UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO

Engenharia de Software II
Glauco de Figueiredo Carneiro

Identificação de padrões arquiteturais no projeto crawl4ai: Um comparativo entre as estratégias de busca e identificação.

Carlos Daniel Lima de Gois
Felipe Osni Santos Moura
João Pedro Cardoso Arruda
Nicolas Matheus Ferreira de Jesus
Samuel Bastos Borges Pinho
Vinícius Vasconi Villas Boas Micska
Vitor Leonardo Sena de Lima

SÃO CRISTÓVÃO
08/11/2025

1. Introdução

Nosso objetivo é estudar as arquiteturas de software utilizando como base de pesquisa um projeto real e bem avaliado da plataforma GitHub. Nossa metodologia consistiu da escolha de 3 modelos de redes neurais da plataforma Hugging Face para rodar sobre o projeto procurando os indicativos das arquiteturas, e como forma de validar os resultados, um dos grupos fez uma leitura breve e identificação manual de padrões arquiteturais no repositório do projeto. Esperamos com esse trabalho, praticar os conceitos de Engenharia de Software, e também experimentar com os diferentes métodos e estratégias ao abordar um novo projeto.

2. Projeto escolhido

O projeto escolhido pela equipe foi o Crawl4AI (disponível em *[1]*) que consiste em síntese de um web scrapper para LLMs. Ele faz uma busca profunda e otimizada em diversos sites e retorna um texto markdown para melhor treinamento destas redes neurais. Escolhemos o projeto pelo interesse em explorar o contexto de web scrapping por ser um contexto emergente em computação.

3. Arquiteturas de software

Apresentaremos algumas das arquiteturas para embasar as observações feitas nas próximas sessões.

- **Modular**

  A arquitetura modular consiste na segmentação do código em partes especialistas, facilitando a manutenção e rastreabilidade. Busca-se maior independência possível entre cada um dos módulos, e essa separação permite a testagem mais adequada dos módulos, assim como torna o software mais escalável.

- **Microservices**

  Similar à arquitetura modular, mas se refere ao caso específico onde utilizamos serviços externos no software para simplificar determinadas partes do software aproveitando implementações já existentes.

- **Pipe & Filter (Tubos e Filtros)**

  Embora similar à arquitetura modular, a arquitetura Pipe & Filter (Tubos e Filtros) é um padrão de design focado em descrever como um fluxo de dados sequencial é processado. Ele é composto por "filtros" (componentes de processamento) que transformam os dados e "tubos" (conectores) que transportam os dados de um filtro para o próximo, geralmente em um fluxo unidirecional.

- **Microkernel**

  É um padrão que permite adicionar novas features a aplicação como plugins ao core da aplicação, fornecendo extensibilidade, além de isolamento e separação das features. Caracterizado por dois componentes principais: core system e plugin modules. A lógica da aplicação é dividida entre módulos independentes de plugins e o core system, promovendo a extensibilidade, flexibilidade e isolamento das features e lógicas de processamento da aplicação

- **Event Driven**

  A arquitetura Orientada a Eventos é um padrão arquitetural onde os componentes de software reagem a "eventos" (notificações de que algo aconteceu, como um "pedido criado"). Em vez de um serviço dar uma ordem direta a outro, ele apenas publica o evento em um barramento central (*event broker*). Outros serviços, interessados neste evento, consomem e executam suas tarefas de forma assíncrona, promovendo um desacoplamento total entre quem produz a informação e quem a consome.

4. Padrões de projeto

- **Adapter**
  - **Identificação:** Em aberto (Não identificado pelos modelos ou processo manual neste trecho do documento).

- **Facade**
  - **Identificação:** Em aberto (Não identificado pelos modelos ou processo manual neste trecho do documento).

- **Strategy**
  - **Identificação:** Em aberto (Não identificado pelos modelos ou processo manual neste trecho do documento).

- **Decorator**

  - **Identificação:** Detectado com sucesso parcial em `async_webcrawler.py` através da Análise de Código Fonte.
  - **Modelo: CodeLlama-7b**

- **Cache**

  - **Identificação:** Detectado com sucesso em `model_loader.py` devido ao uso do decorador `@lru_cache` através da Análise de Código Fonte.
  - **Modelo: CodeLlama-7b**

- **Factory Method**

  - **Identificação:** Detectado com sucesso em `model_loader.py`, onde as funções `load_*` encapsulam a criação de objetos, através da Análise de Código Fonte.
  - **Modelo: CodeLlama-7b**

- **Semaphore**

  - **Identificação:** Detectado com sucesso em `async_dispatcher.py` devido ao uso de `asyncio.Semaphore` através da Análise de Código Fonte.
  - **Modelo: CodeLlama-7b**

- **Padrão Hook (Plugin)**

  - **Identificação:** Identificado na Issue #1527, que descreve um sistema de "Hooks" para injetar código customizado, promovendo a extensibilidade do sistema, através da Análise de Issues.
  - **Modelo: CodeLlama-7b**

  - **Identificação:** Detectado com sucesso nas evidências do README. O modelo reconheceu o uso de módulos “hooks” e “plugins” que estendem o comportamento do sistema principal sem alterar o núcleo.
  - **Modelo: Qwen2.5-0.5B**

- **Concorrência (Race Condition)**

- **Identificação:** Um bug clássico de *race condition* foi identificado na Issue #1572, onde múltiplas tarefas disputavam um recurso único (aba de navegador), através da Análise de Issues.
- **Modelo: CodeLlama-7b**

- **Streaming / Statelessness**
  - **Identificação:** Identificados na Issue #1212 como princípios de design da API para garantir eficiência e escalabilidade, através da Análise de Issues.
  - **Modelo: CodeLlama-7b**

5. Modelos da Hugging Face

Como mencionado anteriormente, fizemos a escolha de 3 modelos no hugging faces para análise dos dados. A divisão das equipes com os respectivos modelos segue:

Equipe	Modelo
João Felipe	**MiniMax-M2** Descrição: Link: MiniMaxAI/MiniMax-M2 · Hugging Face
David Daniel	**Qwen2.5-Coder-0.5B-Instruct** Descrição: Link: https://huggingface.co/Qwen/Qwen2.5-0.5B
Samuel Vitor	**CodeBERT-base e CodeLlama-7b-hf** Descrição: Link: https://huggingface.co/microsoft/codebert-base Link: https://huggingface.co/codellama/CodeLlama-7b-hf
Vinícius Nicolas	*Não se aplica (Equipe responsável pela comparação manual)*

6. Estratégias

6.1. Análise das Issues do repositório

(Vitor Leonardo / Modelo: CodeLlama-7b)

Metodologia

A estratégia de análise de issues foi um processo iterativo para superar desafios de amostragem e hardware.

1. **Abordagem Inicial (Rejeitada):** A análise de issues recentes abertas foi descartada por gerar um **viés de amostragem**, ignorando discussões arquiteturais mais antigas e já fechadas.
2. **Abordagem Intermediária (Rejeitada):** Uma tentativa de análise "sob demanda" falhou devido a um erro de ambiente (`NameError`), pois o script não era portátil.
3. **Metodologia Final (Dividir para Conquistar):** Para superar o estouro de memória da GPU (`CUDA out of memory`) ao tentar sintetizar 55 issues de uma só vez, foi adotada uma abordagem multifásica:
    - **Fase 1 (Agrupamento Temático):** Um prompt leve (contendo apenas títulos e padrões) foi enviado ao CodeLlama para agrupar as 55 issues em temas macro, evitando o estouro de memória.
    - **Fase 2 (Resumo por Tema):** O CodeLlama gerou resumos focados para cada tema, processando apenas o contexto das issues daquele grupo.
    - **Fase 3 (Síntese Final):** Os resumos temáticos (curtos e densos) foram combinados para gerar o relatório consolidado de forma gerenciável.

Esta metodologia híbrida, combinando curadoria humana na seleção de dados com uma síntese em etapas pela IA, foi essencial para gerenciar os recursos computacionais.

(Nicolas Ferreira / feito a mão)

Metodologia

A metodologia para a análise do crawl4ai sem a utilização do LLM consistiu na leitura pura e entendimento lógico do código apresentando pelo projeto escolhido.

- **Início:** O primeiro passo antes de partir para a leitura propriamente dita do código foi a pesquisa e entendimento de padrões arquitetônicos de projetos de software e fazendo pequenas anotações para a utilização mais a frente.

- **Entendimento do Projeto:** Com as anotações feitas, o passo inicial feito foi ler documentos que normalmente apresentam e falam mais sobre o projeto, ou seja o README.md, e entendo melhor sobre o projeto e vendo sua organização com os arquivos disponibilizados, foi feita a investigação de cada pasta adicionada a página inicial. Tendo poucos arquivos que continham código, não levando a pasta principal crawl4ai.
- **Leitura do Código:** Foi feita uma busca dentro da pasta crawl4ai de códigos para a retenção maior de informações e escolhas de possíveis padrões, focando principalmente em arquivos .py “soltos” dentro dessa pasta.
- **Final:** O último passo feito foi ver quais arquiteturas tinham fortes indícios e elaborar argumentação com partes do código e sua distribuição de tarefas.

(Carlos Daniel / Modelo: Qwen2.5-0.5B)

Metodologia

A estratégia adotada concentrou-se na identificação de **padrões arquiteturais** por meio de evidências textuais (documentação e estrutura do repositório).

Abordagem Inicial (Rejeitada): A primeira tentativa consistiu em utilizar diretamente o modelo **Qwen 2.5 Coder** no ambiente Google Colab para gerar descrições detalhadas dos padrões arquiteturais a partir do conteúdo integral do projeto. Essa abordagem mostrou-se inviável devido à limitação de memória e tempo de execução do Colab, que resultava em travamentos durante o carregamento de modelos maiores.

Ajuste da Estratégia:

Optou-se então por uma **abordagem textual e iterativa**, focando na leitura automática do **README.md** e da **árvore de diretórios** do projeto. Essa estratégia reduziu o volume de dados processados, tornando possível a execução integral do modelo no Colab em CPU, sem necessidade de GPU. O prompt foi configurado para instruir o Qwen a responder **exclusivamente em formato JSON**, garantindo uma saída estruturada e objetiva.

Fase de Execução:
O modelo foi utilizado através de um *pipeline* de geração textual. O script executou três etapas principais:

Coleta de Evidências: Leitura do arquivo README.md e mapeamento da estrutura de pastas do repositório clonado.

Análise com Qwen: Envio dessas evidências ao modelo para inferir padrões arquiteturais (máximo de cinco por execução).

Pós-Processamento (Parsing): Interpretação da resposta do modelo em formato JSON. Quando o modelo não produziu uma saída válida, o código ativou automaticamente um **modo heurístico**, reconstruindo o resultado com base em palavras-chave encontradas nas evidências (“webhook”, “plugin”, “docker” etc.).

6.2. Análise do código fonte

(Vitor Leonardo / Modelo: CodeLlama-7b)

Metodologia

Para analisar o código-fonte, foi desenvolvida uma metodologia de "Fato-Interpretação" em duas etapas para aumentar a confiabilidade e reduzir a "alucinação" do modelo.

1. **Preparação:** O esqueleto do código (apenas `import`, `class`, `def` e decoradores) foi extraído de cada arquivo-alvo.
2. **Passo A (Busca de Fatos):** Um prompt detalhado foi enviado ao CodeLlama, pedindo que respondesse apenas "Sim" ou "Não" para 13 fatos concretos relacionados a padrões (ex: "FATO 3 (Singleton): O código tem lógica para garantir que apenas uma instância de uma classe seja criada?").
3. **Passo B (Interpretação dos Fatos):** A resposta do Passo A ("Relatório de Fatos") foi usada como contexto para um segundo prompt. Este pedia ao modelo que agisse como um "Arquiteto de Software" e listasse apenas os padrões confirmados pelos fatos marcados como "Sim".

O objetivo era forçar o modelo a basear suas conclusões em evidências estruturais objetivas, em vez de semelhanças textuais fracas.

(Nicolas Ferreira / feito a mão)

Metodologia

Para fazer essa análise mais profunda o foco foi redirecionado para partes mais específicas do projeto, como os módulos async_webcraler, async_config, browser_adaptive e etc.

- **Escolha:** Vendo a divisão de vários documentos que são importantes, foi feita uma divisão rápida de quais arquivos deveriam ter mais cuidado para que o foco e evolução do trabalho ficasse dentro do prazo. Com isso as classes que ficaram mais em foco foram a content_scraping_strategy, content_filter_Strategy, markdown_generation_strategy, extraction_strategy, proxy_strategy, chunking_strategy e, como dito antes, async_webcraler, async_config e browser_adaptive.
- **Definição do arquivo central:** Dado todo o contexto e arquivos de focos, a ser feita a leitura foi notado que um dos arquivos tipo um peso maior, pelo fato de servir como um núcleo central de todo o projeto, esse foi o Async_webcrawler, principalmente com sua classe AsyncWebCrawler.
- **Análise final:** Com o núcleo definido, a identificação de mais arquiteturas teve sua via facilitada, pois como uma tinha dependência de várias outras ficou mais fácil de identificar cada uma de suas funções e consequentemente os tipos de projetos arquiteturais que demonstraram ser.

6.3. Extração de classes e funções por scripting

7. Resultados

(Vitor Leonardo / Modelo: CodeLlama-7b)

Resultados da Análise de Issues:

A análise das 55 issues selecionadas revelou uma arquitetura moderna, focada em extensibilidade, performance e separação de responsabilidades.

- **Microsserviços:** Identificada na Issue #1550, que declara explicitamente o objetivo de criar uma arquitetura de microsserviços.
- **Clean Architecture:** Mencionada textualmente na Issue #1525 durante uma refatoração do transporte de comunicação.
- **Containers / Orquestração:** Identificada na Issue #1274, que propõe o uso de *Devcontainers* para criar um ambiente de desenvolvimento padronizado e isolado.
- **API Gateway:** Inferido pela IA na Issue #1513, que descrevia a criação de um ponto de entrada único para gerenciar requisições.

(Nicolas Ferreira / feito a mão)

Resultado da Busca e Análise dos Códigos:

Olhando e lendo o código e estabelecendo uma lógica para ele, juntamente de observar suas dependências e importações, foi possível identificar que o projeto `crawl4ai` adota uma combinação de arquiteturas bem definidas, voltadas para modularidade, extensibilidade e processamento assíncrono. Com três principais estilos arquiteturais:

- **Arquitetura Microkernel**
  - **a.** O núcleo do sistema é o AsyncWebCrawler, que centraliza o controle do fluxo de execução, cache, logs e chamadas assíncronas. Ao seu redor, há diversos módulos externos e configuráveis, chamados de "estratégias" (como ScrapingStrategy, ExtractionStrategy, ProxyStrategy, etc.), que estendem o comportamento padrão sem modificar o código do núcleo.

- **Arquitetura em Camadas**
  - a. Outra característica observada é a organização em camadas bem definidas, visível tanto na estrutura do diretório quanto no fluxo lógico do código:
    - i. Camada de Interface: responsável por interação e inicialização (cli.py, hub.py).
    - ii. Camada de Aplicação/Core: controle principal do processo de crawling (async_webcrawler.py, adaptive_crawler.py).
    - iii. Camada de Domínio: define as regras e estratégias de negócio (content_filter_strategy.py, markdown_generation_strategy.py, extraction_strategy.py).
    - iv. Camada de Infraestrutura: fornece suporte técnico, como log, cache e banco assíncrono (async_logger.py, async_database.py, config.py).

- **Arquitetura Pipe and Filter**
  - **a.** Dentro do AsyncWebCrawler, especialmente no método arun(), é possível perceber uma sequência encadeada de etapas:
    - i. o HTML é baixado, limpo, filtrado, convertido em Markdown, dividido em blocos (chunks) e finalmente analisado.
    - ii. Cada etapa processa a saída da anterior e a entrega para a próxima, seguindo o padrão Pipe and Filter.
    - iii. Esse modelo facilita a extensão do pipeline, pois novas transformações podem ser adicionadas sem quebrar o fluxo existente.
    - iv. Além disso, ele combina naturalmente com a programação assíncrona usada no projeto (async/await), garantindo eficiência e paralelismo no processamento.

(Vinícius Vasconi / feito a mão)

Resultado da Busca e Análise dos Códigos:

A estratégia utilizada foi principalmente a análise das dependências do software, e busca pelas bibliotecas para identificar onde e como eram utilizadas.

Em um primeiro momento, retornei ao primeiro commit para analisar os arquivos em uma versão simplificada. No primeiro commit, havia o arquivo “requirements.txt”, de onde pude extrair informações sobre quais bibliotecas estavam sendo utilizadas.

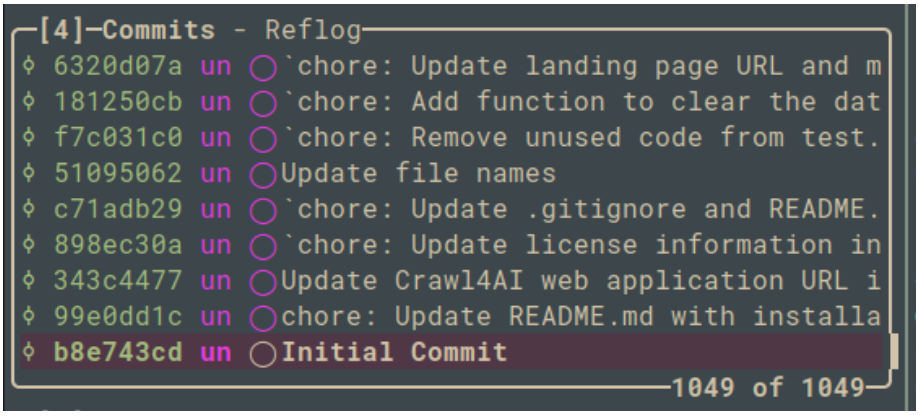

Figura 1: Tabela de commits

Especulação inicial e possíveis arquiteturas associadas às dependências:

1. **Pydantic:** Biblioteca em python utilizada para melhor estruturar os dados utilizados no projeto. Pela sua natureza de encapsulamento é muitas vezes utilizadas em arquiteturas que dependem de comunicação entre camadas, pois garante a correta conversão e validação de tipos.
2. **FastAPI:** Utilizado na criação de APIs com Python e foi pensado para ser compatível com o protocolo ASGI (Asynchronous Server Gateway Interface). Levando em consideração o caráter do projeto podemos inferir algumas arquiteturas, como microserviços e a própria relação com a arquitetura de camadas possivelmente presente dada a biblioteca anterior.
3. **aioSQLite:** Interface para comunicação assíncrona com banco de dados, indica arquitetura baseada em eventos (EDA), que não podem fazer chamadas bloqueantes, pois caso ocorram o event loop fica travado esperando a chamada terminar, o que prejudica o funcionamento.

Minha segunda estratégia foi utilizar um script para extrair todas as classes e funções definidas nos arquivos. Utilizando a biblioteca OS do python e o AST para análise do código, fiz a conversão para mermaid mas não consegui exibir o diagrama pois tive problemas com o compilador. A intenção era visualizar a relação entre as classes para entender possíveis padrões de projeto e padrões arquiteturais.

Essa estratégia usando scripting se mostrou ineficiente pois o resultado tinha muitas linhas e era muito difícil de acompanhar o que estava relacionado com o que e como.

Voltando para a primeira estratégia que rendeu frutos, apliquei no repositório no estado atual, mas dessa vez busquei as dependências no dockerfile, onde tive duas bibliotecas revelaram funcionalidades que chamaram atenção:

- **PDF parsing:** No contexto de web scraping, ter uma biblioteca para manipulação de imagens e pdfs, envolvendo compressão, dá fortes indícios da arquitetura de pipe & filter, já que a imagem deve ser identificada, depois comprimida, depois processada, indicando a ideia da arquitetura.
- **REDIS local database:** Identifiquei em uma das pastas a utilização do banco de dados local **REDIS**, fiz uma pesquisa e encontrei que esse tipo de banco é comumente utilizado como cache, pois é um banco local de baixa latência. Dada essa natureza, é um forte indicativo da utilização de um padrão proxy.

Por fim, o indício de uso da arquitetura de *microservices* vem da leitura do arquivo `env.txt` no diretório root do projeto, onde linhas como *“OPENAI_API_KEY = "YOUR_OPENAI_API""* indicam a possibilidade de configurar a API de uma LLM.

(Vitor Leonardo / Modelo: CodeLlama-7b)

Resultados da Análise de Issues:

- **Concorrência (Race Condition):** Um bug clássico de *race condition* foi identificado na Issue #1572, onde múltiplas tarefas disputavam um recurso único (aba de navegador).
- **Streaming / Statelessness:** Identificados na Issue #1212 como princípios de design da API para garantir eficiência e escalabilidade.
- **Padrão Hook (Plugin):** Identificado na Issue #1527, que descreve um sistema de "Hooks" para injetar código customizado, promovendo a extensibilidade do sistema.

Resultados da Análise de Código Fonte:

- **Cache:** Detectado com sucesso em `model_loader.py` devido ao uso do decorador `@lru_cache`.
- **Factory Method:** Detectado com sucesso em `model_loader.py`, onde as funções `load_*` encapsulam a criação de objetos.

- **Semaphore:** Detectado com sucesso em `async_dispatcher.py` devido ao uso de `asyncio.Semaphore`.
- **Decorator:** Detectado com sucesso parcial em `async_webcrawler.py`.

(Nicolas Ferreira / feito a mão)

Resultado da Busca e Análise dos Códigos:

Durante a leitura do código também foi possível identificar alguns padrões de projeto como:

- Strategy
- Decorator
- Factory
- Observer
- Template Method

(Vitor Leonardo / Modelo: CodeLlama-7b)

Limitações da Estratégia de Issues:

- **Viés de Amostragem:** A metodologia é altamente sensível à seleção de dados. Uma análise ingênua focada apenas em issues recentes/abertas falhou em identificar padrões arquiteturais-chave definidos no início do projeto.
- **Limitação de Memória (Hardware):** A principal limitação técnica foi o estouro de memória da GPU (`CUDA out of memory`) ao tentar sintetizar o contexto combinado de 55 análises. Isso exigiu a criação da metodologia "Dividir para Conquistar".

Limitações da Estratégia de Código Fonte:

A metodologia "Fato-Interpretação" expôs duas fraquezas significativas no modelo CodeLlama:

- **Falha Crítica (Alucinação Persistente):** O modelo falhou em seguir as instruções. No arquivo `browser_manager.py`, ele respondeu "Não" a todos os 13 fatos (incluindo o de Singleton) no Passo A. No entanto, no Passo B, ele ignorou sua própria análise e **afirmou ter encontrado o padrão Singleton**, alucinando uma detecção totalmente infundada.

- **Inconsistência:** O modelo pode não seguir suas próprias conclusões. Em `async_webcrawler.py`, ele respondeu "Sim" para "Factory Method" no Passo A, mas misteriosamente ignorou essa informação e não a listou no relatório final

do Passo B.

8. Comparação dos modelos

9. Conclusão

10. Histórico de atividades

Aluno	Atividade
Felipe Osni Santos Moura - 202100011397	
João Pedro Cardoso Arruda	
Nicolas Matheus Ferreira de Jesus - 202200014444	Responsável por uma das análises manuais do projeto crawl4ai, identificando arquiteturas de projeto como microkernel, piper and filter, camadas e modular. 1)Fiz busca e pesquisa para entender melhor as arquiteturas de projeto; 2)Análise de arquivos dentro do projeto, focando principalmente na pasta crawl4ai e no arquivo async_webcrawler 3)Leitura de todos os arquivos importados pelo async_webcrawler e raciocínio de seu funcionamento 4)Juntando todos indícios apresentados e com a compreensão dos projetos arquitetônicos foi possível fazer a identificação das arquiteturas
Samuel Bastos Borges Pinho - 202300083945	

Vinícius Vasconi Villas Boas Micska - 202300038940	1) Dei o ponto de partida para trabalhar na análise manual, criando um arquivo para documentar os achados que foi de grande importância na organização das ideias do documento 2) Fiz a análise das dependências do software com base nos requirements e no docker file 3) Fiz uma breve pesquisa sobre as arquiteturas para apresentação da base teórica 4) Organizei a versão inicial do documento de apresentação.
Vitor Leonardo Sena de Lima - 202200014622	Responsável por duas grandes análises usando o modelo CodeLlama 7B: 1) Analisei as discussões históricas do projeto no GitHub (Issues) para identificar a evolução da arquitetura e as decisões de design; 2) Analisei o código-fonte para detectar padrões de projeto (como Singleton e Factory), desenvolvendo métodos especializados para garantir a precisão do modelo.
Carlos Daniel Lima de Gois 202200078746	

11. Referências

11.1. *[1] Link para o repositório do projeto: https://github.com/unclecode/crawl4ai*

11.2. *[2] Link para o vídeo: https://drive.google.com/drive/folders/1SjETTMOI9PYRgJS8b1KvXLs6LUR6Iy9V?usp=sharing*

11.3. *[3] Referência de arquitetura Modular:* ***O que é arquitetura modular e por que ela é importante / Introdução Versão 1.0 – Arquitetura Modular – Manual do Arquiteto de Software****. Disponível em: <https://modular.arquiteturadesoftware.online/o-que-e-arquitetura-modular-e-por-que-ela-e-importante-introducao-versao-1-0/>. Acesso em: 8 nov. 2025.*

11.4. *[4] Referência de arquitetura Microkernel: LAGO, J.* ***Padrões de Arquitetura de Software — Parte III****. Disponível em: <https://jeziellago.medium.com/padr%C3%B5es-de-arquitetura-de-software-parte-iii-9e2fae850b5>. Acesso em: 12 nov. 2025.*

11.5. *[5] Referência de arquiteturas variadas:* ***Padrões arquiteturais: arquitetura de software descomplicada | Alura****. Disponível em: <https://www.alura.com.br/artigos/padroes-arquiteturais-arquitetura-software-descomplicada?srsltid=AfmBOoofBpVGE0OA4LmfiY8qRQ20LUbMhsUKmEtLM2xZUVBj7HCp4ul6>. Acesso em: 12 nov. 2025.*

11.6. *[6] Referência de arquiteturas variadas: SALMON, A.* ***Entenda tudo sobre os tipos de arquiteturas de software****. Disponível em: <https://truechange.com.br/blog/tipos-de-arquiteturas-de-software/>.*