



Especificações de Requisitos

1. Análise do Sistema: identificar os problemas e estabelecer requisitos de info.
 2. Projeto do Sistema: criar especificações do projeto
 3. Programação: traduzir especificações em código de software do programa
 4. Teste: avalia se o sistema é capaz de produzir os resultados certos
 5. Produção: mudar do velho sistema para o novo
 6. Mantenção: opera e avalia o sistema

Métodos

- 1. Tradicional: desenvolvimento organizado em estágios formais
- 2. Protótipos: processo iterativo de construção de sistemas interativos
- 3. Ágil: divide projetos grandes em sub-projetos pequenos

Ato: Entidade externa (pessoa ou não) que interage com o sistema

User Story: definição de alto-nível de um requisito (de interação)

"It's a [user], y want [function], no that [value]"
WHO? WHAT? WHY?

Requisitos y complementares:

1. Regra do Negócio: define ou restringe um aspecto do negócio, com a intenção de afirmar a estrutura ou influenciar o comportamento do negócio.
 2. Requisito Técnico: preocupa-se com os aspectos técnicos que o sistema tem de cumprir, como questões de desempenho, fiabilidade e disponibilidade.
 3. Restrição: limita o grau de liberdade na procura por uma solução.

Arquitetura de Informação

- É o campo que soluciona a sobrecarga de informação e a multiplicação de canais de acesso
 - Toma em consideração os espaços ricos em informação e o seu design para maximizar a eficácia, qualquer que seja o objetivo do utilizador
- (1) O princípio que organiza os padrões inherentes nos dados, tornando o complexo claro
 - (2) A pessoa que cria a estrutura ou mapa de informação que permite aos outros encontrar o seu caminho pessoal para o conhecimento
 - (3) O foco na clareza, entendimento/compreensão e ciência da organização da informação
 - (4) O design estrutural de ambientes de informação partilhada
 - (5) A história da organização, rotulagem, pesquisa e sistemas de navegação nos ecossistemas digitais, físicos e mistos
 - (6) A arte e ciência de formar produtos de informação e experiência para apoiar usabilidade, facilidade de pesquisa e compreensão
 - (7) O foco em trazer os princípios do design e da arquitetura para o ambiente digital

- Objetivos**
- Correto os objetos de informação e utilizadores peristos
 - Identificar conceitos e caminhos para acesso e navegação
 - Criar ferramentas e sistemas para as pessoas organizarem info.
 - Conectar vários tipos de informação, aplicações, plataformas e canais

Relacionados: Engenharia de Usabilidade; Ciência de Informação;
Engenharia de Fatores Humanos; Design Visual; Design de Interacção

- Sistemas**
- 1. Organização: como a informação é categorizada
 - 2. Representação: como a informação é representada
 - 3. Navegação: como se move entre/através da informação
 - 4. Pesquisa: como a informação é procurada

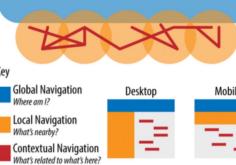
Esquema de Organização: define as características partilhadas dos itens e o seu agrupamento

- | | | | |
|--------------|---|-----------------|--|
| Exato | <ul style="list-style-type: none"> - Alfabético - Cronológico - Geográfico | Ampliúdo | <ul style="list-style-type: none"> - Categórico - Hierárquico - Dependente da audiência |
|--------------|---|-----------------|--|

• Etiquetas podem ser textuais ou icónicas

Sistema de Navegação:

1. Onde estou?
2. O que posso fazer?
3. Para onde posso ir?



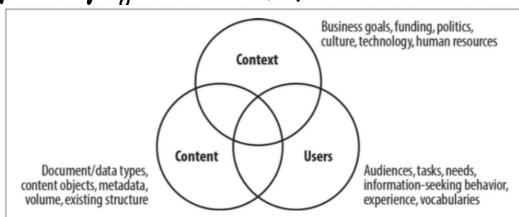
- Tipos
- Global: presente em todas as páginas
 - Local: subseção imediata
 - Contextual: associação de conceitos existentes

Dutos: mapa do site; caminho ("breadcrumb"); pesquisa
Aranhadas: Personalização; Customização; Visualização; Navegação Social

- A pesquisa é um mecanismo de navegação avançada

Processo de A.I.:

1. Pesquisa: inventário de conteúdo & ordenação de cátions
2. Design: mapa do site & wireframes
3. Implementação: prototipagem em papel & teste de usabilidade
4. Avaliação



Sitemap: mostra a relação entre elementos de informação (páginas) e pode ser usado para retratar sistemas de organização, navegação e rotulagem

Wireframe: caracteriza como é que uma página individual ou template deve parecer, de uma perspectiva estrutural e arquitetônica

Wireflow: combina wireframes e diagramas de fluxo para fornecer uma visão ao nível da ideia de organização das páginas (estrutura) e documenta fluxos complexos e trajeto do utilizador (fluxo) - interações

Especificação da Base de Dados

Modelo Conceptual do Domínio: contém a identificação e descrição das entidades do domínio e das relações entre elas

Modelo Conceptual dos Dados: obtido usando um diagrama de classes UML que contém as classe, associações, multiplicidade e atributos
↓
atributos, associações e restrições

Como obter o Modelo Conceptual?

1. Identificar tipos de entidades
2. Identificar relações
3. Identificar atributos
4. Aplicar convenções de nomes

Orientações:

1. explorar conceitos do domínio e as suas relações
2. identificar entidades de dados e atribuir-lhes atributos

Esquema Relacional: derivado do Modelo Conceptual (de Dados), inclui esquemas de relações, atributos, domínios, chaves primárias, chaves externas e regras de integridade

Como validar? Identificar todas as dependências funcionais e normalizar todos os esquemas de relações (se necessário)

table-one (id, atributo1 → Table2, atributo2 UK NN)

Rule 1	Classes are mapped into relation schemas
Rule 2	Class attributes are mapped to attributes of relations.
Rule 3	Operations of classes are generally not mapped. They can nevertheless be mapped to <i>stored procedures</i> , stored and executed in the global context of the database involved.
Rule 4	Objects are mapped into tuples of one or more relations.
Rule 5	Each object is uniquely identified. If the identification of an object is defined explicitly by the OID (<i>object identifier</i>) stereotype, associated with one or more attributes, this attribute is mapped to primary key in the relation schema. Otherwise, we assume implicitly that the corresponding primary key is derived from a new attribute with the name of the relation and common suffix (e.g. "PK", "ID").
Rule 6:	The mapping of many-to-many associations involves the creation of a new relation schema, with attributes acting together as primary key, and individually as foreign key for each of the schemas derived from the classes involved.
Rule 7:	The mapping of one-to-many associations involves the introduction, in the relation schema corresponding to the class that has the constraint "many", of a foreign key attribute for the other schema.
Rule 8:	The mapping of one-to-one associations has in general two solutions. The first corresponds to the fusion of the attributes of the classes involved in one common schema. The second solution is to map each of the classes in the corresponding schema and choose one of the schemas as the most suitable for the introduction of a foreign key attribute for the other schema. This attribute should also be defined as unique within that schema.
Rule 9:	Association navigability in general has no impact on the mapping process. The exception lies in one-to-one associations, when they are complemented with navigation cues it helps in the selection of the schema that should include the foreign key attribute.
Rule 10:	Aggregation and composition associations have a minimal impact on the mapping process, which may correspond to the definition of constraints cascade ("CASCADE") in changing operations and/or removal of tuples.
Rule 11:	The mapping of generalization associations in general presents three solutions. The first solution consists in crushing the hierarchy of classes in a single schema corresponding to the original superclass. This solution is appropriate when there is a significant distinction in the structure of sub-classes and/or when the semantics of their identification is not strong. The second solution is to consider only schemas corresponding to the sub-classes and duplicate the attributes of the super-class in these schemas; in particular it works if the super-class is defined as abstract. The third solution is to consider all the schemas corresponding to all classes of the hierarchy, resulting in a mesh of connected schemas and maintained at the expense of referential integrity rules. This solution has the advantage of avoiding duplication of information among different schemas, but suggests a dispersion of information by various schemas, and might involve a performance penalty in query operations or updating of data by requiring the execution of various join operations (i.e. "JOIN") and/or validation of referential integrity.

Dependência: raiz de vários problemas associados com esquemas relacionais

(identificar) → dependências funcionais → ajustar → decomposição

*• Se uma relação está numa forma normal, então uns problemas
não evitados/minimizados*

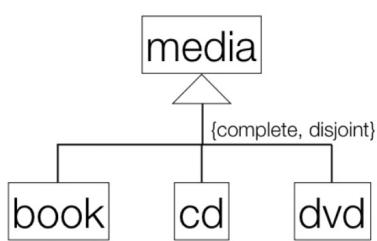
BCNF > 3NF

BCNF: para qualquer dependência funcional ...
ou (1) $X \rightarrow Y$ é trivial ($Y \subseteq X$)
(2) X é uma super-chave da relação

*3
BC*

Superclass approach

```
media(id, type CHK {book, cd, dvd} ...)
```



ER approach

```
media(id, ...)  
book(id->media, ...)  
cd(id->media, ...)  
dvd(id->media, ...)
```

Object Oriented

```
book(id, [media attributes], ...)  
cd(id, [media attributes], ...)  
dvd(id, [media attributes], ...)
```

Índices

Índices: estruturas de dados secundárias usadas para melhorar o acesso aos dados

Índices B-Tree: usam uma estrutura de dados tipo árvore que mantém os dados ordenados e permite a pesquisa, ordenação e pesquisa num intervalo/alcance em tempo logarítmico

Índices Hash: usam uma função hash para mapear chaves para valores — Nó não considerado quando um operador de igualdade é usado (nem ordenação nem intervalos/alcances)

CREATE INDEX name ON Table (a, b)

funciona quando se pesquisa por (a, b) ou (a)

Índices Únicos: fornecem a unicidade dos valores da(s) coluna(s) — não são permitidas múltiplas linhas da Tabela com valores de índice iguais — criados automaticamente para colunas PK e UK

Agrupamento: reordenação física dos dados no disco baseada na informação do índice — vantajoso quando múltiplos registos são lidos em conjunto e um índice pode agrupá-los (irrelevante para acessos aleatórios)

Cardinalidade: unicidade dos valores de dados contidos numa coluna particular — menor cardinalidade, mais valores duplicados numa coluna

"Full Text Search"

• É necessário indexar cada palavra individualmente

Passo-Chave: definir o que é um documento no sistema de pesquisa e que informação é relevante

Tipo Tívector: os textos são separados em "tokens", uma representação normalizada de palavras — um tívector armazena "tokens" distintos, bem "stop words" e com o registo do número de cada "token"
↑ PESQUISA SOBRE

Tipo Tquery: representação de pesquisas (plain-totquery)

@@: operador que verifica se um tívector corresponde a uma tquery

setweight: adiciona um peso (importância de 'A' a 'D') a um determinado tívector

b-rank[-cd]: retornam uma pontuação para cada linha retornada para uma determinada correspondência entre uma tquery e um tívector

• Deve adicionar-se uma coluna às tabelas em que vai ser realizado FTS que contenha os valores tívector de cada linha, atualizadas automaticamente com um gatilho — coluna "search"

Índices GIN: melhores para dados estáticos — pesquisa rápida

Índices GIST: melhores para dados dinâmicos — atualização rápida

Gatilhos e FDU

- permitem para forçar o cumprimento de regras de integridade
- São identificadas e descritas para serem confiáveis pelo servidor BD
- Podem receber valores de tipos simples ou compostos
- Podem retornar valores de tipos simples ou compostos

Funções Definidas pelo Utilizador: estendem a funcionalidade do servidor

- reduzem o número de viagens entre a aplicação e o servidor
 - aumentam o desempenho da aplicação (pré-compilados)
 - podem ser reutilizados em várias aplicações
-
- desenvolvimento lento de software (conhecimento especializado)
 - dificultam gestão de versões e debug
 - código menos portátil para outros SGBD

Gatilhos: regras evento-condição-ação

Evento: alteração à base de dados que ativa o gatilho

Condição: query ou teste que corre quando o gatilho é activado

Ação: procedimento que é executado quando a condição é verdadeira

→ antes, depois ou em vez do evento

→ pode referir-se aos valores novos ou antigos

cada registo modificado
realizada uma vez para **ou** todos os registos modificados

Transações

- As transações englobam múltiplos passos numa única "operação tudo-ou-nada", garantindo a integridade dos dados com acessos concorrentes

Propriedades ACID:

1. Atomicidade: múltiplas operações são tratadas como uma unidade indivisível
 2. Consistência: a BD move-se entre estados consistentes
 3. Isolamento: o resultado de múltiplas transações executadas concorrentemente é o mesmo se cada transação fosse executada isoladamente
 4. Durabilidade: os efeitos de uma transação "committed" devem persistir na base de dados
- Uma transação é um conjunto de operações consideradas uma só
 - Uma transação ou nenhuma falha na sua globalidade
 - Os passos intermédios não são visíveis para outras transações concorrentes
 - Uma transação move a base de dados entre estados consistentes

- Gestão de Transações
- Recuperação de Falhas
- Controlo de Concorrência

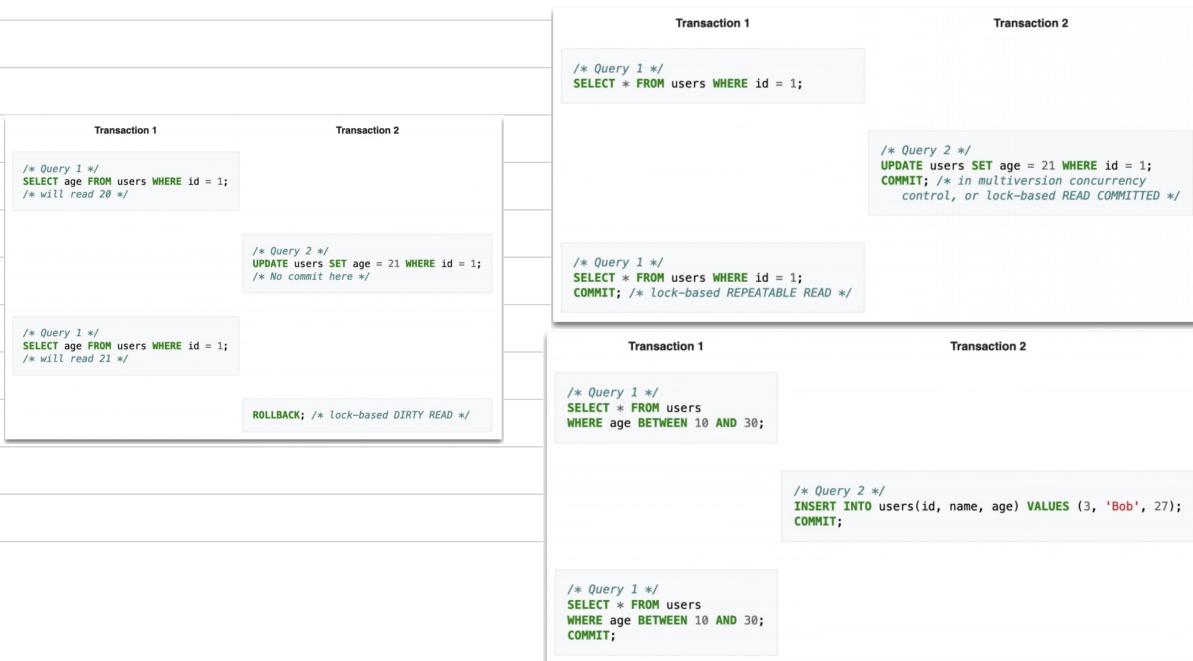
Problemas de Concorrência

Lertras Suja: uma transação lê dados escritos por uma transação corrente uncommitted

Lertras Não-Repetitivas: uma transação re-lê dados e descobre que os dados foram modificados por outra transação — a mesma query retorna resultados diferentes durante a transação

Lertras Fantasma: uma transação re-excuta uma query e descobre que os resultados mudaram por outra transação — os valores não mudaram mas linhas diferentes são retornadas

Anomalia de Serialização: o resultado de fazer "commit" de um grupo de transações é inconsistente com todas as ordenações possíveis de correr essas transações, numa de cada vez



Níveis de Isolamento

"Read Uncommitted": os registros ainda não "committed" podem ser lidos
COMPORTA-SE como

"Read Committed": as transações só vêm dados "committed" no momento da leitura, nunca vêm dados alterados e não "committed" por outras transações concorrentes - bloqueios longos de escrita e bloqueios curtos de leitura DEFAULT EM POSTGRESQL

"Repeatable Read": as transações só vêm dados "committed" antes do início da transação, nunca vêm dados não "committed" ou alterações "committed" durante a execução da transação por transações concorrentes - bloqueios longos de escrita e de leitura

DEFAULT EM SQL STANDARD

"Serializable": as transações só vêm dados "committed" antes da transação começar e nunca vêm dados ou alterações não "committed"

- A restrição dos níveis de isolamento define a concorrência dos acessos
- O ideal é o menor nível de isolamento que garante consistência dos dados
- As transações devem ser declaradas READ ONLY quando possível

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

Aplicações Web

Internet: rede de comunicação global feita de redes físicas interligadas – uma rede de redes

World Wide Web: sistema distribuído de informações que corre sobre a Internet

Aplicação Web: sistema de software, baseado em padrões e tecnologias web, acessível através de um navegador web

"Packet Switching": método em que os dados são divididos em pequenos pacotes e enviados separadamente

Routing: equipamento de hardware central usado para ligar redes através de diferentes tecnologias físicas

Internet Protocol (IP): oferece uma rede virtual, escondendo as redes físicas subjacentes – sistema de endereçamento (endereços IP) e estrutura de datagramas (pacotes)

Transmission Control Protocol (TCP): oferece uma entrega de pacotes fiável e ordenada entre aplicações em computadores diferentes

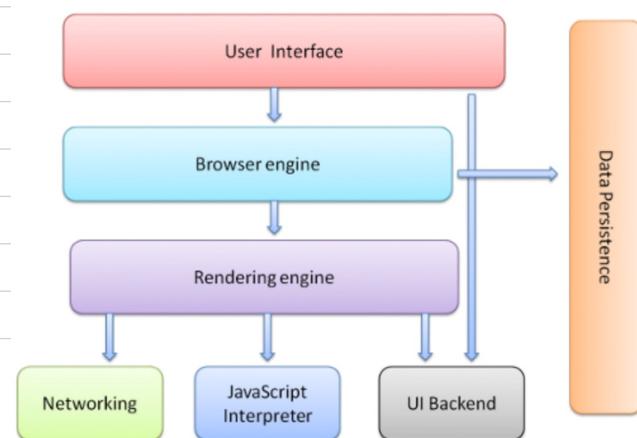
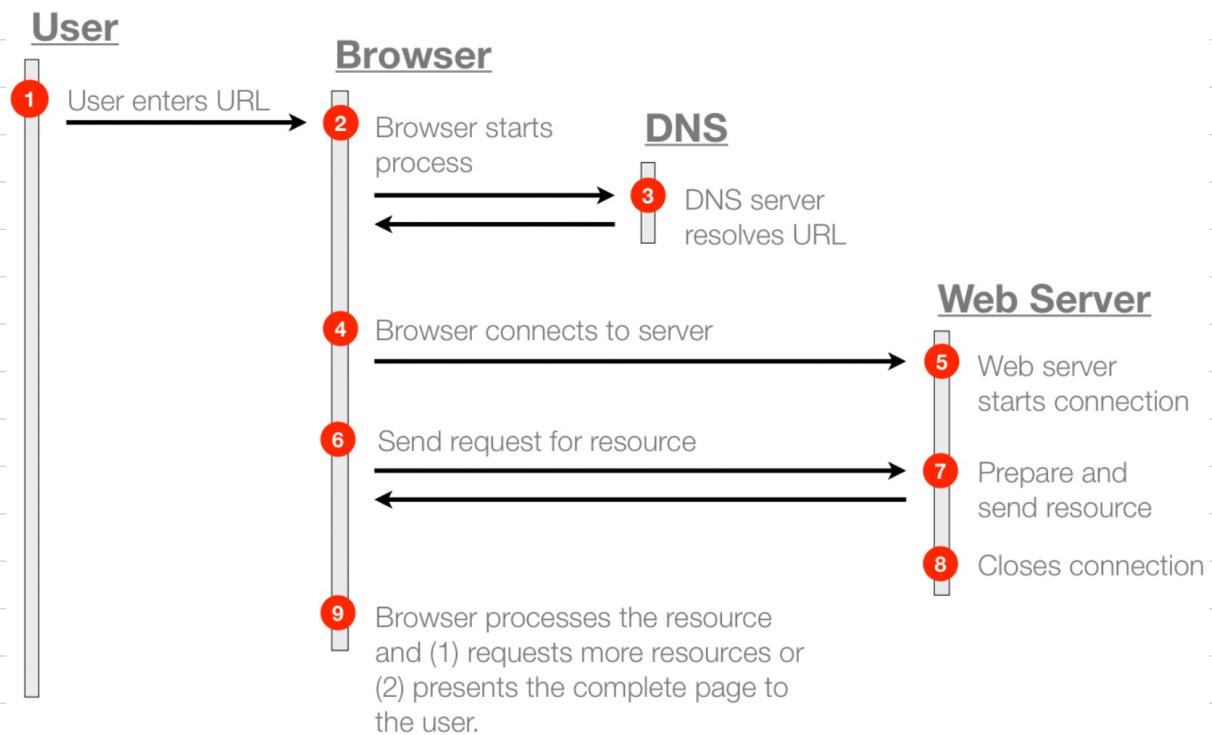
Domain Name System (DNS): traduz nomes simbólicos legíveis em endereços numéricos (IP)

WHOIS Protocol: protocolo de pergunta-resposta usado para interrogar bases de dados que contêm informações sobre recursos da internet, tal como nomes de domínio ou endereços IP

A arquitetura web segue um modelo cliente-servidor

Servidor: máquina que executa aplicações à espera de pedidos de clientes

Cliente: navegador web que inicia a sessão de comunicação com o servidor



- User Interface - browser controls
- Browser engine - mapping
- Rendering engine - HTML & CSS
- Networking - network calls
- JS Interpreter - execute javascript
- UI Backend - drawing widgets
- Data Persistence - saves data

1. HTTP request: "Give me resource X"

Client

Server

2. HTTP response: "Here is resource X ..."

Uniform Resource Locator (URL): usado para identificar os recursos disponíveis na web — endereço único | localizar

Sintaxe: protocolo://máquina:porto/diretório/ficheiro.tipo

HyperText Transfer Protocol (HTTP): usado para especificar como é que os clientes comunicam com os servidores para aceder a recursos web | definir

Métodos:

1. GET — pede o recurso ao servidor
2. HEAD — pede só os cabeçalhos (sem o conteúdo)
3. POST — submete dados para serem processados pelo recurso
4. PUT — faz upload de dados no recurso especificado
5. DELETE — apaga o recurso especificado

Códigos de Estado:

1. 1XX — Informacional
2. 2XX — Sucesso
3. 3XX — Redirecionamento
4. 4XX — Erro do Cliente
5. 5XX — Erro do Servidor

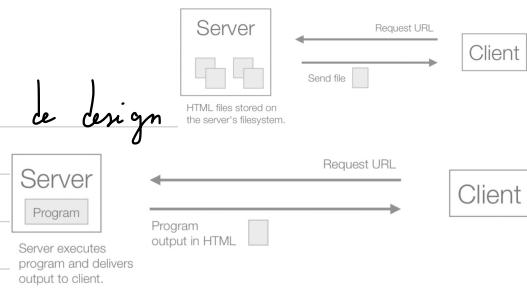
Cookies: ficheiros do lado do cliente gerados pelo servidor e armazens ao lado do pedido

Versões: ficheiros do lado do servidor com identificadores únicos da versão

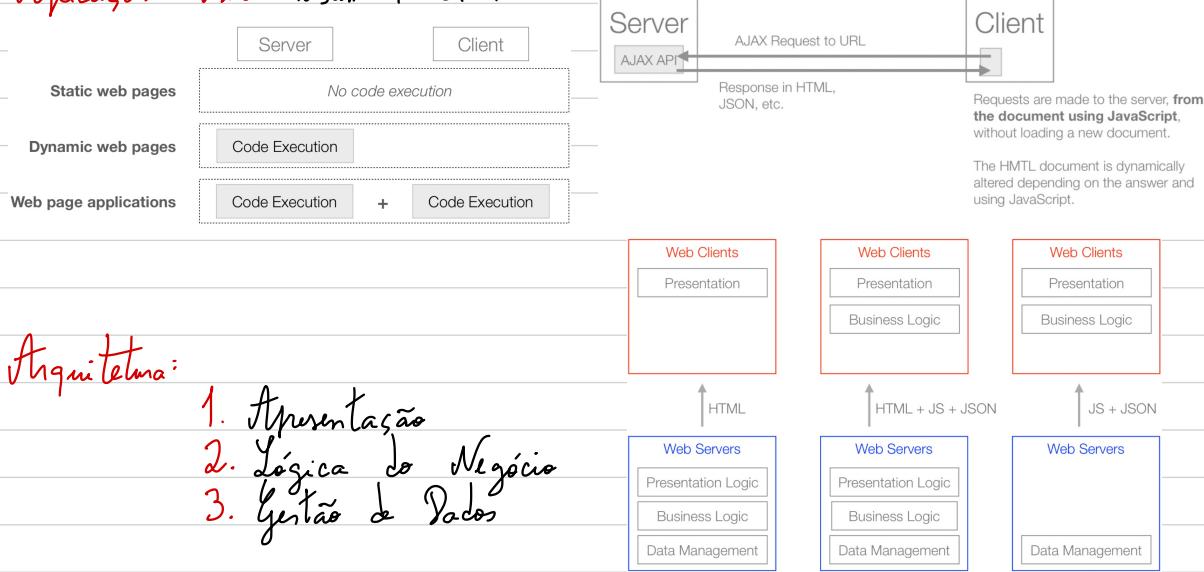
HyperText Markup Language (HTML): usado para representar e interligar documentos na web — define o conteúdo e a estrutura

Páginas Web Estáticas: construídas em tempo de design

Páginas Web Dinâmicas: tempo de execução



Aplicações Web usam AJAX



Arquitetura:

1. Apresentação
2. Lógica do Negócio
3. Gestão de Dados

Aplicações Web Multi-Página: cada página é preparada no servidor e não os detalhes da apresentação são enviados para o browser - cliente magro

- estilo REST; independente do cliente; consistente entre browsers
- desenvolvimento responsivo; código fragmentado; não atualizável

Aplicações Web de Página Única: os recursos web são carregados ou adicionados dinamicamente à página - cliente gordo

- melhor experiência do utilizador; menor consumo; interfaces e código reutilizável
- necessidade de JavaScript; sem histórico; nem REST; mais dependências

Aleatoriedadem Mista: usam diferentes páginas web e APIs de dados para as atualizações — usam pedidos assíncronos para melhorar desempenho e experiência.

"Rendering":

1. Yo do lado do servidor

- a. Estático — HTML corresponde aos ficheiros pré-construídos no servidor
- b. Dinâmico — HTML resulta de aplicações executadas no servidor

2. Dos lados do cliente e do servidor

- a. "Rendering" do lado do servidor com hidratação — HTML construído dinamicamente no servidor com atualizações no cliente
- b. "Rendering" do lado do cliente com "pre-rendering" no servidor — HTML preparado no servidor, mas o cliente assume o controle

3. Yo do lado do cliente — só um esqueleto mínimo é servido pelo servidor, toda a lógica da aplicação e o "rendering" são feitos no cliente

Device Capabilities – Which features are supported.

Networking – DNS, TCP/IP.

Template Design – Reusable, modular solutions.

Web Protocols – HTTP, REST.

Content Presentation – HTML, CSS.

Data Formats – HTML, JSON, XML.

Interface Programming – Also HTML and CSS but a lot of JavaScript.

Web Servers – Apache, IIS, nginx, lighttpd, node.js.

Performance – How to improve response speed.

Data Storage – SQL, NOSQL, cache system, file system.

Resilience – How to make reliable systems.

Server-side Programming – Languages, libraries, frameworks.

Security – Critical and complex in web systems.

Content Management Systems – Reusable building blocks (e.g. Wordpress).

Navigation Design – How to organize content and navigation.

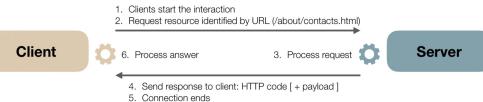
Library and framework management

Graphics – Images, SVG, Canvas.

Building and packaging – How to distribute web products: hosting, CDNs.

Servidor: PHP (Laravel) e PostgreSQL

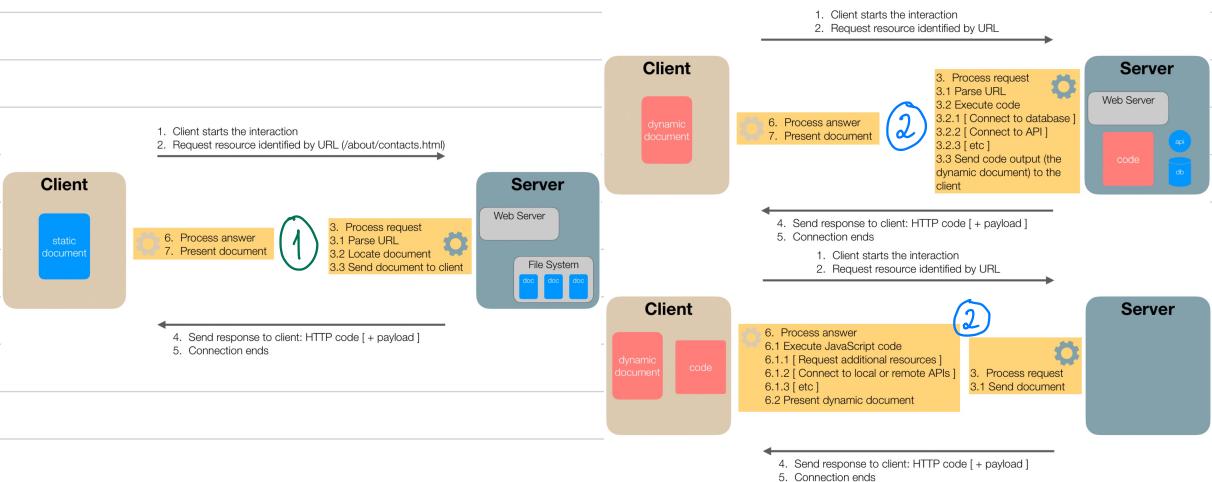
Cliente: HTML, CSS, Java Script, AJAX



Server - Side

Web - Dev

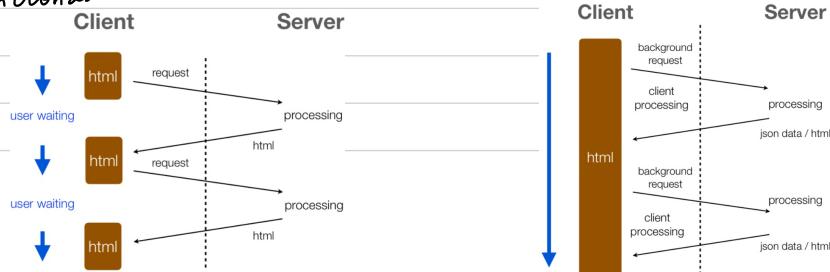
Documentos Web Estáticos: não têm processamento envolvido
Documentos Web Dinâmicos: preparados quando pedidos



- rápidos, mas rígidos (não atualizações / personalizações e difíceis de manter)
- S. complexos e atualizadores, mas lentos
- C. complexos e interativos, mas perados

Vistas: só accedem a dados para apresentar, o output é HTML
Asões: mudam dados e redirecionam para uma vista

AJAX: conjunto de técnicas de desenvolvimento que implementam interações web assíncronas



Recursos Web

Recursos Web: representam os endpoints da aplicação web

1. Vistas: mostram a interface do utilizador - retornam HTML
2. Ações: processam informações e redirecionam para uma vista
3. AJAX: disponivel para chamadas assíncronas - retornam JSON/HTML

OpenAPI: uniformiza a descrição de APIs

O documento é um objeto JSON e pode ser representado em JSON/YAML

Servidores: especificam o servidor da API e o URL base

Todos os caminhos da API não relativos ao URL do servidor

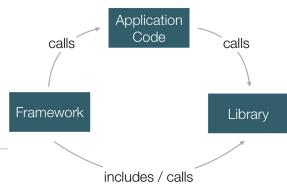
Tags: usadas para agrupar operações da API - módulos definidos

Caminhos: recursos/endpoints que a API expõe

Operações: métodos HTTP usados para manipular os caminhos

```
[/login:]  
[post:]  
  
requestBody:  
  required: true  
  content:  
    application/x-www-form-urlencoded:  
      schema:  
        type: object  
        properties:  
          email:           # <!-- form field name  
            type: string  
          password:       # <!-- form field name  
            type: string  
        required:  
          - email  
          - password
```

Frameworks



Framework: controla o fluxo da aplicação e chama código específico
Biblioteca: usada pelo código da aplicação para suportar funcionalidades

- +: rapidez de implementação; facilidade; experiência; manutenção
- : menor independência; pior desempenho; dependência; " prisão / bloqueio"

Arquitetura Tri-Partida:

1. Apresentação: interface com o usuário, visões e interações
2. Lógica: coordenação da aplicação, processamento de dados e do fluxo
3. Dados: gestão de informações, persistência e consistência

Frameworks Yenê-Yide:

1. Micro: foco em definir as rotas de pedidos HTTP para um "callback"
2. Full-Stack: repleto de funcionalidades (rotas, templates, acessos, mapeamento)
3. Componente: coleção de bibliotecas especializadas de uso único

Componentes Nucleares:

1. Routing de Pedidos: maneja pedidos/acessos HTTP para funções específicas
2. Mecanismo de Templates: estrutura e reforma a apresentação da lógica
3. Acesso a Dados: uniformiza o acesso, mapeamento e configuração dos dados

Componentes Comuns:

1. Segurança: proteção contra ataques comuns de segurança web
2. Serviços: gestão e configuração de serviços
3. lidar com Erros: captura e gerencia erros ao nível da aplicação
4. Scalability: gerar rapidamente interfaces CRUD baseadas no modelo

Laravel

Rotas: definem como lidar com pedidos a URLs

Modelos: mapeamento das tabelas para suportar operações sobre os dados

Vistas: definidas para a apresentação

Artisan: interface da linha de comandos

1. Modelos → 2. Controladores → 3. Rotas → 4. Vista

- Desenvolvimento rápido da aplicação
- Convenção em vez de configuração
- Simplicidade

Composer: pacote de PHP gestor de dependências — composer.json

composer.lock: fica as versões usadas no projeto

.env: ficheiro de configurações que configura o acesso à base de dados

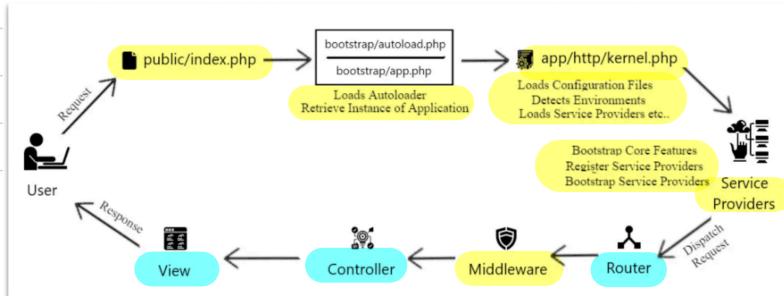
Seeder: cria a base de dados a partir do ficheiro SQL

Modelo: controla o acesso à base de dados — usado para interagir

Controlador: onde a lógica de cada recurso é implementada

Vista: define a interface do utilizador

Rota: define como um pedido web é processado



Rotas {

- Web Route::get Route::post ... Route::match Route::any
- API Route::redirect Route::permanentRedirect Route::view

· /URI/{parâmetro}/{opcional?} → REGEX → name()

· O comportamento das rotas deve ser lidado através do método **404** do controlador

Controlador: organiza a lógica de uma ou mais rotas na mesma classe

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

· \$request → validate ([... => ..., ... => ...])

Vista: permite a separação da camada de apresentação (das camadas do modelo e do controlador)

{} de variável {}

return view (...) → with (... , ...)
return view (... , [... => ...])

@verbatim @if @brief @else @unless @inset @empty
@auth @guest @production @env @for @foreach @while
@include: inclui uma vista noutra vista
@section: define uma secção de conteúdo
@yield: mostra os conteúdos de uma dada secção

Banco de Dados:

1. SQL cru: select; update; insert; delete; statement
2. Constructor de Querries: DB::table (...) → where (...) → get()
3. Eloquent ORM: cada tabela tem um modelo correspondente

hasMany

belongs To

- Os modelos não são usados para ir buscar dados
- Cada modelo funciona como um construtor de querries especializado
- save() armazena uma instância na base de dados

Migrações: up para adicionar novos dados e down para reverte

Autenticação: Users (username, password, remember_token)

"Gates" ≈ rotas (baseadas em fechos) - para ações não relacionadas com modelos ou recursos

"Policies" ≈ controladores (definem lógica) - para ações sobre modelos ou recursos
AUTODISCOVERY → can (...) → authorize
@can
@cannot

Client - Side | Web - Tech

Os navegadores fazem pedidos aos servidores, que produzem e retornam documentos para serem interpretados e mostrados

HTML: conteúdo e estrutura

CSS: organização e apresentação

JavaScript: comportamento e interação

Uma Texto: informação legível ligada de forma sem restrições
forma de ligar e acessar a informação de vários tipos como nós

HTML5 {
· Linguagem de Marcação
· Definição do Modelo do Documento
· APIs para suportar interação do JavaScript com o DOM

Microdados: extensão para definir novos atributos e incluir dados simples em documentos HTML — anotar conteúdo com etiquetas

itemscope: cria um item

itemprop: define uma propriedade

→ define conceitos e relações

itemid: associa um vocabulário com um item

itemref: define um identificador único global

→ API de Geo-Localização — única ou contínua

→ API de Armazenamento — persistente no cliente em pares chave-valores

→ API de Web Sockets — comunicação bidirecional cliente-servidor

→ API WebRTC — comunicação entre clientes em tempo real

→ API de Trabalhadores Web PWA — suportam execução de scripts em 2º plano

desenvolvíveis; instaláveis; ligáveis; independentes; progressivos; responsivos; seguros

Desempenho Web

• 80% do tempo de resposta do utilizador final é gasto no front-end

1. Fazer menos pedidos HTTP
 - a. mapas de imagens → proliferação de erros; limitações de acessibilidade
 - b. Sprites CSS → difícil de manter
 - c. combinar scripts e folhas de estilos
2. Optimizar imagens
3. Usar uma "Content Delivery Network" (CDN)
4. Adicionar um cabeçalho de expiração
5. Gérejar componentes
6. Tornar JavaScript e CSS externos
7. Reduzir pesquisas DNS
8. Minimizar JavaScript e CSS
9. Evitar redirecionamentos
10. Remover scripts duplicados
11. Configurar ETags
12. Tornar AJAX "cacheable"

Client - Side Web - Dev

- Existem frameworks de HTML, CSS e javaScript
- O problema central que visam é gerir o estado da aplicação e sincronizar a interface do utilizador

Tailwind CSS: em vez de criar classes sobre componentes, as classes são criadas sobre estilos específicos — "utility classes"

- Força "utility classes"; design responsivo; variações de estado
- HTML verboso; maior pegada; desafios de consistência (falta de reunião)

Vue.js: framework de javaScript do lado do cliente

- v-model: via uma ligação nos dois sentidos — se o input muda, os dados mudam e vice-versa
- v-bind: pode mudar as propriedades de um elemento
- v-on: usado para capturar eventos do DOM e escrever código JS @class

Ferramentas:

1. Desenvolvimento de Código: controlo do código fonte (backups, equipa); ferramentas de desenvolvimento do browser (inject/debug), linters (destaca erros e violações); bundlers/facotres (preparam para produção)
2. Transformação do Código: permitem usar funcionalidades mais recentes (Babel, PostCSS) ou outras linguagens (Sass/SCSS, TypeScript)
3. Testagem: correm testes automaticamente
4. Produção: publicam automaticamente numa aplicação web

Acessibilidade & Usabilidade

Acessibilidade: suporta inclusão social para pessoas com incapacidade

Tecnologias de Assistência:

1. Leitores de Tela
2. Ampliadores de Tela
3. Aumentadores de Texto
4. Dispositivos de Input Alternativos

Usabilidade: combinação de fatores que afetam a experiência do utilizador com a aplicação — cria interfaces de utilizador eficazes

- Facilidade de Aprendizagem
- Eficiência de Uso
- Memorabilidade
- Frequência e severidade dos erros
- Utilização Subjetiva

Padrões de Design:

1. Acondição
2. Barra de Progresso
3. Barra de Navegação da Esquerda
4. Barra de Navegação do topo
5. Paginação de Itens

... Registo Previsões; Divulgação Progressiva; Breadcrumbs; Registo de Contas; Marcador de Campo Obrigatório

Técnicas: Testagem; Prototipagem; Avaliação Heurística