Como estruturar um software?

Estruturar um software envolve a criação de uma arquitetura sólida que organize e distribua as diferentes partes do sistema de forma eficiente. Aqui estão algumas etapas e práticas para estruturar um software de maneira eficaz:

Compreender os Requisitos: Antes de começar a estruturar o software, é fundamental entender completamente os requisitos funcionais e não funcionais do sistema. Isso inclui entender as necessidades dos usuários, os objetivos do negócio e quaisquer restrições técnicas.

Definir a Arquitetura: Com base nos requisitos, defina a arquitetura do software. Isso inclui decidir sobre o modelo de arquitetura (por exemplo, arquitetura em camadas, arquitetura orientada a serviços, arquitetura de microsserviços), identificar os principais componentes do sistema e estabelecer as relações entre eles.

Decompor em Módulos ou Componentes: Divida o sistema em módulos ou componentes menores, cada um responsável por uma função específica. Isso ajuda a simplificar a complexidade do sistema e facilita o desenvolvimento e a manutenção.

Definir Interfaces: Defina interfaces claras entre os diferentes módulos ou componentes do sistema. Isso permite que os componentes se comuniquem de maneira eficiente e independente, facilitando a substituição e a extensão de componentes no futuro.

Organizar o Código: Dentro de cada módulo ou componente, organize o código de maneira lógica e coesa. Use padrões de design e princípios de programação para garantir que o código seja modular, coeso e de fácil compreensão.

Gerenciar Dependências: Gerencie cuidadosamente as dependências entre os diferentes componentes do sistema. Evite acoplamento excessivo e dependências cíclicas, e use técnicas como injeção de dependência para tornar o sistema mais flexível e fácil de testar.

Garantir a Qualidade do Código: Mantenha altos padrões de qualidade de código, seguindo boas práticas de codificação, realizando revisões de código regulares e escrevendo testes automatizados abrangentes.

Documentar o Sistema: Documente a arquitetura, os componentes, as interfaces e outras partes importantes do sistema. Isso ajuda os desenvolvedores a entenderem o sistema e facilita a integração de novos membros na equipe.

Revisão e Refatoração: Regularmente revise a estrutura do software e, se necessário, faça refatorações para melhorar a organização e a eficiência do código. Isso ajuda a manter o software flexível e adaptável às mudanças nos requisitos e nas tecnologias.

Iterar e Evoluir: Lembre-se de que a estrutura do software não é estática e pode precisar ser ajustada à medida que o sistema evolui. Esteja preparado para iterar na estrutura do software conforme necessário para atender aos requisitos em constante mudança do sistema.

♣ Quais são os componentes de software?

Bibliotecas: Conjuntos de funções ou classes que oferecem funcionalidades específicas, como manipulação de arquivos, processamento de dados, comunicação de rede, entre outros. As bibliotecas podem ser reutilizadas em vários projetos para evitar a duplicação de código e acelerar o desenvolvimento.

Frameworks: Conjuntos de bibliotecas e ferramentas que fornecem uma estrutura para o desenvolvimento de software em uma determinada linguagem de programação ou plataforma. Os frameworks podem incluir componentes para lidar com tarefas comuns, como gerenciamento de banco de dados, autenticação de usuário e renderização de interfaces de usuário.

Componentes de Interface de Usuário (UI): Elementos de interface de usuário, como botões, campos de texto, barras de progresso e caixas de diálogo, que são utilizados para criar a interface gráfica de um aplicativo. Esses componentes são frequentemente encapsulados em bibliotecas ou frameworks de UI para facilitar o desenvolvimento de interfaces consistentes e visualmente atraentes.

Serviços: Componentes que fornecem funcionalidades específicas por meio de uma interface de programação de aplicativos (API), geralmente acessada remotamente por outros componentes ou sistemas. Exemplos de serviços incluem serviços da web, APIs RESTful, serviços de autenticação e serviços de armazenamento em nuvem.

Módulos: Unidades de código independentes que realizam uma função específica dentro de um sistema maior. Os módulos são frequentemente organizados em uma estrutura hierárquica e podem ser agrupados em pacotes ou namespaces para facilitar a organização e o gerenciamento do código.

Plugins: Componentes opcionais que podem ser adicionados a um sistema para estender sua funcionalidade. Os plugins geralmente são carregados dinamicamente durante a execução do sistema e podem ser ativados ou desativados conforme necessário.

Drivers e Bibliotecas de Dispositivos: Componentes que fornecem acesso a dispositivos de hardware específicos, como placas de vídeo, impressoras, dispositivos de armazenamento e periféricos de entrada. Esses componentes são essenciais para permitir que o software interaja com o hardware subjacente do sistema.

♣ Quais são os tipos de arquitetura de software?

Arquitetura em Camadas: Neste modelo, o sistema é dividido em camadas horizontais, onde cada camada realiza uma função específica e se comunica com as camadas adjacentes. As camadas geralmente incluem a interface do usuário, lógica de negócios e acesso a dados. Esse tipo de arquitetura promove a modularidade e a reutilização de código.

Arquitetura Cliente-Servidor: Neste modelo, o sistema é dividido em dois componentes principais: o cliente, que solicita serviços ou recursos, e o servidor, que fornece esses serviços ou recursos em resposta às solicitações do cliente. A comunicação entre o cliente e o servidor geralmente é realizada por meio de uma rede.

Arquitetura Orientada a Serviços (SOA): Nesta abordagem, o sistema é composto por serviços independentes que podem ser acessados remotamente por outros componentes ou sistemas por meio de uma interface de programação de aplicativos (API). Os serviços são geralmente autônomos, independentes de plataforma e podem ser reutilizados em vários contextos.

Arquitetura Orientada a Microserviços: Semelhante à SOA, mas com um foco maior na granularidade e na independência dos serviços. Neste modelo, o sistema é dividido em pequenos serviços independentes, cada um executando uma função específica e comunicando-se por meio de protocolos leves, como HTTP ou mensagens assíncronas.

Arquitetura Monolítica: Neste modelo, todo o sistema é desenvolvido como um único código-fonte e implantado como um único artefato. Embora possa ser mais simples de desenvolver e implantar inicialmente, a arquitetura monolítica pode se tornar difícil de escalar e manter à medida que o sistema cresce em complexidade.

Arquitetura em Event-Driven: Nesta abordagem, os componentes do sistema comunicam-se entre si por meio de eventos assíncronos. Cada componente pode ser produtor, consumidor ou ambos de eventos, o que permite uma maior flexibilidade e escalabilidade do sistema.

Arquitetura Baseada em Pipeline: Neste modelo, o processamento de dados é organizado em uma série de estágios sequenciais, chamados de pipeline. Cada estágio do pipeline realiza uma operação específica nos dados e os passa para o próximo estágio para processamento adicional.

♣ Quais são as 3 camadas de software?

“MVC”

Camada de Apresentação (ou Interface do Usuário): Esta é a camada que interage diretamente com os usuários finais do sistema. Ela é responsável por apresentar informações ao usuário e capturar entradas do usuário. Geralmente, inclui elementos de interface do usuário, como formulários, botões, menus e outros controles. A camada de apresentação pode ser implementada em uma variedade de tecnologias, como HTML/CSS/JavaScript para aplicativos da web, frameworks de interface gráfica para aplicativos de desktop ou componentes de interface de usuário para aplicativos móveis.

Camada de Lógica de Negócios (ou Lógica de Aplicativo): Esta camada contém a lógica de processamento central do sistema. Ela é responsável por implementar as regras de negócios e as funcionalidades específicas do domínio da aplicação. A camada de lógica de negócios geralmente é independente da interface do usuário e dos detalhes de armazenamento de dados. Isso permite que ela seja reutilizada em diferentes interfaces de usuário e adaptada a diferentes tecnologias de armazenamento de dados.

Camada de Acesso a Dados (ou Persistência): Esta camada é responsável por interagir com o sistema de armazenamento de dados subjacente, seja um banco de dados relacional, um sistema de arquivos ou qualquer outro mecanismo de persistência de dados. Ela fornece funcionalidades para recuperar, armazenar, modificar e excluir dados do sistema de armazenamento de dados. A camada de acesso a dados encapsula as complexidades de interação com o armazenamento de dados e fornece uma interface limpa e consistente para a camada de lógica de negócios.

♣ Quais são os tipos de software?

Software de Sistema:

Sistemas Operacionais: Software que fornece uma interface entre o hardware do computador e os programas de aplicativos. Exemplos incluem Windows, macOS e Linux.

Drivers: Software que permite a comunicação entre o sistema operacional e dispositivos de hardware específicos, como placas gráficas, impressoras e dispositivos de armazenamento.

Firmware: Software embutido em dispositivos de hardware que fornece controle básico e funcionalidades de baixo nível. Exemplos incluem firmware de BIOS e firmware de dispositivos embarcados.

Software de Aplicativos:

Aplicativos de Produtividade: Software usado para realizar tarefas específicas de produtividade, como processamento de texto, planilhas, apresentações e gerenciamento de e-mail. Exemplos incluem Microsoft Office, Google Workspace e LibreOffice.

Software de Navegador da Web: Aplicativos usados para acessar e interagir com a internet, como navegadores da web (Chrome, Firefox, Safari).

Software de Design Gráfico e Edição de Mídia: Ferramentas usadas para criar e editar gráficos, imagens, vídeos e áudio. Exemplos incluem Adobe Photoshop, Illustrator, Premiere Pro e Audition.

Software de Desenvolvimento: Ferramentas usadas por desenvolvedores de software para escrever, depurar e testar código. Exemplos incluem IDEs (Ambientes de Desenvolvimento Integrado) como Visual Studio, IntelliJ IDEA, e editores de texto como Sublime Text e Visual Studio Code.

Software de Jogos: Aplicativos de entretenimento interativo, geralmente para entretenimento de usuários. Exemplos incluem jogos para PC, consoles de videogame e dispositivos móveis.

Software de Utilitário:

Antivírus e Software de Segurança: Aplicativos usados para proteger o sistema contra vírus, malware e outras ameaças à segurança.

Ferramentas de Backup e Recuperação de Dados: Software usado para fazer cópias de segurança de dados e restaurar dados perdidos em caso de falha do sistema.

Ferramentas de Otimização do Sistema: Aplicativos usados para melhorar o desempenho e a eficiência do sistema, como limpadores de registro, otimizadores de disco e ferramentas de gerenciamento de memória.

Software Embutido:

Embarcado em Dispositivos: Software incorporado em dispositivos eletrônicos específicos, como sistemas embarcados em carros, eletrodomésticos, dispositivos médicos, eletroeletrônicos, entre outros.

Software de Controle Industrial: Software usado para controlar processos industriais automatizados em fábricas e instalações industriais.

♣ O que é um Plano de Projeto de Software?

Objetivos do Projeto: Declaração clara dos objetivos do projeto, incluindo o que será entregue, quem será beneficiado e quais problemas serão resolvidos.

Escopo do Projeto: Definição clara do escopo do projeto, incluindo as funcionalidades e características que serão incluídas no software, bem como quaisquer limitações ou exclusões.

Cronograma: Estimativa de tempo para cada fase do projeto, incluindo datas de início e término, marcos importantes e entregáveis ​​principais. Isso pode ser representado em forma de cronograma de Gantt ou outra forma de visualização temporal.

Recursos: Identificação dos recursos necessários para o projeto, incluindo pessoal, hardware, software, infraestrutura e outras ferramentas e materiais necessários para completar o projeto.

Orçamento: Estimativa de custos para o projeto, incluindo custos de pessoal, custos de hardware/software, despesas gerais e quaisquer outros custos associados ao projeto.

Riscos e Mitigações: Identificação dos principais riscos que podem afetar o projeto e planos para mitigar ou lidar com esses riscos, caso ocorram.

Comunicação e Stakeholders: Detalhes sobre como a comunicação será gerenciada dentro da equipe do projeto, com as partes interessadas e com outras partes envolvidas. Isso pode incluir canais de comunicação, frequência de atualizações e responsabilidades dos membros da equipe.

Controles e Monitoramento: Procedimentos para monitorar o progresso do projeto, avaliar o desempenho em relação ao cronograma e ao orçamento, e implementar ajustes ou correções conforme necessário para garantir que o projeto permaneça no caminho certo.

♣ Quais são os 3 principais padrões da arquitetura web?

Modelo-Visão-Controlador (MVC):

O padrão MVC separa uma aplicação em três componentes principais:

Modelo: Representa a camada de dados, contendo a lógica de negócios e o acesso aos dados. Ele é responsável por gerenciar o estado da aplicação.

Visão: Responsável pela apresentação dos dados ao usuário final. É onde a interface do usuário é definida e gerada.

Controlador: Responsável por receber as solicitações do usuário, interagir com o modelo e selecionar a visão apropriada para ser apresentada ao usuário. Ele atua como o intermediário entre o modelo e a visão.

Modelo-Visão-ViewModel (MVVM):

O padrão MVVM é uma variação do MVC, com foco na separação entre a lógica de apresentação e a lógica de negócios.

Modelo: Mantém o estado e a lógica de negócios da aplicação, assim como no MVC.

Visão: Responsável pela apresentação dos dados ao usuário final, mas agora é vinculada a um objeto de visão de modelo (ViewModel).

ViewModel: Age como um adaptador entre a visão e o modelo, expondo os dados e comandos necessários para a visão e manipulando a interação do usuário para atualizar o modelo.

Arquitetura em Camadas:

A arquitetura em camadas divide uma aplicação em várias camadas com base nas responsabilidades funcionais.

Camada de Apresentação (Interface do Usuário): Responsável pela apresentação de informações ao usuário final. Esta camada inclui a interface do usuário e a lógica de apresentação.

Camada de Lógica de Negócios: Contém a lógica de negócios da aplicação, incluindo a validação de dados, processamento de regras de negócios e interação com o banco de dados.

Camada de Acesso a Dados: Responsável pela interação com o banco de dados ou outras fontes de dados. Ela gerencia a persistência e a recuperação de dados para a aplicação.