Asynchronous Programming and Promises

Promises. Async / Await

SoftUni Team
Technical Trainers

Software University

SoftUni Foundation

Software University
http://softuni.bg

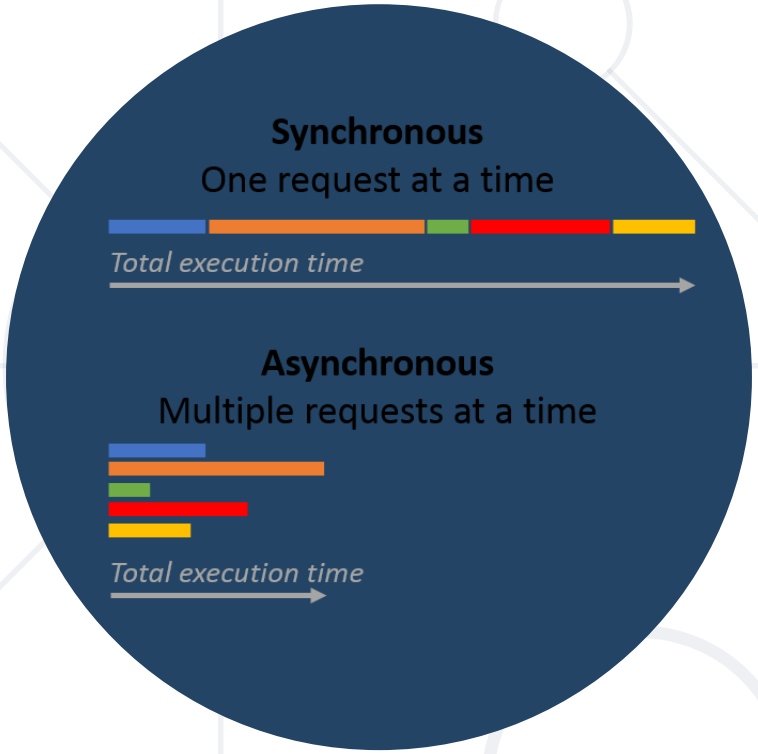
Table of Contents

Have a Question?

SoftUni Foundation

sli.do

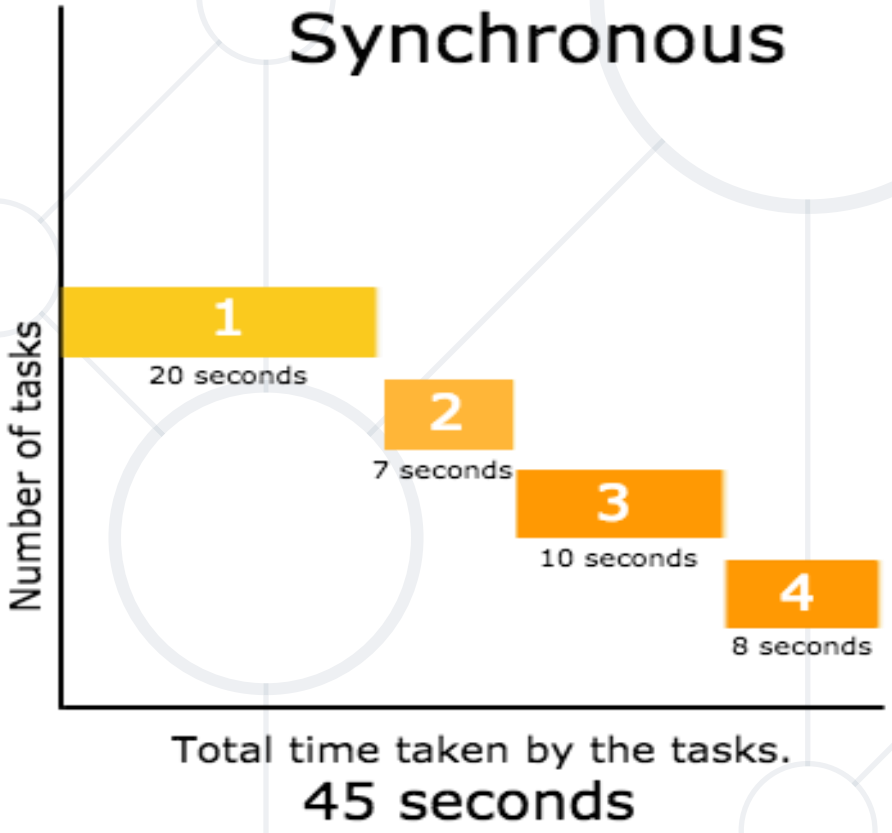
#JSCORE

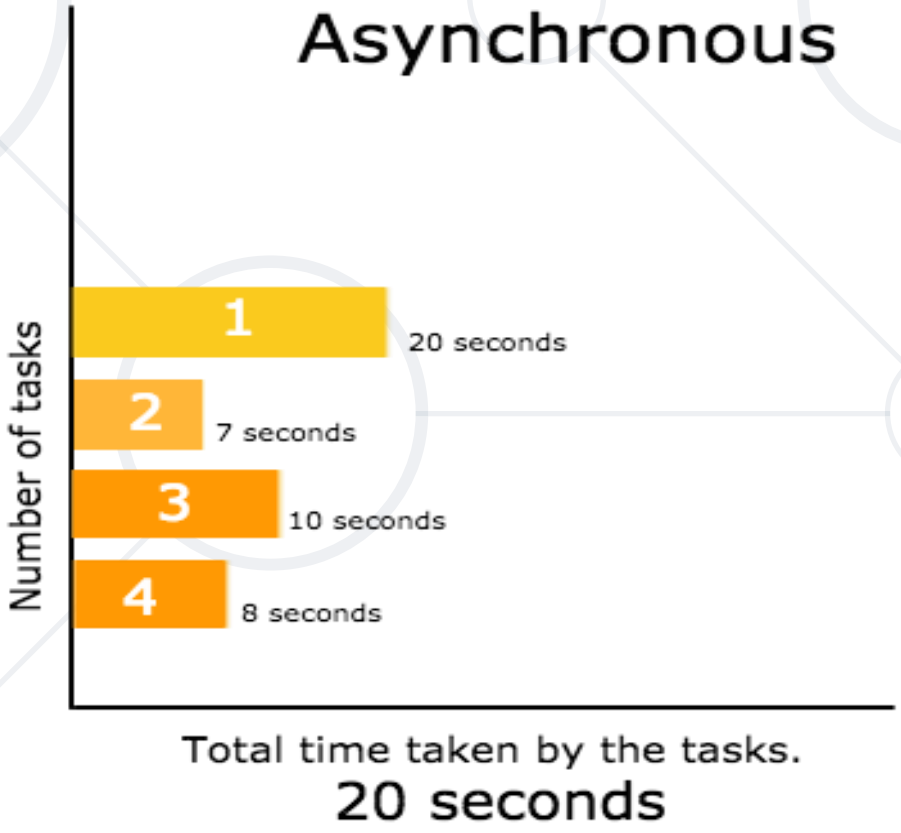


Asynchronous Programming

Asynchronous Programming

- **Asynchronous programming** deals with the needs to run several tasks (pieces of code) in parallel, in the same time





Asynchronous Programming

SoftUni Foundation

- Synchronous programming
  - Each line of code is executed only when the preceding line has finished it's work
- Asynchronous Programming
  - In asynchronous programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes
  - The tasks may finish in order of execution but it is not guaranteed

Asynchronous Programming – Example

SoftUni Foundation

```javascript
// Say "Hello."
console.log("Hello.");
// Say "Goodbye" two seconds from
now.
setTimeout(function() {
  console.log("Goodbye!");
}, 2000);
// Say "Hello again!"
console.log("Hello again!");
```

```javascript
function getData() {
  var data;
  $.get("example.php",
    function(response) {
      data = response;
    }
  );
  return data;
}

var data = getData();
console.log("The data is: " +
data);
```

- What is wrong with these examples?

Asynchronous Programming in JS

SoftUni Foundation

- Asynchronous does not mean the same thing as concurrent or multi-threaded
- JavaScript can have asynchronous code, but it is **generally single-threaded** (except when using Web Workers)
- Asynchronous code needs to be structured in a different way than synchronous code, and the most basic way to do that is with callback functions
- In current versions of JavaScript we have two more options
  - Promises
  - Async / Await patern

Do we need asynchronous programming in JS?

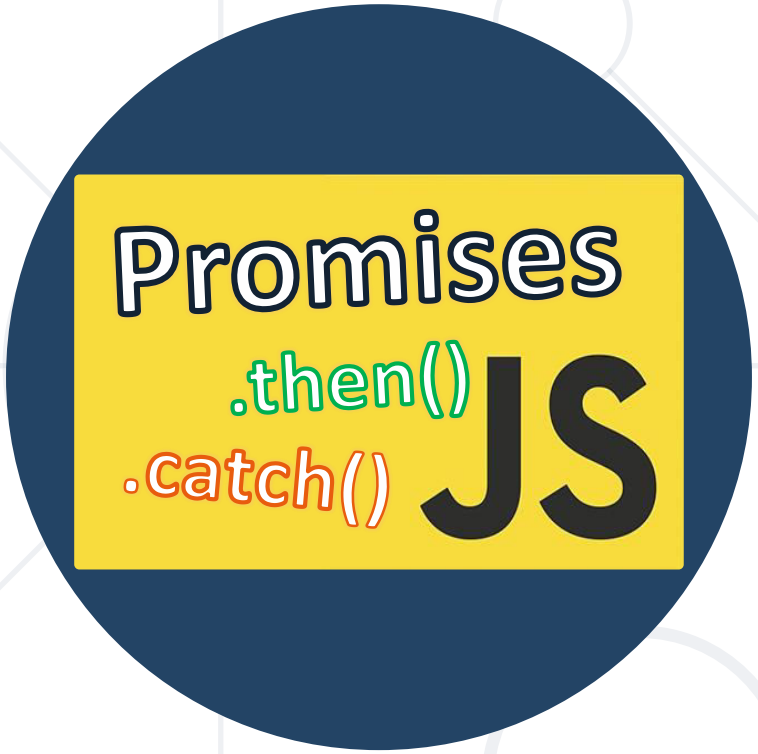
SoftUni Foundation

- Some people decide that dealing with asynchronous code is too complicated to work with, so they try to make everything synchronous
- When doing an AJAX call, it is possible to set an option to make the call synchronous rather than asynchronous (although this option is slowly losing browser support)
- There are also synchronous alternatives to many asynchronous functions in Node.js

Do we need asynchronous programming in JS?

SoftUni Foundation

- Trying to avoid asynchronous code and replacing it with synchronous code is almost always a bad idea in JavaScript
- JavaScript only has a single thread (except when using Web Workers)
- The webpage will be unresponsive while the script is running. If you use a synchronous AJAX call, then the user will not be able to do anything while they are running
- The issue is even worse when using server-side JavaScript: the server will not be able to respond to any requests while waiting for synchronous functions to complete, which means that every user making a request to the server will have to wait to get a response



Promises in JS

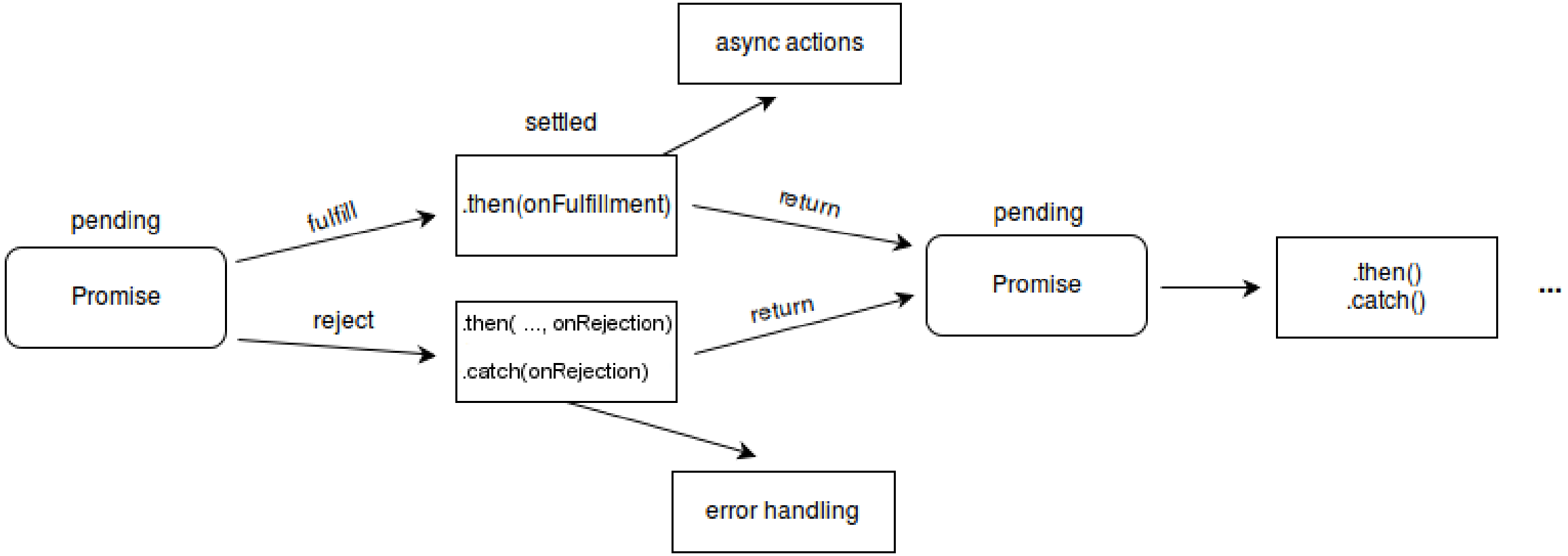
Objects Holding Asynchronous Operations

What is a Promise?

SoftUni Foundation

- A promise is an object holding an asynchronous operation
  - A result which may be available now, or in the future, or never
  - Promises let asynchronous methods return values like synchronous methods, instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future
- Promises may be in one of these states:
  - **Pending** – operation still running (unfinished)
  - **Fulfilled** – operation finished (and the result is available)
  - **Failed** – operation is failed (and an error is available)
- Promises in JS use the **Promise** object

What is a Promise?



Promise Methods

SoftUni Foundation

- Promise.all(iterable)
  - Returns a promise that either fulfills when all of the promises in the iterable argument have fulfilled or rejects as soon as one of the promises in the iterable argument rejects
  - fulfilled with an array of the values from the fulfilled promises in the same order as defined in the iterable
  - rejected with the reason from the first promise in the iterable that rejected

Promise Methods

SoftUni Foundation

- Promise.race(iterable)
  - Returns a promise that fulfills or rejects as soon as one of the promises in the iterable fulfills or rejects, with the value or reason from that promise
- Promise.reject(reason)
  - Returns a Promise object that is rejected with the given reason
- Promise.resolve(value)
  - Returns a Promise object that is resolved with the given value

JS Promises – Syntax

```
let p = new Promise(function(resolve, reject) {
  // Do an async task and then resolve or reject
  if (/* operation successful */)
    resolve('Success!');
  else /* operation failed */
    reject('Failure!');
});
```

```
p.then(function(result) {
  /* process the result (when the promise is resolved) */
})
.catch(function(error) {
  /* handle the error (when the promise is rejected) */ });
```

Promise.then() – Example

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

Resolved after 500 ms

```
Before promise
After promise
Then returned: done
```

Check your solution here: https://judge.softuni.bg/Contests/360

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

Rejected after 500 ms

```
Before promise
After promise
fail
```

Check your solution here: https://judge.softuni.bg/Contests/360

Asynchronous function with Promise – Example

SoftUni Foundation

```
function getAsync(url)
{
    return new Promise((resolve, reject) =>
    {
        const xhr = new XMLHttpRequest();
        xhr.open("GET", url);
        xhr.onload = () => resolve(xhr.responseText);
        xhr.onerror = () => reject(xhr.statusText);
        xhr.send();
    });
}
```

Promises with jQuery AJAX

jQuery Promise

SoftUni Foundation

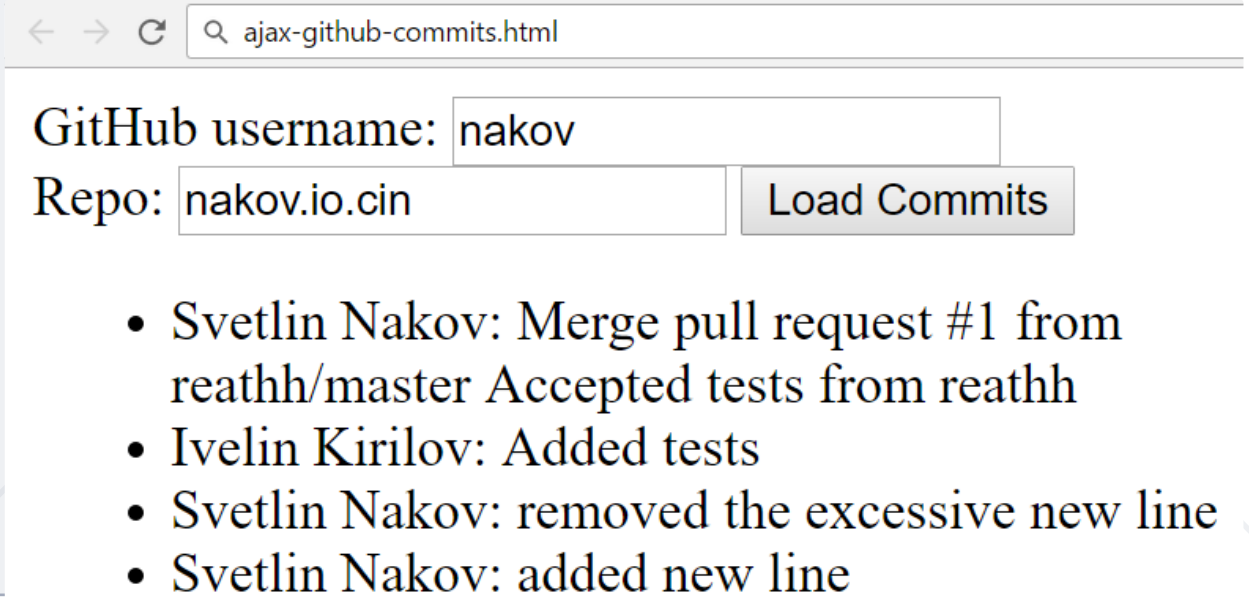
- The Deferred object, introduced in jQuery 1.5, is a chainable utility object created by calling the jQuery.Deferred() method. It can register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function
- jQuery 3.0 and newer Deferred object is a Promises/A+ implementation
- Deferred object is thanable, so can be casted to native JS Promise, but some of the arguments passed to than() method will be discarded

jQuery Promise interface

- As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the XMLHTTPRequest object. This jQuery XHR object, or "jqXHR“, implements the Promise interface.
- jqXHR.done(function(data, textStatus, jqXHR) {});
  - An alternative construct to the success callback option
- jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});
  - An alternative construct to the error callback option, the .fail() method replaces the deprecated .error() method
- jqXHR.always(function(data|jqXHR, textStatus, jqXHR|errorThrown) { }); (added in jQuery 1.6)
  - An alternative construct to the complete callback option, the .always() method replaces the deprecated .complete() method.
  - In response to a successful request, the function's arguments are the same as those of .done(): data, textStatus, and the jqXHR object For failed requests the arguments are the same as those of .fail(): the jqXHR object, textStatus, and errorThrown
- jqXHR.then(function(data, textStatus, jqXHR) {}, function(jqXHR, textStatus, errorThrown) {});
  - Incorporates the functionality of the .done() and .fail() methods, allowing (as of jQuery 1.8) the underlying Promise to be manipulated

Problem: Load GitHub Commits with AJAX

```
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() {
    // AJAX call …
  }
</script>
```



Solution: Load GitHub Commits with AJAX

SoftUni Foundation

```
function loadCommits() {
  $("#commits").empty();
  let url = "https://api.github.com/repos/" +
    $("#username").val() + "/" +
    $("#repo").val() + "/commits";
  $.get(url)
    .then(displayCommits)
    .catch(displayError);
  function displayCommits(commits) { … }
  function displayError(err) { … }
}
```

jQuery AJAX methods return promises

Solution: Load GitHub Commits with AJAX (2)

SoftUni Foundation

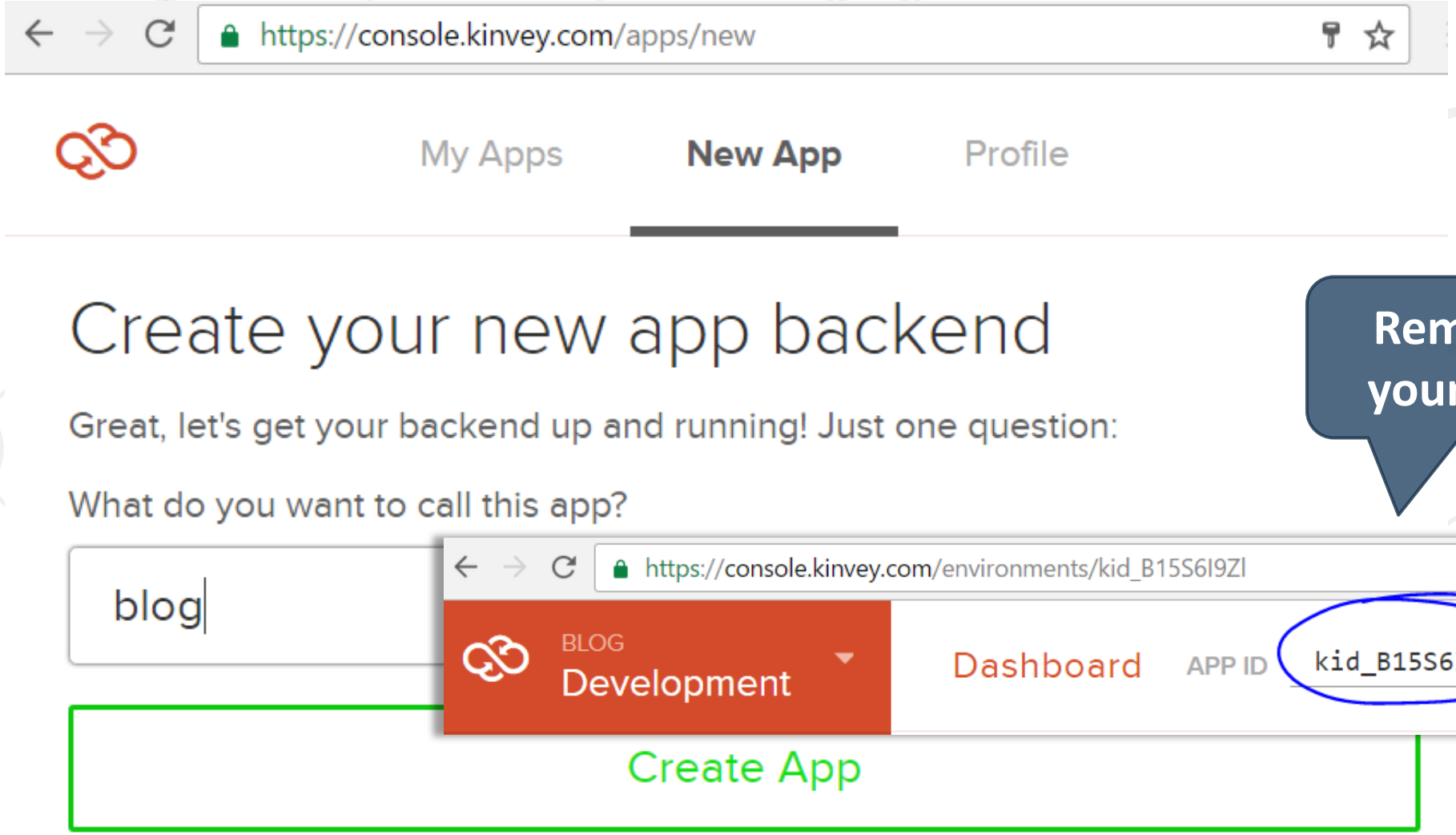
```
function displayCommits(commits) {
  for (let commit of commits)
    $("#commits").append($("<li>").text(
      commit.commit.author.name + ": " +
      commit.commit.message
    ));
}
function displayError(err) {
  $("#commits").append($("<li>").text("Error: " +
    err.status + ' (' + err.statusText + ')'));
}
```

Check your solution here: https://judge.softuni.bg/Contests/360

Problem: Blog (Posts with Comments)

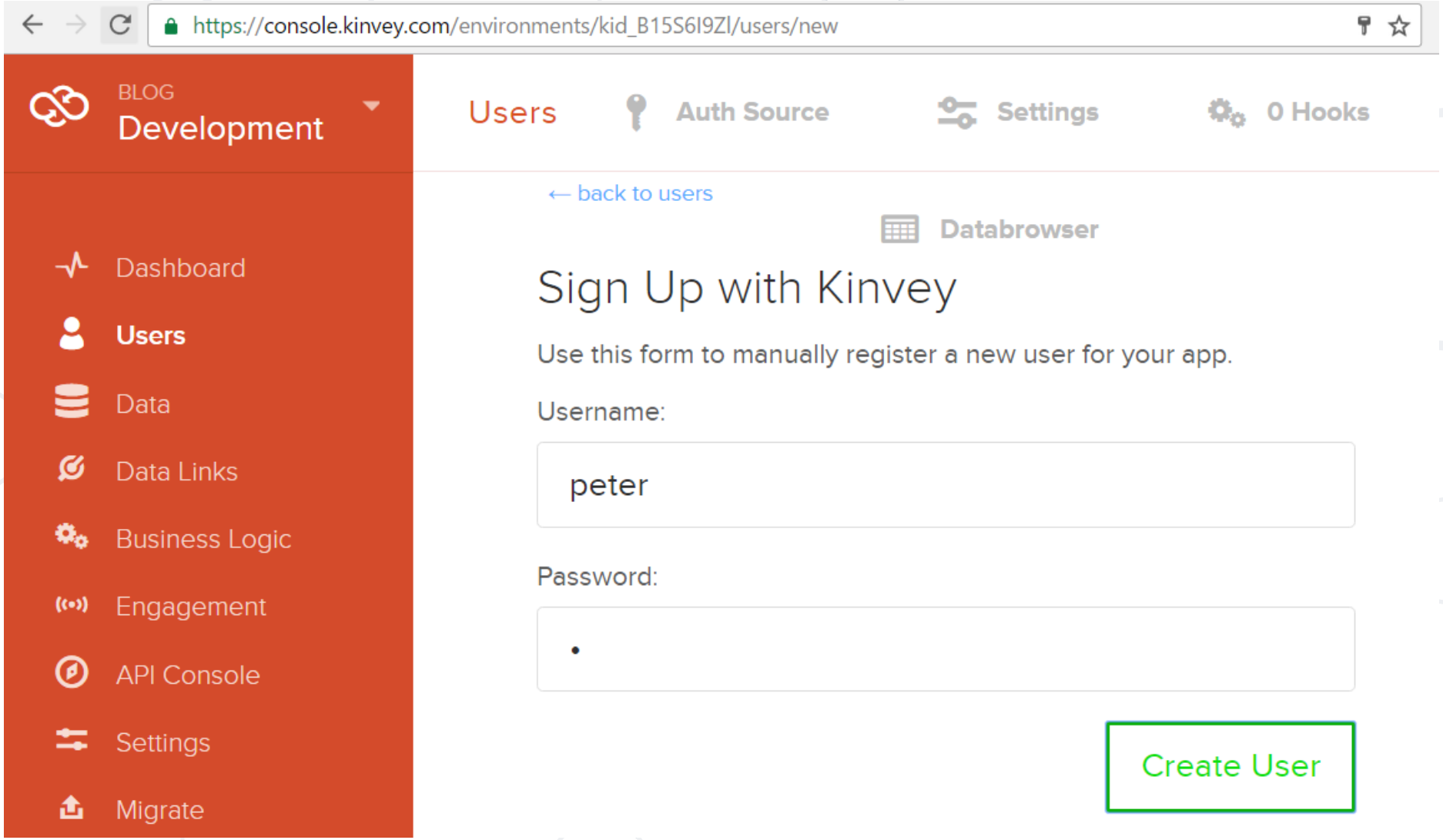
- Create a Kinvey app
  - Create user "peter" with password "p"
  - Create posts "Post1" and "Post2"
  - Create comments "Com1a" and "Com1b" for "Post1"
  - Create comments "Com2a", "Com2b" and "Com2c" for "Post2"
- Write a JS app to display all posts and view selected post along with its comments

Solution: Blog – Create a Kinvey App



Remember your **app ID**

Solution: Blog – Create User "peter"



Solution: Blog – Create the First Post

SoftUni Foundation

Insert your Kinvey App ID here

```
POST /appdata/kid_B15S6I9Zl/posts/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Type: application/json

{ "title":"Post1", "body":"Post #1 body" }
```

Base64(user:pass)

```
{ "title": "Post1", "body": "Post #1 body", …,
  "_id": "582cde77209db9d9730bab03" }
```

Remember the post _id

Solution: Blog – Create the Comments

SoftUni Foundation

```
POST /appdata/kid_B15S6I9Zl/comments/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Type: application/json

{ "text":"Com1a", "post_id":"582cde77209db9d9730bab03" }
```

Use post _id from the previous request

```
POST /appdata/kid_B15S6I9Zl/comments/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Content-Type: application/json

{ "text":"Com2a", "post_id":"582cde77209db9d9730bab03" }
```

Solution: Blog – Create the Second Post

- Create the second post "Post2" + its 3 comments
- The "posts" and "comments" collections should look like these:

_id	_acl	_kmd	title	body
582ce30adb630ca5056856d6	{"creator":"582cde6b..	{"lmt":"2016-11-16T22:57:09.7..	"Post2"	"Post #2 body"
582cde77209db9d9730bab03	{"creator":"582cde6b..	{"lmt":"2016-11-16T22:57:13.9..	"Post1"	"Post #1 body"

_id	_acl	_kmd	text	post_id
582ce37b209db9d9730be951	{"creator":"582cde6b..	{"lmt":"2016...	"Com2c"	"582ce30adb630ca5056856d6"
582ce375db630ca505685855	{"creator":"582cde6b..	{"lmt":"2016...	"Com2b"	"582ce30adb630ca5056856d6"
582ce36d209db9d9730be92a	{"creator":"582cde6b..	{"lmt":"2016...	"Com2a"	"582ce30adb630ca5056856d6"
582ce2b76ce7ea9e15ca61e3	{"creator":"582cde6b..	{"lmt":"2016...	"Com1b"	"582cde77209db9d9730bab03"
582ce154d474d37548133d04	{"creator":"582cde6b..	{"lmt":"2016...	"Com1a"	"582cde77209db9d9730bab03"

Solution: Blog – HTML Code

```html
<script src="jquery-3.1.1.min.js"></script>
<script src="blog.js"></script>

<h1>All Posts</h1>
<button id="btnLoadPosts">Load</button>
<select id="posts"></select>
<button id="btnViewPost">View</button>

<h1 id="post-title">Post Details</h1>
<ul id="post-body"></ul>
<h2>Comments</h2>
<ul id="post-comments"></ul>
```

Solution: Blog – JS Code

```
$(document).ready(function() {
  const kinveyAppId = "kid_B15S6I9Zl";
  const serviceUrl = "https://baas.kinvey.com/appdata/" +
kinveyAppId;
  const kinveyUsername = "peter";
  const kinveyPassword = "p";
  const base64auth = btoa(kinveyUsername + ":" +
kinveyPassword);
  const authHeaders = { "Authorization": "Basic " + base64auth
};
  $("#btnLoadPosts").click(loadPostsClick);
  $("#btnViewPost").click(viewPostClick);
  function loadPostsClick() { … }
  function viewPostClick() { … }
  …
});
```

Solution: Blog – Load Posts

```
function loadPostsClick() {
  let loadPostsRequest = {
    url: serviceUrl + "/posts",
    headers: authHeaders,
  };
  $.ajax(loadPostsRequest)
    .then(displayPosts)
    .catch(displayError);
}
```

Solution: Blog – Display Posts as Options

```
function displayPosts(posts) {
  $("#posts").empty();
  for (let post of posts) {
    let option = $("<option>")
      .text(post.title)
      .val(post._id);
    $("#posts").append(option);
  }
}
```


```
▼<select id="posts">
    <option value="582cde77209db9d9730bab03">Post1</option>
    <option value="582ce30adb630ca5056856d6">Post2</option>
  </select>
```

Solution: Blog – Handle AJAX Errors

```
function displayError(err) {
  let errorDiv = $("<div>").text("Error: " +
    err.status + ' (' + err.statusText + ')');
  $(document.body).prepend(errorDiv);
  setTimeout(function() {
    $(errorDiv).fadeOut(function() {
      $(errorDiv).remove();
    });
  }, 3000);
}
```

Solution: Blog – Load Post Comments Query

SoftUni Foundation

- Kinvey allows querying collections:

```
https://baas.kinvey.com/appdata/kid_B15S6I9Zl/comments
?query={"post_id":"582cde77209db9d9730bab03"}
```

GET https://baas.kinvey.com/appdata/kid_B15S6I9Zl/comments?query={"post_id":"582cde77209db9d9730bab03"} Params Send Save

Authorization • Headers (2) Body Pre-request Script Tests • Code

Authorization	Basic cGV0ZXI6cA==	
Content-Type	application/json	
key	value	

Bulk Edit Presets

Body Cookies Headers (7) Tests (0/6) Status: 200 OK Time: 950 ms

Pretty Raw Preview JSON Save Response


```
1 [
2   {
3     "_id": "582ce154d474d37548133d04",
4     "text": "Com1a",
5     "post_id": "582cde77209db9d9730bab03",
```

Solution: Blog – [View Post] Button Click

```
function viewPostClick() {
  let selectedPostId = $("#posts").val();
  if (!selectedPostId) return;
  let requestPosts = $.ajax({
    url: serviceUrl + "/posts/" + selectedPostId,
    headers: authHeaders });
  let requestComments = $.ajax({ url: serviceUrl +
`/comments/?query={"post_id":"${selectedPostId}"}`,
    headers: authHeaders });
  Promise.all([requestPosts, requestComments])
    .then(displayPostWithComments)
    .catch(displayError);
}
```

Solution: Blog – Display Post with Its Comments

```
function displayPostWithComments([post, comments]) {
  $("#post-title").text(post.title);
  $("#post-body").text(post.body);
  $("#post-comments").empty();
  for (let comment of comments) {
    let commentItem = $("<li>")
      .text(comment.text);
    $("#post-comments")
      .append(commentItem);
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/360

Async / Await

Simplified Promises

Async Functions

SoftUni Foundation

- The async function declaration defines an asynchronous function, which returns an AsyncFunction object
- An asynchronous function is a function which operates asynchronously via the event loop, using an implicit Promise to return its result
- The syntax and structure of your code using async functions is much more like using standard synchronous functions

Async Functions

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
async function asyncCall() {
  console.log('calling');
  var result = await resolveAfter2Seconds();
  console.log(result);
  // expected output: 'resolved'
}
asyncCall();
```

Async Functions

SoftUni Foundation

- An async function can contain an await expression that pauses the execution and waits for the passed Promise's resolution
- After resolution the async function's execution resumes and resolved value is available
- The await keyword is only valid inside async functions
- The purpose of async/await functions is to simplify the behavior of using promises synchronously
- async/await is similar to combining generators and promises

Async Functions

SoftUni Foundation

- Do not confuse await for Promise.then()
- If you wish to await two or more promises in parallel, you must still use Promise.then()

```
var resolveAfter2Seconds = function() {
  console.log("starting slow promise");
  return new Promise(resolve => {
    setTimeout(function() {
      resolve(20);
      console.log("slow promise is done");
    }, 2000);
  });
};
```

```
var resolveAfter1Second = function() {
  console.log("starting fast promise");
  return new Promise(resolve => {
    setTimeout(function() {
      resolve(10);
      console.log("fast promise is done");
    }, 1000);
  });
};
```

Sequential Start

- Execution suspends 2 seconds for the first await, and then again an other 1 second for the second await.
- The second timer is not created until the first has already fired

```
var sequentialStart = async function() {
  console.log('==SEQUENTIAL START==');

  const slow = await resolveAfter2Seconds();
  const fast = await resolveAfter1Second();

  console.log(slow);
  console.log(fast);
}
```

Concurrent Start

- Both timers are created and then awaited. The timers are running concurrently but the await calls are still running in series, meaning the second await will wait for the first one to finish
- This leads the code to finish in 2 rather than 3 seconds, which is the time the slowest timer needs

```
var concurrentStart = async function() {
  console.log('==CONCURRENT START==');
  const slow = resolveAfter2Seconds();
  // starts timer immediately
  const fast = resolveAfter1Second();

  console.log(await slow);
  console.log(await fast);
  // waits for slow to finish, even though
  fast is already done!
}
```

```
var stillConcurrent = function() {
  console.log('==PROMISE ALL START==');
  Promise.all([resolveAfter2Seconds(),
    resolveAfter1Second()])
    .then((messages) => {
      console.log(messages[0]); // slow
      console.log(messages[1]); // fast
    });
}
```

Parallel Start

- If you wish to await two or more promises in parallel, you must still use Promise.then()

```
var parallel = function() {
  console.log('==PARALLEL with Promise.then==');
  resolveAfter2Seconds().then((message)=>console.log(message));
  resolveAfter1Second().then((message)=>console.log(message));
}
```

Error Handling

- If a promise resolves normally, then await promise returns the result. But in case of a rejection, it throws the error, just as if there were a throw statement at that line

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

```
async function f() {
  let response = await fetch();
}

// f() becomes a rejected promise
f().catch(alert);
```

Live Exercises in Class (Lab)

Practice: Promises, AJAX, Async / Await

Summary

SoftUni Foundation

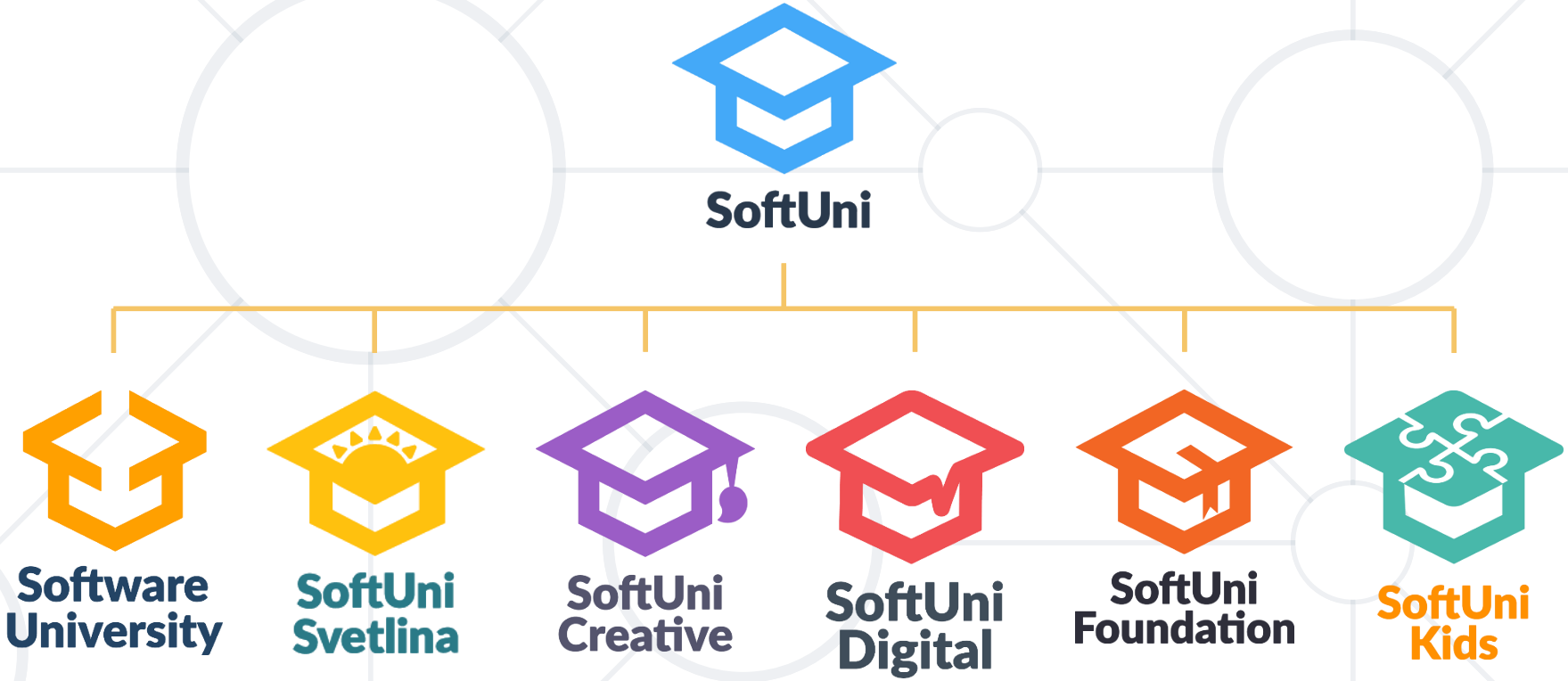
- Promises hold operations (background tasks)
  - Can be resolved or rejected

```
p.then(function(result) {
  /* process the result */ })
.catch(function(error) {
  /* handle the error */ });
```

  - jQuery AJAX works with promises

```
$.ajax(request)
.then(function(data) { … })
.catch(function(error) { … });
```

Questions?



https://softuni.bg/trainings/2082/js-applications-november-2018

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

One SOFTW

WORLD OF MYTHS

Lukanet.com

codexio

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license