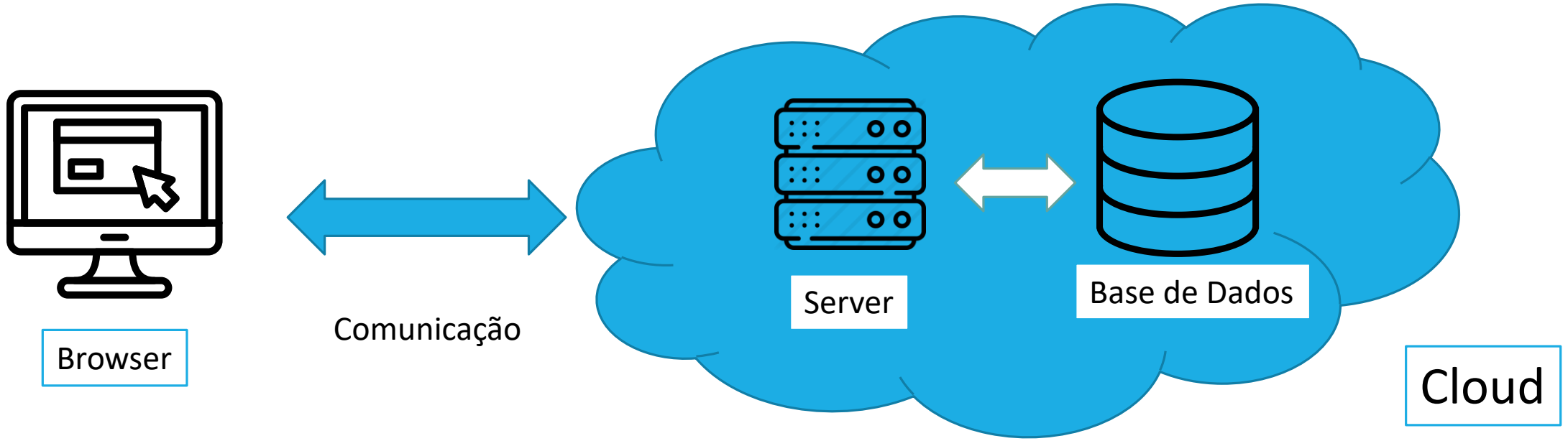
Programação & Serviços Web
React – Aula 5
2019/2020

Cliente - Servidor

- A maioria das aplicações web não funcionam sozinhas
- Muitas delas têm um backend na “Cloud”



Comunicação Cliente-Servidor

- Existem vários problemas neste tipo de comunicações
  - Internet lenta
  - Servidor offline
  - Outros problemas
- Quando se cria uma aplicação, é necessário reconhecer a natureza **assíncrona** da comunicação
  - A informação não está disponível imediatamente, e a aplicação não deve estar eternamente à sua espera.

Hypertext Transfer Protocol (HTTP)

- Protocolo de comunicação entre cliente-servidor
- Permite receber documentos html
- Verbos utilizados
  - HEAD
  - GET
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT

Web Services

- Sistemas desenhado para suportar a interoperabilidade de sistemas conectados através de uma rede
- Na prática, são usadas duas abordagens:
  - SOAP (Simple object access protocol)
    - Usa WSDL (Web Services Description Language)
    - Baseado em XML
  - REST (Representational State Transfer)
    - Usa os standards da web
    - Troca dados através de XML ou JSON
    - Mais simples comparado com o SOAP

REST

- Estilo de arquitetura de software para sistemas distribuídos (como a world wide web) .
- Coleção de princípios de arquitetura de rede que descrevem como os recursos são definidos e abordados

- Princípios básicos do design:
  - Usar métodos HTTP
  - Sem estado
  - Expor URIs semelhante à estrutura de diretórios
  - Transferir dados usando XML, JSON ou ambos

REST and HTTP

- O REST permite capturar as características da web que fizeram a web bem-sucedida
  - URI
  - Procolo Http
  - Fazer um **Request** – Receber **Resposta** – **Mostrar** resposta

Resources

- A abstração principal das informações no REST é um Recurso
- Um recurso é um mapeamento conceitual para um conjunto de entidades
  - Qualquer informação pode ser um recurso:
    - Um documento/imagem
    - Um serviço do tempo (ex. “o tempo em Viseu”)
    - Uma coleção de outros resources
    - Etc...
- Representado através de um identificador (URI)
  - http://omeuservidor.com/heroi/123

Naming Resources

- REST utiliza o URI para identificar os recursos
  - http://omeuservidor.com/comida/
  - http://omeuservidor.com/comida/123
  - http://omeuservidor.com/clientes
- Quanto mais se avança no caminho, vai-se do mais genérico ao mais específico.

Verbos

- Representam as acções para ser feitas nos recursos
  - Correspondem às operações CRUD
- HTTP Get <> READ
- HTTP Post <> CREATE
- HTTP PUT <> UPDATE
- HTTP DELETE <> DELETE

Identifique

HTTP GET www.omeuservidor.com/users/

Devolve a lista de utilizadores

HTTP GET www.omeuservidor.com/users/1

Devolve o utilizador 1

HTTP GET www.omeuservidor.com/heroi/12

Devolve o super-herói 12

HTTP GET www.omeuservidor.com/heroi/12/super_power

Devolve o super_power do herói 12

Identifique

HTTP DELETE www.omeuservidor.com/users/1

Elimina o utilizador 1

HTTP DELETE www.omeuservidor.com/heroi/12

Elimina o super-herói 12

HTTP POST www.omeuservidor.com/users

Adiciona um novo utilizador

HTTP PUT www.omeuservidor.com/users/1

Atualiza os dados do utilizador com o ID 1

Representações

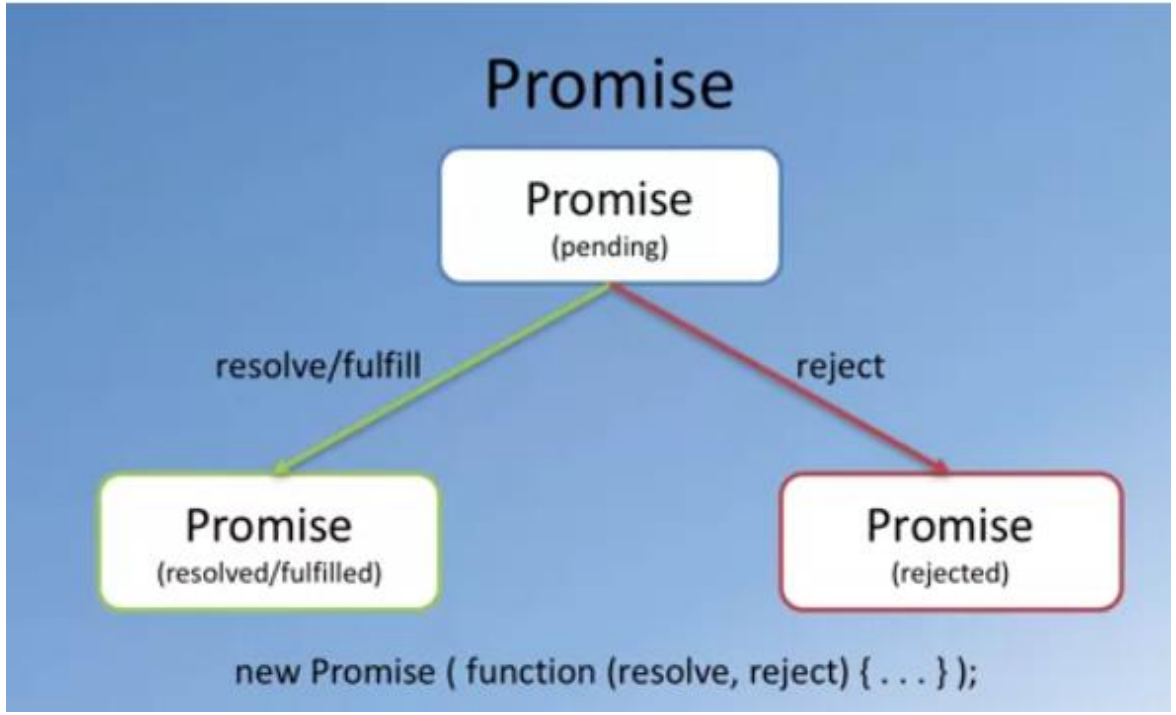
- Os dados podem ser devolvidos ao cliente nos seguintes formatos:
  - JSON
  - XML
- É comum haver múltiplas representações para o mesmo tipo de dados
  - O cliente pede os dados no formato em que quiser

Stateless Server

- O servidor não deve decorar o estado do cliente
  - Todos os requests são um novo request para o cliente
- Os clientes é que devem gerir o seu próprio estado
  - Usando cookies, base de dados do lado do cliente, etc
  - Cada request deve incluir informação suficiente para o servidor devolver a informação pedida
- No caso do React, usa-se o **state** para gerir o estado da aplicação

Promises

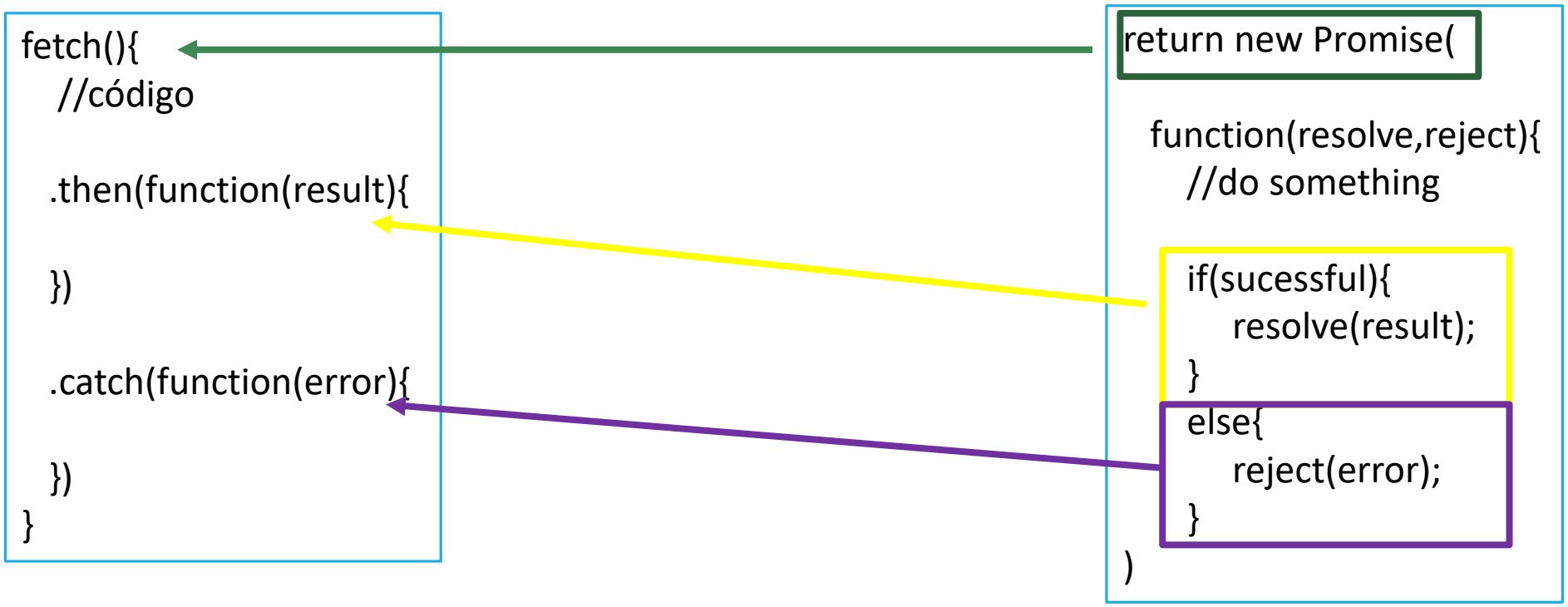
- Mecanismo que suporta a computação assíncrona
- Representa um valor que:
  - Se sabe agora
  - Só se sabe no futuro
  - Nunca se vai saber



Porque usar Promises?

- Para que a aplicação não fique à espera de uma resposta (indicamos apenas o que fazer quando ela chegar, se chegar)
- Resolve o problema do “callback hell” (callbacks encadeadas)
- Os promises podem ser sequenciais (faz isto, depois isto, depois aquilo)
- Pode imediatamente devolver:
  - Promise.resolve(result)
  - Promise.reject(error)

Exemplo de um Promise



Fetch

- XMLHttpRequest() está demasiado desatualizado
  - Foi envolvido em basicamente todas as bibliotecas Javascript
- A **API Fetch** é uma substituição moderna ao XMLHttpRequest()
  - Fornece uma interface para fazer fetch a recursos
  - Mais poderoso e flexível
  - Baseado em promises

Abracções do Fetch

- **Request** : Representa um request a um recurso
- **Response**: Representa a resposta a um request
- **Headers**: Representa os cabeçalhos do request/response, permitindo que s faça uma query a eles e fazer acções diferentes dependendo dos resultados
- **Body**: Fornece métodos relacionados com o corpo do request/response, permitindo declarar qual é o tipo de conteúdo e como é que deve ser usado

Utilização do Fetch

- Fetch(baseUrl + 'users')
  - .then(response => response.json())
  - .then(data => console.log(data))
  - .catch(error => console.log(error.message));

Resultado em Raw. Se soubermos que é JSON, podemos logo converter

Caso o response.json() tenha falhado, a execução salta para o catch

Caso contrário, salta para o then

O **catch** não é obrigatório. Apenas o **then** é.

Exemplo de um Fetch GET

```
const BASE_URL = "https://us-central1-league-of-heroes-backoffice.cloudfunctions.net/api/";
```

```
export function GetUsers() {
    return fetch(BASE_URL + "Users/", {
        method: 'GET',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
    })
    .then(response => {
        if (response.status !== 200) {
            return []
        }
        return response.json()
    });
}
```

```
componentDidMount() {
    GetUsers()
    .then(res => {
        this.setState({ list_users: res })
    })
}
```

Exemplo de um Fetch POST

```
const BASE_URL = "https://us-central1-league-of-heroes-backoffice.cloudfunctions.net/api/";
```

```
export function UpdateSuperhero(id, list) {
    return fetch(BASE_URL + "Users/" + PRIVATE_ID, {
        method: 'POST',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(list)
    })
    .then(response => {
        if (response.status !== 200) {
            return []
        }
        return response.json()
    });
}
```

```
removeHero = (elementId) => {
    //código de eliminar o super-heroi e o
remover da lista de top
    UpdateSuperhero(PUBLIC_ID,
this.state.list_of_heroes);
    UpdateTop(PUBLIC_ID,
this.state.list_of_heroes);
}
```

Ficha 5

Leitura Complementar

- https://reactjs.org/docs/faq-ajax.html
- https://www.robinwieruch.de/react-fetching-data (Até à secção How to test data fetching in React?)