquiteturas e ferramentas disponíveis para utilizarmos e oque diferencia de um sistema bom, escalável e adaptável de algo feito por IA que nem ao certo sabemos como funciona.

Referencias

Artigo:
https://dev.to/indiamaraenes/principios-de-arquitetura-de-software-design-de-sistemas-escalaveis-e-flexiveis-885

Videos:

- MVC Descomplicado em 3 Minutos: Model, View e Controller Explicados!
- back end, front end, padrão camadas, mvc, rest
- Monorepo vs Multi-repo: Which Code Management Strategy is Right for You?
- Por que Monorepos Estão Mudando a Maneira como Desenvolvemos Software