Asynchronous Programming and Promises

Fetch API, Promises, async/await

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

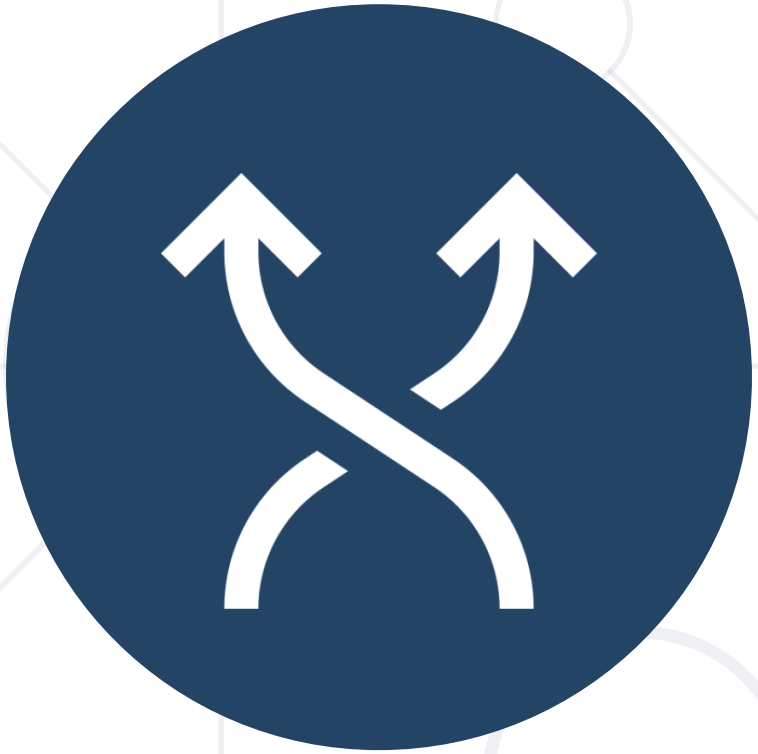
Have a Question?

Software University

sli.do

#js-front-end

Table of Contents



Synchronous vs Asynchronous

Asynchronous Programming

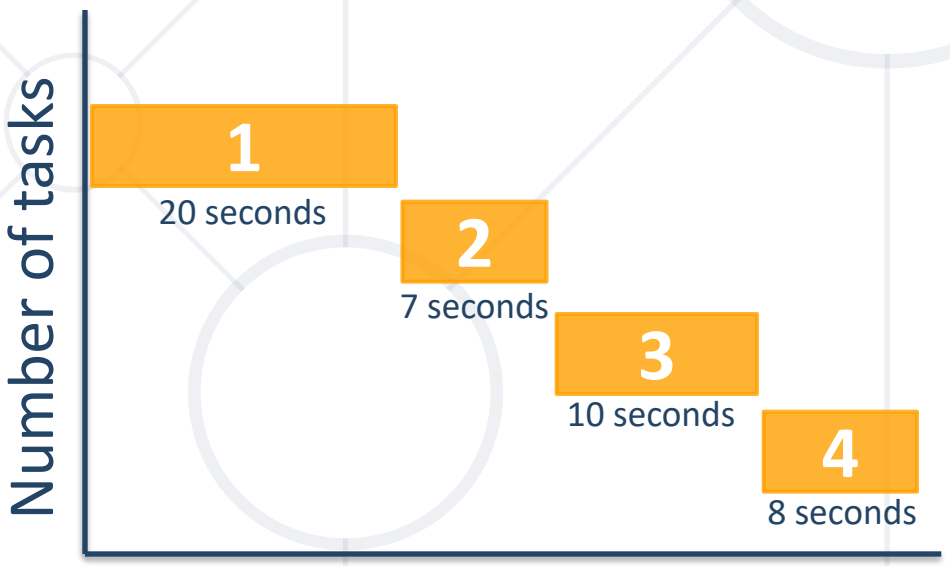
Asynchronous Programming in JS

- 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**

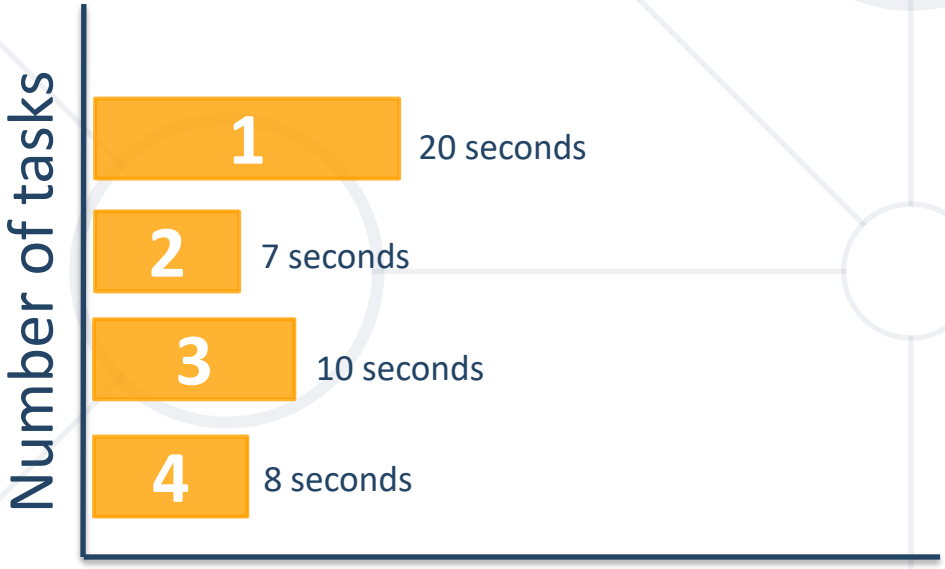
Asynchronous Programming

- 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

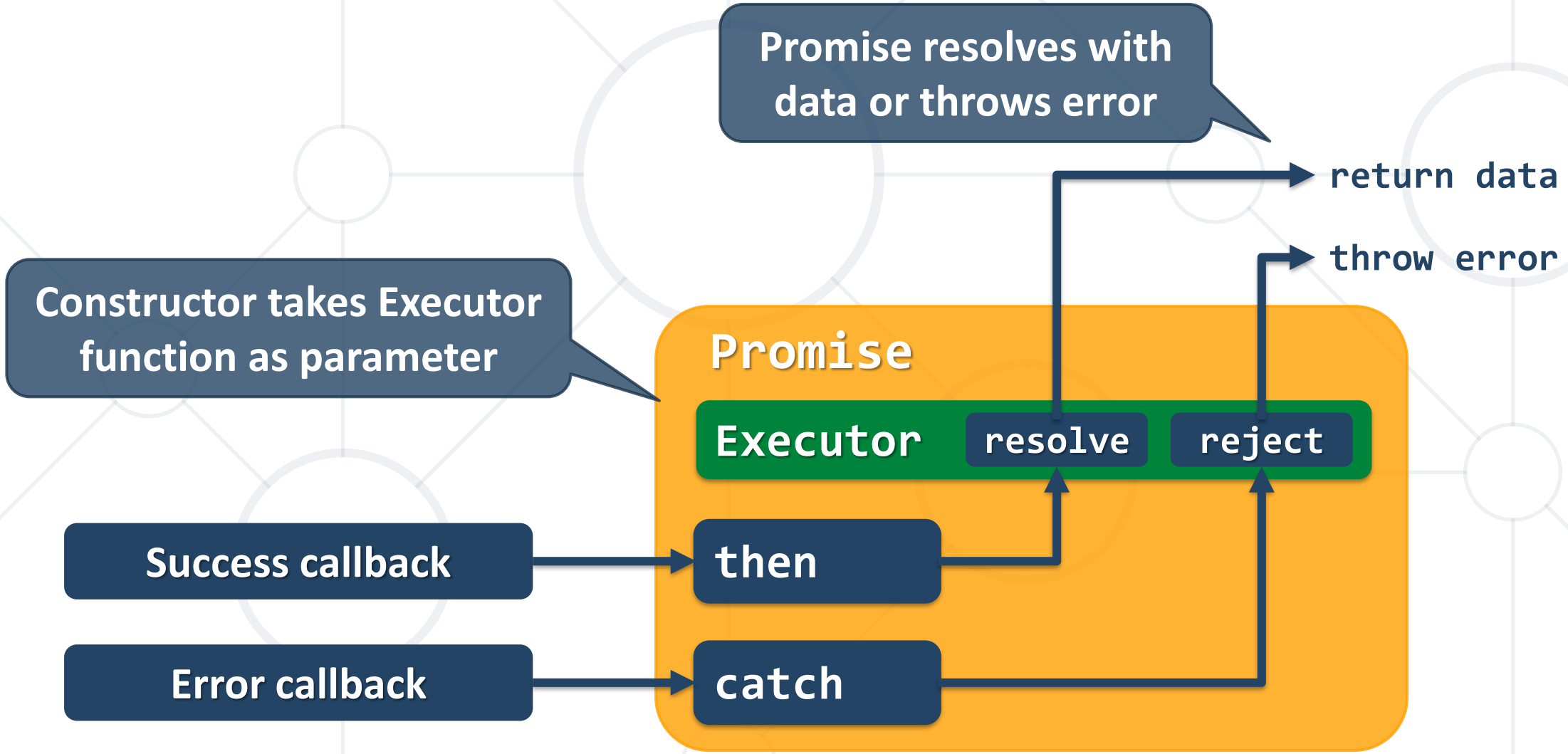
Objects Holding Asynchronous Operations

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);
});

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

Resolved after 500 ms

```
// 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

Client

Web Server

HTTP request (initial page load)

HTTP response (HTML page)

Browser

JS App

Static

REST

AJAX: Workflow

Client

Web Server

Browser

JS App

UI Interaction

AJAX handler

Static

REST

AJAX request

Returns data as JSON

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

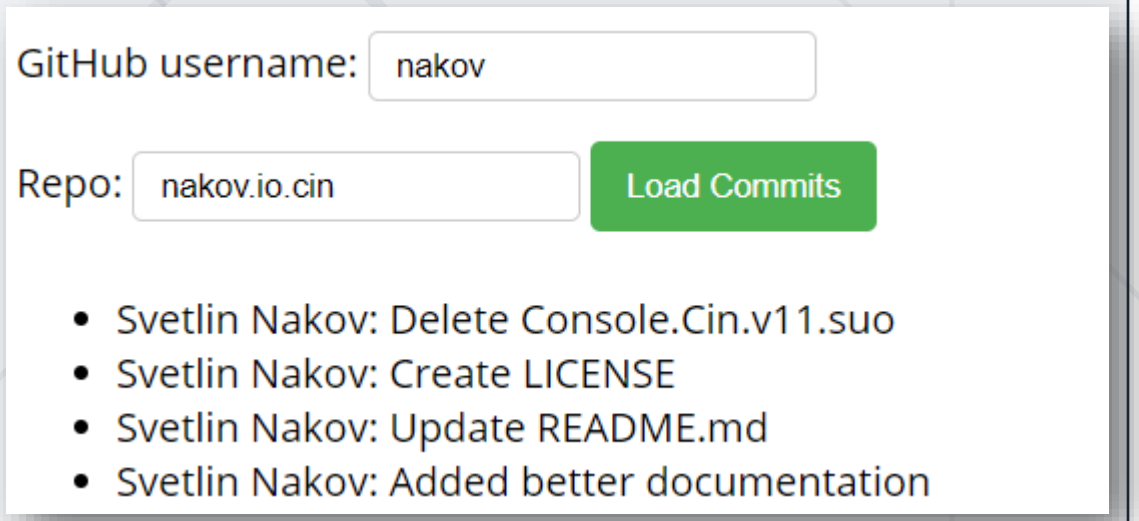
```
fetch('/url/:id', {
    method: 'patch',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(data),
})
```

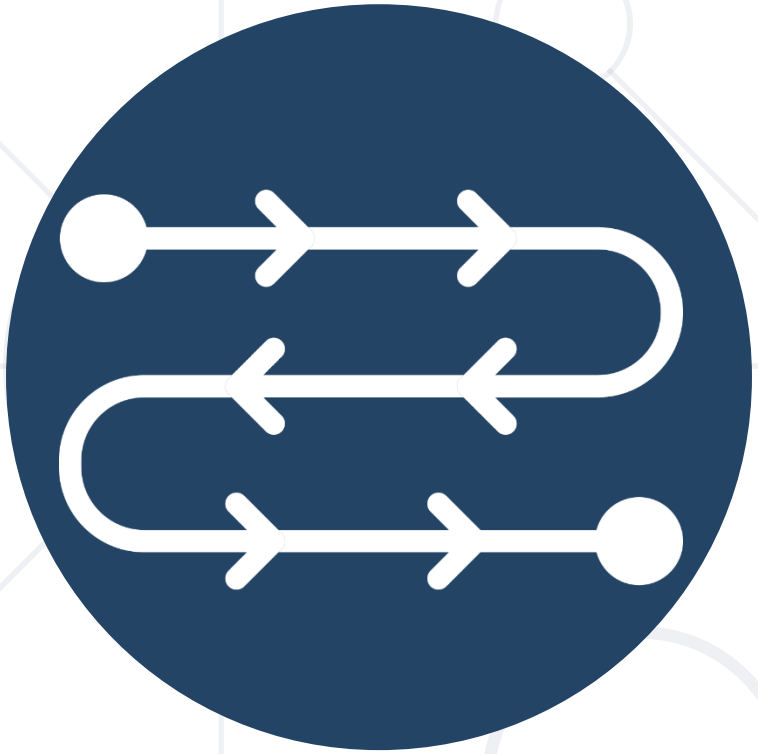
DELETE Request

```
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>
```





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

```
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);
    });
}
```

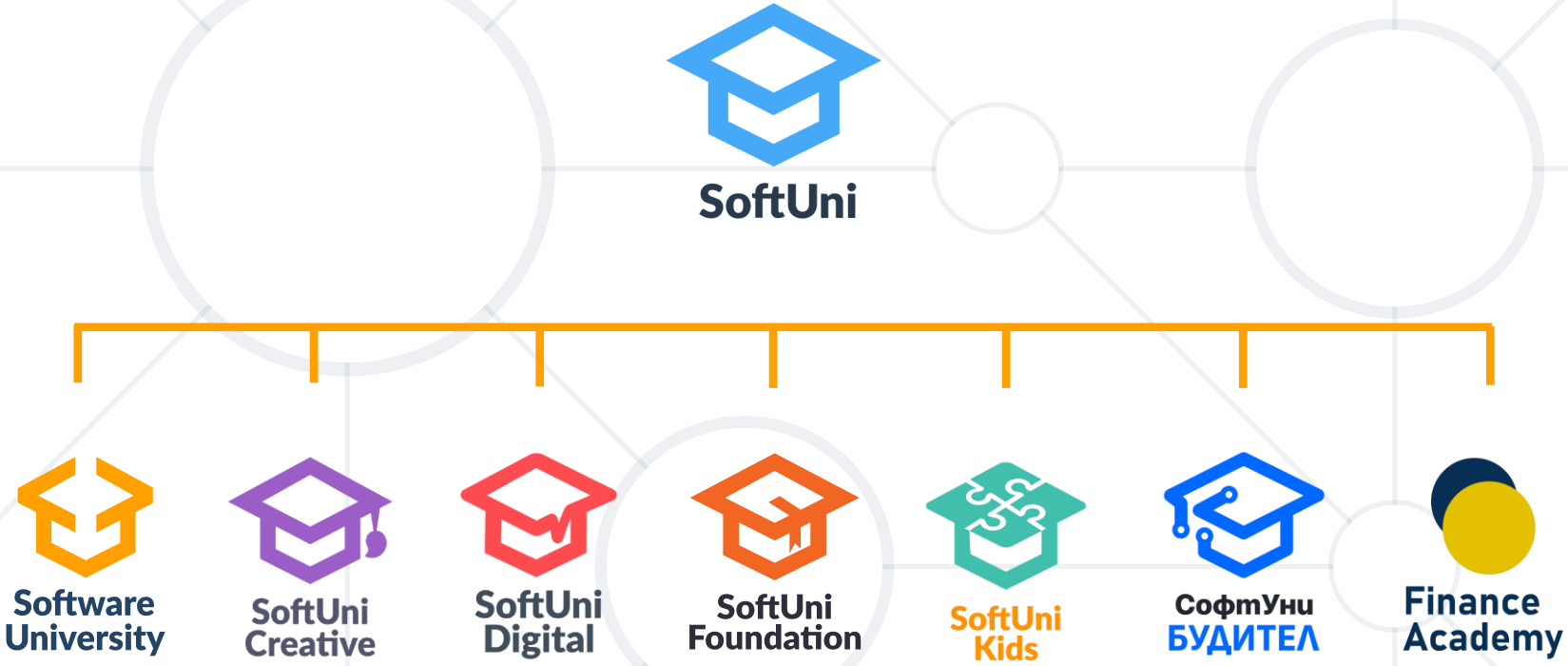
- Async / Await

```
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

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

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

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