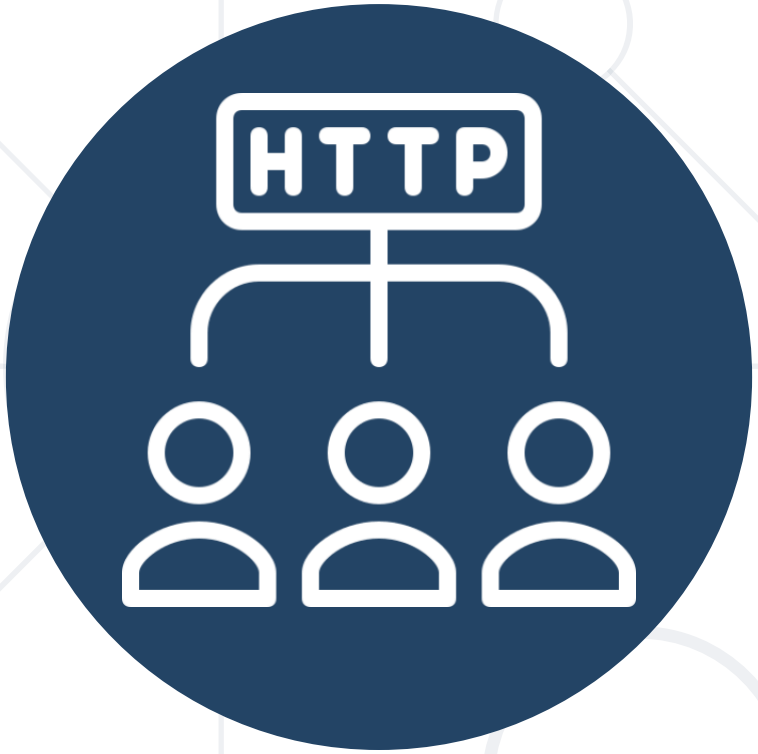
Table of Contents

Have a Question?

Software University

sli.do

#js-front-end

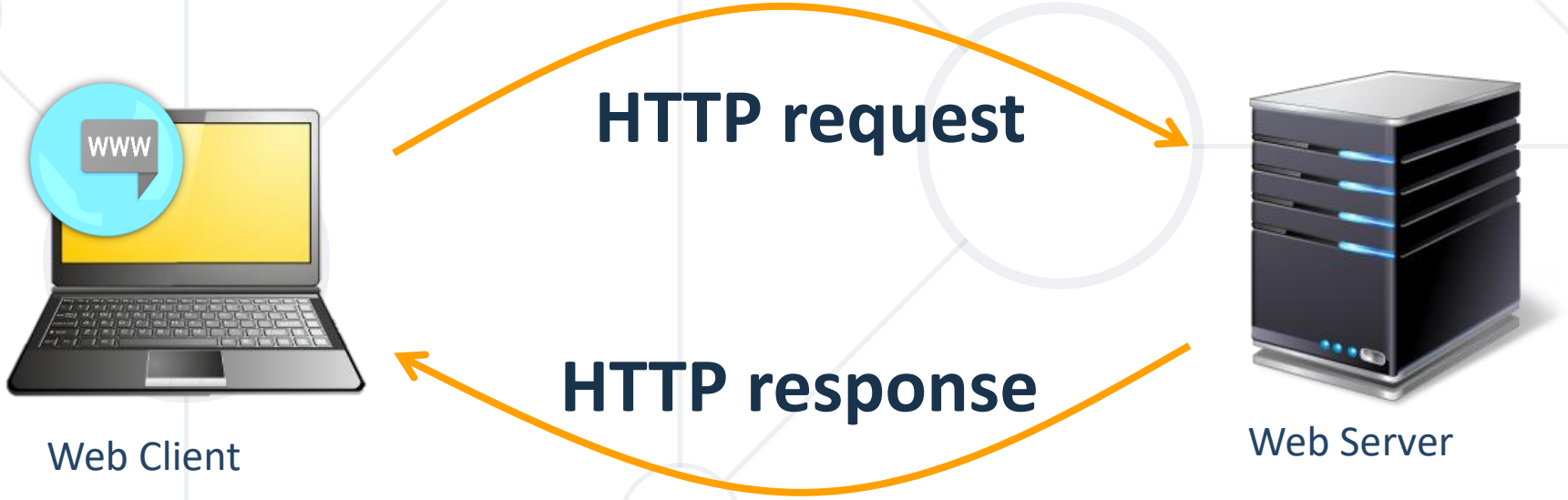


HTTP Overview

Hypertext Transfer Protocol

HTTP Basics

- HTTP (**H**yper **T**ext **T**ransfer **P**rotocol)
  - Text-based client-server protocol for the Internet
  - For transferring Web resources (HTML files, images, styles, etc.)
  - Request-response based



HTTP Request Methods

- **HTTP** defines **methods** to indicate the desired action to be performed on the identified resource

Method	Description
GET	Retrieve / load a resource
POST	Create / store a resource
PUT	Update a resource
DELETE	Delete (remove) a resource
PATCH	Update resource partially
HEAD	Retrieve the resource's headers
OPTIONS	Returns the HTTP methods that the server supports for the specified URL

HTTP GET Request – Example

```
GET /users/testnakov/repos HTTP/1.1
Host: api.github.com
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36
Connection: Keep-Alive
Cache-Control: no-cache
<CRLF>
```

HTTP request line

HTTP headers

The request body is empty

HTTP POST Request – Example

```
POST /repos/testnakov/test-nakov-repo/issues HTTP/1.1
Host: api.github.com
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
<CRLF>
{"title":"Found a bug",
 "body":"I'm having a problem with this.",
 "labels":["bug","minor"]}
<CRLF>
```

HTTP request line

HTTP headers

The request body holds the submitted data

HTTP Response – Example

```
HTTP/1.1 200 OK
Date: Fri, 11 Nov 2016 16:09:18 GMT+2
Server: Apache/2.2.14 (Linux)
Accept-Ranges: bytes
Content-Length: 84
Content-Type: text/html
<CRLF>
<html>
  <head><title>Test</title></head>
  <body>Test HTML page.</body>
</html>
```

HTTP response status line

HTTP response headers

HTTP response body

HTTP Response Status Codes

Status Code	Action	Description
200	OK	Successfully retrieved resource
201	Created	A new resource was created
204	No Content	Request has nothing to return
301 / 302	Moved	Moved to another location (redirect)
400	Bad Request	Invalid request / syntax error
401 / 403	Unauthorized	Authentication failed / Access denied
404	Not Found	Invalid resource
409	Conflict	Conflict was detected, e.g. duplicated email
500 / 503	Server Error	Internal server error / Service unavailable

Content-Type and Disposition

- The **Content-Type** / **Content-Disposition** headers specify how the HTTP request / response body should be processed

JSON-encoded data

```
Content-Type: application/json
```

UTF-8 encoded HTML page. Will be shown in the browser

```
Content-Type: text/html; charset=utf-8
```

```
Content-Type: application/pdf

Content-Disposition: attachment;
filename="Financial-Report-April-2016.pdf"
```

This will download a PDF file named Financial-Report-April-2016.pdf

Browser Developer Tools

Elements Console Sources Network Timeline Profiles Application Security Audits

View: Preserve log Disable cache Offline No throttling

Filter Regex Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name
wikipedia.org
www.wikipedia.org
Wikipedia_wordmark@1.5x.png
Wikipedia-logo-v2@1.5x.png
sprite-bookshelf_icons@1.5x.png?89f815b53cd5e1c90bdbf33cab3d7dd7315c1542
index-abc278face.js
gt-ie9-c84bf66d33.js
sprite-project-logos@1.5x.png?7a6de5f6f8800ddf02f77b943e0074fb541294db
bg-20c36dfd.json
wikipedia.ico

10 requests | 75.5 KB transferred | Finish: 471 ms | DOMContentLoaded: 41...

Headers Preview Response Cookies Timing

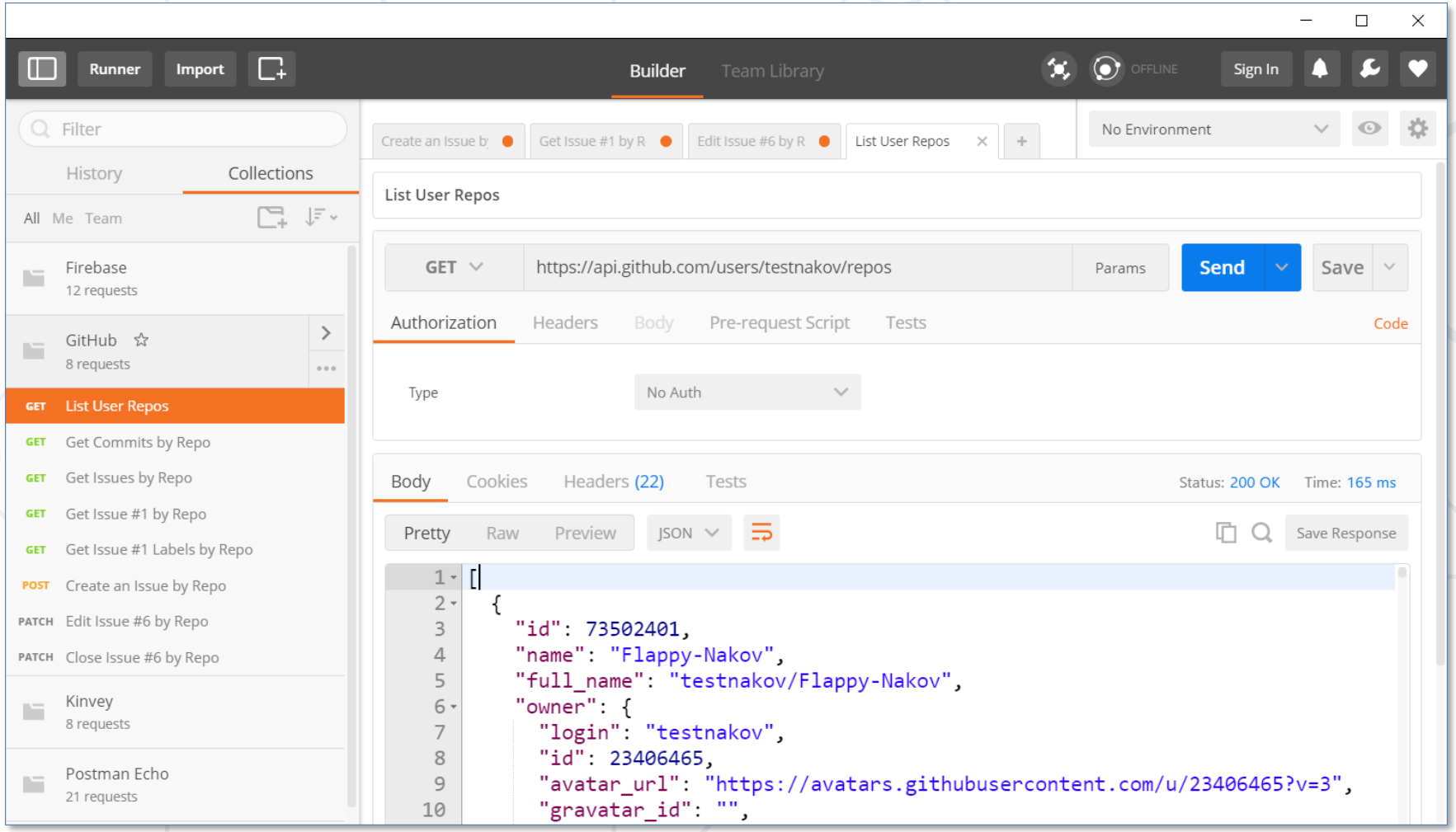
General

Request URL: https://www.wikipedia.org/
Request Method: GET
Status Code: 200
Remote Address: 91.198.174.192:443

Response Headers

accept-ranges: bytes
age: 40884
backend-timing: D=216 t=1478774829320624
cache-control: s-maxage=86400, must-revalidate, max-age=3600
content-encoding: gzip
content-length: 18969
content-type: text/html
date: Fri, 11 Nov 2016 22:08:33 GMT
etag: W/"13d53-5406d9c558709"
last-modified: Thu, 03 Nov 2016 23:06:42 GMT
server: mw1255.eqiad.wmnet
set-cookie: CP=H2; Path=/; secure
set-cookie: WMF-Last-Access=11-Nov-2016;Path=/;HttpOnly;secure;Expires=Tue, 13 Dec 2016 12:00:00 GMT
status: 200
strict-transport-security: max-age=31536000; includeSubDomains; preload
vary: Accept-Encoding
via: 1.1 varnish-v4, 1.1 varnish-v4, 1.1 varnish-v4
x-analytics: https=1
x-cache: cp1054 hit/1, cp3032 hit/7, cp3043 hit/853595
x-cache-status: hit
x-client-ip: 93.123.90.93

Postman

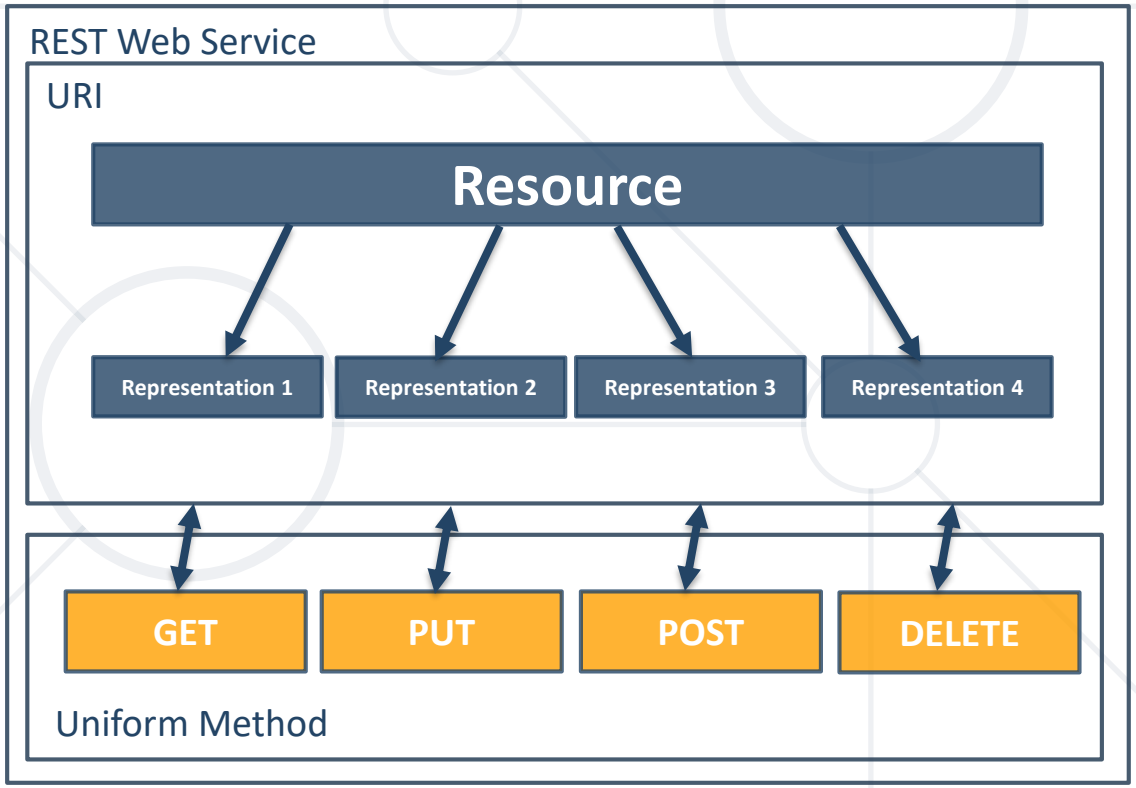


Read more about Postman REST Client

REST and RESTful Services

REST and RESTful Services

- **Re**presentational **S**tate **T**ransfer (**REST**)
  - Architecture for **client-server communication** over HTTP
  - Resources have **URI** (address)
  - Can be **created**/**retrieved**/**modified**/**deleted**/etc.
- RESTful API/RESTful Service
  - Provides access to **server-side resources** via **HTTP** and **REST**



REST and RESTful Services – Example

- Create a new post

POST	http://some-service.org/api/posts

- Get all posts / specific post

GET	http://some-service.org/api/posts

GET	http://some-service.org/api/posts/17

- Delete existing post

DELETE	http://some-service.org/api/posts/17

- Replace / modify existing post

PUT/PATCH	http://some-service.org/api/posts/17

Accessing GitHub Through HTTP

GitHub REST API

GitHub API

- List user's all public repositories:

GET	https://api.github.com/users/testnakov/repos

- Get all commits from a public repository:

GET	https://api.github.com/repos/testnakov/softuniada-2016/commits

- Get all issues/issue #1 from a public repository

GET	/repos/testnakov/test-nakov-repo/issues

GET	/repos/testnakov/test-nakov-repo/issues/1

Github: Labels Issue

- Get the first issue from the "**test-nakov-repo**" repository
- Send a **GET** request to:
  - https://api.github.com/repos/testnakov/test-nakov-repo/issues/:id
  - Where **:id** is the current issue

GitHub API (2)

- Get all labels for certain issue from a public repository:

GET	https://api.github.com/repos/testnakov/test-nakov-repo/issues/1/labels

- Create a new issue to certain repository (with authentication)

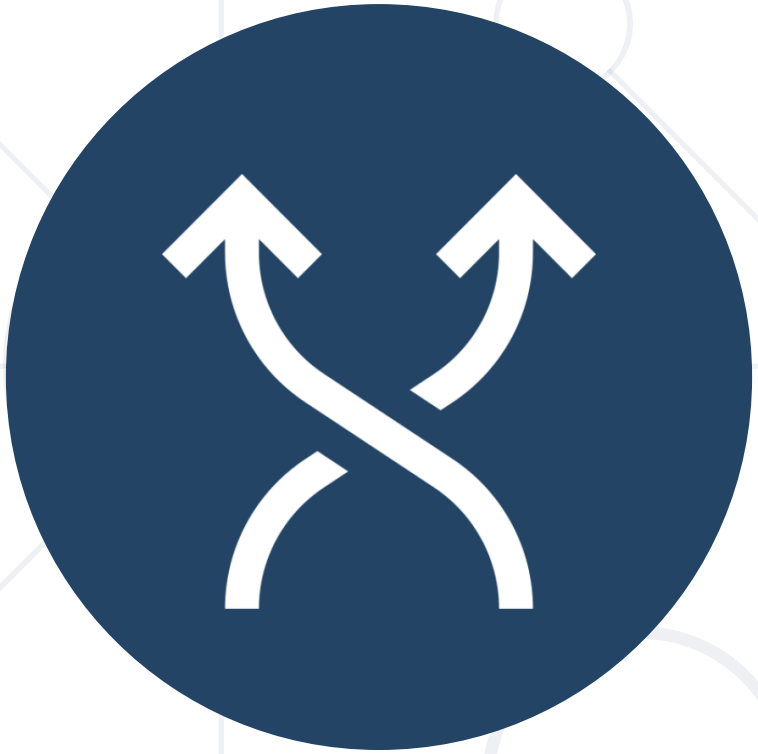
POST	https://api.github.com/repos/testnakov/test-nakov-repo/issues

Headers	Authorization: Basic base64(user:pass)

Body	{"title":"Found a bug", "body": "I'm having a problem with this."}

Github: Create Issue

- Create an issue when you send a "**POST**" request
- Use your Github account **credentials** to submit the issue



Synchronous vs Asynchronous

Asynchronous Programming

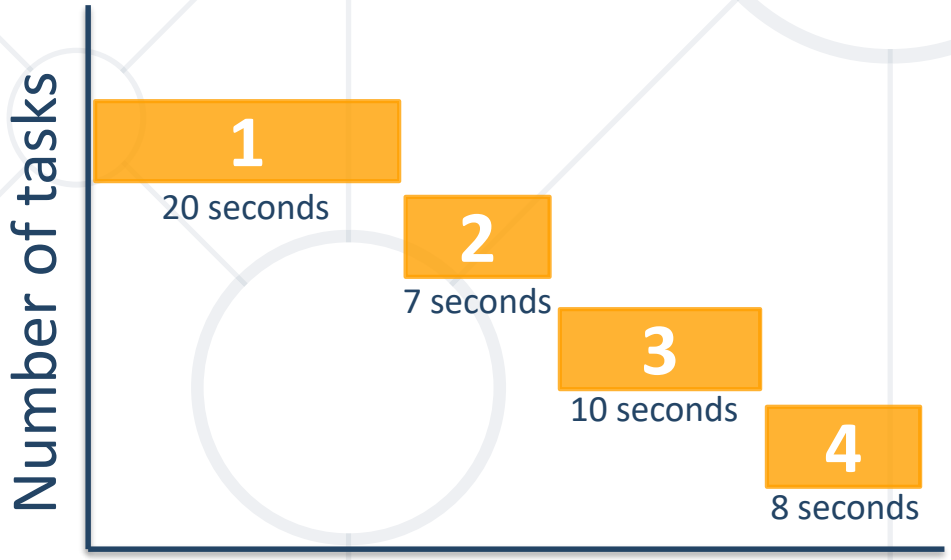
Asynchronous Programming in JS

- Structured using **callback functions**
- In current versions of JS there are:
  - **Callbacks**
  - **Promises**
  - **Async Functions**
- Not the same thing as **concurrent** or **multi-threaded**
- **JS code** is generally **single-threaded**

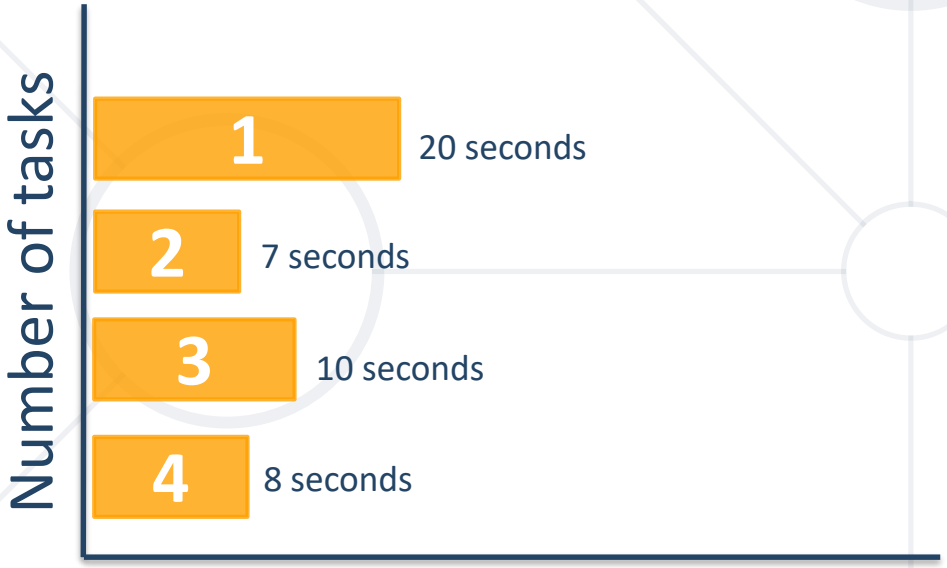
Asynchronous Programming

- Runs several tasks (pieces of code) in parallel, **at the same time**

Synchronous



Asynchronous



Asynchronous Programming – Example

- The following commands will be executed as follows:

```
console.log("Hello.");

setTimeout(function() {
  console.log("Goodbye!");
}, 2000);

console.log("Hello again!");
```

```
// Hello.
```

```
// Hello again!
```

```
// Goodbye!
```

Callbacks

- Function **passed** into another function as an **argument**
- Then **invoked** inside the outer function to complete some kind of routine or action

```
function running() {
    return "Running";
}
function category(run, type) {
    console.log(run() + " " + type);
}
category(running, "sprint"); //Running sprint
```

Callback function

Promises

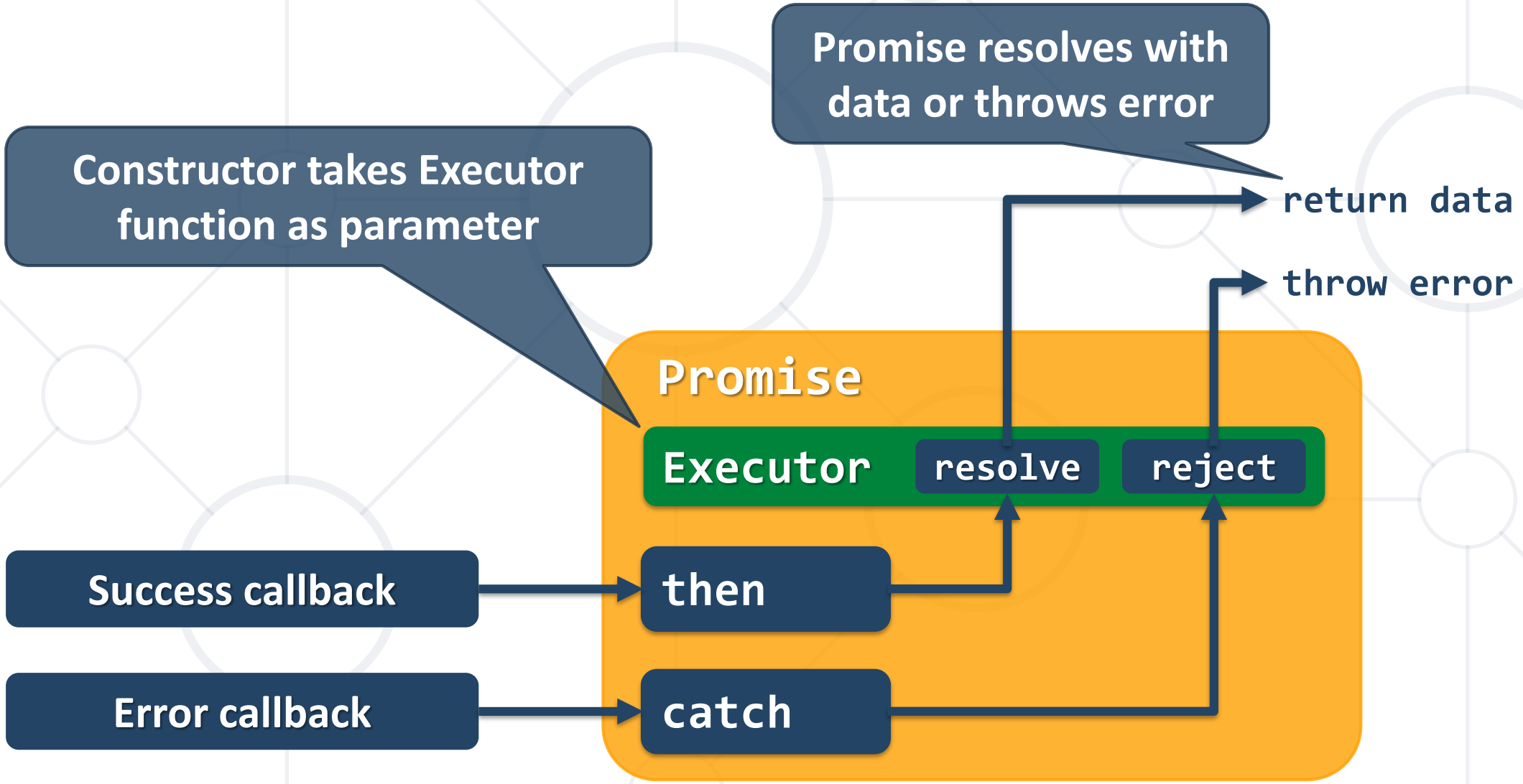
Objects Holding Asynchronous Operations

What is a Promise?

- A promise is an **asynchronous action** that **may complete** at some point and **produce a value**
- States:
  - **Pending** - operation still running (unfinished)
  - **Fulfilled** - operation finished (the result is available)
  - **Failed** - operation failed (an error is present)
- Promises use the **Promise class**

```
new Promise(executor);
```

Promise Flowchart



Promise.then() – Example

```
console.log('Before promise');
```

```
new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('done');
  }, 500);
})
.then(function(res) {
  console.log('Then returned: ' + res);
});
```

Resolved after 500 ms

```
console.log('After promise');
```

```
// Before promise
```

```
// After promise
```

```
// Then returned: done
```

Promise.catch() – Example

```
console.log('Before promise');
```

```
new Promise(function (resolve, reject) {
  setTimeout(function () {
    reject('fail');
  }, 500);
})
  .then (function (result) { console.log(result); })
  .catch (function(error) { console.log(error); });
```

Rejected after 500 ms

```
console.log('After promise');
```

Popular Promise Methods

- `Promise.reject`(reason)
  - Returns an **object** that is **rejected** with the given **reason**
- `Promise.resolve`(value)
  - Returns an object that is **resolved** with the given **value**
- `Promise.finally`()
  - The handler is called when the promise is settled
- `Promise.all`(iterable)
  - Returns a **promise**
    - Fulfills when **all** of the promises **have fulfilled**
    - Rejects as soon as **one** of them **rejects**

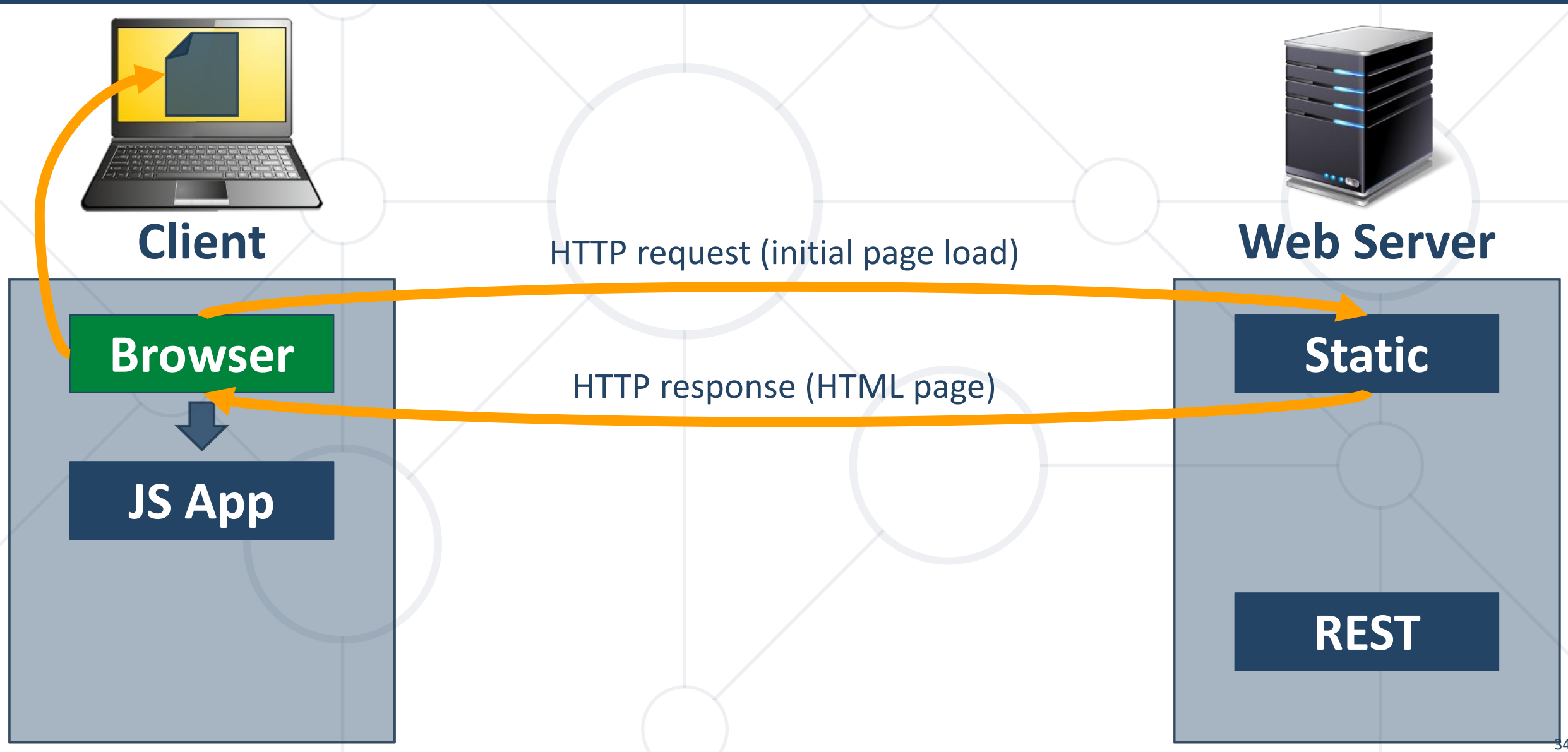
AJAX

Connecting to a Server via Fetch API

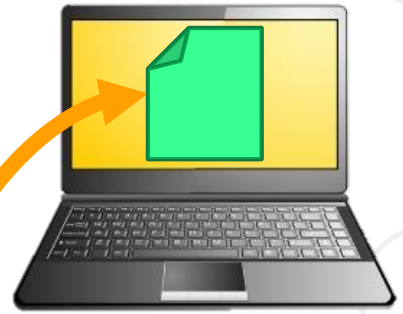
What is AJAX?

- **Asynchronous JavaScript And XML**
  - Background loading of **dynamic content/data**
  - Load data from the Web server and **render** it
- Some **examples** of AJAX usage:
  - **Partial page rendering**
    - Load HTML fragment + show it in a ** **
  - **JSON service**
    - Loads JSON object and displays it

AJAX: Workflow



AJAX: Workflow



Client

Web Server

Browser

JS App

UI Interaction

AJAX handler

AJAX request

Returns data as JSON

Static

REST

What is Fetch?

- The **Fetch API**:
  - Allows making network requests
  - Uses **Promises**
  - Enables a **simpler** and **cleaner** API
  - Makes code more readable and maintainable

```
fetch('./api/some.json')
  .then(function(response) {…})
  .catch(function(err) {…})
```

Basic Fetch Request

- The response of a **`fetch()`** request is a **Stream** object
- The **reading** of the stream happens **asynchronously**
- When the **`json()`** method is called, a **Promise** is **returned**
  - The **response status** is checked (should be **200**) **before parsing** the response as **JSON**

```
if (response.status !== 200) {
  // handle error
}
response.json()
  .then(function(data) { console.log(data)})
```

GET Request

- **Fetch API** uses the **GET** method so that a direct call would be like this

```
fetch('https://api.github.com/users/testnakov/repos')
  .then((response) => response.json())
  .then((data) => console.log (data))
  .catch((error) => console.error(error))
```

Problem: GitHub Repos

- Execute an **AJAX GET** Request to load **all repos** of a **user**
- Use the **Fetch API**
- Use the following **URL**:
  - https://api.github.com/users/testnakov/repos
- In the **first then()** block map the response to **text**
- In the **second then()** block **display** the content in a **div**

POST Request

- To make a **POST** request, we can set the **method** and **body** parameters in the **fetch()** options

```
fetch('/url', {
    method: 'post',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(data),
})
```

PUT Request

```
fetch('/url/:id', {
    method: 'put',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(data),
})
```

PATCH Request

```
fetch('/url/:id', {
    method: 'patch',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(data),
})
```

DELETE Request

```
fetch('/url/:id', {
    method: 'delete',
})
```

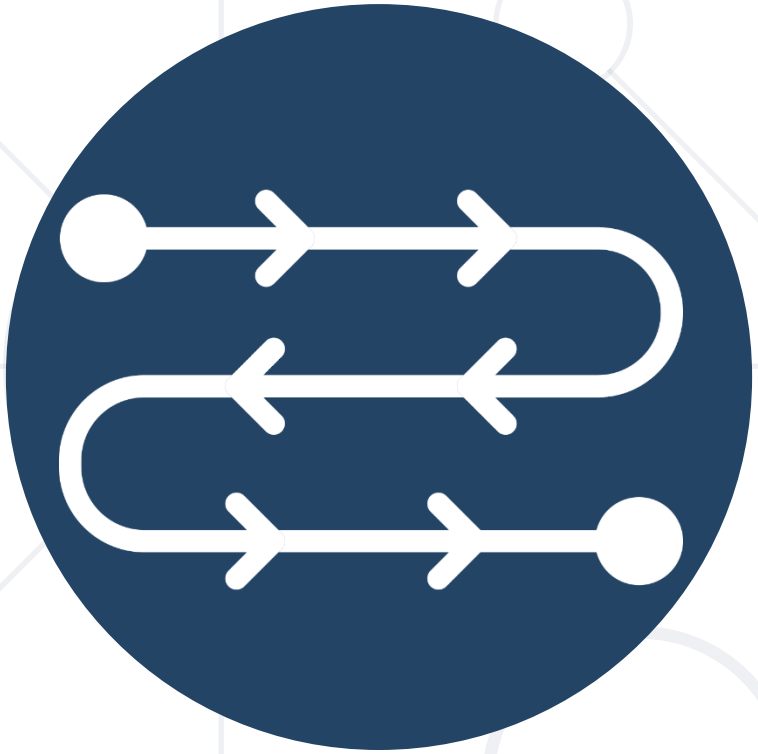
Problem: Load GitHub Commits

```
GitHub username:
<input type="text" id="username" value="nakov" /> <br>
Repo: <input type="text" id="repo" value="nakov.io.cin" />
<button onclick="loadCommits()">Load Commits</button>
<ul id="commits"></ul>
<script>
  function loadCommits() {
    // Use Fetch API
  }
</script>
```

GitHub username: nakov

Repo: nakov.io.cin Load Commits

- Svetlin Nakov: Delete Console.Cin.v11.suo
- Svetlin Nakov: Create LICENSE
- Svetlin Nakov: Update README.md
- Svetlin Nakov: Added better documentation



Async / Await

ES6 Simplified Promises

Async Functions

- Returns a **promise**, that can await other promises in a way that **looks synchronous**
- Contains an **await** expression that:
  - Is **only valid** inside **async functions**
  - **Pauses** the execution of that function
  - Waits for the Promise's **resolution**

Async Functions (2)

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
```

```
Expected output:
// calling
// resolved
```

```
async function asyncCall() {
  console.log('calling');
  let result = await resolveAfter2Seconds();
  console.log(result);
}
```

Error Handling

```
async function f() {
  try {
    let response = await fetch();
    let user = await response.json();
  } catch (err) {
    // catches errors both in fetch andresponse.json
    alert(err);
  }}
```

```
async function f() {
  let response = await fetch();
}
// f() becomes a rejected promise
f().catch(alert);
```

Async/Await vs Promise.then

- Promise.then

```
function logFetch(url) {
  return fetch(url)
    .then(response => {
      return response.text()
    })
    .then(text => {
      console.log(text);
    })
    .catch(err => {
      console.error(err);
    });
}
```

- Async/Await

```
async function logFetch(url) {
  try {
    const response =
       await fetch(url);
    console.log(
      await response.text()
    );
  }
  catch (err) {
    console.log(err);
  }
}
```

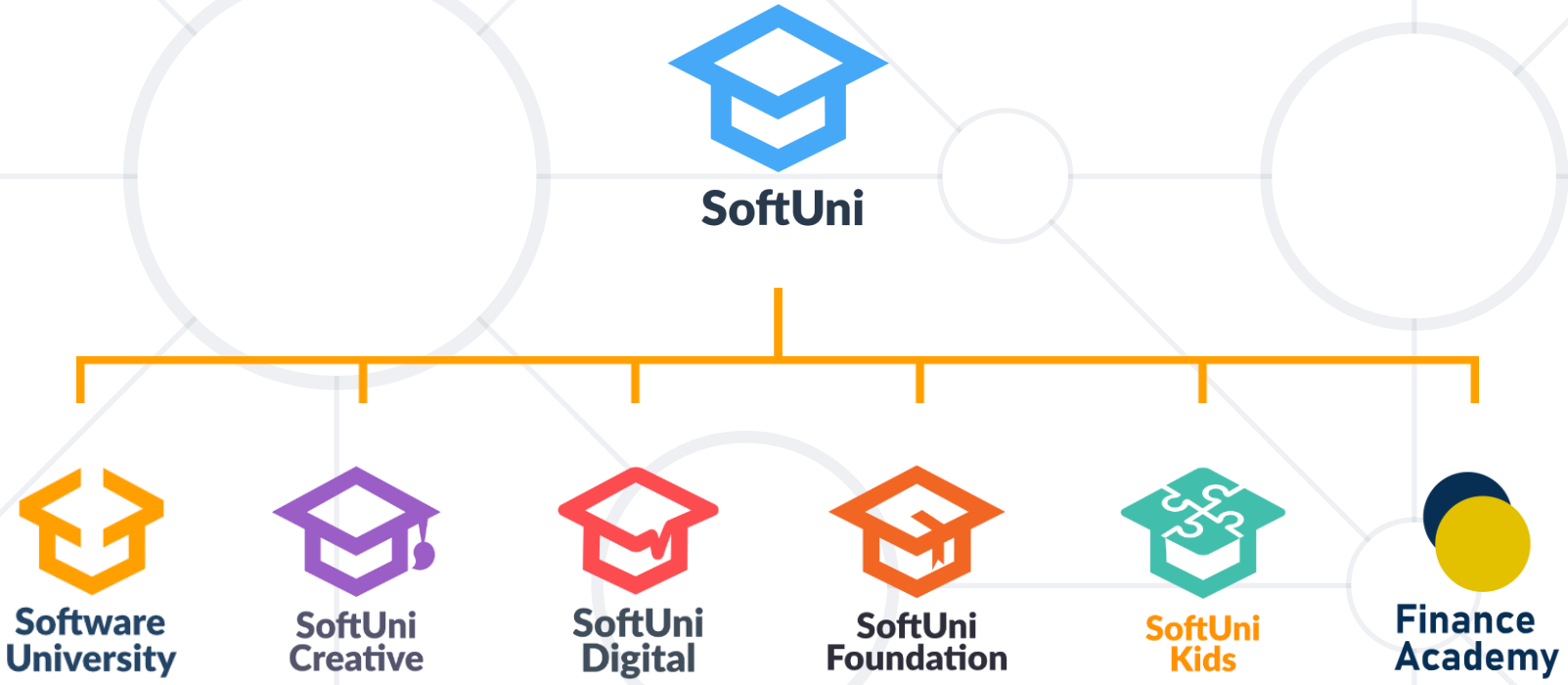
Summary

- **HTTP** is text-based request-response protocol
- **RESTful** services address resources by URL
  - Provide **CRUD** operations over HTTP
- **Asynchronous** programming
- **Promises** hold operations – **resolve** & **reject**
- **AJAX** & **Fetch** API – connect to a server
- ES6 **Async**/**Await** Expression

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

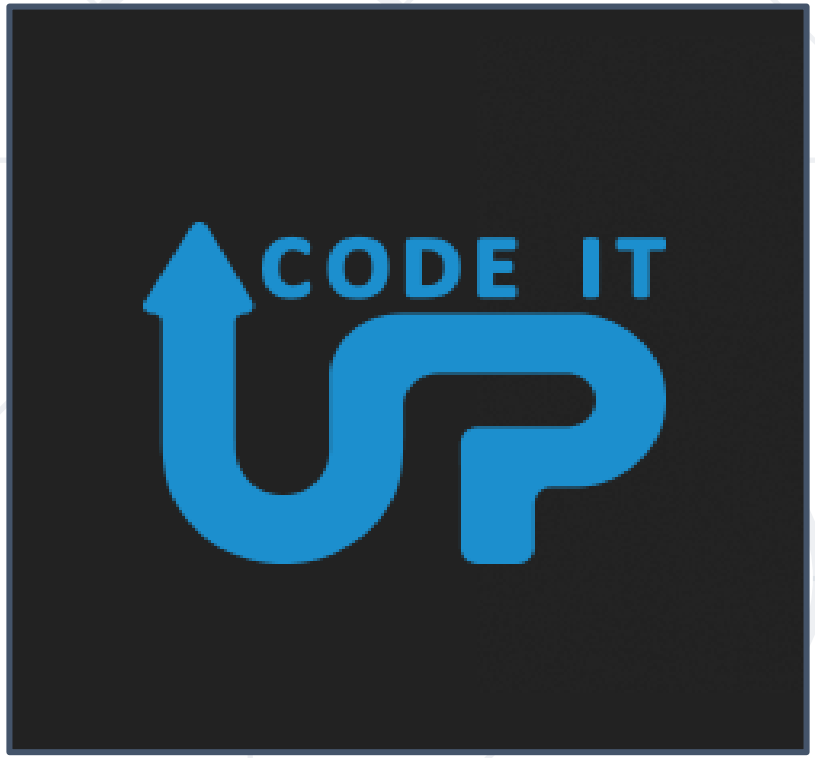
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg