

Desenvolvimento Web

Trilha JavaScript com Angular e Node

Instrutor: Júlio Pereira Machado (julio.machado@pucrs.br)



Web



Web

De acordo com o W3C:

"A World Wide Web (WWW) é um espaço de informações no qual os itens de interesse, referidos como recursos, são identificados por identificadores globais chamados Uniforme Resource Identifiers (URI)".

<https://www.w3.org/>

Web

URI

`http://weather.example.com/oaxaca`

Identifies

Resource

Oaxaca Weather Report

Represents

Representation

Metadata:

Content-type:

`application/xhtml+xml`

Data:

```
<!DOCTYPE html PUBLIC "...  
    "http://www.w3.org/...  
<html xmlns="http://www...  
<head>  
<title>5 Day Forecaste for  
Oaxaca</title>  
...  
</html>
```

HTTP

- Comunicar-se com servidores e aplicativos web se dá através do protocolo *Hypertext Transfer Protocol*
- Protocolo de nível de aplicação no modelo requisição/resposta
- Protocolo textual
- Protocolo baseado em mensagens de requisição/resposta no modelo cliente/servidor
- Protocolo sem manutenção de estado
- Mais informações:
 - <https://tools.ietf.org/wg/httpbis/>
 - <https://developer.mozilla.org/pt-BR/docs/Web/HTTP>

HTTP

Versões (em uso):

- 1.1 (RFCs [7230](#), [7231](#), [7232](#), [7233](#), [7234](#), [7235](#))
- 2 (RFC [7540](#))

HTTP

- Um URL *Uniform Resource Locator* é utilizado para identificar elementos em um servidor web

- Ex.: `http://java.sun.com/index.html`

- Formato geral:

`protocol ":" "//" host [":" port] [path ["?" query]]`

HTTP

Uma requisição HTTP consiste de:

- Uma linha inicial
- Um ou mais campos de cabeçalho
- Uma linha em branco
- Possivelmente um corpo da mensagem

Uma resposta HTTP consiste de:

- Uma linha de status com seu código (ver [RFC](#), [Wikipédia](#)) e mensagem associada
- Um ou mais campos de cabeçalho
- Uma linha em branco
- Possivelmente um corpo da mensagem

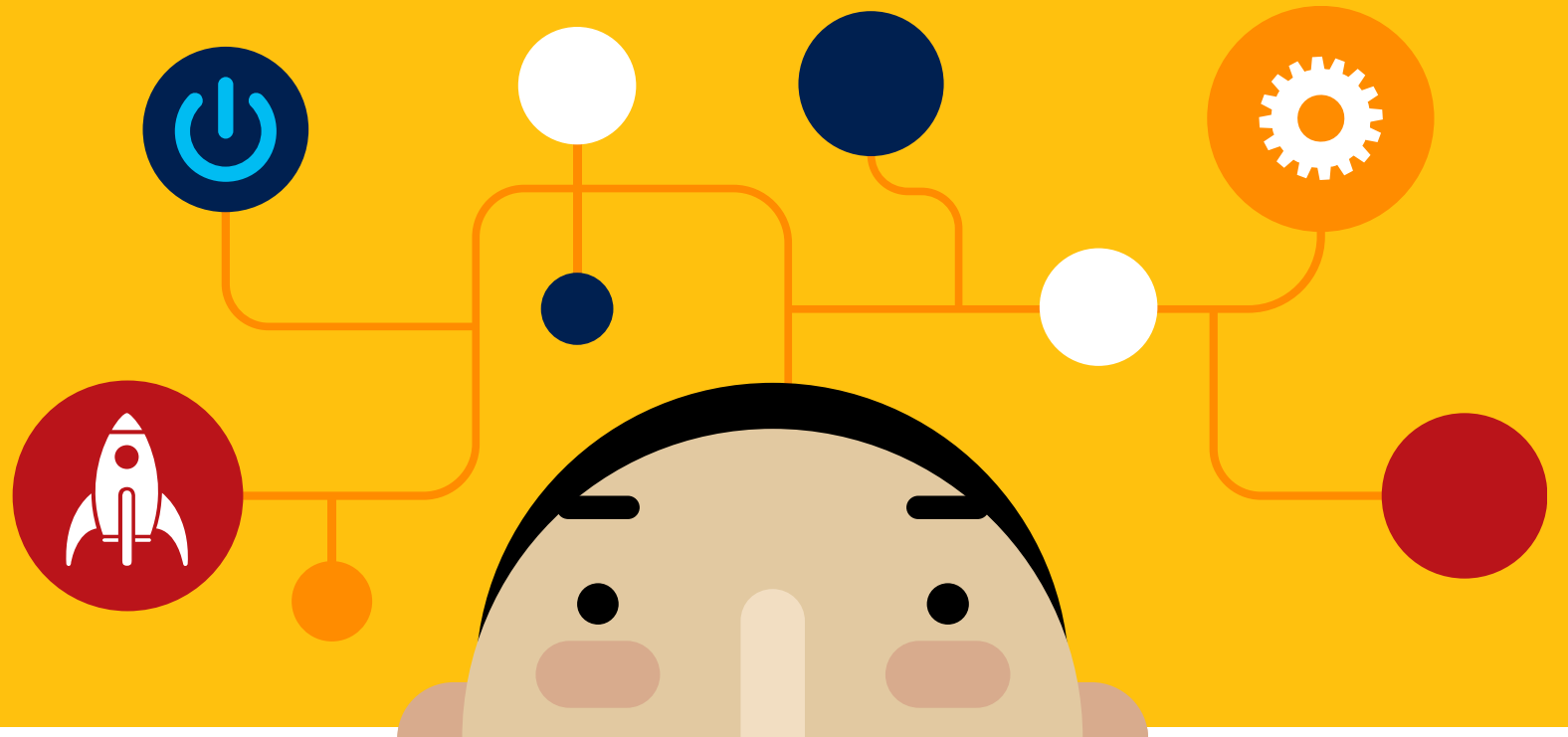
HTTP

HTTP Method ↕	RFC ↕	Request Has Body ↕	Response Has Body ↕	Safe ↕	Idempotent ↕	Cacheable ↕
GET	RFC 7231 ↗	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231 ↗	No	No	Yes	Yes	Yes
POST	RFC 7231 ↗	Yes	Yes	No	No	Yes
PUT	RFC 7231 ↗	Yes	Yes	No	Yes	No
DELETE	RFC 7231 ↗	No	Yes	No	Yes	No
CONNECT	RFC 7231 ↗	Yes	Yes	No	No	No
OPTIONS	RFC 7231 ↗	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231 ↗	No	Yes	Yes	Yes	No
PATCH	RFC 5789 ↗	Yes	Yes	No	No	No

[https://en.wikipedia.org/wiki/Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

Laboratório

- Experimente as ferramentas de desenvolvedor do navegador para observar o fluxo das requisições e respostas do protocolo HTTP



Web Services REST



SOA

- Arquitetura Orientada a Serviço consiste em três elementos principais:
- Serviços : representam funcionalidades de negócio independentes e podem ser implementados por diversas tecnologias em diversas plataformas;
- Infraestrutura: permite combinar os serviços de maneira flexível;
- Políticas e processos: para lidar com sistemas distribuídos heterogêneos.
- Ideias trabalhadas desde a década de 1990

SOA

- Um *serviço* é um pedaço de funcionalidade corporativa independente
- A funcionalidade pode ser simples ou complexa
- A idéia é que as interfaces externas dos sistemas devem ser projetadas de forma que as pessoas de negócio possam entendê-las
- A interface e o contrato de um serviço devem ser bem definidos

SOA

- Um *fornecedor* é um sistema que implementa um serviço de tal forma que outros sistemas podem consumi-lo
- Um *consumidor* é um sistema que usa um serviço fornecido

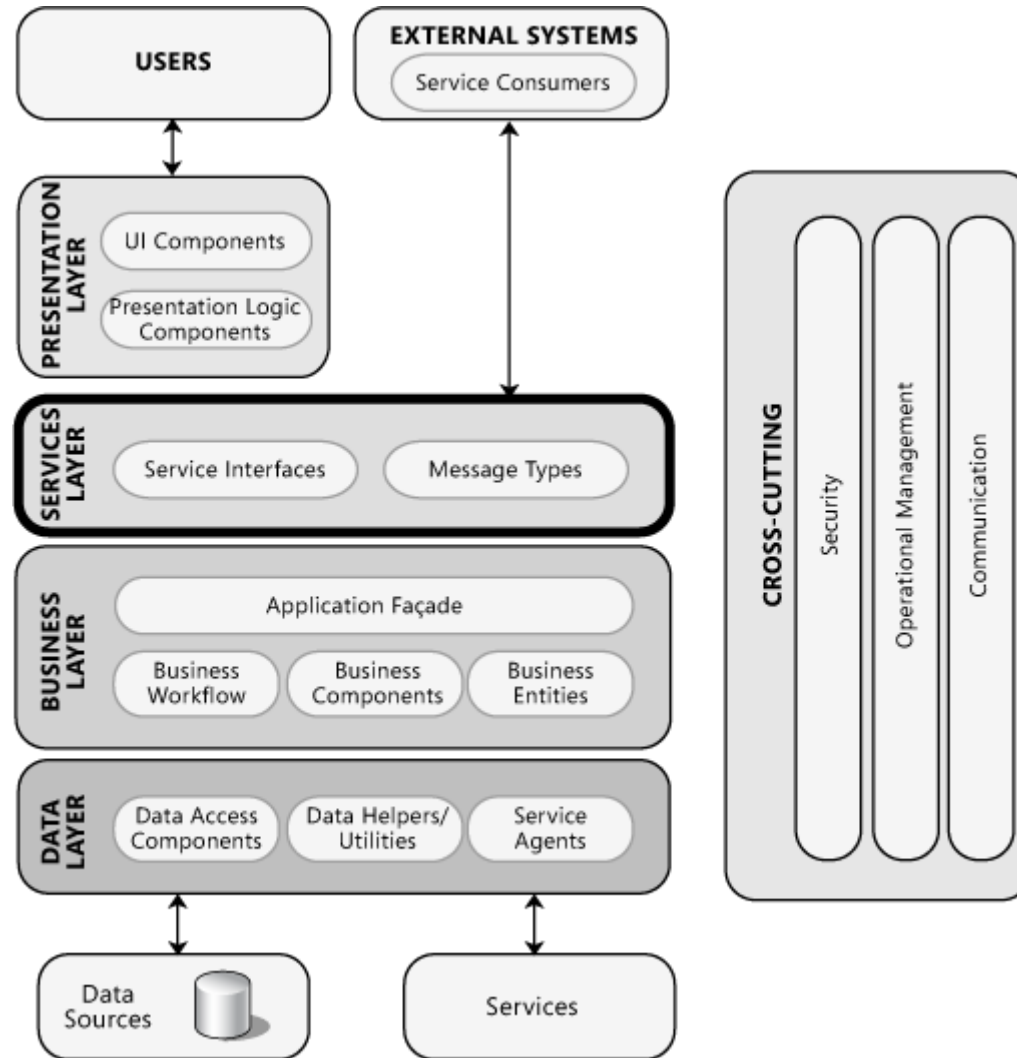
Web Services

- W3C:
- “Um Web Service é um sistema de software cujo propósito é suportar de maneira interoperável a interação máquina-máquina sobre uma rede de comunicação. Ele possui uma interface descrita em um formato processável por máquinas. Outros sistemas interagem com ele de acordo com a interface através de mensagens, tipicamente sobre um protocolo padrão da internet via serialização em conjunto com outros padrões web relacionados.”

Web Services

- Características:
 - Objetos remotos
 - Residem em um servidor Web e têm um endereço URL
 - Trabalham sobre o modelo de requisição/resposta
 - Utilizam protocolos que facilitam a comunicação entre sistemas
 - Independente do sistema operacional e da linguagem de programação (web services interoperáveis)
 - São objetos para soluções fracamente acopladas

Web Services



REST

- Representational State Transfer é um estilo arquitetural
 - <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Web services baseados em REST são chamados de RESTful
- Características:
 - Serviços sem estado
 - Baseados no protocolo HTTP/HTTPS
 - Dados e funcionalidades são considerados recursos acessados via URIs
- Exemplo:
 - <http://predic8.com/rest-demo.htm>
 - <http://jsonplaceholder.typicode.com/>
 - <https://reqres.in/>
 - <https://randomuser.me/>
 - <http://petstore.swagger.io/>

REST

- Arquitetura baseada em quatro princípios:
- Identificação dos recursos através de URIs
- Interface uniforme de acesso aos recursos
 - Operações de criação, leitura, alteração e remoção
 - Implementadas via verbos do HTTP
- Mensagens autodescritivas
- Iteração com manutenção de estado através de hiperlinks

REST

- Questões para o desenvolvedor:
 - Definir quais são os “recursos” expostos
 - Definir o formato das URIs para os recursos
 - Decidir quais verbos do HTTP serão realmente utilizados
 - Estabelecer a real semântica da aplicação de cada verbo sobre um recurso
 - Definir o retorno das requisições em caso de sucesso e de falha

REST

- Exemplo: URI de coleção
 - `http://exemplo/recursos/`
 - GET: lista as URIs e outros detalhes dos elementos da coleção
 - PUT: substitui a coleção por uma outra
 - PATCH: não é muito utilizado
 - POST: adiciona um novo elemento na coleção, retornando a URI para o novo elemento
 - DELETE: remove a coleção inteira

REST

- Exemplo: URI de elemento
- `http://exemplo/recursos/123`
- GET: obtém a representação de um elemento específico da coleção
- PUT: substitui um membro específico da coleção ou, se ele não existe, cria um novo
- PATCH: atualiza (podendo ser só uma parte) do membro específico da coleção
- POST: geralmente não utilizado; trata o elemento da coleção como uma própria coleção, adicionando um novo elemento nele
- DELETE: remove o elemento da coleção

REST

- Descrição e descoberta:
- WADL (Web Application Description Language)
 - <https://wadl.java.net/>
 - Especificação de linguagem para descrição de serviços REST
 - Criada inicialmente para implementação JavaEE
- RAML (RESTful API Modeling Language)
 - <http://raml.org/>
 - Especificação de linguagem para descrição de serviços REST
- Swagger
 - <http://swagger.io/>
 - Especificação de linguagem para descrição de serviços REST
 - Formato de arquivo JSON (<http://json.org/>) e YAML (<http://yaml.org/>)
- OpenAPI
 - <https://www.openapis.org/>

Web Services REST e Node



Consumindo Serviços REST

- Um código TypeScript rodando no ambiente Node tem várias opções para consumir serviços REST:
 - Módulo http da API nativa do Node
 - Módulo request <https://www.npmjs.com/package/request>
 - Módulo node-fetch <https://www.npmjs.com/package/node-fetch>
 - Módulo r2 <https://www.npmjs.com/package/r2>
 - Módulo axios <https://www.npmjs.com/package/axios>
 - etc

Consumindo Serviços REST

- Exemplo:
 - Módulo node-fetch <https://www.npmjs.com/package/node-fetch>
 - Instalação:
 - `npm install node-fetch --save`
 - `npm install @types/node-fetch --save-dev`

Criando Serviços REST

- Utilizando Node com Express e pacotes adicionais, serviços REST podem ser configurados e implementados de maneira eficiente
- Express suporta:
 - Verbos usuais do HTTP
 - Extração de elementos da URI
 - Manipulação de conteúdo de corpo da requisição em JSON e outros formatos de serialização
 - Autenticação com *JSON Web Tokens*
 - etc

Express



Express

- Framework para aplicações Web baseadas em Node.js
- Abstrai vários elementos da infraestrutura
 - Por exemplo, a manipulação das requisições e respostas do HTTP
- Ajuda a organizar uma aplicação Node.js dentro do padrão Model-View-Controller (MVC)
- Código aberto

<http://expressjs.com/>

express

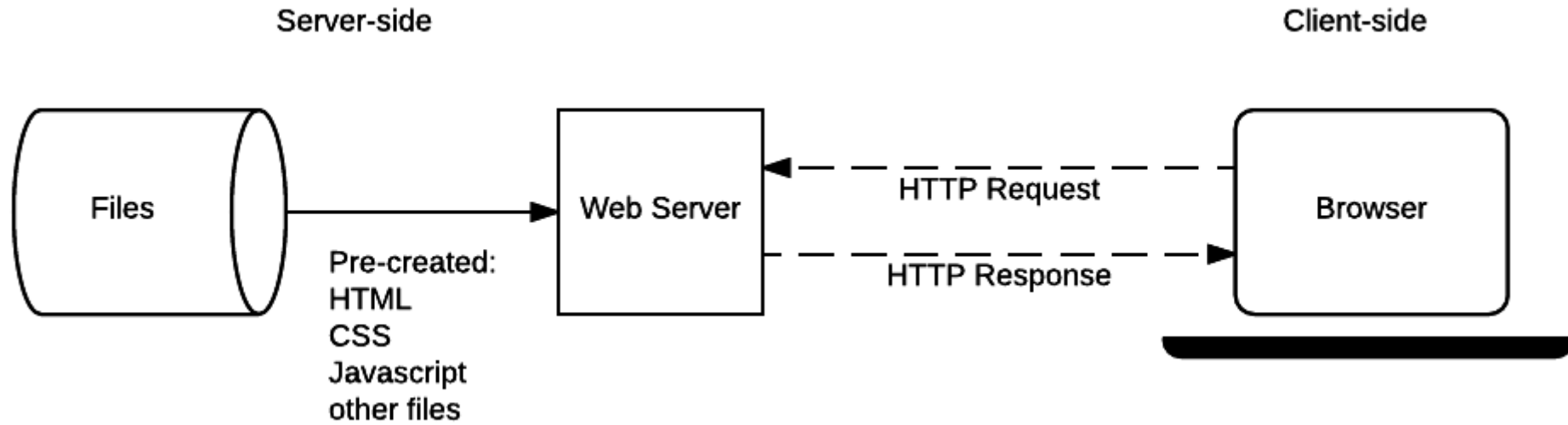
Express

- Express é distribuído via NPM
- Deve ser instalado como um pacote local ao projeto que irá utilizá-lo
 - Instalação
 - `npm install express --save`
 - `npm install @types/express --save-dev`

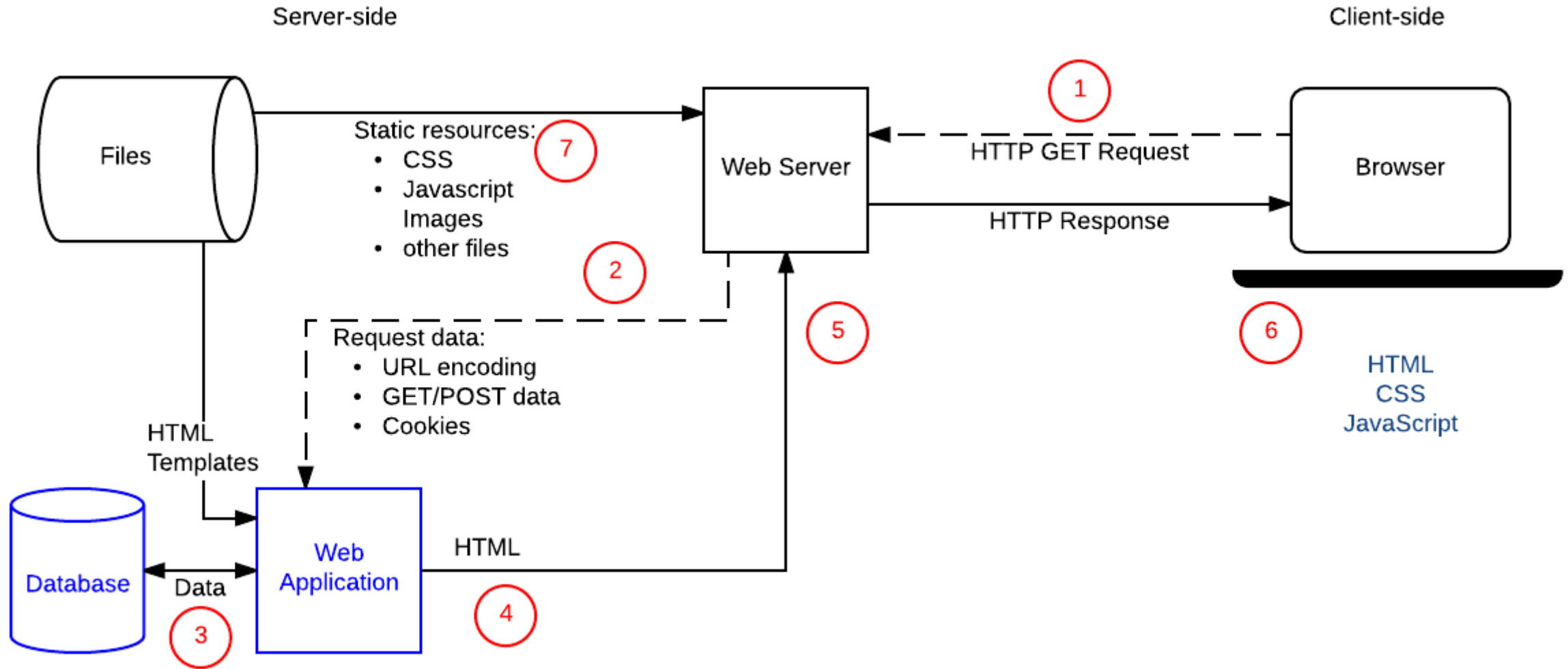
Construindo um Servidor Web

- Um dos módulos básicos do Node.js é o módulo HTTP, o qual fornece a implementação de um servidor e cliente do protocolo HTTP
- O servidor Web fornecido pelo módulo HTTP é bastante flexível, mas requer bastante código de configuração para ser utilizado
- É comum utilizar outros pacotes, como *Express* para configurar um servidor Web

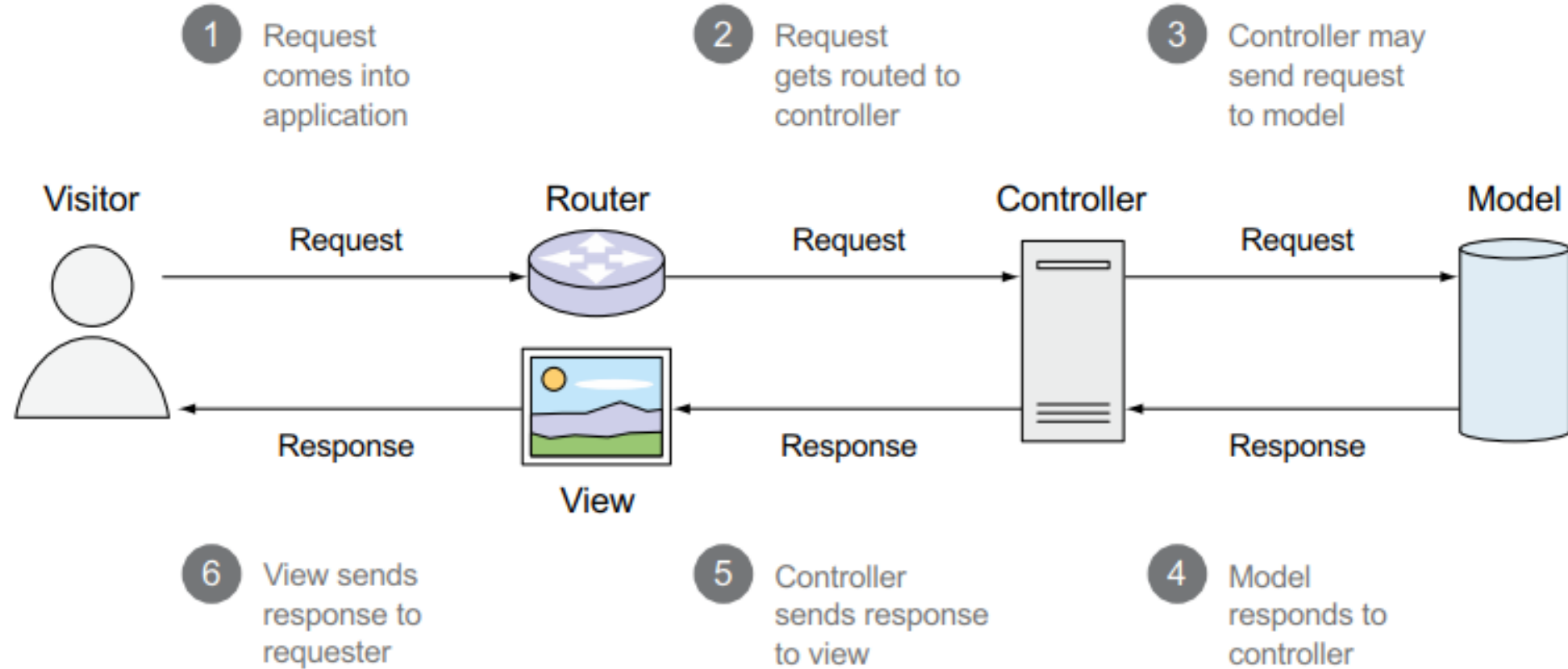
Sites Estáticos



Sites Dinâmicos



Express - MVC



Express

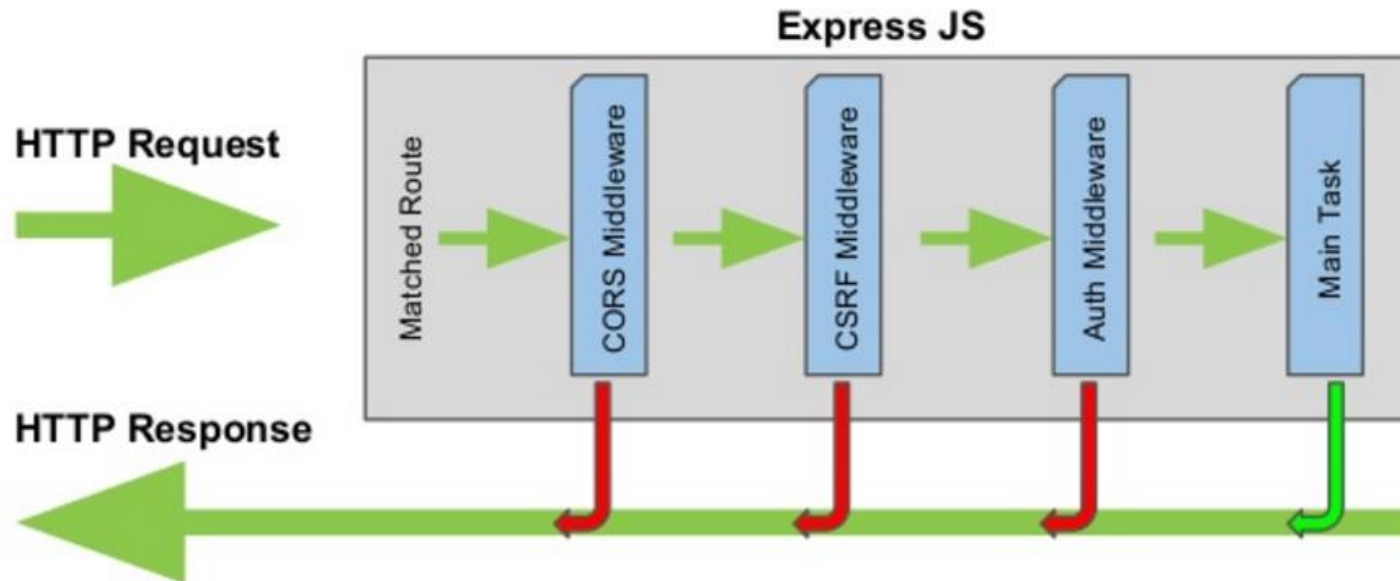
- Provê suporte para roteamento de URIs, fornecimento de arquivos estáticos, fornecimento de arquivos dinâmicos via *view engines* baseadas em *templates* (requer pacotes adicionais)
- Quase que a totalidade das funcionalidades (conhecidas no ambiente Express como “middleware”) são fornecidas por módulos de terceiros, por exemplo
 - Processamento do corpo de uma requisição – *Body Parser*
 - Manipulação de cookies – *Cookie Parser*
 - Utilização de sessões – *Express Sessions*
 - Configuração de autenticação – *Passport*
 - Veja mais em <http://expressjs.com/en/resources/middleware.html>

Express - Exemplo

```
import express from 'express';  
const app = express();  
  
app.get('/', function (req, res) {  
  res.send('Hello World!');  
});  
  
app.listen(8080, function () {  
  console.log('App na porta 8080!');  
});
```

Express - Middleware

- O processamento de uma requisição até a construção da resposta se dá no Express através de um *pipeline*
 - Cada componente configurado do *middleware* realiza um processamento sobre a requisição/resposta e a repassa para o próximo componente
 - Formato do callback: *function (request, response, next)*
- A ativação do *middleware* se dá através de chamadas *app.use()*



Express - Middleware

- Um dos middleware bastante utilizados é a configuração de rotas para arquivos estáticos

```
import express from 'express';
const app = express();

app.use(express.static(path.join(__dirname, 'public')));

app.listen(8080, function () {
  console.log('App na porta 8080!');
});
```

Express - Roteamento

- Uma regra de roteamento mapeia solicitações via HTTP para um URI em uma função de callback específica
 - Requisições HTTP podem ser GET/POST/PUT/DELETE, etc
 - URIs descrevem o recurso solicitado
- Definições de roteamento tem a forma geral:

```
app.METODO(CAMINHO, TRATADOR);
```

- app é a instância de express
- METODO é o método de express associado ao verbo do HTTP
 - Express possui vários métodos get(), post(), etc, que podem ser utilizados
- CAMINHO é a parte da URI que se deseja realizar o roteamento
- TRATADOR é a função de callback

Express - Roteamento


- Exemplo:

- Express direciona uma requisição HTTP GET como:
- `http://localhost:8888/index`
- Para uma função de callback que irá tratar da requisição e gerar a resposta:

```
app.get('/index', function (req, res) {});  
app.get('/index', async function (req, res) {});
```

- Para uma função de callback que irá tratar da requisição, gerar a resposta e passar para o próximo tratador de middleware:

```
app.get('/index', async function (req, res, next) {});
```



Express - Roteamento

- Vários exemplos:

```
app.get('/', function (req, res) {  
  res.send('Hello World!');  
});
```

```
app.post('/', function (req, res) {  
  res.send('Got a POST request');  
});
```

```
app.put('/user', function (req, res) {  
  res.send('Got a PUT request at /user');  
});
```

```
app.delete('/user', function (req, res) {  
  res.send('Got a DELETE request at /user');  
});
```

Express - Roteamento

- A string que define um caminho a ser roteado suporta expressões regulares para definir um conjunto de URIs compatíveis
 - Express utiliza o pacote <https://www.npmjs.com/package/path-to-regexp> (acesse para pesquisar na documentação os formatos disponíveis)
- Exemplos:
 - `/ab?cd` mapeia `acd` e `abcd`
 - `/ab+cd` mapeia `abcd`, `abbc`, `abbbcd`, etc
 - `/ab*cd` mapeia `abcd`, `abxcd`, `ab1cd`, `abqualquercoisacd`, etc
 - `/ab(cd)?e` mapeia `abe`, `abcde`
- Testador de rotas:
 - <http://forbeslindesay.github.io/express-route-tester/>

Express - Roteamento

- Roteamento pode utilizar objeto *Router* para facilitar a descrição de rotas

```
app.route('/livro')
  .get(function (req, res) {
    res.send('Obter um livro qualquer')
  })
  .post(function (req, res) {
    res.send('Adicionar um livro')
  })
  .put(function (req, res) {
    res.send('Atualizar um livro')
  });
```

Express - Roteamento

- Roteamento pode utilizar classe *express.Router* para facilitar a descrição de rotas modulares

```
//arquivo de módulo rotas.js
import express from 'express';
const router = express.Router();

router.get('/', (req, res) => { res.send('Birds home page')});
router.get('/about', (req, res) => { res.send('About birds')});

export default router;
```

```
//arquivo de módulo index.js
...
import router from ('./rotas.js');
app.use('/birds', router);
// URIs /birds e /birds/about
```

Express – Requisição URI

- Dois tipos de dados estão disponíveis via URI
 - Parâmetros da URI – são segmentos nomeados do caminho da URI; acessar via propriedade `params` do objeto de requisição
 - Query Strings – acessar via propriedade `query` do objeto de requisição

http:// www.some-domain.com :8080 /person/2 ?detailed=true #home

Protocolo	Domínio / Endereço	Porta	Caminho	Query String	Lado-cliente
-----------	--------------------	-------	---------	--------------	--------------

Express – Requisição URI

- Para definir rotas que mapeiam parâmetros recebidos na URI, especifica-se o nome da chave que irá ser populada com o símbolo ":"
- Exemplo:

```
Caminho do roteamento: /users/:userId/books/:bookId  
URI requisitada: http://localhost:3000/users/34/books/8989  
req.params: { "userId": "34", "bookId": "8989" }
```

```
app.get('/users/:userId/books/:bookId', function (req, res) {  
  res.send(req.params);  
});
```

Express – Requisição Corpo

- Express não suporta diretamente o processamento rico do corpo da requisição
- Utiliza-se pacotes adicionais para processar o corpo da requisição de forma adequada
 - Body Parser – utilizado para processar corpo codificado nos formatos URL encoded e JSON
 - Multer (e outros) – utilizado para processar upload de arquivo

Express – Requisição Corpo

- Um dos middleware mais utilizados é o *body-parser*
 - <http://expressjs.com/en/resources/middleware/body-parser.html>
 - `npm install body-parser --save`
 - `npm install @types/body-parser --save-dev`
- Body Parser permite manipular os dados disponíveis via a propriedade **body** do objeto de requisição
 - Fornece diferentes tipos de *parsers* tais como JSON, raw, texto, URL-encoded
 - No caso de Web Services REST, o mais utilizado é o JSON
- Body Parser deve ser carregado antes de qualquer rota que necessita acesso

Express – Requisição Corpo

- Exemplo:

```
import express from 'express';
import * as bodyParser from 'body-parser';

const app = express();
const jsonParser = bodyParser.json();

app.post('/api/users', jsonParser, function (req, res) {
  if (!req.body) return res.sendStatus(400);
  // Obter dados do objeto de requisição em req.body
})
```

Express - Resposta

- O método de resposta fornece múltiplas alternativas de envio de dados

MÉTODO	DESCRIÇÃO
download()	Transfere o arquivo indicado com um anexo
end()	Finaliza o processo de resposta; utilizado para gerar respostas sem envio de dados
json()	Envia conteúdo JSON; chama internamente <code>JSON.stringify</code> sobre o parâmetro de entrada
redirect()	Envia uma mensagem de redirecionamento para outro recurso
render()	Envia o resultado do processamento de um <i>template view</i>
send()	Envia uma resposta de vários tipos
sendFile()	Envia um arquivo como um fluxo binário
sendStatus()	Configura o código de status da resposta HTTP e envia a mensagem no corpo da resposta

Express - Resposta Exemplo

```
app.get('/', function (req, res) {  
  res.send({mensagem : "Hello World!"}); //Como fica com res.json?  
});
```

Express - Resposta

- Para configurar diretamente um código de status do HTTP para uma determinada resposta, utiliza-se o método **status**
- Exemplo:

```
res.status(403).end();  
res.status(400).send('Bad Request');
```

Express – Tratamento de Exceções

- Utiliza-se funções de middleware para definir um tratamento de exceções
- O tratador é definido como último elemento do pipeline do middleware
- Express já possui um tratador de exceções padrão
 - Comportamento padrão é retornar toda a pilha da exceção para o cliente (no caso de um ambiente de depuração e desenvolvimento)
- É possível definir um (ou vários) tratador próprio, por exemplo:

```
app.use(function (err, req, res, next) {  
  if (res.headersSent) {  
    return next(err);  
  }  
  console.error(err.stack)  
  res.status(500).send('Falha no servidor!');  
});
```

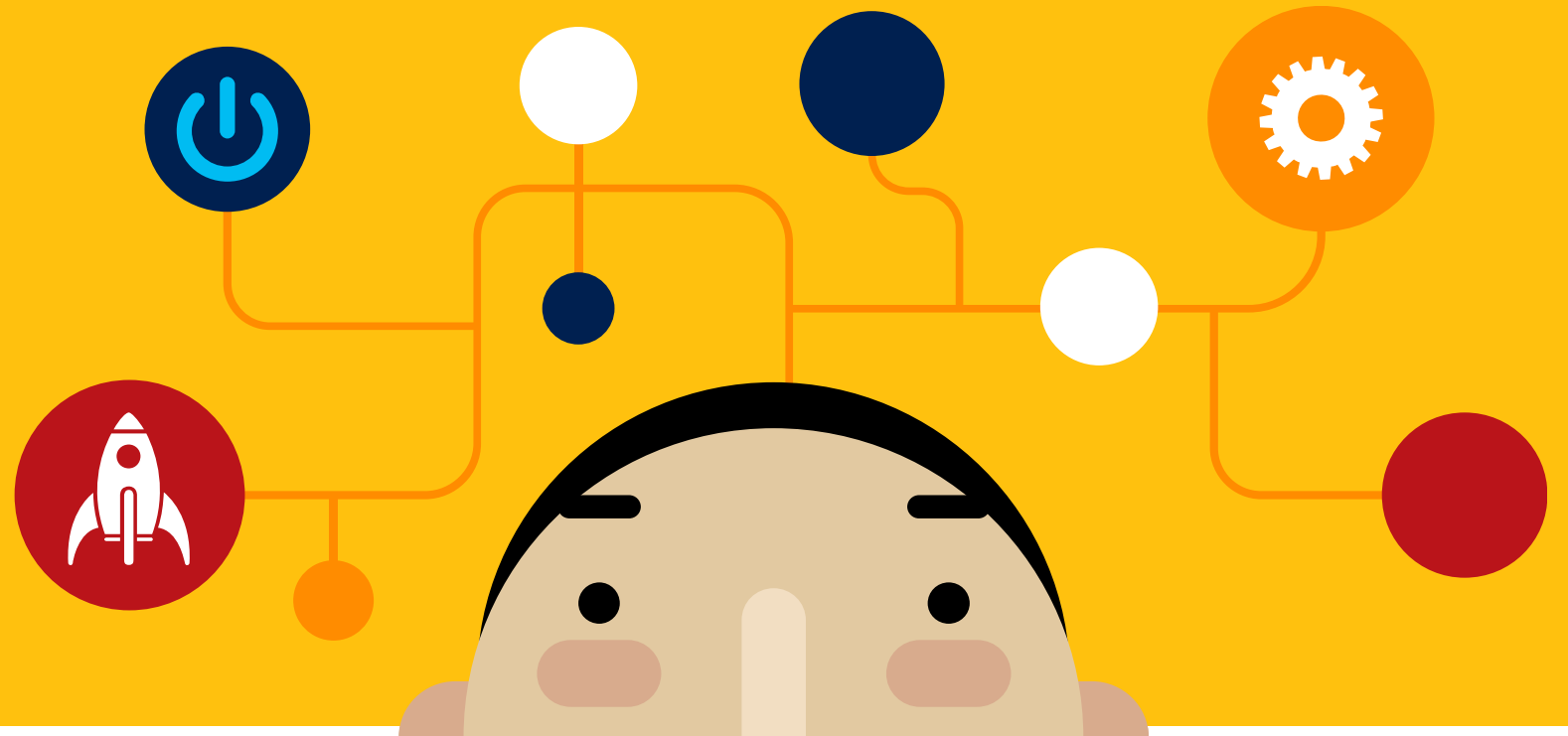
Express – Tratamento de Exceções

- Cuidado!
- O tratador captura exceções de código síncrono
- Para código assíncrono deve ser utilizado função *next()* para propagar o objeto de erro

```
router.get('/user/:id', async (req, res, next) => {  
  try {  
    const user = await getUserFromDb({ id: req.params.id })  
    res.json(user);  
  } catch (e) {  
    next(e);  
  }  
});
```

Laboratório

- Abra as instruções do arquivo Lab06_TypeScript_REST



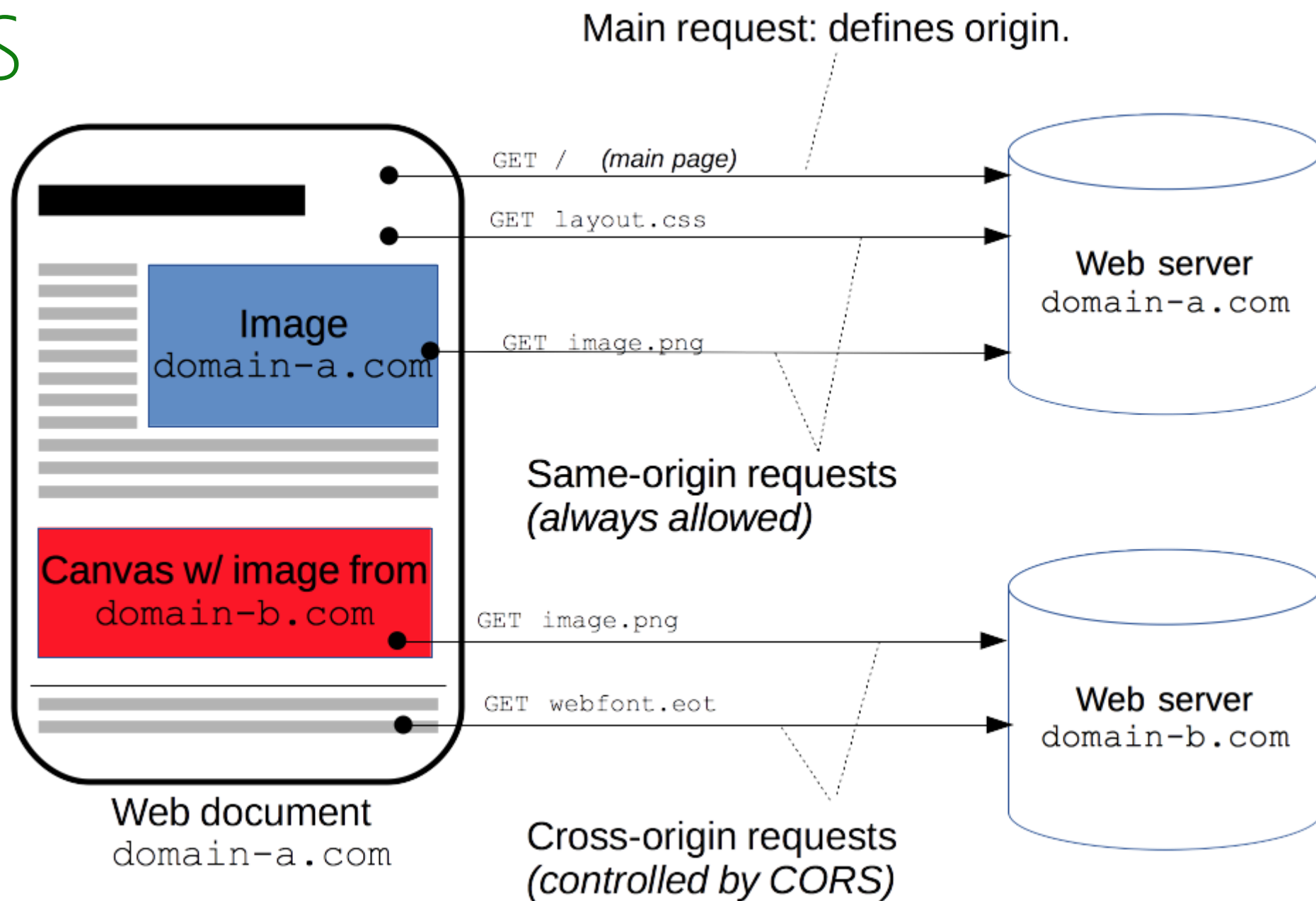
CORS



CORS

- Cross-Origin Resource Sharing
- Especificação que permite acesso entre recursos com origens a partir de servidores diferentes
- É essencial para permitir que clientes acessem um serviço web
- Mais informações:
 - <https://enable-cors.org/>
 - <https://www.w3.org/TR/cors/>
 - <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
 - <http://expressjs.com/en/resources/middleware/cors.html>
 - `npm install cors --save`
 - `npm install @types/cors --save-dev`

CORS



Restify



Restify

- Framework para criação de Web Services REST baseados em Node.js
 - Inclui facilidades a mais que o Express para a implementação de serviços (tratadores de erros, suporte a hyperlinks, etc)
- Instalação:
 - `npm install restify --save`
 - `npm install @types/restify --save-dev`

<http://restify.com/>

