Introduction to Node.js

Overview, Modules, Web Server, Request and Response

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-web

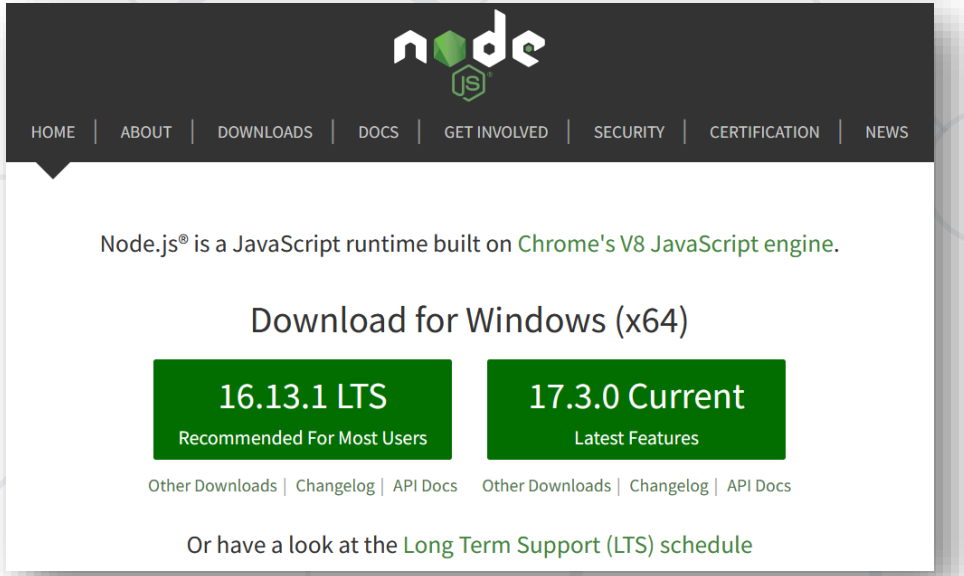
Introduction to Node.js

Node.js Overview

- A runtime environment for JS that runs on the server
- Advantages
  - **One language** for server and client
  - **Asynchronous** and **Event** Driven
  - Very **fast**
  - Efficient **package manager**

Installation

- Go to http://nodejs.org and install the latest version



- To check the currently installed version of the node, type in the **command prompt/terminal**:

```
node -v
```

Environment Setup

- From the **terminal**

```
node        // Starts REPL
let a = 5
let b = 3
a + b       // 8
```

- Interpret code from a **file**
  - Save the script to **index.js**
  - Execute from the terminal:

```
node index.js
```

NPM Packages

Node.js **projects** are usually set up as **NPM packages**

- From the **terminal**, inside the **target directory**

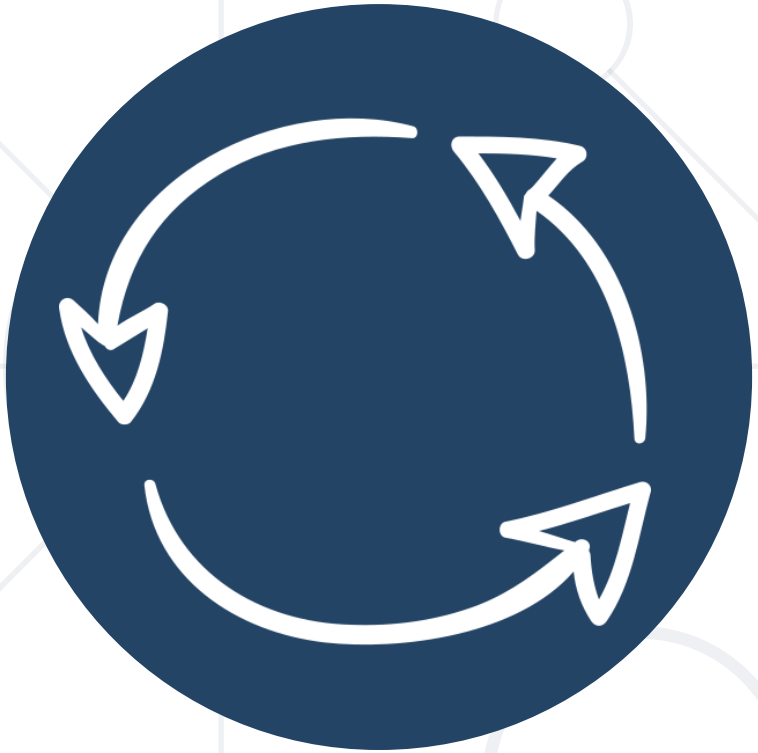
```
npm init
```

- Answer **questions** to initialize the project
- A **package.json** file will be created with initial configuration
- To bypass all questions (take default values):

```
npm init -y
```

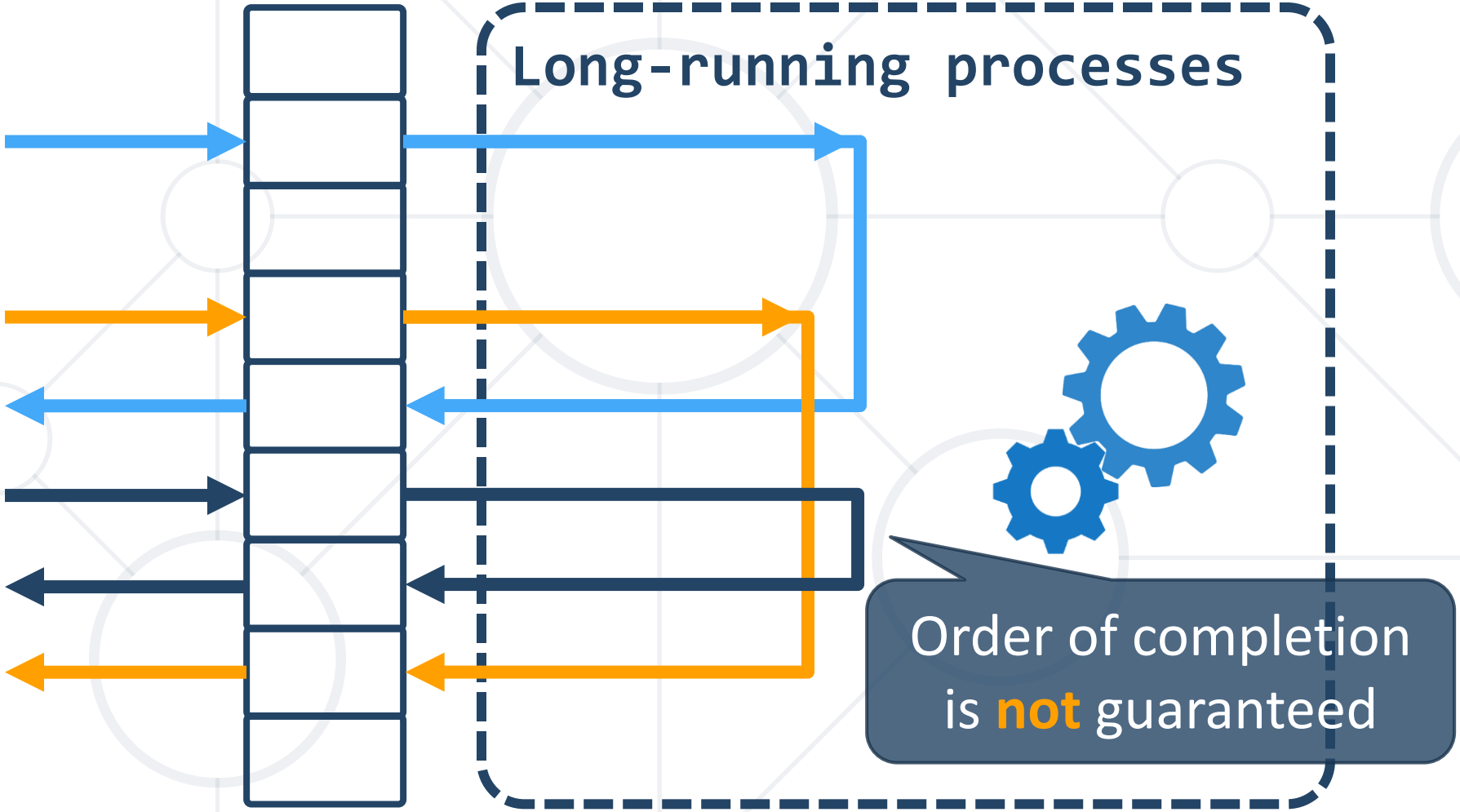
Configuration (Package.json)

```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "Node.js demo project",
  "main": "index.js",
  "engines": {          // Sets versions of Node.js
    "node": ">= 6.0.0",           and other commands
    "npm": ">= 3.0.0" },
  "scripts": {          // Defines a set of node scripts
    "start": "node index.js" },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



Event Loop

The Event Loop



Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

bar(8)

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

foo(10)

bar(8)

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

return

bar(8)

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

bar(8)

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

return

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack calls

```
function foo(x) {
    return x * x;
}
function bar(y) {
    return foo(y + 2);
}

bar(8);
```

Stack

```
function init(el){
    el.addEventListener(
        "click",
        handler
    );
}
```

Stack calls

SoftUni Foundation

Stack

Browser APIs

Hidden implementation

Stack calls

SoftUni Foundation

Stack

init

Browser APIs

Hidden implementation

Software University

Stack calls

SoftUni Foundation

Stack

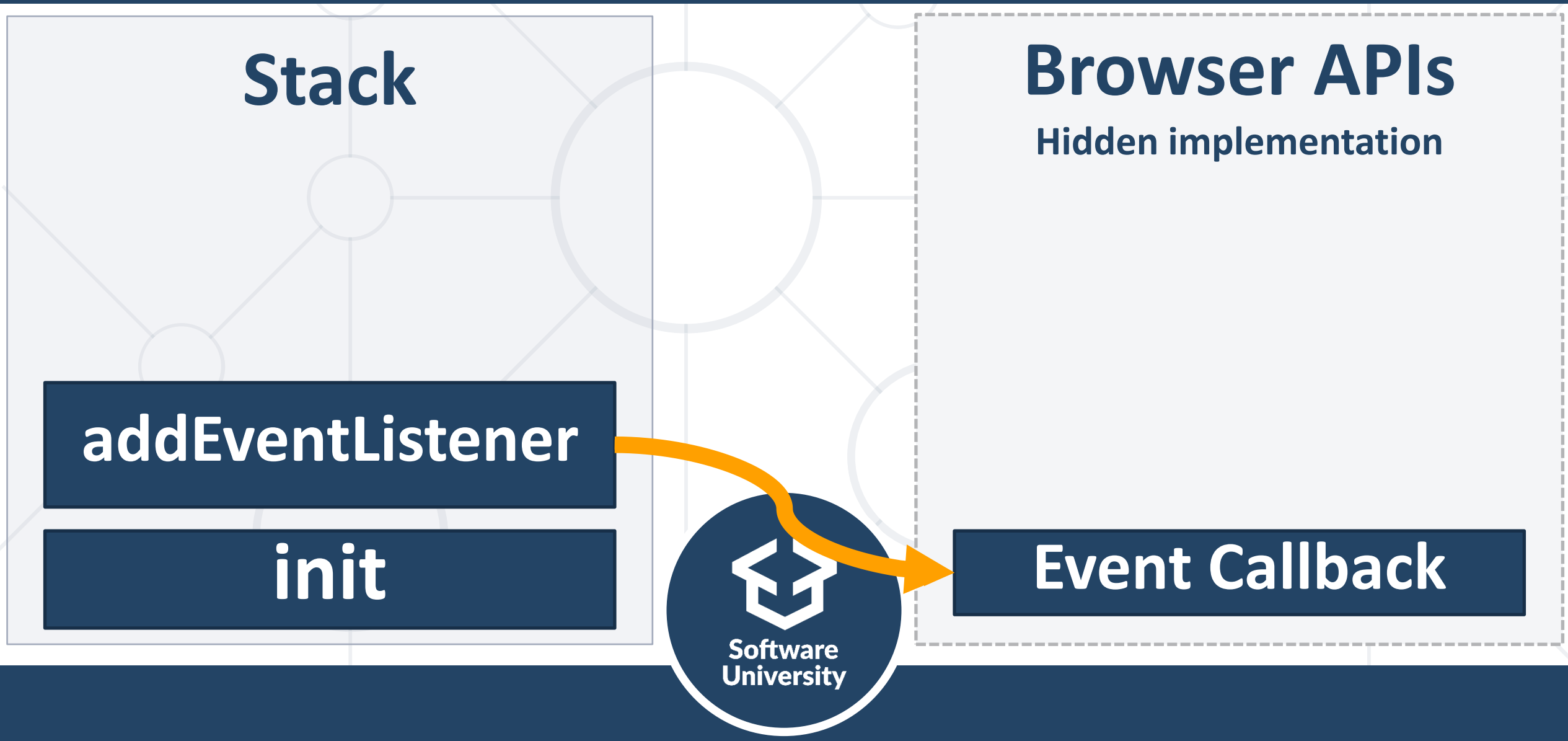
addEventListener

init

Browser APIs

Hidden implementation

Stack calls


Stack
addEventListener
init
Browser APIs
Hidden implementation
Event Callback
Software University

Stack calls

SoftUni Foundation

Stack

return

init

Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

SoftUni Foundation

Stack

return

Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

SoftUni Foundation

GC

Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

SoftUni Foundation

Stack

Browser APIs

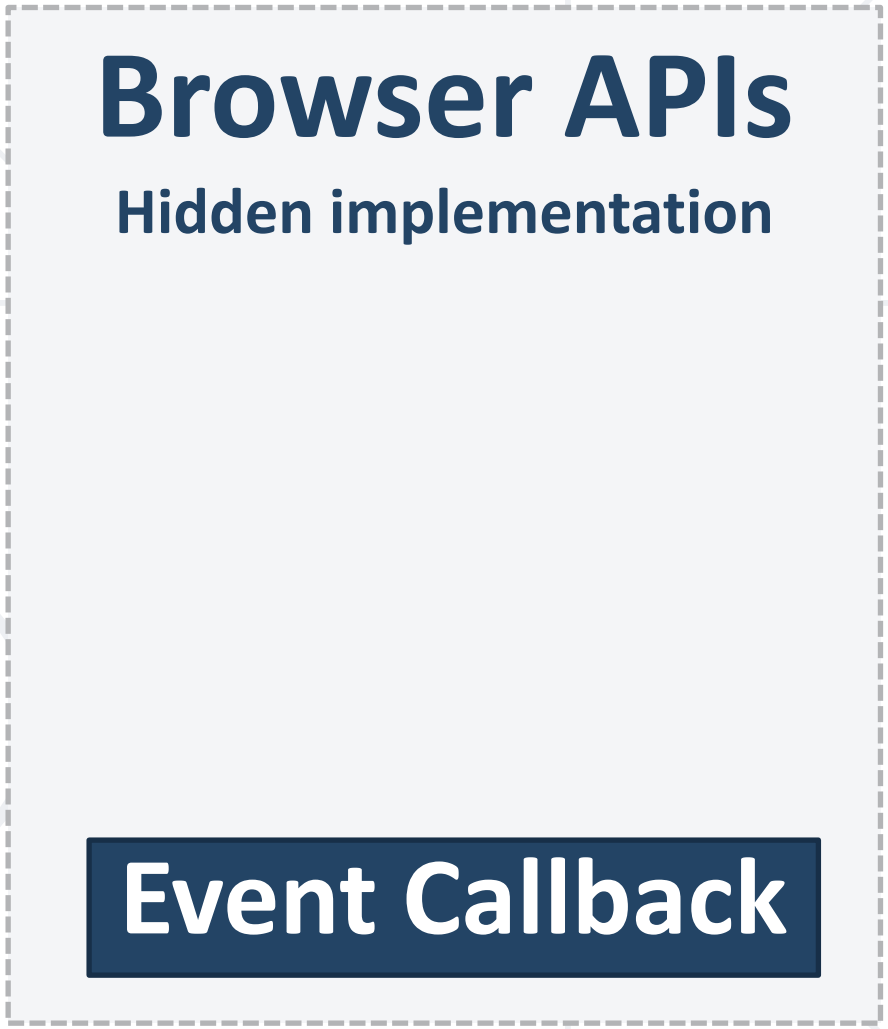
Hidden implementation

Event Callback

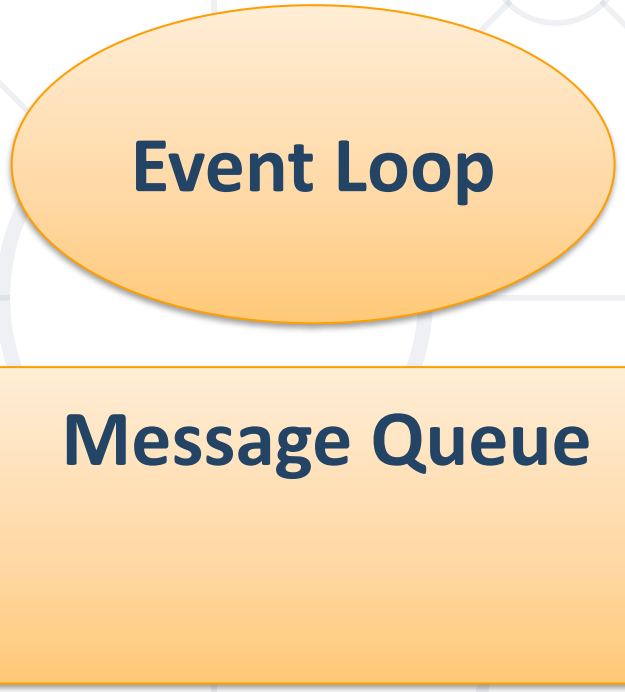
Software University

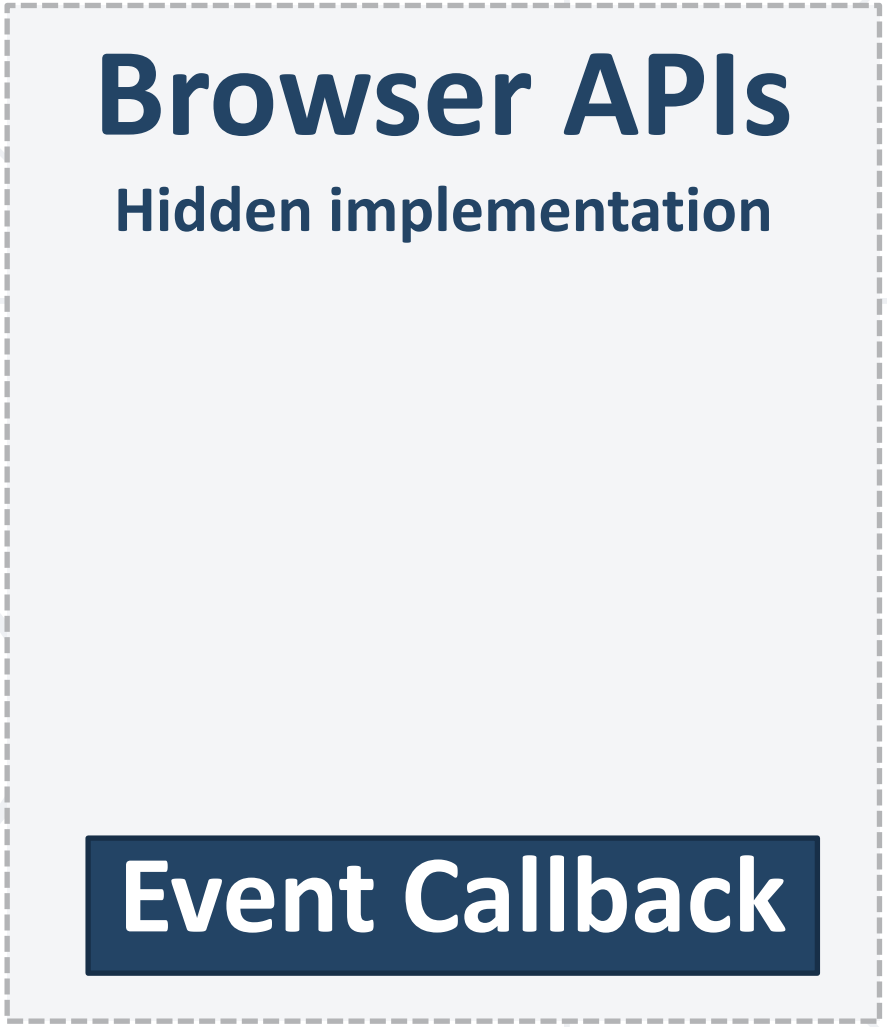
Stack calls

Message Queue

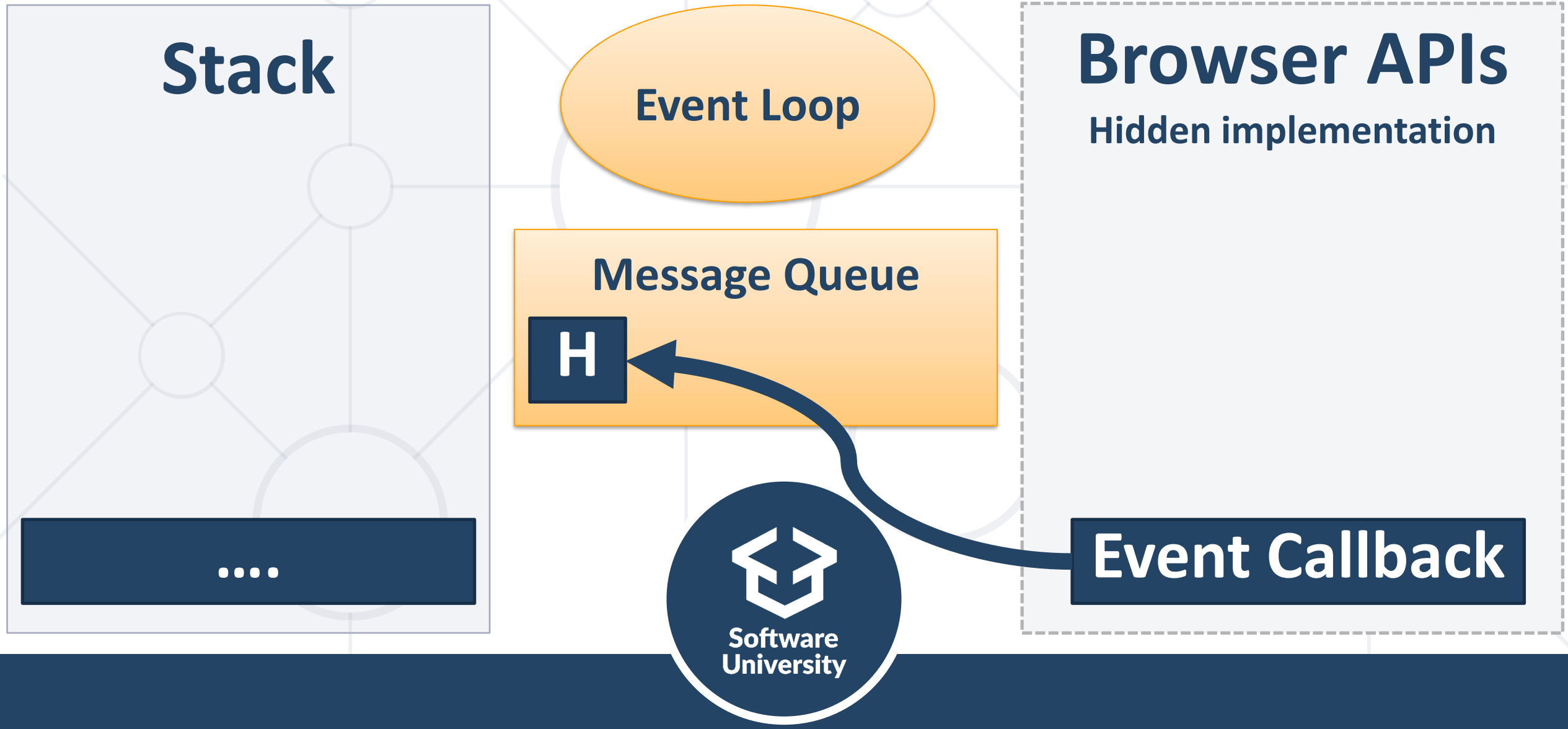


Stack calls





Stack calls

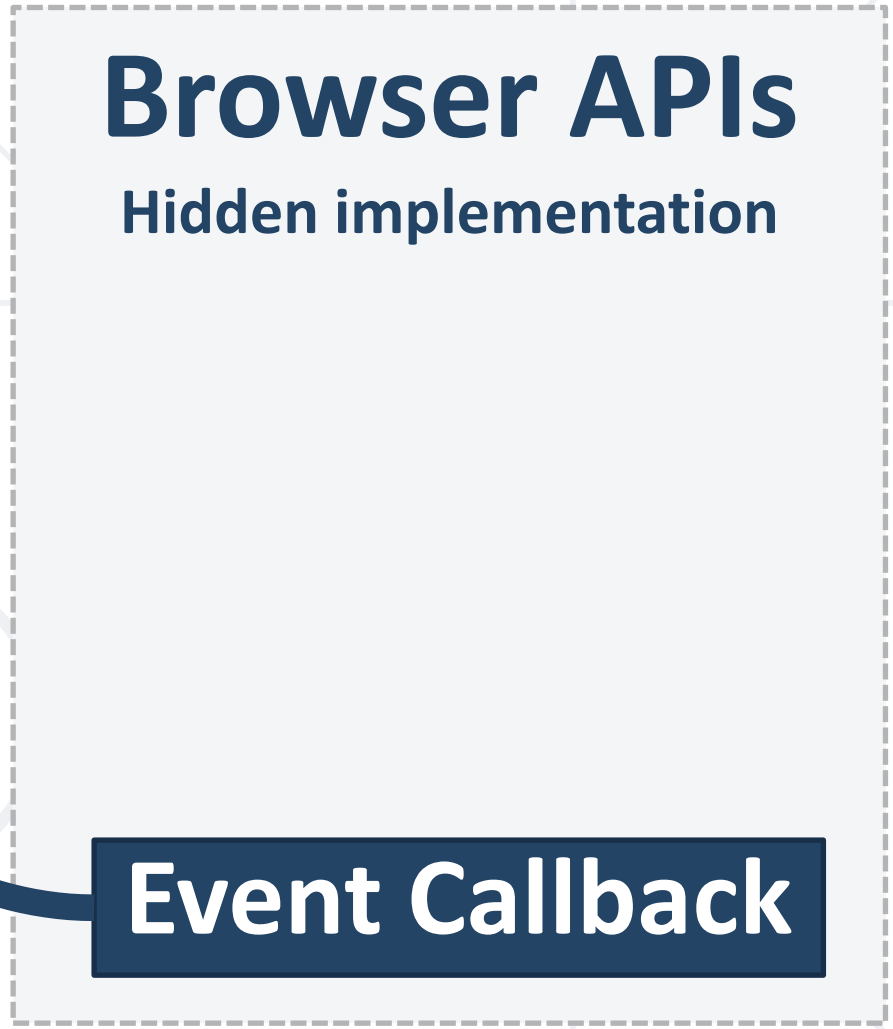

Stack
....
Event Loop
Message Queue
H
Software University
Browser APIs
Hidden implementation
Event Callback

Stack calls


Stack


Event Loop


Message Queue
H
H


Browser APIs
Hidden implementation

Event Callback


Software
University

Stack calls

SoftUni Foundation

Stack

Event Loop

Message Queue

H H

Browser APIs

Hidden implementation

Event Callback

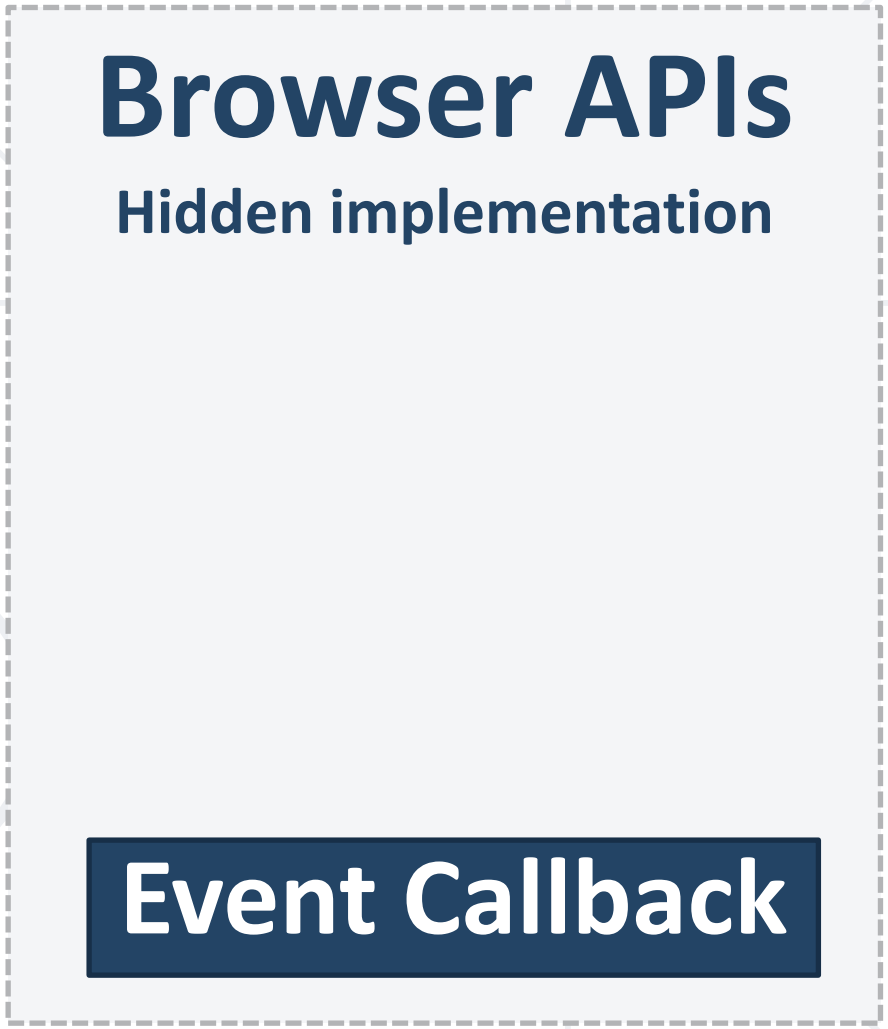
Software University

Stack calls


Stack
handler


Event Loop


Message Queue
H
H


Browser APIs
Hidden implementation
Event Callback

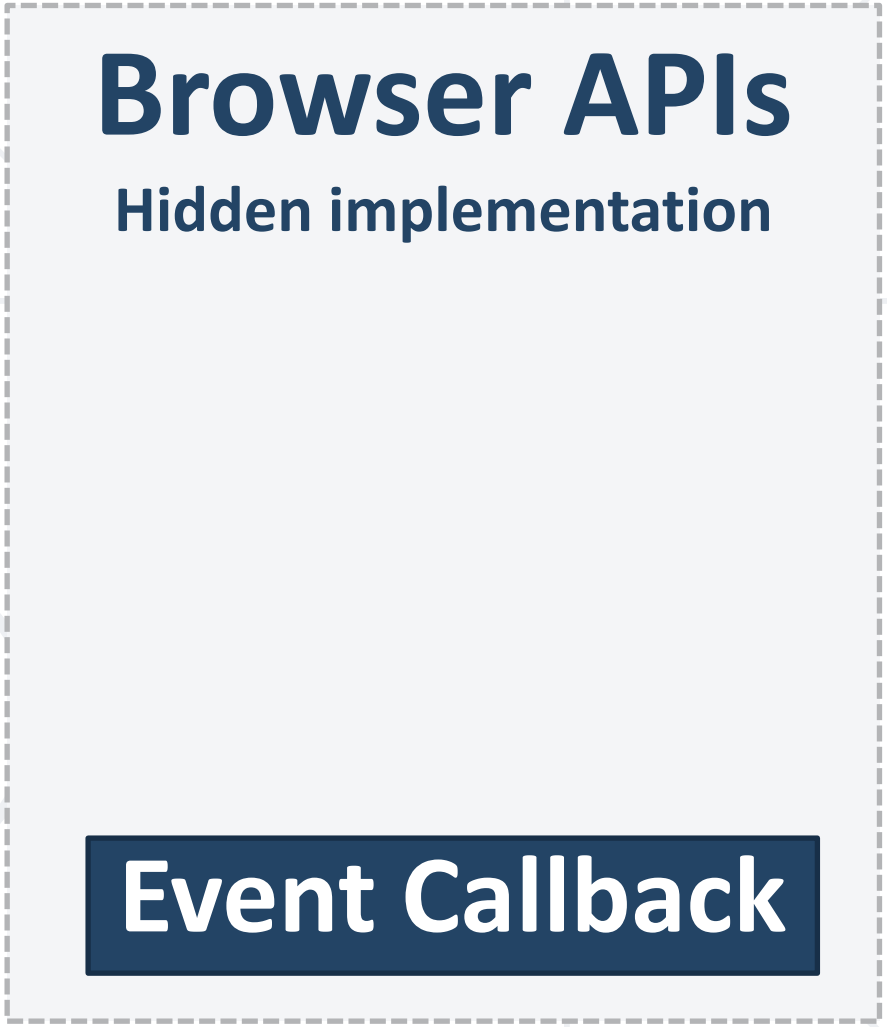

Software
University

Stack calls


Stack
handler


Event Loop


Message Queue
H


Browser APIs
Hidden implementation
Event Callback


Software
University

Stack calls

SoftUni Foundation

Stack

return

Event Loop

Message Queue

H

Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

Event Loop

Message Queue

H

Browser APIs

Hidden implementation

Event Callback

Stack calls

SoftUni Foundation

Stack

Event Loop

Message Queue

H

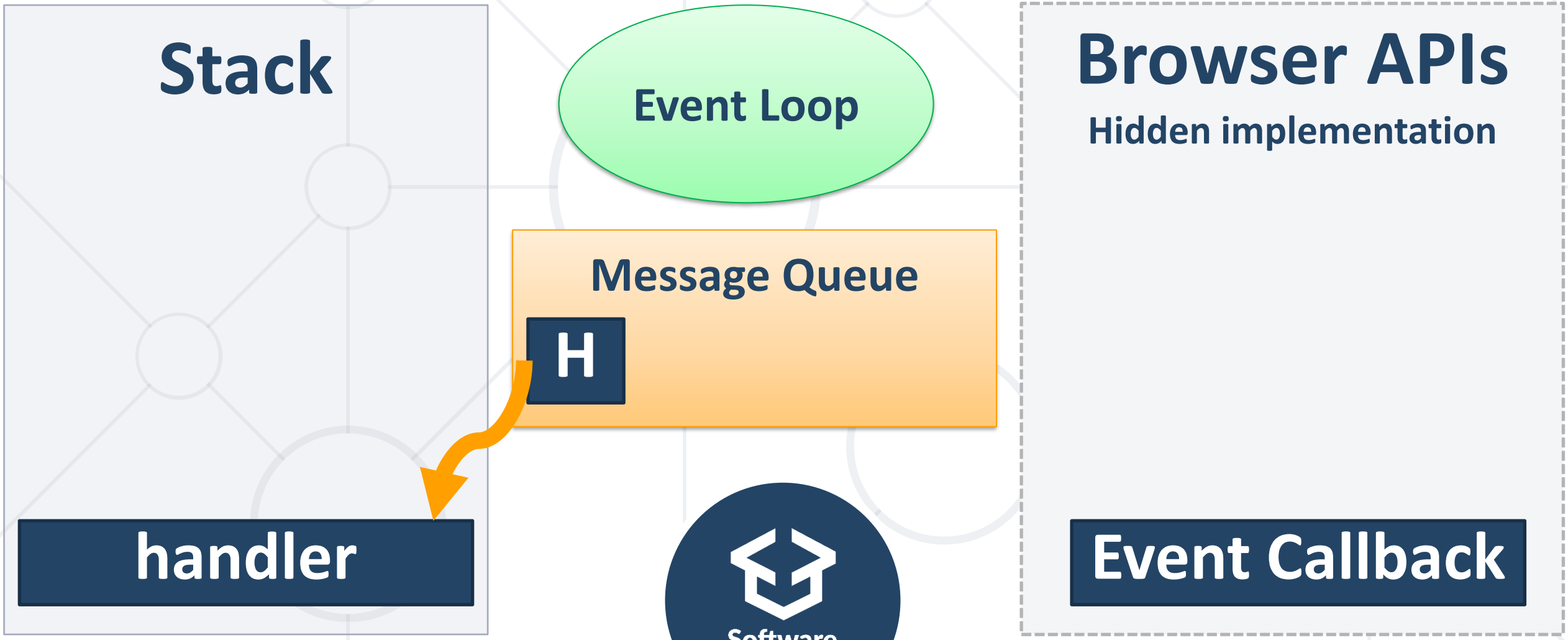
Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

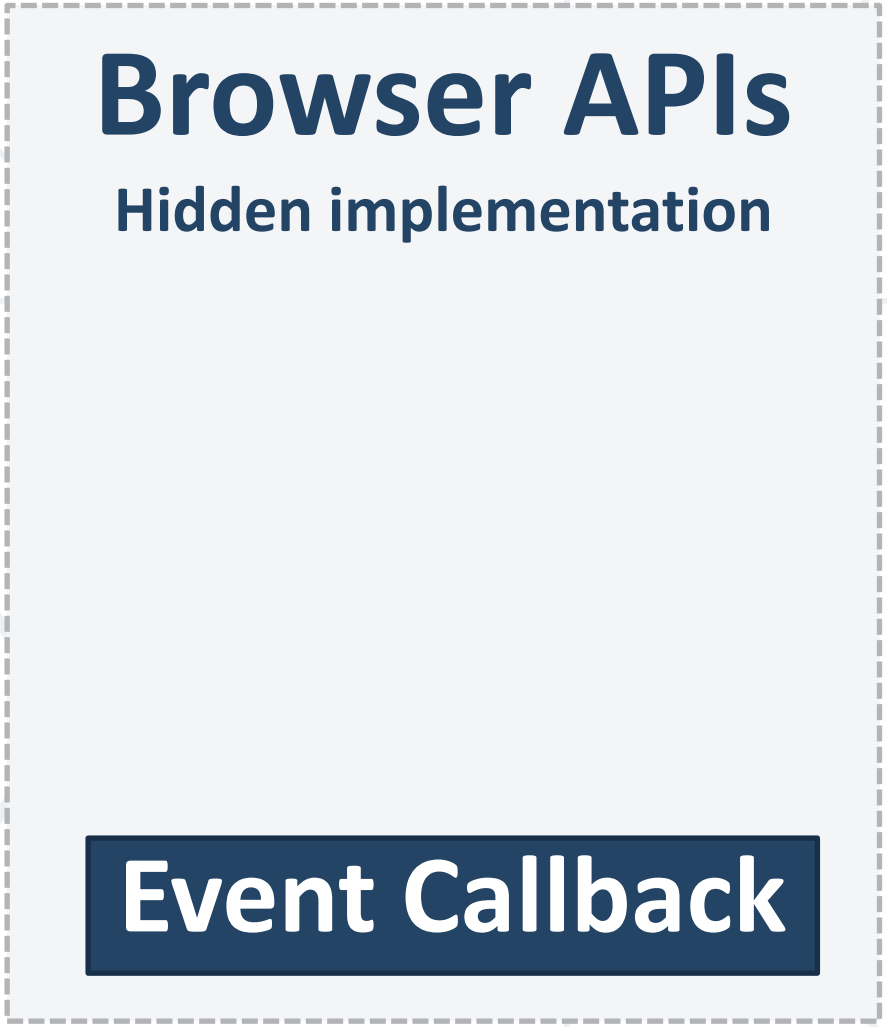

Stack
Event Loop
Browser APIs
Hidden implementation
Message Queue
H
handler
Event Callback
Software University

Stack calls


Stack
handler


Event Loop


Message Queue


Browser APIs
Hidden implementation
Event Callback


Software
University

Stack calls

SoftUni Foundation

Stack

return

Event Loop

Message Queue

Browser APIs

Hidden implementation

Event Callback

Software University

Stack calls

Event Loop

Message Queue

Browser APIs

Hidden implementation

Event Callback

Stack calls

SoftUni Foundation

Stack

Event Loop

Message Queue

Browser APIs

Hidden implementation

Event Callback

Software University

Modules

Modules

- Allow larger **apps** to be **split** and **organized**
- Each module has its **own context**
  - It **cannot pollute** the **global scope**
- Node.js includes **three types** of modules
  - **Core** Modules
  - **Local** Modules
  - **Third-Party** Modules

Local Modules

- Created **locally** in the Node.js application
- Include **different functionalities** in **separate** folders
- Use `module.exports` to expose a **function**, **object** or **variable**

```
module.exports = myModule
```

- Loaded using the `require()` function

```
const myModule = require('./myModule.js');
```

Third-Party Modules

- Installed from Node Package Manager (**NPM**)
- Run from the terminal

```
npm install express --save-exact
```

- To use in your code

```
const express = require('express');
```

- To install globally (for use from the terminal)

```
npm install mocha -g
```


47

Core Modules

- Includes all **functionalities** of Node.js
- Load **automatically** when Node.js process starts
- Need to be **imported** in order to be used

```
const module = require('module');
```

- Commonly used modules are
  - **http** - used to create Node.js server
  - **url**, **querystring**, **path**, **fs**

URL Module

- Provides utilities for URL **resolution** and **parsing**

```
const url = require('url');
```

- Parses an address with the **parse()** function
  - Returns an **object** with **info** about the **url**

```
let urlObj = url.parse(req.url);
```

- **Splits** web address into **readable** parts

URL Parts

- Host '**localhost:8080**'

```
let host = urlObj.host
```

- Path '**/home**'

```
let path = urlObj.pathname
```

- Search/query '**?year=2017&month=february**'

```
let query = urlObj.query
```

```
let search = urlObj.search
```

Query String Module

- Provides utilities for **parsing** and **formatting** URL query strings

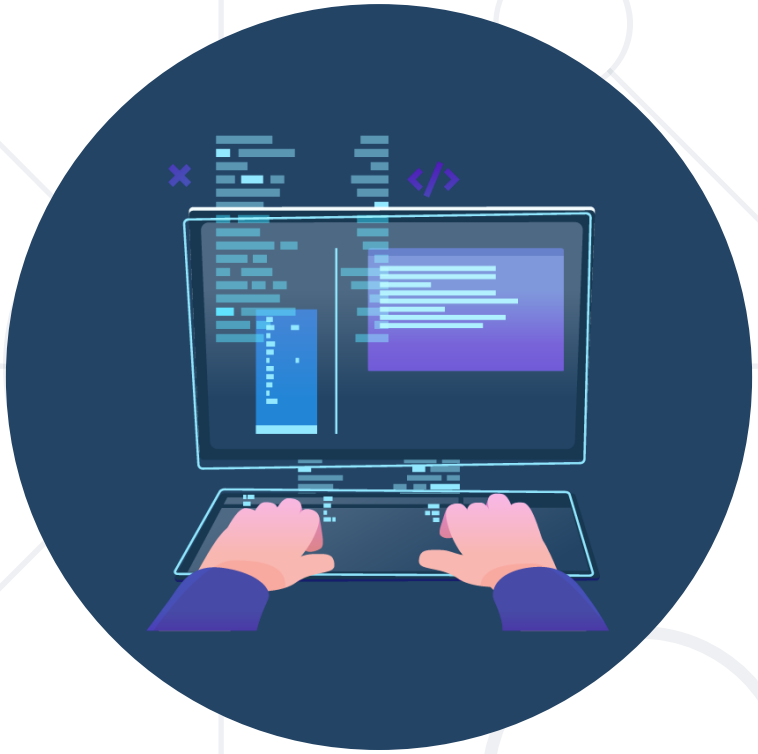
```
const queryString = require('querystring');
```

- Parses a query string into an object

```
const qs = querystring
  .parse('year=2017&month=february');
```

```
const year = qs.year;          // 2017
```

```
const month = qs.month;      // february
```



Node.js Web Server

Web Servers

- All **physical** servers have **hardware**
- The hardware is controlled by the **operating system**
- **Web servers** are **software** products that use the operating system to **handle web requests**
  - Web servers **serve** Web content
- The requests are **redirected to other software** products (ASP.NET, PHP, etc.), depending on the webserver **settings**

Node.js Web Server

- Creating a simple Node.js web server

```
const http = require('http');

http.createServer((req, res) => {
  res.write('Hi!');
  res.end();
}).listen(1337);

console.log('Node.js server running on port 1337');
```

Request & Response Wrappers

The Request Wrapper

- Used to **handle** incoming http requests
- Properties
  - **httpVersion** - '1.1' or '1.0'
  - **headers** - object for request headers
  - **method** - 'GET', 'POST', etc.
  - **url** - the URL of the request

56

Request Wrapper Example

Software University

```
const http = require('http');
const url = require('url');
const port = 1337;

http.createServer((req, res) => {
  let path = url.parse(req['url']).pathname;
  if (path === '/') {
     // TODO: Send 'Welcome to home page!'
  }
}).listen(port);
```

The Response Wrapper

- Used to **retrieve** a **response** to the **client**
- Functions
  - Create **response header**
  - Send the actual **content** to the **client**
  - **End** the response

Response Wrapper Example

```
const http = require('http');
const port = 3000;

http.createServer((req, res) => {
  res.writeHead(200, { // Response Status Code
    'Content-Type': 'text/plain'
  });
  res.write('Hello from Node.js'); // UTF-8 Encoding
  res.end(); // Always End the Response
}).listen(port);
```

Live Exercises

Summary

- Node.js is a **fast** and **asynchronous** efficient **package manager**
- Applications can be **organized** using **module**
- NPM allows quick access to **external modules**
- **Web Servers** transfer resources to the **Client**
- The **Request/Response** Wrappers

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

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

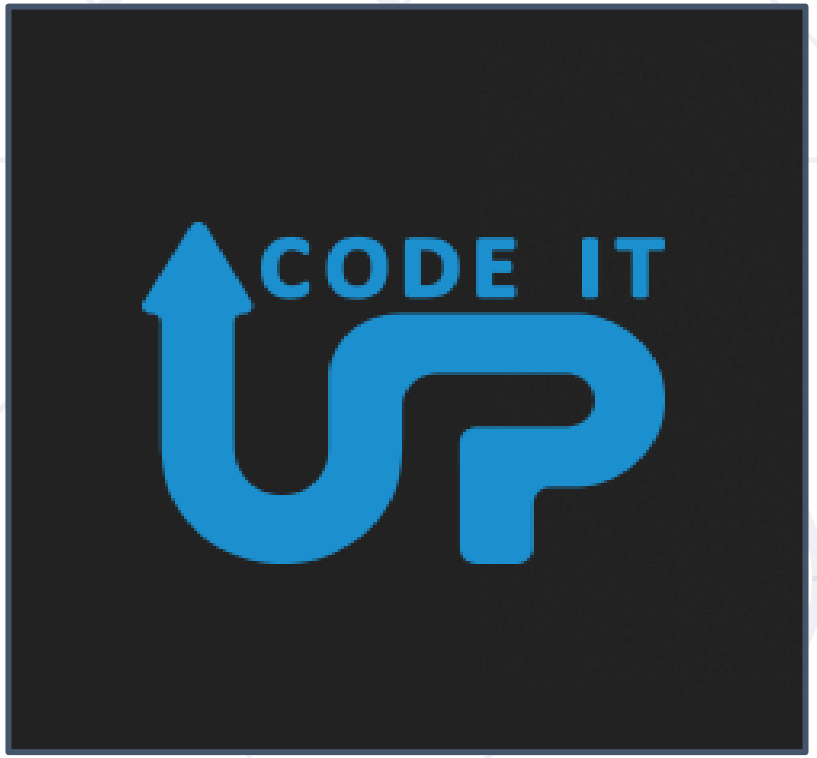
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

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

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://about.softuni.bg/
- © Software University – https://softuni.bg

Streams and Utilities

Streams, Pub/Sub Pattern, Events, FS Module

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

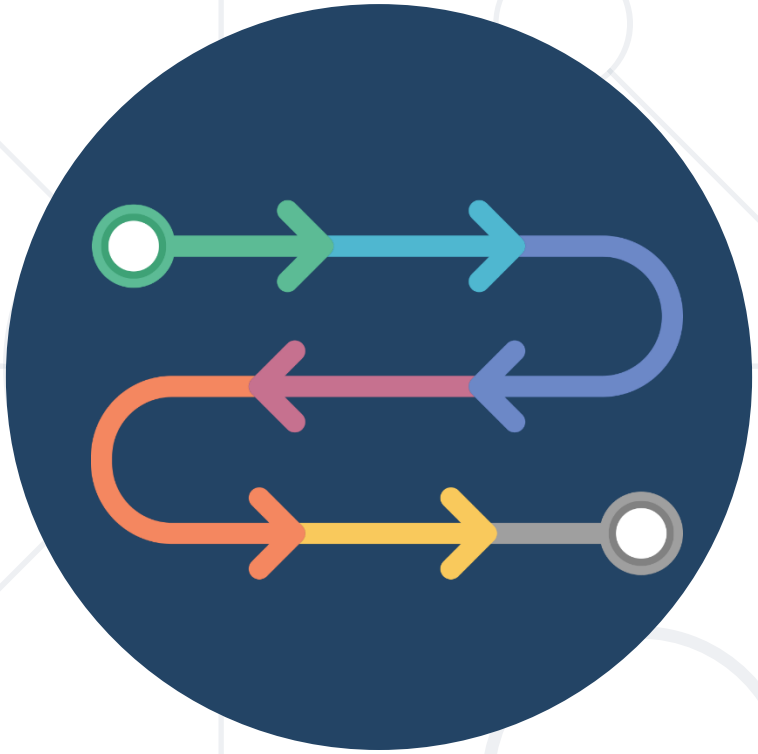
Table of Contents

Have a Question?

Software University

sli.do

#js-web



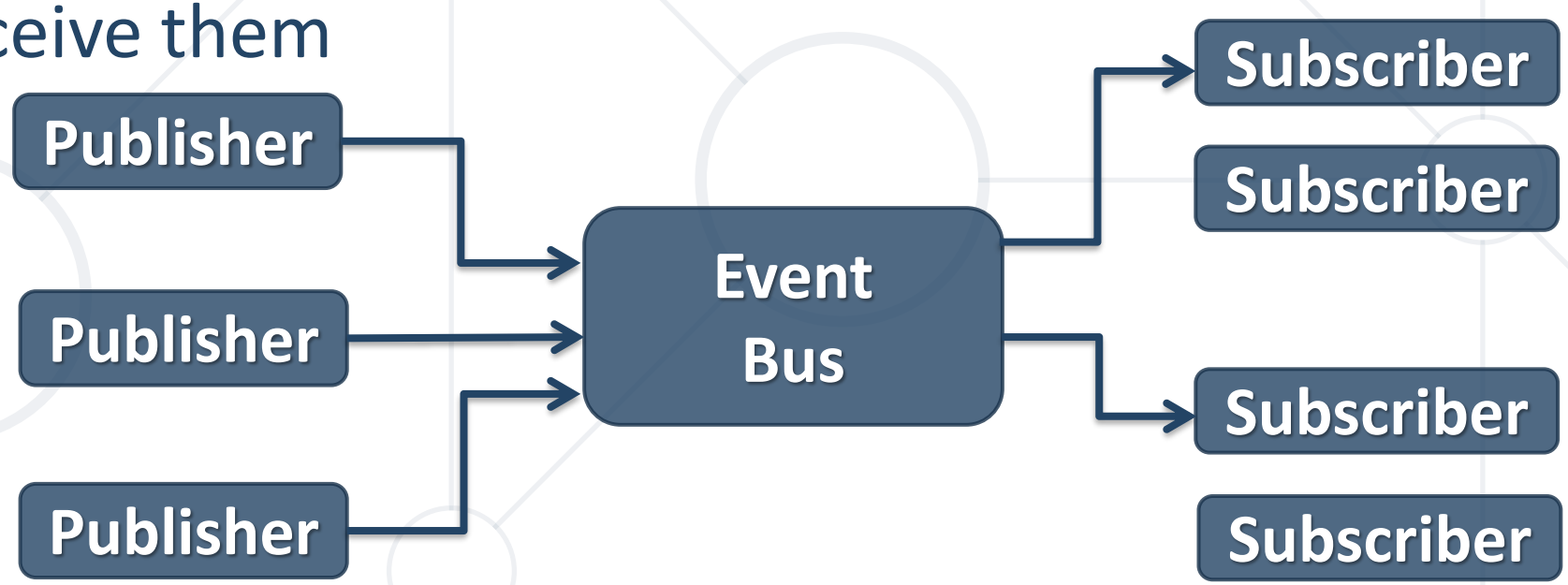
Publish-Subscribe Pattern

What is Pub/Sub?

- Used to **communicate messages** between different system components without them knowing anything about each other's **identity**
  - **Senders** (publishers), do not program the messages to be sent directly to specific **receivers** (subscribers)
  - Subscribers express interest in **one or more events**, and only **receive messages** that are of **interest**

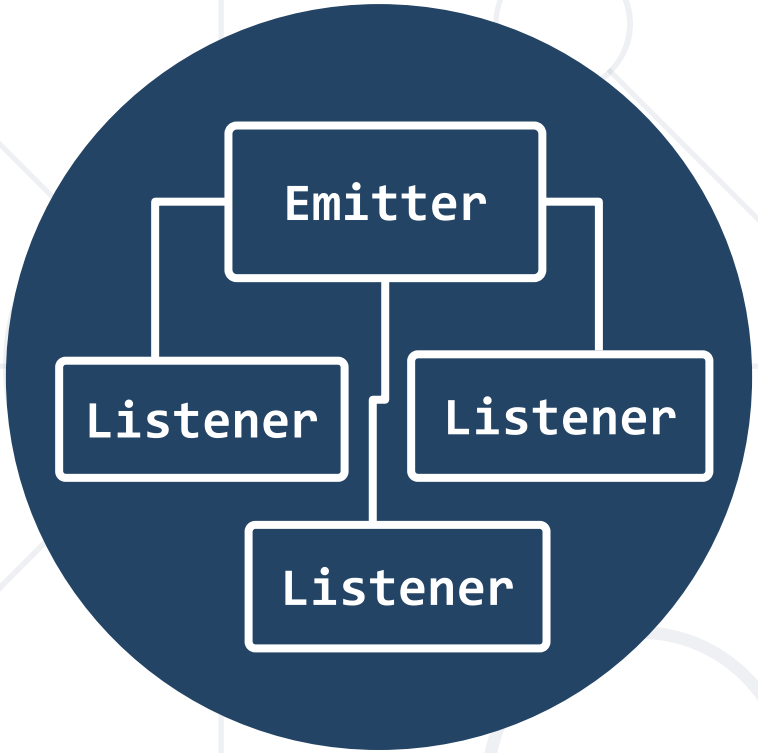
Pub/Sub Example

- An **intermediary** (called a "**message broker**" or "**event bus**")
  - Receives **published** messages
  - Forwards them to the **subscribers** who are registered to receive them



Advantages

- Decouple and Scale Independently
  - Makes software more **flexible**
- Eliminate Polling
  - Promotes **faster response time** and **reduces the delivery latency**
- Simplify Communication
  - Reduces complexity by **removing** all the **point-to-point connections** with a single connection



Events

Events

- Require module "**events**"

```
const events = require('events');
let eventEmitter = new events.EventEmitter();

eventEmitter.on('click', (a, b) => {
  console.log('A click has been detected!');
  console.log(a + ' ' + b); // outputs 'Hello world'
});

eventEmitter.emit('click', 'Hello', 'world');
```

- Events are **not** asynchronous

Streams

Streams

- **Collections of data** that is not available at once
  - Data may come **continuously** in **chunks**
- Types
  - **Readable** - can only be read (process.stdin)
  - **Writeable** - can only be written to (process.stdout)
  - **Duplex** - both Readable and Writeable (TCP sockets)
  - **Transform** - the output is computed from the input (zlib, crypto)

Readable Stream

- Functions
  - `read()` - get chunks from the stream
  - `pause()` - switch to **paused** mode
  - `resume()` - switch to **flowing** mode
- Events - used when the stream is **flowing**
  - `data` - chunk is available for reading
  - `end` - no more data
  - `error` - an exception has occurred

Readable Stream (2)

- **HTTP Request** is a readable stream

```
const http = require('http');

http.createServer((req, res) => {
  if (req.method === 'POST') {
    let body = '';
    req.on('data', data => { body += data });
    req.on('end', () => {
      console.log(body);
    });
  }
}).listen(5000);
```

Writable Stream

- Functions
  - `write()` - send chunks to the stream
  - `end()` - close the stream
- Events
  - `drain` - stream can receive more data
  - `finish` - all data has been flushed (buffer is empty)
  - `error` - an exception has occurred

Writable Stream (2)

- **HTTP Response** is a writeable stream

```
const fs = require('fs');
const server = require('http').createServer();

server.on('request', (req, res) => {
  const src = fs.createReadStream('./bigfile.txt');
  src.on('data', data => res.write(data));
  src.on('end', () => res.end());
});

server.listen(5000);
```

Piping Streams

- The `pipe()` function allows a readable stream to **output directly** to a writable stream
  - **Event listeners** are automatically added

```
const fs = require('fs');
const server = require('http').createServer();

server.on('request', (req, res) => {
  const src = fs.createReadStream('./bigfile.txt');
  src.pipe(res);
});
server.listen(5000);
```

Duplex and Transform Streams

- **Duplex** stream
  - Implements both the **Readable** and **Writeable** interfaces
  - Example - a TCP socket
- **Transform** stream
  - A special kind of duplex stream where the output is a **transformed** version of the input
    - http://codewinds.com/blog/2013-08-20-nodejs-transform-streams.html

Streams

- Transforms with Gzip

```
const fs = require('fs');
const zlib = require('zlib');

let readStream = fs.createReadStream('index.js');
let writeStream = fs.createWriteStream('index.js.gz');

let gzip = zlib.createGzip();

readStream.pipe(gzip).pipe(writeStream);
```

- https://nodejs.org/api/zlib.html

FS Module

Working with the File System

Software University

- The `fs` module gives you access to the **file system**

```
const fs = require('fs');
```

- All functions have **synchronous** and **asynchronous** variants

```
const data = fs.readFileSync('./package.json', 'utf8');
console.log(data);
```

```
const data = fs.readFile('./package.json', 'utf8',
(err, data) => {  // Handle possible errors
  console.log(data); });
```

Working with the File System (2)

Software University

- List files in a directory

```
let data = fs.readdirSync('./myDir', 'utf8');
console.log(data);
```

```
let data = fs.readdir('./myDir', 'utf8', (err,
data) => {
  if (err) {
    console.log(err);
    return;
  }
  console.log(data);
});
```

The result is an **array of strings**, containing all filenames

Working with the File System (3)

- `Create` a directory

```
fs.mkdirSync('./myDir');
```

```
fs.mkdir('./myDir', err => {
  if (err) {
    console.log(err);
    return;
  }
});
```

Working with the File System (4)

Software University

- **Delete** directory

```
fs.rmdirSync('./myDir');
```

```
fs.rmdir('./myDir', err => {
  if (err) {
    console.log(err);
    return;
  }
});
```

- Full API docs: https://nodejs.org/api/fs.html

Working with the File System (5)

Software University

- Rename file or directory

```
fs.renameSync('./oldName', './newName');
```

```
fs.rename('./oldName', './newName', err => {
  if (err) {
    console.log(err);
    return;
  }
});
```

Working with the File System (6)

- `Write` a file

```
const fs = require('fs');
let filePath = './data.txt';
let data = 'Some text';
```

```
fs.writeFileSync(filePath, data);
```

```
fs.writeFile(filePath, data, err => {
  if (err) {
    console.log(err);
    return;
  }
});
```

Working with the File System (7)

Software University

- **Delete** file

```
fs.unlinkSync('./target.txt');
```

```
fs.unlink('./target.txt', err => {
  if (err) {
    console.log(err);
    return;
  }
});ч
```

Debugging

Debugging & Watching in Node.js

Software University

- Debugging in Node.js
  - The V8 **debug protocol** is a **JSON** based protocol
- **IDEs** with a debugger
  - Webstorm
  - Visual Studio
  - Node-inspector (not working with latest version)
- Watching with **Nodemon**

Live Exercises

Summary

- Node.js has various useful **utility** modules
- **Streams** allow working with **big data**
- **Events** simplify **communication** within a large application
- **Pub/Sub** pattern is used to **communicate messages**
- The `fs` module gives you access to the **file system**

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

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

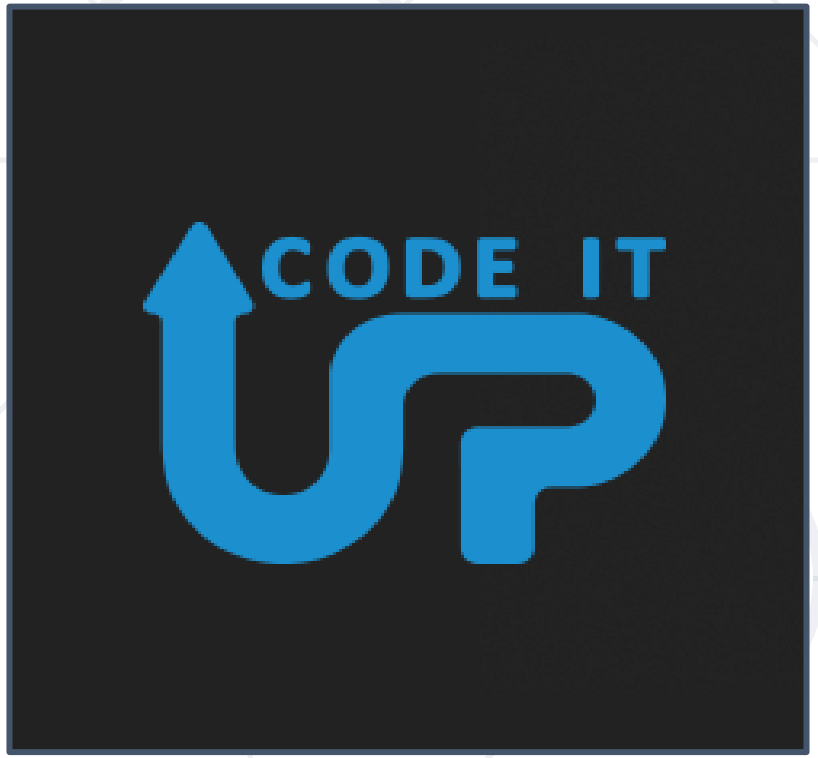
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

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

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://about.softuni.bg/
- © Software University – https://softuni.bg

Intro to Express.js and View Engines

Router, Static Files, Middleware, Handlebars

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-web

Introduction to Express.js

Introduction to Express.js

Software University

```
npm install express
```

```
const express = require('express')
const app = express();
const port = 3000;
app.get('/', (req, res) => {
  res.status(200);
  res.send('Welcome to Express.js!');
})
app.listen(port, () => console.log(`Express running
on port: ${port}...`));
```

Create a new **instance** of the **application**

Router in Express.js

Router

- Routing has the following syntax

```
app.METHOD(PATH, HANDLER)
```

- Where
  - **app** is an **instance** of express
  - **METHOD** is an HTTP **request** method, in **lowercase**
  - **PATH** is a path on the **server**
  - **HANDLER** is the function **executed** when the route is **matched**

Route Methods

```
// GET method route
app.get('/', (req, res) => {
  res.send('GET request to the homepage')
})
// POST method route
app.post('/', (req, res) => {
  res.send('POST request to the homepage')
})
// PUT method route
app.put('/', (req, res) => {
  res.send('PUT request to the homepage')
})
```

Route Methods

Software University

```
// All methods route
app.all('/about', (req, res, next) => {
  console.log('Middleware execution..')
  next()
}, (req, res) => {
  res.send('Show about page.')
})
```

The next **handler** to be **called**

Shows the **about** page **after** middleware **execution**

Router Paths

- Paths can **contain** special characters:

```
app.get('*',(req, res) => {
  res.send('Matches everything')
})
app.get('/ab*cd', (req, res) => {
  res.send('abcd, abANYTHINGcd')
})
app.get(/.*fly$/, (req, res) => {
  res.send('butterfly, dragonfly')
})
```

Based on string **patterns**

Based on **regular expressions**

Extracting Parameters

- Paths can have **parameters**

```
app.get('/users/:userId', (req, res) => {
  const paramsObj = req.params
  res.send(paramsObj) })
```

- You can also **validate** parameters with **regular expressions** (it is not recommended)

```
app.get('/users/:userId(\\d+)', (req, res) => {
  const paramsObj = req.params
  res.send(paramsObj) })
```

Chainable Routes

- You can create **chainable** route handlers using '**app.route()**'

```
app.route('/home')
  .get((req, res) => {
    res.send('GET home page') })
  .post((req, res) => {
    res.send('POST home page') })
  .all((req, res) => {
    res.send('Everything else')
  })
```

Better for **ordering** routes

Always place '**all**' as a **final** method

Router Responses

- Responses
  - res.**download** - **prompt** a file to be **downloaded**

```
app.get('/pdf', (req, res) => {
  res.download('FULL PATH TO PDF') })
```

  - res.**end** - end the response **process**
  - res.**json** - send a **JSON** response

Router Responses

- res.**redirect** - redirect a request (to **another** page)

```
app.get('/about/old', (req, res) => {
  res.redirect('/about') })
```

- res.**sendFile** - send a **file** as an **octet**-stream

```
app.get('/file/:fileName', (req, res) => {
  const fileName = req.params.fileName
  res.sendFile("PATH TO FILE" + fileName) })
```

- res.**render** - render a **view template**

Modular Routers

- You can use `express.Router` for modular route handlers
  - **Mounted** on a route (e.g. '/about')
  - Can use middleware, specific **only** to that router

```
const express = require('express')
const router = express.Router()
router.use(/* add middleware */)
router.get(/* define route handlers */)
app.use('/about', router)
```

Middleware

Middleware

- **Function** that has **access** to
  - The **request** and **response** object
  - The **next** middleware in the application's **request-response cycle**
- Different **kinds** of middleware exist
  - Application, route, error

```
const app = express()
app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next() })
```

Next **handler** to be called

Custom Middleware

- Middleware can be for **specific** path

```
app.use('/user/:userId', (req, res, next) => {
  const userId = req.params.userId
  // TODO: Check if user exists in db/session
  let userExists = true
  if (!userExists) { res.redirect('/login') }
  else { next() }
})
app.get('/user/:userId', (req, res) => {
  res.send('User home page!')
})
```

Custom Middleware

- Can be on Application-level:

```
app.use(function (req, res, next) {
  console.log('Time:', Date.now()) next()
});
```

- Can be used for error-handling

```
app.use(function (err, req, res, next) {
  console.error(err.stack)
  res.status(500).send('Something broke!')
});
```

Third-Party Middleware

```
app.set('view engine', 'pug');
app.set('views', __dirname + '/views');
app.use(cookieParser());
app.use(session({secret: 'magic unicorns'}));
app.use(passport.initialize());
app.use(passport.session());
app.use(express.static(config.rootPath +'/public'));
```

Static Files

Static Files

Software University

- Serving static files

```
app.use(express.static('public'))
app.use('/static', express.static('public'))
app.use('/static', express.static(__dirname + '/public'))
```

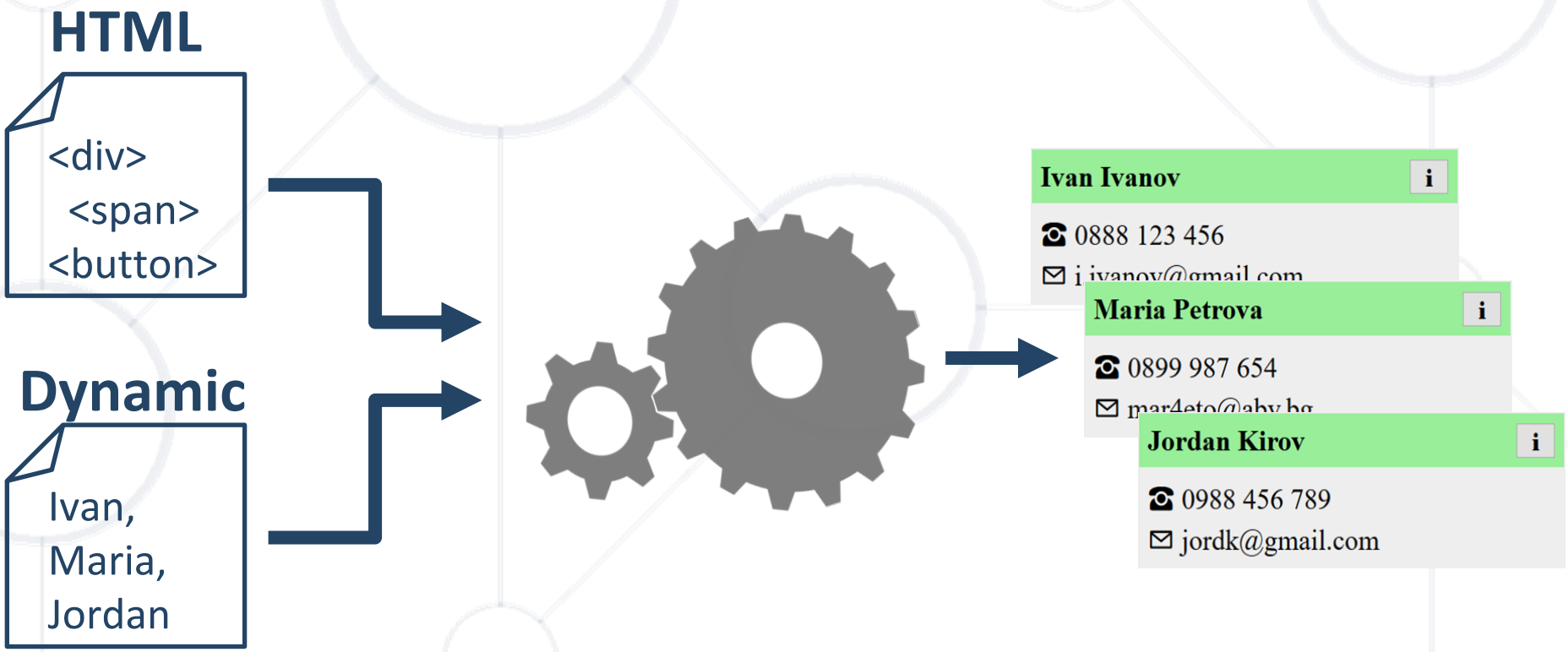
- And all **files** from the directory will be **public**

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

Templating Concepts

Templating

- Allows similar content to be **replicated** in a web page, **without repeating** the corresponding markup everywhere



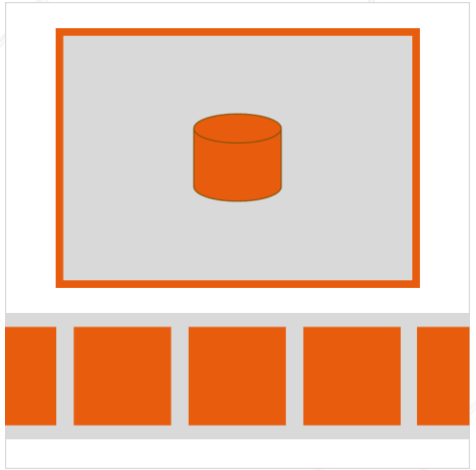
Templating Concepts

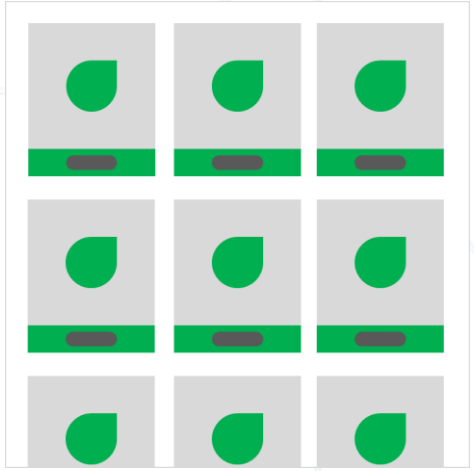
Software University

- The **static parts** of a webpage are stored as **templates**
- The **dynamic content** is kept separately (e.g. in a **database**)
- A **view engine** combines the two
- Benefits
  - **Productivity** - avoid writing the same markup over and over
  - **Easier upkeep** - only change the code in one place
  - **Composability** - a single element can be used on multiple pages

Examples

- Display articles in a blog
- Display a gallery of photos
- Visualize user profiles
- Show items in a catalog





Server View Engines

- Server view engines **return** ready-to-use **HTML** to the **client** (the browser)
  - They parse the **data** to **HTML** on the **server**
  - Web applications, created with **server** view engines are **not** real **SPA** apps (In **most** cases)
- Famous View Engines
  - Pug, Mustache, Handlebars, EJS, Vash

Templating with Handlebars

Handlebars

- Based on **Mustache** specification
- Expressions are **initialized** with '{{' and finish with '}}'

```
<div class="entry">
 <h1>{{title}}</h1>
  <div class="body">
   {{body}}
  </div>
</div>
```

```
<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    This is my first post!
  </div>
</div>
```

Integration in Express

```
npm install express-handlebars
```

```
const app = require('express')()
const exphbs= require('express-handlebars')
const handlebars = exphbs.create({ extname: '.hbs'});

app.engine('.hbs', handlebars.engine)
app.set('view engine', '.hbs')
```

```
app.get('/', function (req, res) {
  res.render('home');
})
```

Call the template

Handlebars directory structure

- Must have a **views** folder containing all the Handlebars templates:

- The **layouts** folder inside the views folder will contain the layouts or the template wrappers.
- The **main.hbs** file is the main layout.
- ***Note:*** The default name of the folders can be changed with an appropriate setting

For-Loops

- A template can be **repeated** for every entry in an **array**

```
const context = {
  contacts: [
    { name: 'Maria Petrova', email: 'mar4eto@abv.bg'},
    { name: 'Jordan Kirov', email: 'jordk@gmail.com'} ]};
```

```
<ul id="contacts">
  {{#each contacts}}
    <li>{{name}}: {{email}}</li>
  {{/each}}
</ul>
```

The expression inside the loop uses each **entry as context**

Conditional Statements

```
{{#if sunny}}
  The sky is clear
{{else}}
  The sky is overcast
{{/if}}
```

Variable to check for **truthiness**

Will be shown if the array is **empty**

```
<ul id="contacts">
  {{#each contacts}}
    <li>{{name}}: {{email}}</li>
  {{else}}
    <i>(empty)<i>
  {{/each}}
</ul>
```

Partials

- **Templates** that can be **inserted into** other templates

```
<div id="contacts">
  {{#each contacts}}
    {{> contact}}
  {{else}}
    <i>(empty)<i>
  {{/each}}
</div>
```

HTML Escaping

- By default, any strings that are evaluated will be **HTML-escaped**
- To prevent this, use the "**triple-stash**"

```
title: "All about <p> Tags"
body: "<p>This is a post about &lt;p&gt; tags</p>"
```

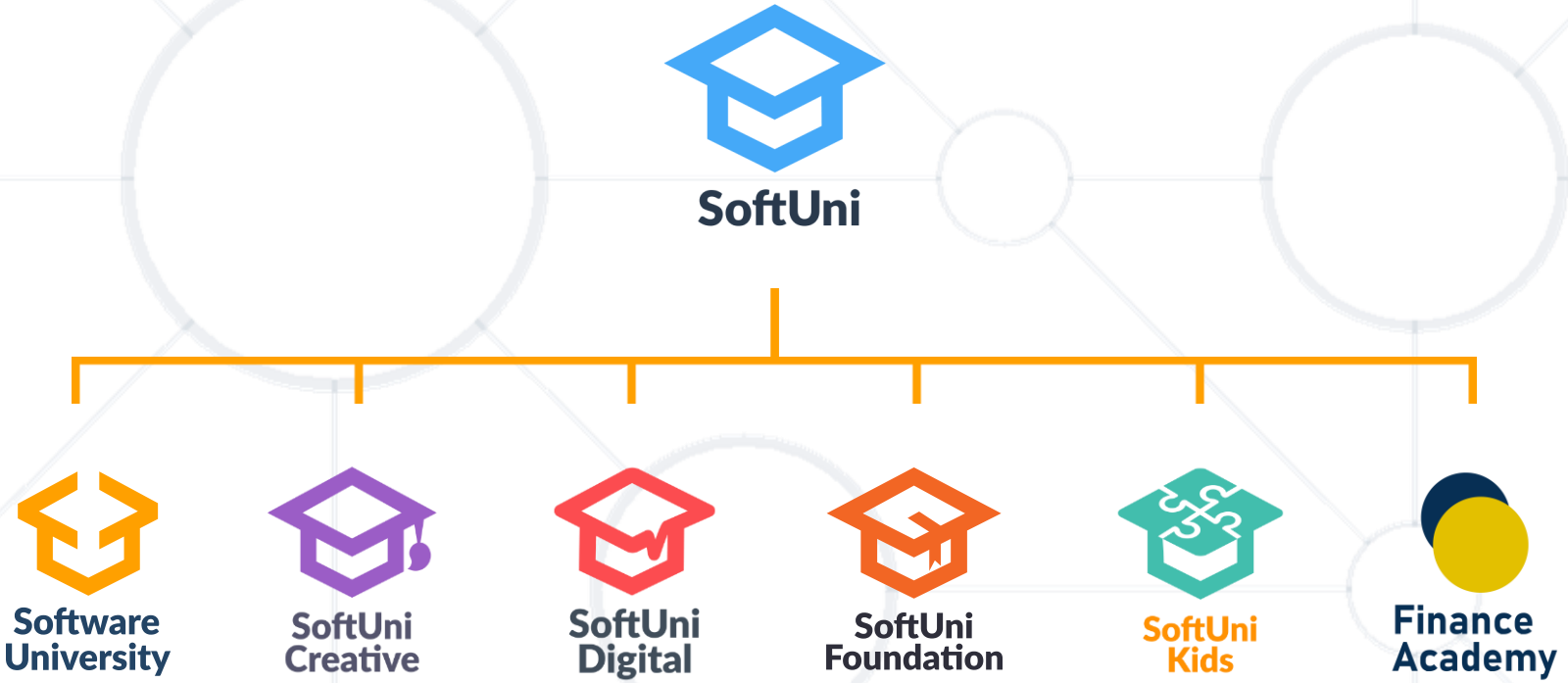
```
<h1>{{title}}</h1>
  <div class="body">
    {{{body}}}
  </div>
```

```
<h1>All About &lt;p&gt; Tags</h1>
  <div class="body">
    <p>This is a post about &lt;p&gt; tags</p>
  </div>
```

Summary

- Express.js is a **fast** web **framework** for Node.js
- **Middlewares** can **manipulate** requests and responses
- Templates **speed up** and **simplify** development
- View Engines **render templates**
- **Handlebars** offers effective templates and simple helper functions

Questions?



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

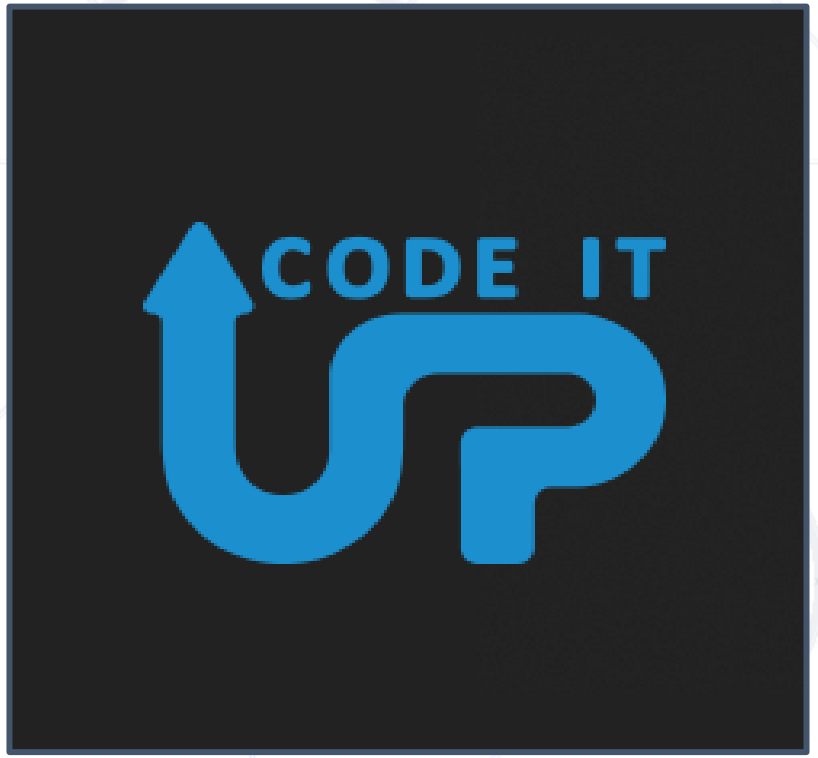
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

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

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://about.softuni.bg/
- © Software University – https://softuni.bg

NoSQL and MongoDB

NoSQL vs SQL, MongoDB, Mongoose

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

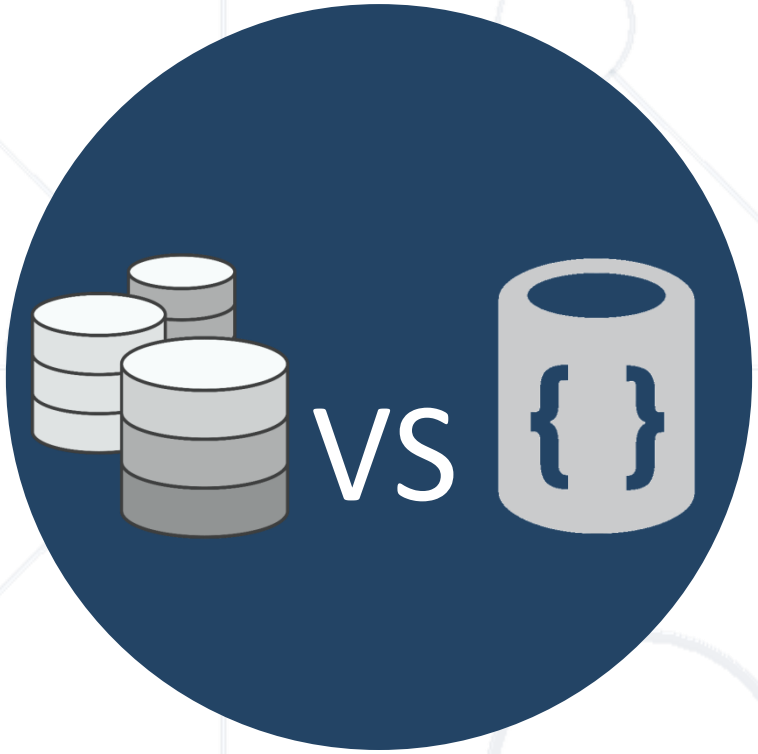
Table of Contents

Have a Question?

Software University

sli.do

#js-web



Relational and NoSQL Databases

Relational Database

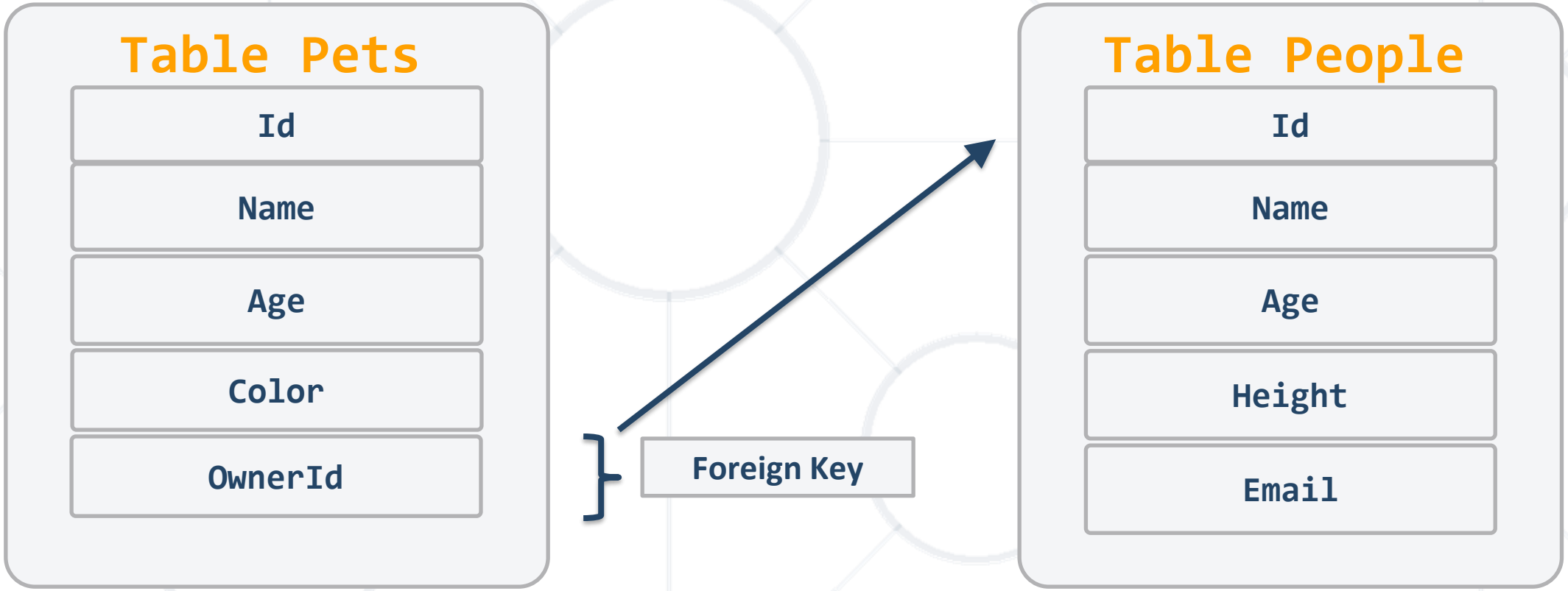
Software University

- Organize data into one or more **tables** of **columns** and **rows**
- Unique **key** identifying each **row** of data
- Almost all relational databases use **SQL** to **extract** data

```
SELECT * FROM Students
```

- **Relations** between tables are done using **Foreign Keys (FK)**
- Such databases are **Oracle**, **MariaDB**, **SQL Server**, etc…

Relational Database – Example



Non-relational Database (NoSQL)

- Key-value **stores**

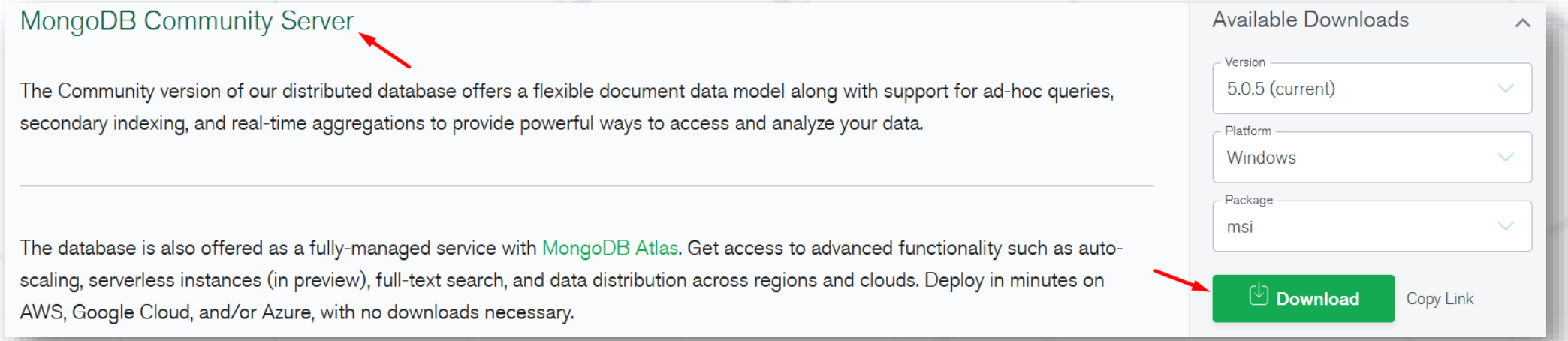
```
{
    "_id": ObjectId("59d3fe7ed81452db0933a871"),
    "email": "peter@gmail.com",
    "age": 22
}
```

- **SQL** query is **not** used in NoSQL systems
- More **scalable** and **provide** superior **performance**
- Such databases are **MongoDB**, **Cassandra**, **Redis**, etc..

MongoDB Overview

Install MongoDB

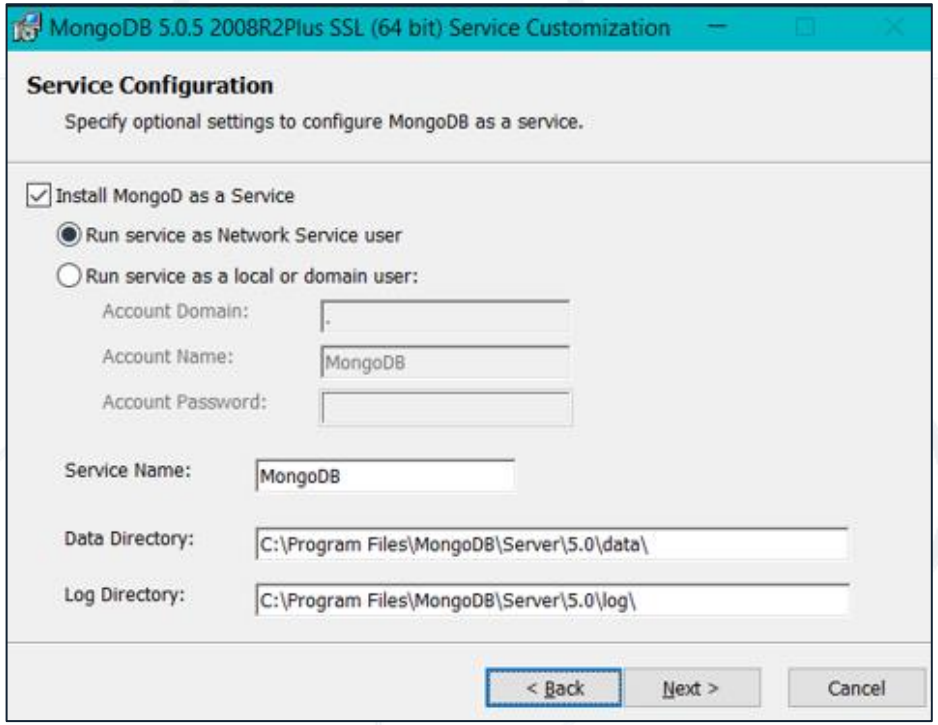
- Download from: mongodb.com/try/download/community



- The package includes **MongoDB Compass**
- When **installed**, MongoDB needs a **driver** (for every project)
  - Install MongoDB **driver** for Node.js `npm install mongodb`
  - We will be using **Mongoose** (includes a driver)

MongoD Windows Service

- During installation, configure the **MongoDB service**:



Manual Service Configuration

- Required if you **skipped** the service installation (and for Linux)
  - Go to installation folder and **run** a command prompt as an **administrator**
  - Type the following command

Usually in C:\Program Files\MongoDB\Server\3.4\bin

```
<path to mongod.exe> mongod --dbpath <path to store data>
```

  - Additional information at https://docs.mongodb.com/manual/tutorial/

Working with MongoDB Shell Client

- Start the shell from **another** CLI
  - Type the command **mongo**

```
show dbs
```

```
use mytestdb
```

```
db.mycollection.insertOne({"name":"George"})
```

```
db.mycollection.find({"name":" George"})
```

```
db.mycollection.find({})
```

  - Additional information at
    - https://docs.mongodb.com/manual/reference/mongo-shell/

Working with MongoDB GUI

- Choose one of the many (**Compass** is included in the installer)
- For example
  - Compass- https://www.mongodb.com/products/compass
  - Robo 3T- https://robomongo.org/download
  - NoSQLBooster- https://nosqlbooster.com

Working with MongoDB from Node.js – Example

Software University

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;
const connectionStr = 'mongodb://localhost:27017';
const client = new MongoClient(connectionStr,
{useUnifiedTopology: true});
client.connect(function(err) {
  const db = client.db('testdb');
  const people = db.collection('people');
  people.insertOne({ 'name': 'Ivan' }, (err, result) => {
    people.find({ name: 'Ivan' }).toArray((err, data) => {
      console.log(data);
    });
  });
});
```

Mongoose Overview

Mongoose Overview

Software University

- Mongoose is an object-document **model** module in Node.js for MongoDB
  - It **provides** a straight-forward, **schema-based** solution to **model** your application data
  - Includes build-in type **casting** and **validation**
  - **Extends** the native **queries** (much **easier** to use)
  - To **install** type in terminal/CMD (for every project)

```
npm install mongoose --save
```

Working with Mongoose in Node.js

Software University

- Load the following module

```
const mongoose = require('mongoose')
```

- **Connecting** to the database

```
async function main(){
    await mongoose.connect('mongodb://localhost:27017/testdb', {
          useNewUrlParser: true,
          useUnifiedTopology: true
    });
    console.log("Database connected")
}
main();
```

MongoDB Hosting

- Host a **database** in the largest MongoDB **cloud** service
- Go to '**mongo atlas**' and register - https://www.mongodb.com/cloud/atlas
- You can **store** up to 500 MB of **content**

Mongoose Models

Mongoose Models

- Mongoose **supports** models
  - Fixed **types** of documents
    - Used like object **constructors**
  - Needs a **mongoose.Schema** call

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
    firstName: String,
    lastName: String,
    facultyNumber: { type: String, required: true },
    age: Number
});

const Student= mongoose.model('Student', studentSchema);
```

Mongoose Models - Example

```
const myPerson = new Student({
        firstName: "John",
        lastName: "Peterson",
        facultyNumber: "5014sa",
        age: 25
    });

await myPerson.save();
const data = await Student.find({});
console.log(data); /* [{_id: new ObjectId("6139c6faf79365e5e54645bf"),
                        firstName: 'John',
                        lastName: 'Peterson',
                        facultyNumber: '5014sa',
                        age: 25, __v: 0}] */
```

Model Methods

- Since Mongoose models are just JavaScript **object constructors,** they can have **methods**

```
const studentSchema = new mongoose.Schema({…});

studentSchema.methods.getInfo = function() {
   return `I am ${this.firstName} ${this.lastName}`;
};
```

Do **not** use arrow functions

Model Virtual Properties

- Not all properties **need** to be **persisted** in the **database**
- Mongoose provides a way to **create** properties, that are accessible on all models, but are **not persisted** to the database
  - And they have both **getters** and **setters**

```
studentSchema.virtual('fullName').get(function () {
  return this.firstName + ' ' + this.lastName
});
```

Property Validation (1)

- With Mongoose developers can **define** custom **validation** on their **properties**
  - Validate records when trying to **save**

```
studentSchema.path('firstName')
  .validate(function () {
      return this.firstName.length >= 2
      && this.firstName.length <= 10
}, 'First name must be between 2 and 10 symbols long!')
```

Error message as second param

Property Validation (2)

- Mongoose has several built-in validators.
  - All **Schema-**Types have the built-in required validator.
  - Numbers have **min** and **max** validators.
  - Strings have **enum**, **regex**, **minLength**, and **maxLength** validators.

```
facultyNumber: {
   type: String,
   required: [true, 'FacultyNumber is required']
 }
```

Error message

- You can configure the error message for individual validators in your schema.

Exemplary validations

- Mongoose replaces **{VALUE}** with the value being validated.

```
const studentSchema = new mongoose.Schema({
  age: {
     type: Number,
     min: [0, 'Must be at least 0, got {VALUE}'],
     max: 50 }
});
```

Error message as second param

```
const studentSchema = new mongoose.Schema({
  facultyNumber: {
     type: String,
     enum: {
      values: ['50121', '50122', '50123'],
      message: '{VALUE} is not valid'
    } }
});
```

Exporting Modules

- Having all model definitions in the **main** module is **no** good
  - That is the reason Node.js has **modules** in the first place
  - In folder **models**, file **Student.js**

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
    firstName: { type: String, required: true },
    age: { type: Number }
});

module.exports = mongoose.model('Student', studentSchema);
```

Exporting and Using Modules

Software University

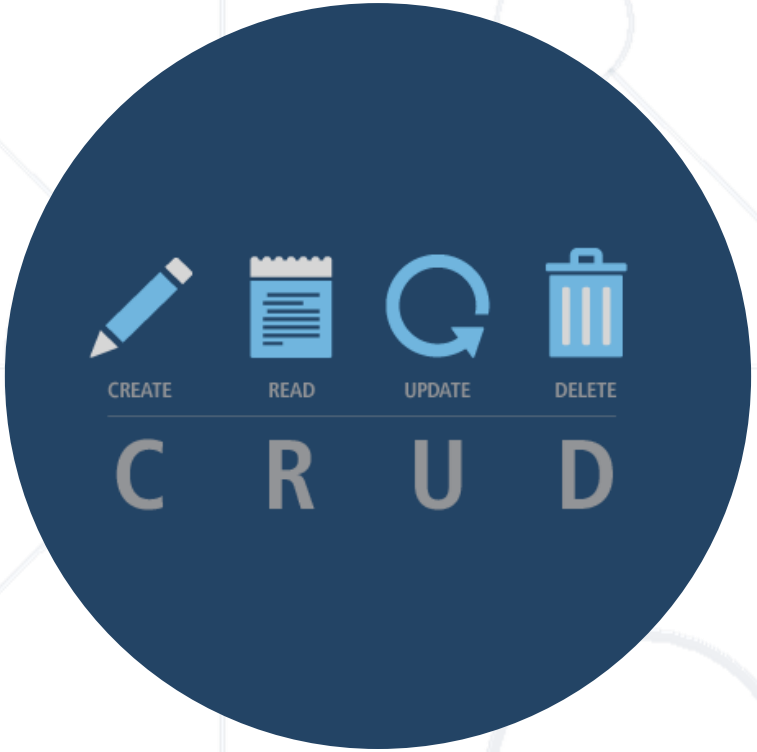
- Export the model definition:

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({ /* … */ });

module.exports = mongoose.model('Student', studentSchema);
```

- We can put each **model** in a different **module**, and **load** all models where they are needed

```
const Student = require('./models/Student');
```



CRUD with Mongoose

CRUD with Mongoose

Software University

- Mongoose supports **all** CRUD operations
  - Create (Persist data)

```
new Student({}).save(callback)
```

  - Read (Extract data)

```
Student.find({})
Student.findOne({conditions}, {options}, callback)
Student.findById(id, {options}, callback)
```

CRUD with Mongoose

Software University

- Update (Modify data)

```
Student
  .findByIdAndUpdate(id, {$set: {prop: newVal}}, callback)
Student
  .updateOne({filter}, {$set: {prop: newVal}}, callback)
Student
  .updateMany({filter}, {$set: {prop: newVal}}, callback)
```

- Delete (Remove data)

```
Student.findByIdAndDelete(id, callback)
Student.deleteOne({conditions}, {options}, callback)
Student.deleteMany({conditions}, {options}, callback)
```

Create Example

```
const mongoose = require('mongoose');
const connectionStr = 'mongodb://localhost:27017/unidb';
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true, minlength: 3 },
  age: Number
});
const Student = mongoose.model('Student', studentSchema);
mongoose.connect(connectionStr).then(() => {
  new Student({ name: 'Petar', age: 21 })
    .save()
    .then(student => {
      console.log(student._id)
    });
});
```

You can also use Student.create()

Read Example

```
Student
    .find({})
    .then(students => console.log(students))
    .catch(err => console.error(err))
Student
    .find({name: 'Petar'})
    .then(students => console.log(students))
Student
    .findOne({name: 'Petar'})
    .then(student => console.log(student))
```

Always handle errors

Update Example

```
Student
    .findById('57fb9fe1853ab747b0f692d1')
    .then((student) => {
      student.name = 'Stamat'
      student.save()
    });
Student
    .findByIdAndUpdate('57fb9fe1853ab747b0f692d1', {
      $set: { name: 'Petar' }
    }).then(students => console.log(students))
Student
    .updateOne(
      { name: 'Petar' },
      { $set: { name: 'Kiril' } }). then(students =>
console.log(students))
```

Remove & Count Example

```
Student
    .findByIdAndDelete('57fb9fe1853ab747b0f692d1').then()
Student
    .deleteOne({ name: 'Stamat' }).then()
Student
    .countDocuments()
    .then(console.log)
Student
    . countDocuments({ age: { $gt: 19 } })
    .then(console.log)
```

Remove by criteria

Get the count by criteria

Mongoose Queries

Mongoose Queries

Software University

- Mongoose defines **all** queries of the native MongoDB driver in a more **clear** and **useful** way
  - Instead of

```
{
  $or: [
    {conditionOne: true},
    {conditionTwo: true}
  ]
}
```

  - Do

```
.where({ conditionOne: true })
.or({ conditionTwo: true })
```

Mongoose Queries Example

- Mongoose supports **many** queries
  - For equality/non-equality

```
Student.findOne({'lastName':'Petrov'})
```

```
Student.find({}).where('age').gt(7).lt(14)
```

```
Student.find({}).where('facultyNumber').equals('12399')
```

  - Selection of some properties

```
Student.findOne({'lastName':'Kirilov'}).select('name age')
```

Mongoose Queries Example 2

- Sorting

```
Student.find({}).sort({age:-1})
```

- Limit & skip

```
Student.find({}).sort({age:-1}).skip(10).limit(10)
```

- Different methods could be **stacked** one upon the other

```
Student.find({})
       .where('firstName').equals('gosho')
       .where('age').gt(18).lt(65)
       .sort({age:-1})
       .skip(10)
       .limit(10)
```

Model Population

Population Definition

- Population is the process of **automatically replacing** the **specified paths** in the document with document(s) from **other** collection(s)
- We may **populate** a single document, multiple documents, plain object, multiple plain objects, or all objects returned from a query

Example

- We create **two models** that **reference** each other

```
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  facultyNumber: String,
  teacher: { type: Schema.Types.ObjectId, ref: 'Teacher' },
  subjects: [{ type: Schema.Types.ObjectId, ref: 'Subject' }]
});
const subjectSchema = new mongoose.Schema({
  title: String,
  students: [{ type: Schema.Types.ObjectId, ref: 'Student' }]
});
const Student = mongoose.model('Student', studentSchema);
const Subject = mongoose.model('Subject', subjectSchema);
```

Population

- To load all the data **referenced** with the entity use **populate()**

```
Student.findOne({ name: 'Peter' })
   .populate('subjects')
   .then(student => {
      console.log(student.subjects)
    })
```

Will return an array of **objects** and **NOT** Id's

- You can also load **multiple** paths

```
Student.findOne({ name: 'Peter' })
    .populate('subjects')
    .populate('teacher')
    .then(student => {
       console.log(student.teacher)
       console.log(student.subjects)
    })
```

Query Conditions

- Populate based on a **condition**

```
Subject.
  find({})
  .populate({
    path: 'students',
    match: { age: { $gte: 19 }},
    select: 'name facultyNumber',
    options: { limit: 3 }
  })
```

- More on populate here - mongoosejs.com/docs/populate.html

Summary

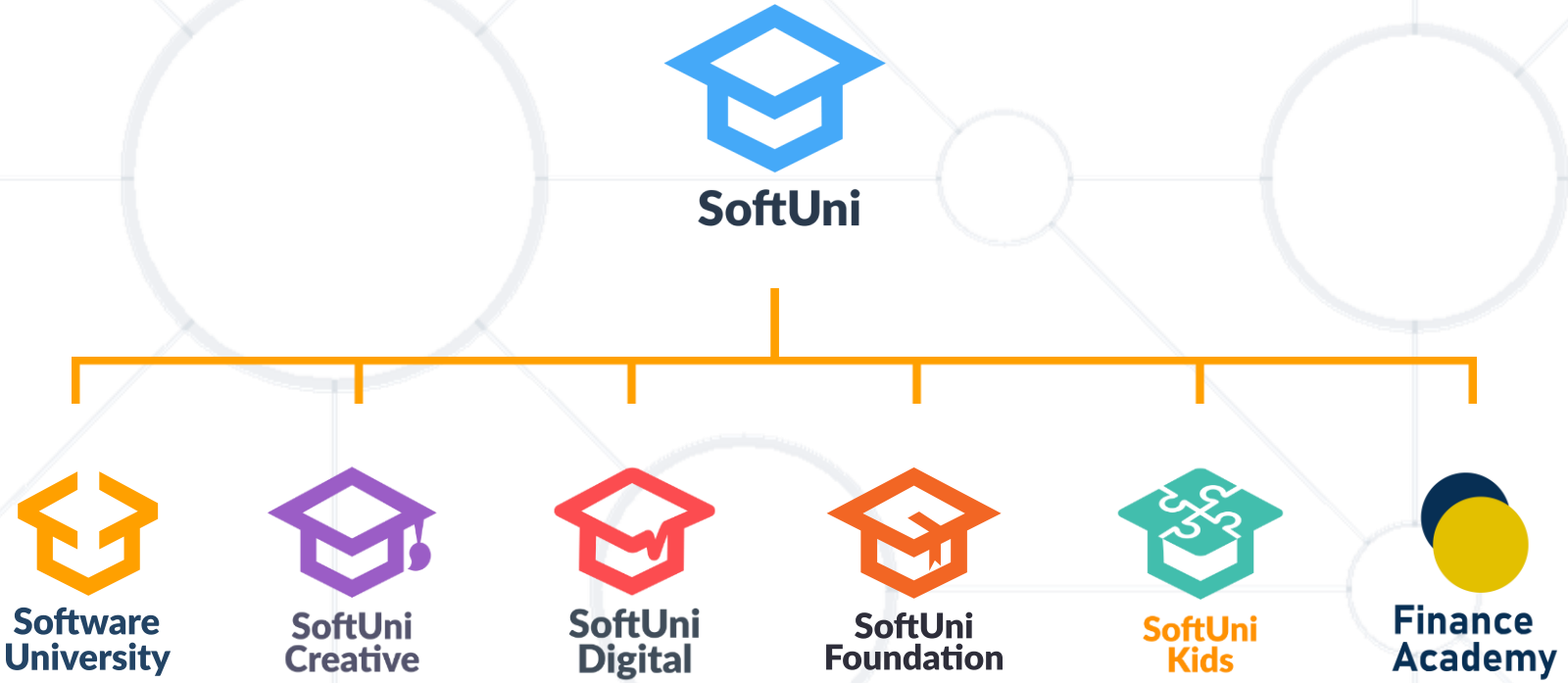
- NoSQL databases provide **superior** performance
- Mongoose gives us a **schema-based** solution

```
const modelSchema = new mongoose.Schema({
   propString: String
});
```

- Mongoose supports all **CRUD** operations
- Chaining queries with Mongoose is possible

```
Student.find({}).where('firstName').equals('gosho')
.where('age').gt(18).lt(65).sort({age:1}).skip(10)
.limit(10)
```

Questions?



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

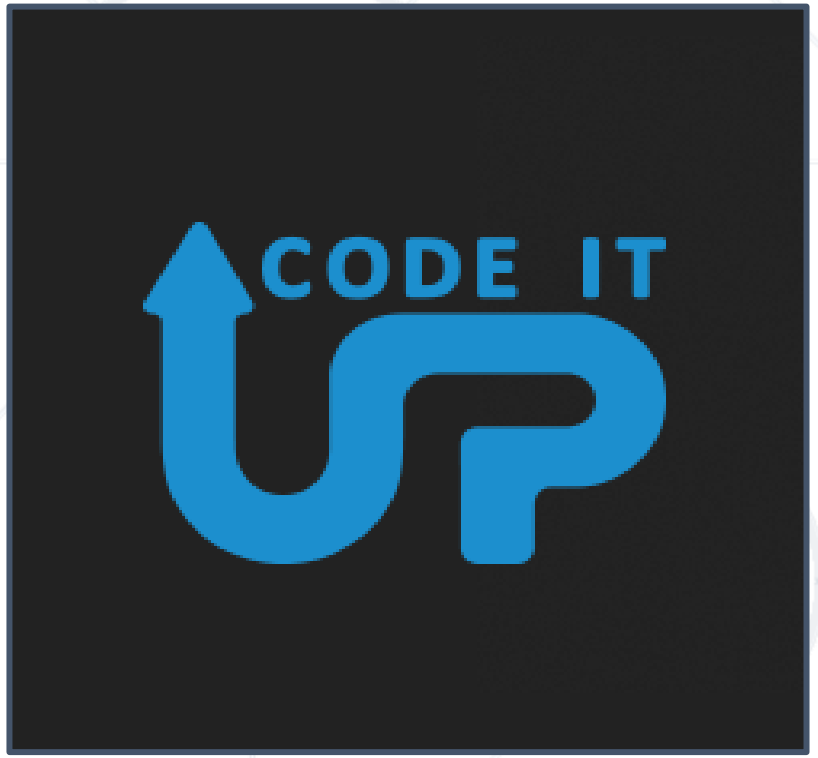
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

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

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://about.softuni.bg/
- © Software University – https://softuni.bg

Sessions and Authentication

State Persistence and Application Security

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-web

Cookies and Sessions

HTTP Communication

- HTTP is **stateless**
  - The Server and Client **don't remember** each other across requests
- To preserve state, **cookies** are stored on the **Client**
- **Session cookie** - exists on the **Server**
  - It can **store information** about a Client
  - Used to **persist state** across requests
  - Matched to a Client by their **cookie**

Session vs Cookie

- **Session** is preferred when you need to store **short-term** information/values
- **Cookies** are preferred when you need to store **long-term** information/values
- Session is **safer** because is stored on the server
- Cookies are not very safe. Expiration **can be set**, and it can last for years

Using Cookies

- You can use **cookie-parser** middleware for Express

```
npm install cookie-parser --save-exact
```

```
// use in an express app
const cookieParser = require('cookie-parser')
app.use(cookieParser())

app.get('/setCookie', (req, res) => {
  res.cookie("message", "hello")
  res.end('Cookie set')
})

app.get('/readCookie', (req, res) => {
  res.json(req.cookies)
})
```

Using Sessions

- You can use **express-session** middleware for Express

```
npm install express-session --save-exact
```

```
// use in an express app
const session = require('express-session')
app.use(session({ secret: 'my secret',
                  resave: false,
                  saveUnitialized: true,
                  cookie: {secure: true } }))
app.get('/setSession', (req, res) => {
  req.session.message = "hello"
  res.end('Session set')
})

app.get('/readSession', (req, res) => {
  res.json(req.session)
})
```

Live Demo

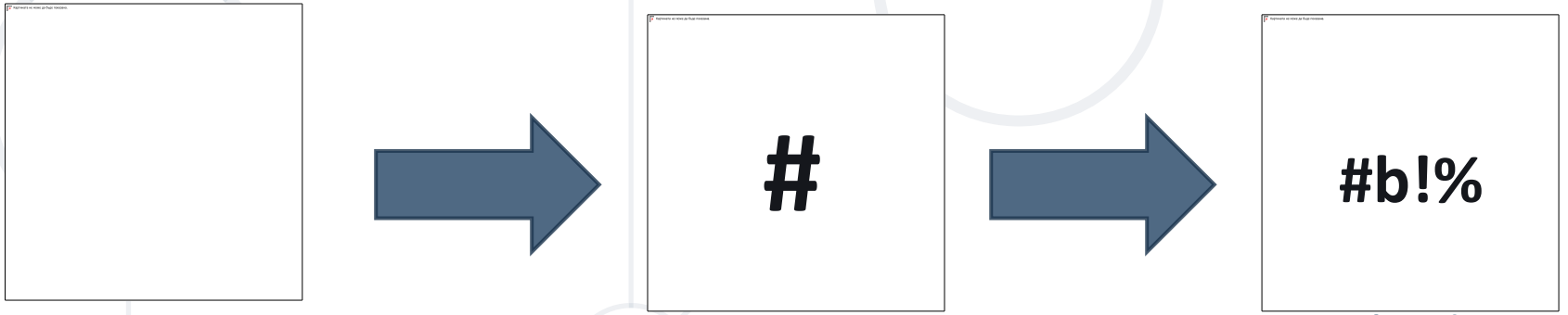
Authentication Concepts

Application Security

- **Authentication** is an important part of **application security**
- It serves to verify whether the **client** is who or what it declares itself to be
- It's built on several **layers of abstraction**
  - Cookies → Sessions → Security
- Authentication is **a cross-cutting concern**, best handled away from business logic
  - Request → Authentication → Business Logic → Response

Bcrypt

- **Bcrypt** is a password hashing function
  - Besides incorporating **salt** to protect against **rainbow table** attacks, **bcrypt** is an adaptive function
    - Over time, the iteration count can be increased to make it slower, so it remains resistant to **brute-force search attacks** even with increasing computation power



Bcrypt

- Installation

```
npm install bcrypt
```

- Hash password

```
const bcrypt = require('bcrypt');
const saltRounds = 9;
const myPlainTextPassword = "password123";

bcrypt.genSalt(saltRounds, (err, salt) => {
    bcrypt.hash(myPlainTextPassword, salt, (err, hash) => {
        console.log(hash);
        // $2b$09$pdhUAoT4qE0tmku.ZkXWROeLcJCy.LDRq.1I4IVImjrUTGuUbYQMi
    })});
```

Bcrypt

Software University

- Check password

```
const myPlainTextPassword = "password123";
const hash = "$2b$09$pdhUAoT4qE0tmku.ZkXWROeLcJCy.LDRq.1I4IVImjrUTGuUbYQMi";

bcrypt.compare(myPlainTextPassword, hash, (err, res) => {
    console.log(res); // true
});
```

- **Async** way is **recommended** to **hash** and **check** password

Authentication vs. Authorization

- **Authentication**
  - The process of **verifying the identity** of a user or computer
  - Questions: "**Who are you?**", "**How do you prove it?** "
  - Credentials can be a password, smart card, external token, etc...
- **Authorization**
  - The process of determining what a user is **permitted** to do on a computer or network
  - Questions: " **What are you allowed to do?**", "**Can you see this page?**"

Live Demo

JSON Web Token

What is JWT?

- **JSON Web Token** (**JWT**) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object
- This information can be verified and trusted because it is digitally signed
- JWTs can be signed using a secret or a public/private key pair using **RSA** or **ECDSA**

When Should You Use JWT?

- JSON Web Tokens are useful for
  - **Authorization** (a most common scenario) - Once the user is logged in, each subsequent request will include JWT, allowing the user to access routes, services, and resources that are permitted with that token
  - **Information Exchange** - JSON Web Tokens are good way of securely transmitting information between parties. Because they are signed digitally

JWT Structure

- In its compact form, JSON Web Tokens consist of three parts separated by dots (.)
  - **Header**
  - **Payload**
  - **Signature**

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

JWT Usage

- Installation

```
npm install jsonwebtoken
```

- Encode token

```
const jwt = require('jsonwebtoken');

const payloads = { _id, username };
const options = { expiresIn: '2d'};
const secret = 'MySuperPrivateSecret';
const token = jwt.sign(payload, secret, options);

console.log(token);
//eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXkiOiIxMjM0NTY3ODkwIiwibmFtZSI6
IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.xzK8LJQz0lDkJqsng04BYxcUQzxWngyEBP
```

JWT Usage

- Decode token

```
const token = req.cookies['token'] || sessionStorage.getItem('token');
// Depends where you store the token..

const decodedToken = jwt.verify(token, secretKey);

console.log(decodedToken); // { _id: ..., username: ... }
```

- More about JWT, you can find
  - https://jwt.io/
  - https://www.npmjs.com/package/jsonwebtoken

Live Demo

Summary

- Cookies and Sessions
  - Definitions and Usage
  - Cookies vs Sessions
- Authentication Concepts
  - Application Security with bcrypt
- JSON Web Token
  - What is JWT?
  - Structure and Usage

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

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

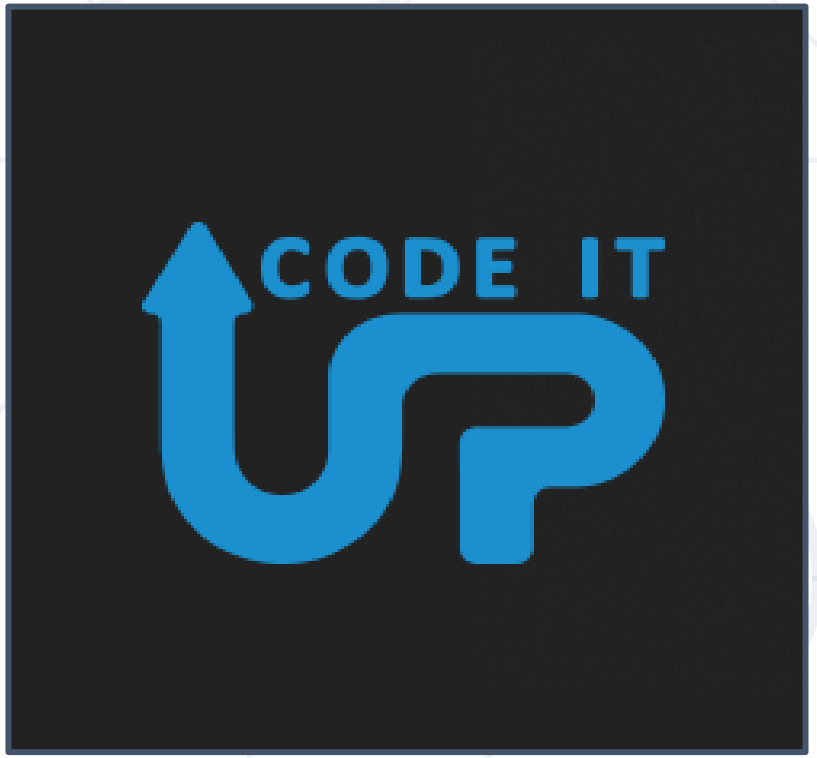
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

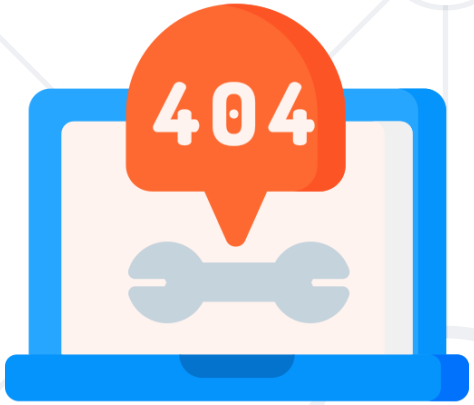
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
  - Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

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://about.softuni.bg/
- © Software University – https://softuni.bg

Validation and Error Handling

Validating User Input and Handle Different Type of Errors



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#js-web

Validation

Validation

- Why validate?
  - **Bigger app** === **more data** you will need from your users at some point of time
  - You should prevent the user from entering something **incorrect**
  - The validation can
    - either **succeed and allow** the data to be written to the database
    - **reject** the input and **return some information**

Validation

- How to validate?
  - **Client-Side**
    - Before any request is sent, we can use **HTML** or **JS** to approve the UX
    - It's optional because the user **can see**, **change** and **disable** the code in the browser
    - This is **not** a protection that secures you against incorrect data being sent to your server

Validation

- How to validate?
  - **Server-side**
    - The code **can't** be **seen**, **changed** or **disabled**, because it happens on the server, not in the browser.
    - **The server** is the place where you should add validation and filter out the invalid data
    - After that, you will be sure you only work with valid data and store the correct information into the database

Validation

- How to validate?
  - **Database**
    - For most database engines there is a **build-in validation** which you can turn on
    - It's **not required**, because there should be no scenario where your database work with invalid data
    - Make sure you have proper **server-side validation** and your database works with correct data

Validation

- **validator.js** - Is a library of string validators and sanitizers
  - Installation and Usage

```
npm install validator
```

    - Server-side usage

```
const validator = require('validator');
const body = req.body;
validator.isEmail(body.email); // true or false
```

    - Client-side usage

```
<script type="text/javascript" src="validator.min.js"></script>
<script type="text/javascript">
  validator.isEmail($('#email').val()); // true or false
</script>
```

Validation

- **express-validator** - Is a set of express.js middlewares that wraps **validator.js** validator and sanitizer functions
  - Installation and usage `npm install express-validator`

```
const { check, validationResult } = require('express-validator');

check('email').isEmail()
check('password').isLength({ min: 5 });

const errors = validationResult(req);

if(!errors.isEmpty()) // Return 422 status and export errors

// Create user...
```

Validation

- **Sanitizers** are functions that implement **sanitization** which is
  - Make sure that the data is in the right format
  - Removing any illegal character from the data
    - **normalizeEmail**: canonicalizes an email address
    - **trim**: trim characters from both sides of the input
    - **blacklist**: remove characters that appear on the blacklist
    - and more...

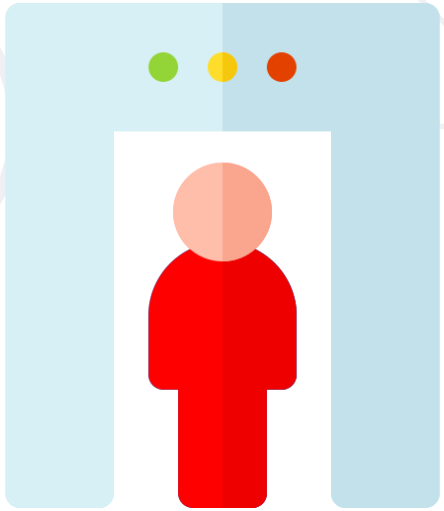
Validation

- **Sanitizing** input is also something that makes sense to be done
  - You can do it in one step by validating

```
const { body } = require('express-validator');
body('email')
      .isEmail() // check if the string is an email (validation)
      .normalizeEmail(), // canonicalizes an email address (sanitization)
body('password')
      .isLength({ min: 5 })
      .isAlphanumeric()
      .trim() // trim characters (whitespace by default) - sanitization
```

Validation

- The sanitization **mutates** the request
- This means that if **req.body.email** was sent
  - with the value "**PeteR@ood.bg**"
  - after the sanitization, its value will be "**peter@ood.bg**"



Validation

- Express-validators allows you to create **custom validations** and that send **custom messages**
- **Custom validator**

```
const { body } = require('express-validator');

app.post('/user', body.('email').custom(value => {
    return User.findUserByEmail(value)
        .then(user => {
            if(user){
                return Promise.reject('E-mail already in use');
            }
        });
}));
```

Validation

- **Custom Sanitizer**
  - Can be implemented by using the method `.customSanitizer()`

```
const { sanitizeParam } = require('express-validator');

app.post('/object/:id', sanitizeParam('id').customSanitizer(value => {
  return ObjectId(value);
}), (req, res) => {
  // Handle the request...
});
```

Mongoose Validation

- Validation is defined in the **SchemaType**
- Validation is middleware
  - Mongoose registers validation as a `pre('save')` hook
  - It's **asynchronously recursive**
  - can be customizable
- **A unique** option for schemas is not a validator
  - It's a convenient helper for building MongoDB unique indexes

Mongoose Save/Validate Hooks

- The **save()** function triggers **validate()** hook
  - all **pre('validate')** and **post('validate')** hooks get called before any **pre('save')** hook

```
schema.pre('validate', function() {
  console.log('this gets printed first');
});
schema.post('validate', function() {
  console.log('this gets printed second');
});
schema.pre('save', function() {
  console.log('this gets printed third');
});
schema.post('save', function() {
  console.log('this gets printed fourth');
});
```

Mongoose Built-in Validators

- All **SchemaTypes** have built-in required validator
  - **Numbers** have min and max validators
  - **Strings** have **enum**, **regex**, **minLength** and **maxLength**

```
const userSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true,
        minLength: 4,
        maxLength: 20
    }
});
```

Mongoose Custom Validators

- If the build-in validators aren't enough, you can define **custom validators** to suit your needs

```
const userSchema = new Schema({
  phone: {
    type: String,
    validate: {
      validator: function(v) {
        return /\d{3}-\d{3}-\d{4}/.test(v);
      },
      message: props => `${props.value} is not a valid phone number!`
    },
    required: [true, 'User phone number required']
  }
});
```

Mongoose Validation Errors

- Errors returned after failed validation contain an **error object** whose values are **ValidatorError** object
  - has a **kind**, **path**, **value** and **message** properties

```
toy.save((err) => {
      assert.equal(err.errors.color.message, 'Color');
      assert.equal(err.errors.color.kind, 'Invalid color');
      assert.eqial(err.errors.color.path, 'color');
      assert.equal(err.errors.color.value, 'Green');
      ...
});
```

Validation

- No matter which approaches you choose, in the end, some of the validations can fail
  - You should **always return** a helpful error **message** to the user
  - **Never reload** the page but always keep the user data inserted because that is a bad user experience
- More info
  - https://express-validator.github.io/docs/
  - https://mongoosejs.com/docs/validation.html

Validation Demo

Error Handing

Error Handling

- Errors in your code should be handled properly
- These errors can be different types
  - **Technical**/**Network** Errors
  - "**Usual**"/"**Expected**" Errors
  - **Bugs**/**Logical** Errors

Error Handling

- **Technical**/**Network** errors
  - MongoDB server might be down
- "**Usual**"/"**Expected**" Errors
  - File can't be read or some database operation fails
- **Bugs**/**Logical**
  - User object used when it doesn't exist
    - These errors are our fault
    - They should be fixed during development

Working with Errors

- An error is a **technical object** in a node application. This built-in error object can be thrown
  - Synchronous code
    - **try-catch**
  - Asynchronous code
    - **then()-catch()**
- In the end in both scenarios, you have to choice
  - Directly handle the error
  - Use **ExpressJS** functionality

Error Handling

- There is a scenario where you **can't continue**, but there is **no technical error**
  - If some user tries to login, but the username does not exist
  - You must check the values and decide what to do
    - Throw an error
    - Directly handle the "error"

Error Handling

- Handling errors synchronously

```
const User = require('../models/User/');

async (req, res, next) => {
    const { username, password } = req.body;
    try{
      const currentUser = await User.findOne({ username });
      // Login...
    } catch (e) {
      // Handle error properly...
    }
};
```

Error Handling

- Handling errors asynchronously

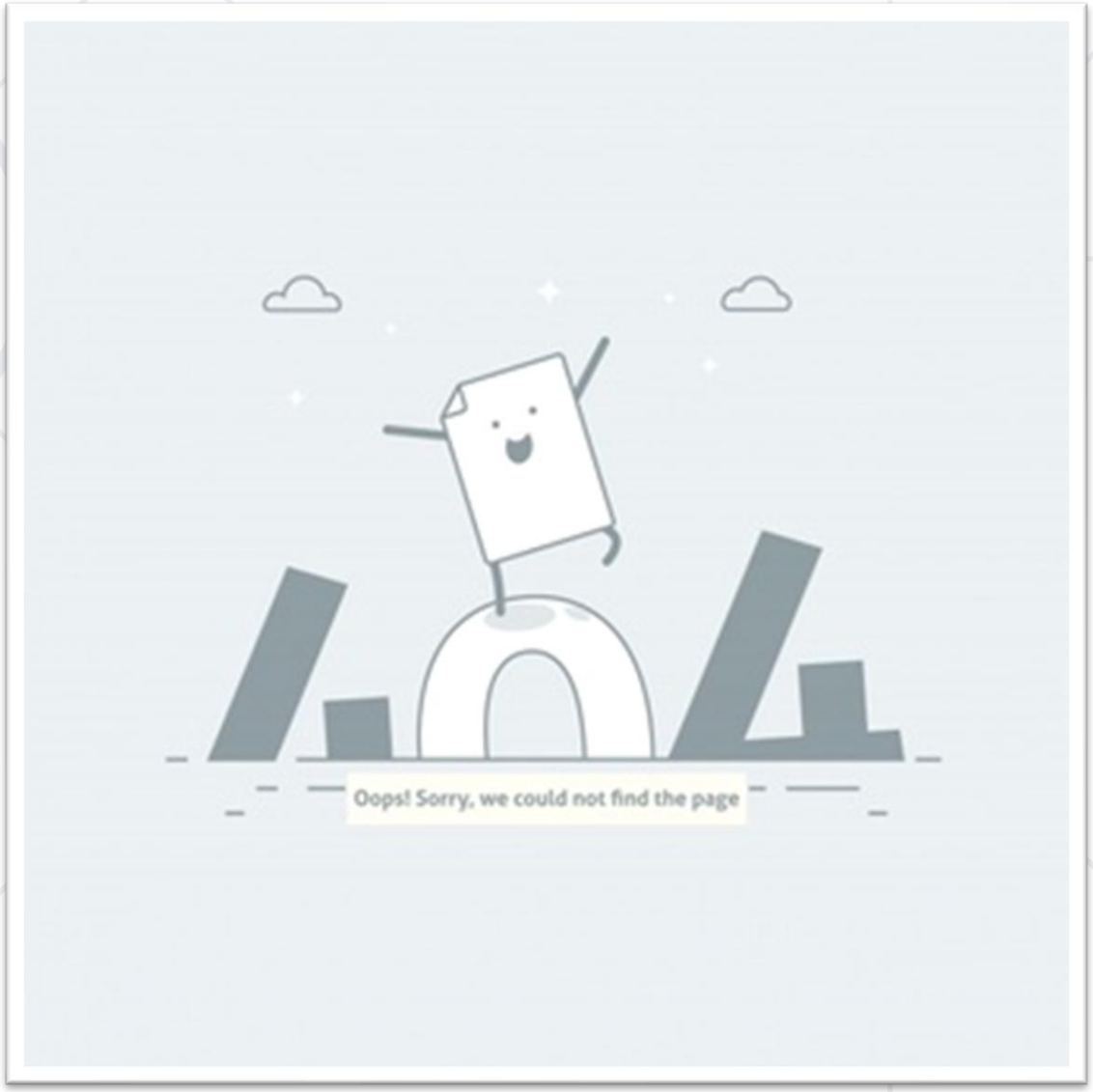
```
Post.findById(postId)
  .then((post) => {
    // Delete post
  })
  .catch(error => {
    if (!error.statusCode) {
       error.statusCode = 500;
    }
    next(error);
  })
```

If status code is missing, then something went wrong with the server

The error is sent to the middleware

Error Handling

- In all cases, you can
  - Return an **error page**
  - Return a response with **error information**
  - **Redirect**



Error Handling Demo

Summary

- Validation
  - Why and how validate data?
  - Validating and sanitization data with express-validator
  - Mongoose validator
- Error Handling
  - Different types of errors

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

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

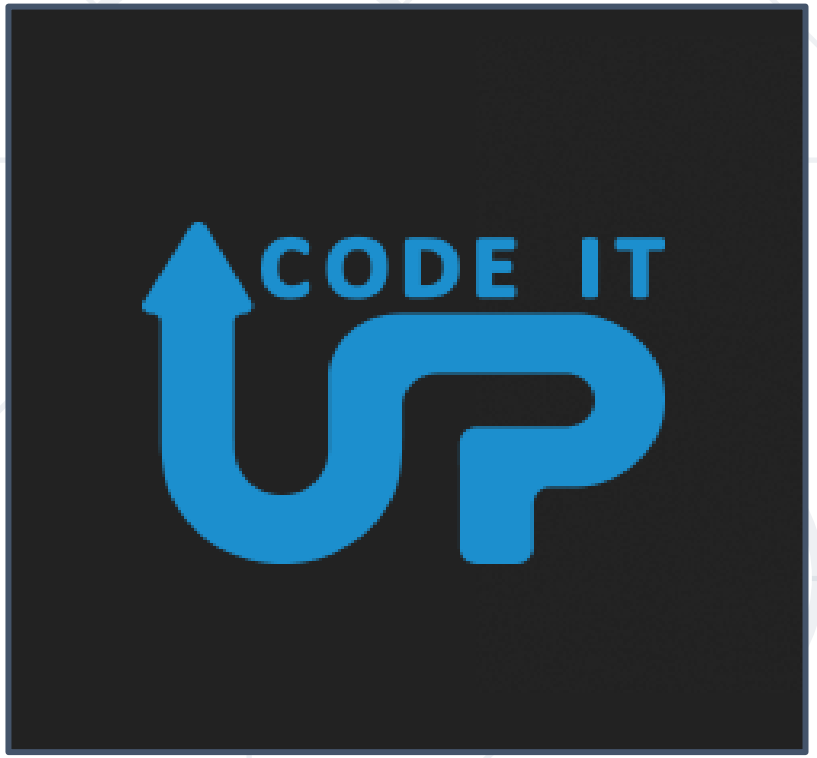
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

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

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://about.softuni.bg/
- © Software University – https://softuni.bg