A JavaScript REST API

Internet Applications, ID1354

Contents

Section

Server-Side Rendering (SSR), Loading the Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button, which means all HTML in the page is read from the server.

Server-Side Rendering (SSR), Loading the Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button, which means all HTML in the page is read from the server.
- Dynamic data is included on the server, before the HTML is sent to the client, for example using a JavaScript program.

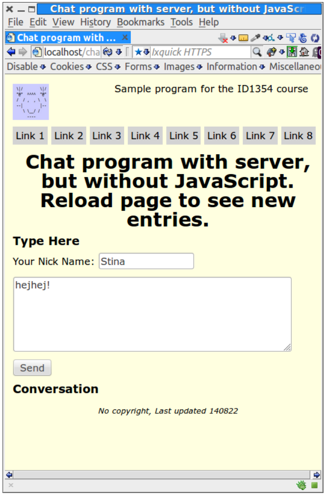
Server-Side Rendering (SSR), Loading the Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button, which means all HTML in the page is read from the server.
- Dynamic data is included on the server, before the HTML is sent to the client, for example using a JavaScript program.
  - This is called server-side rendering, SSR.

Server-Side Rendering (SSR), Loading the Entire Page

- Traditionally, an entire page is loaded when the user clicks a link or a button, which means all HTML in the page is read from the server.
- Dynamic data is included on the server, before the HTML is sent to the client, for example using a JavaScript program.
  - This is called server-side rendering, SSR.
- SSR might be appropriate if the entire page content really must change, but that is often not the case.

A Chat Example


Chat program with server, but without JavaScr
File Edit View History Bookmarks Tools Help
Chat program with ...
localhost/cha
Ixquick HTTPS
Disable Cookies CSS Forms Images Information Miscellaneo

Sample program for the ID1354 course

Link 1 Link 2 Link 3 Link 4 Link 5 Link 6 Link 7 Link 8

Chat program with server, but without JavaScript. Reload page to see new entries.

Type Here

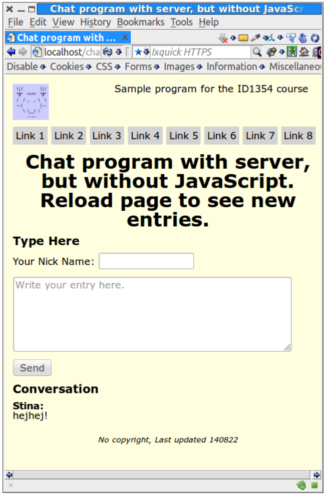
Your Nick Name: Stina

hejhej!

Send

Conversation

No copyright, Last updated 140822


Chat program with server, but without JavaScr
File Edit View History Bookmarks Tools Help
Chat program with ...
localhost/cha
Ixquick HTTPS
Disable Cookies CSS Forms Images Information Miscellaneo

Sample program for the ID1354 course

Link 1 Link 2 Link 3 Link 4 Link 5 Link 6 Link 7 Link 8

Chat program with server, but without JavaScript. Reload page to see new entries.

Type Here

Your Nick Name:

Write your entry here.

Send

Conversation

Stina:
hejhej!

No copyright, Last updated 140822

Consider for example the sample chat application. All that happens when the user clicks **`Send`** is that the new entry is added, the rest of the page is untouched.

Client-Side Rendering (CSR), Loading Only Data

- Today, most web applications use client-side rendering, CSR, where all html is generated on the client, by a JavaScript program running in the browser.

Client-Side Rendering (CSR), Loading Only Data

- Today, most web applications use client-side rendering, CSR, where all html is generated on the client, by a JavaScript program running in the browser.
- Only data is fetched from the server, over HTTP.

Client-Side Rendering (CSR), Loading Only Data

- Today, most web applications use client-side rendering, CSR, where all html is generated on the client, by a JavaScript program running in the browser.
- Only data is fetched from the server, over HTTP.
- An application using CSR is often called a single-page application, SPA, since only the initial load contains an ’html page’.

Client-Side Rendering (CSR), Loading Only Data

- Today, most web applications use client-side rendering, CSR, where all html is generated on the client, by a JavaScript program running in the browser.
- Only data is fetched from the server, over HTTP.
- An application using CSR is often called a single-page application, SPA, since only the initial load contains an 'html page'.
- This course focuses on CSR.

CSR vs SSR

- SSR advantages

CSR vs SSR

- SSR advantages
  - Better for search engines.

CSR vs SSR

- SSR advantages
  - Better for search engines.
  - The initial page load is faster, since with CSR the entire JavaScript program handling the client-side is loaded with the initial page.

CSR vs SSR

- SSR advantages
  - Better for search engines.
  - The initial page load is faster, since with CSR the entire JavaScript program handling the client-side is loaded with the initial page.
  - If the site contains only static pages with no dynamically generated data, CSR might not be meaningful.

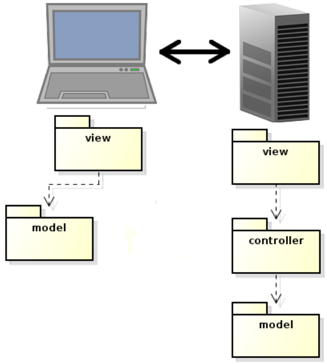
CSR vs SSR

- SSR advantages
  - Better for search engines.
  - The initial page load is faster, since with CSR the entire JavaScript program handling the client-side is loaded with the initial page.
  - If the site contains only static pages with no dynamically generated data, CSR might not be meaningful.
- CSR advantages
  - More user-friendly, richer, user interface.

CSR vs SSR

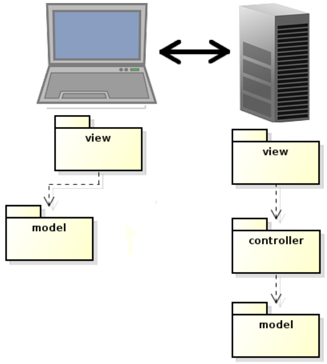
- SSR advantages
  - Better for search engines.
  - The initial page load is faster, since with CSR the entire JavaScript program handling the client-side is loaded with the initial page.
  - If the site contains only static pages with no dynamically generated data, CSR might not be meaningful.
- CSR advantages
  - More user-friendly, richer, user interface.
  - Fast rendering after the initial load.

Repetition: The MVVM Pattern



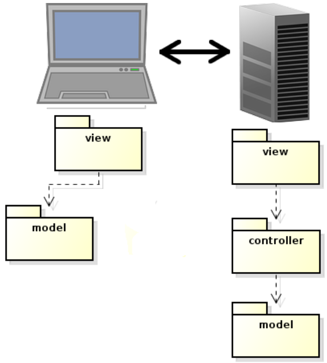
- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.

Repetition: The MVVM Pattern



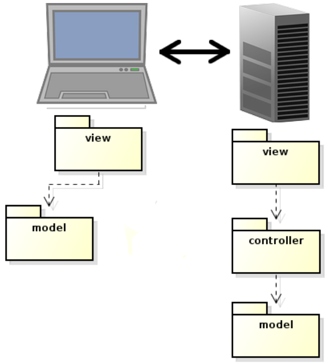
- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.

Repetition: The MVVM Pattern



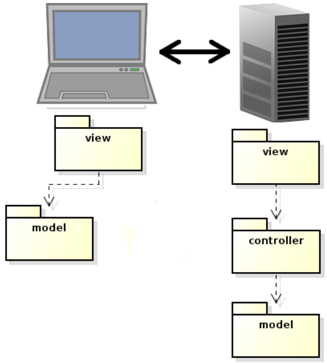
- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.
- Therefore, the viewmodel will always contain the current state of the application.

Repetition: The MVVM Pattern



- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.
- Therefore, the viewmodel will always contain the current state of the application.
- The browser view will reflect the viewmodel state, using the observer pattern.

Repetition: The MVVM Pattern



- The philosophy behind Model-View-ViewModel, MVVM, is to send only state changes from server to client.
- State changes, which means new data, are stored in the viewmodel.
- Therefore, the viewmodel will always contain the current state of the application.
- The browser view will reflect the viewmodel state, using the observer pattern.
- This is very useful for a CSR application.

AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.

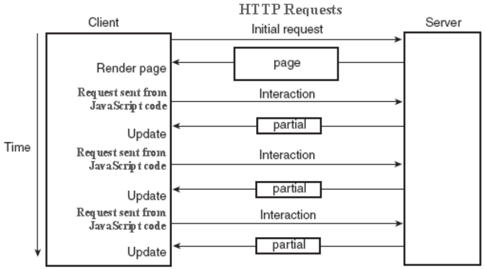
AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.
- AJAX is basically a way to use existing technologies, such as JavaScript, HTTP and XML.
  - No new language or markup.

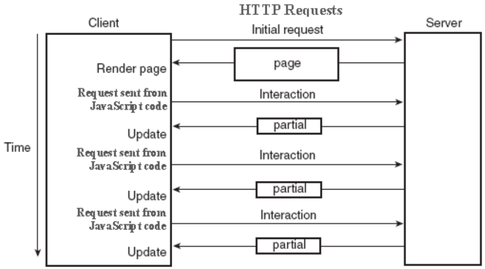
AJAX: To Load Only Data

- The dominating method to request data from the server, without reloading the web page, is **A**synchronous **J**avaScript **A**nd **X**ML, AJAX.
- AJAX is basically a way to use existing technologies, such as JavaScript, HTTP and XML.
  - No new language or markup.
- The only thing specific for AJAX is a JavaScript object, called **`XMLHttpRequest`**, which is standardized by W3C.

How Does It Work?

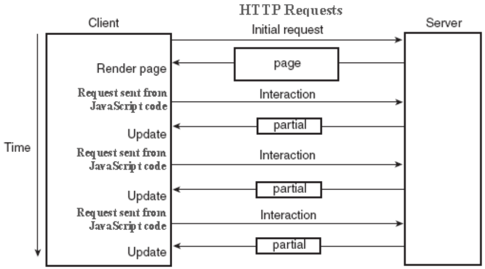


How Does It Work?



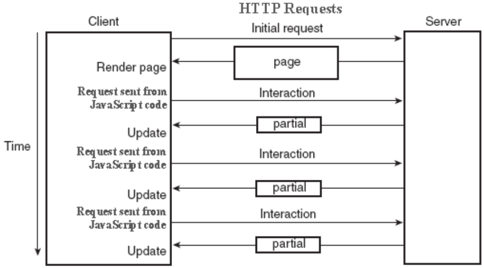
- Web page is loaded only on first request.

How Does It Work?



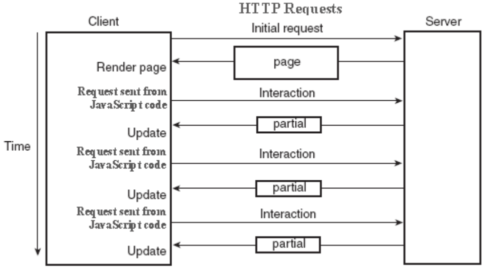
- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using **`XMLHttpRequest`**.

How Does It Work?



- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using **`XMLHttpRequest`**.
- The server returns only data, no markup.

How Does It Work?



- Web page is loaded only on first request.
- Subsequent requests come from JavaScript code, using **`XMLHttpRequest`**.
- The server returns only data, no markup.
- Returned data is available to JavaScript code, and is used to update the web page, by updating the DOM.

How Does It Work? (Cont'd)

- Note that AJAX requests are ordinary HTTP requests.

How Does It Work? (Cont'd)

- Note that AJAX requests are ordinary HTTP requests.
- The server directs the request to the resource specified in the URL, just as when loading a HTML document.

How Does It Work? (Cont’d)

- Note that AJAX requests are ordinary HTTP requests.
- The server directs the request to the resource specified in the URL, just as when loading a HTML document.
- An AJAX request is normally handled by a program, for example written in JavaScript, which generates a response containing the new data.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.

Data Format

- Client and server need to agree on the format of the data included in the HTTP response.
- **J**ava**S**cript **O**bject **N**otation, JSON, is normally used since it is compact and easy to translate to JavaScript objects.

JSON

- The JSON syntax is very simple:

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

```
"firstName":"Olle"
```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with **{** and **}**:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with `{` and `}`:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

  - Arrays are denoted with `[` and `]`:

    ```
    "employees":[
      {"firstName":"Olle", "lastName":"Olsson"},
      {"firstName":"Stina", "lastName":"Nilsson"}
    ]
    ```

JSON

- The JSON syntax is very simple:
  - Data is name/value pairs:

    ```
    "firstName":"Olle"
    ```

  - Data is separated by commas.
  - Objects are denoted with { and }:

    ```
    {"firstName":"Olle", "lastName":"Olsson"}
    ```

  - Arrays are denoted with [and]:

    ```
    "employees":[
      {"firstName":"Olle", "lastName":"Olsson"},
      {"firstName":"Stina", "lastName":"Nilsson"}
    ]
    ```

  - Data types are JavaScript types, for example string, **"abcd"**; integer, **`123`**; boolean, **`false`**

Section

The Server of an SPA

- We are about to develop the server-side of a single-page application, which will be organized as follows.

The Server of an SPA

- We are about to develop the server-side of a single-page application, which will be organized as follows.
  - The server will be a JavaScript program, answering to HTTP requests on a set of URLs.

The Server of an SPA

- We are about to develop the server-side of a single-page application, which will be organized as follows.
  - The server will be a JavaScript program, answering to HTTP requests on a set of URLs.
  - Each of those URLs, together with the HTTP method used in the request, will define a system operation, something the client is requesting the server to do.

The Server of an SPA

- We are about to develop the server-side of a single-page application, which will be organized as follows.
  - The server will be a JavaScript program, answering to HTTP requests on a set of URLs.
  - Each of those URLs, together with the HTTP method used in the request, will define a system operation, something the client is requesting the server to do.
  - For example, a HTTP GET call to the URL **`my.server.com/message`** tells the server to return all messages.

The Server of an SPA

- We are about to develop the server-side of a single-page application, which will be organized as follows.
  - The server will be a JavaScript program, answering to HTTP requests on a set of URLs.
  - Each of those URLs, together with the HTTP method used in the request, will define a system operation, something the client is requesting the server to do.
  - For example, a HTTP GET call to the URL `my.server.com/message` tells the server to return all messages.
- The set of URLs and HTTP methods handled by the server is often called a REST API, or just an API.

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.
    - Addressable resources

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.
    - Addressable resources
    - A uniform, constrained interface

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.
    - Addressable resources
    - A uniform, constrained interface
    - Representation-oriented

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.
    - Addressable resources
    - A uniform, constrained interface
    - Representation-oriented
    - Stateless communication

REST

- While often used just to denote such an API, Representational State Transfer, REST, is in reality much more.
- REST is actually the following set of architectural principles, a REST API is just one possible way to implement the principles.
    - Addressable resources
    - A uniform, constrained interface
    - Representation-oriented
    - Stateless communication
    - Hypermedia as the engine of application state, HATEOS or HATEOAS

Addressable resources

- A resource is any abstraction handled by the application, for example a message or a user in a chat program.

Addressable resources

- A resource is any abstraction handled by the application, for example a message or a user in a chat program.
- Each resource must be addressable via a URI, which in the case of a web application means it must be reachable at a URL.

Addressable resources

- A resource is any abstraction handled by the application, for example a message or a user in a chat program.
- Each resource must be addressable via a URI, which in the case of a web application means it must be reachable at a URL.
- This makes all resources linkable, references to resources can be embedded in for example documents or emails.

Addressable Resources, Contd

An example JSON document representing chat messages:

```
{
  "success": [
    {
      "id": 1,
      "msg": "hi Stina",
      "deletedAt": null,
      "author": "http://localhost:7777/user/2"
    },
    {
      "id": 2,
      "msg": "hi Nisse",
      "deletedAt": null,
      "author": "http://localhost:7777/user/1"
    }
  ]
}
```

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.
- The most important HTTP methods are **`GET`**, **`POST`**, **`PUT`** and **`DELETE`**.

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.
- The most important HTTP methods are **`GET`**, **`POST`**, **`PUT`** and **`DELETE`**.
  - **`POST`** is used to create a resource.

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.
- The most important HTTP methods are **`GET`**, **`POST`**, **`PUT`** and **`DELETE`**.
  - **`POST`** is used to create a resource.
  - **`GET`** is used to read one or more resources.

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.
- The most important HTTP methods are **`GET`**, **`POST`**, **`PUT`** and **`DELETE`**.
  - **`POST`** is used to create a resource.
  - **`GET`** is used to read one or more resources.
  - **`PUT`** is used to update a resource.

A Uniform, Constrained Interface

- Use only methods defined in the communication protocol.
- The most important HTTP methods are **GET**, **POST**, **PUT** and **DELETE**.
  - **POST** is used to create a resource.
  - **GET** is used to read one or more resources.
  - **PUT** is used to update a resource.
  - **DELETE** is used to delete a resource

Advantages of a Uniform, Constrained Interface

- Familiarity, given a URL to a resource you know which operations can be performed on the resource.

Advantages of a Uniform, Constrained Interface

- Familiarity, given a URL to a resource you know which operations can be performed on the resource.
- Interoperability, a service can be called using any HTTP client.

Advantages of a Uniform, Constrained Interface

- Familiarity, given a URL to a resource you know which operations can be performed on the resource.
- Interoperability, a service can be called using any HTTP client.
- Scalability,
  - All HTTP caching facilities can be used.

Advantages of a Uniform, Constrained Interface

- Familiarity, given a URL to a resource you know which operations can be performed on the resource.
- Interoperability, a service can be called using any HTTP client.
- Scalability,
    - All HTTP caching facilities can be used.
    - No problems with duplicate message delivery when using HTTP idempotent methods.

Representation-Oriented

- Client and server exchange representations of the resource identified by the URL, the entire representation, *object*, is transferred in each message.

Representation-Oriented

- Client and server exchange representations of the resource identified by the URL, the entire representation, *object*, is transferred in each message.
- The representation is in the HTTP body, the format could be for example JSON.

Representation-Oriented

- Client and server exchange representations of the resource identified by the URL, the entire representation, *object*, is transferred in each message.
- The representation is in the HTTP body, the format could be for example JSON.
- The format is specified with the content-type HTTP header, for example application/json.

Representation-Oriented

- Client and server exchange representations of the resource identified by the URL, the entire representation, *object*, is transferred in each message.
- The representation is in the HTTP body, the format could be for example JSON.
- The format is specified with the content-type HTTP header, for example application/json.
- Different clients can use different formats.

Stateless Communication

- No client session data stored on the server, all data needed to handle a request must be included in the request.

Stateless Communication

- No client session data stored on the server, all data needed to handle a request must be included in the request.
- Means that, if the application has more than one server, any server can serve any request from any client.

Hypermedia as the Engine of Application State, HATEOAS

- Since all resources are addressable, links to them can be embedded in representations, see the example on the Addressable Resources slide above.

Hypermedia as the Engine of Application State, HATEOAS

- Since all resources are addressable, links to them can be embedded in representations, see the example on the Addressable Resources slide above.
- This, together with the Constrained Interface and Representation-Oriented principles, leads to object graphs in message bodies.

HATEOAS, Cont'd

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

HATEOAS, Cont'd

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

  - The body is an object, delimited by '{' and '}'.

HATEOAS, Cont'd

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

  - The body is an object, delimited by '{' and '}'.
  - The object has four attributes, **`id`**, **`msg`**, **`deletedAt`** and **`author`**.

HATEOAS, Cont’d

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

  - The body is an object, delimited by ’{’ and ’}’.
  - The object has four attributes, **id**, **msg**, **deletedAt** and **author**.
  - The value of the attribute **author** is a reference to another object.

HATEOAS, Cont'd

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

  - The body is an object, delimited by '{' and '}'.
  - The object has four attributes, **`id`**, **`msg`**, **`deletedAt`** and **`author`**.
  - The value of the attribute **`author`** is a reference to another object.
  - All objects have the methods **`get`**, **`post`**, **`put`** and **`delete`**.

HATEOAS, Cont'd

- Consider an HTTP body consisting of a chat message:

```
{
  "id": 1,
  "msg": "hi Stina",
  "deletedAt": null,
  "author": "http://localhost:7777/user/2"
},
```

  - The body is an object, delimited by '{' and '}'.
  - The object has four attributes, **`id`**, **`msg`**, **`deletedAt`** and **`author`**.
  - The value of the attribute **`author`** is a reference to another object.
  - All objects have the methods **`get`**, **`post`**, **`put`** and **`delete`**.
- This is object-oriented HTTP.

Section

We Always Need a Framework!

- Remember that our goal is to only write application specific code.

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).
    - Map an HTTP request to a JavaScript method handling that request.

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).
  - Map an HTTP request to a JavaScript method handling that request.
  - Send exceptions to error handling methods.

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).
  - Map an HTTP request to a JavaScript method handling that request.
  - Send exceptions to error handling methods.
  - Serve static files (html, css, images, etc).

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).
  - Map an HTTP request to a JavaScript method handling that request.
  - Send exceptions to error handling methods.
  - Serve static files (html, css, images, etc).
  - Make security checks, for example if the user is logged in, before handling a request.

We Always Need a Framework!

- Remember that our goal is to only write application specific code.
- But a web application contains a lot of code that is not application specific (infrastructure code).
  - Map an HTTP request to a JavaScript method handling that request.
  - Send exceptions to error handling methods.
  - Serve static files (html, css, images, etc).
  - Make security checks, for example if the user is logged in, before handling a request.
- Thus, it is mandatory to use a web application framework in order to not write infrastructure code.

Why Express.js?

- Express.js is a web application framework for node.js.

Why Express.js?

- Express.js is a web application framework for node.js.
- Small, quite easy to understand.

Why Express.js?

- Express.js is a web application framework for node.js.
- Small, quite easy to understand.
- Handles the most fundamental web application infrastructure code.

Why Express.js?

- Express.js is a web application framework for node.js.
- Small, quite easy to understand.
- Handles the most fundamental web application infrastructure code.
- Eliminates a lot of the infrastructure code from applications written in the course.

Why Express.js?

- Express.js is a web application framework for node.js.
- Small, quite easy to understand.
- Handles the most fundamental web application infrastructure code.
- Eliminates a lot of the infrastructure code from applications written in the course.
- Very widely used and very actively maintained.

Install Express

- Express, like all other node utilities, is installed with npm.

Install Express

- Express, like all other node utilities, is installed with npm.
- Add express to the **`dependencies`** object in **`package.json`**

  ```
  "express": "^4.17.1"
  ```

  `^4.17.1` means any 4.x version bigger than 4.17.1, which is the current version when this presentation is written.

Install Express

- Express, like all other node utilities, is installed with npm.
- Add express to the **`dependencies`** object in **`package.json`**

  ```
  "express": "^4.17.1"
  ```

  `^4.17.1` means any 4.x version bigger than 4.17.1, which is the current version when this presentation is written.
- Install by executing the command **`npm install`** in the project root, the directory where **`package.json`** is located.

The First Express App

```
1 const express = require('express');
2 const app = express();
3 const port = 8001;
4
5 app.get('/', (req, res) =>
6   res.send('Server is up'));
7
8 app.listen(port, () =>
9   console.log(`Server running at ${port}!`));
```

- Line 1 loads express.

The First Express App

```
1  const express = require('express');
2  const app = express();
3  const port = 8001;
4
5  app.get('/', (req, res) =>
6    res.send('Server is up'));
7
8  app.listen(port, () =>
9    console.log(`Server running at ${port}!`));
```

- Line 1 loads express.
- Line 2 creates an express application.

The First Express App

```
1 const express = require('express');
2 const app = express();
3 const port = 8001;
4
5 app.get('/', (req, res) =>
6   res.send('Server is up'));
7
8 app.listen(port, () =>
9   console.log(`Server running at ${port}!`));
```

- Line 1 loads express.
- Line 2 creates an express application.
- The call to **`app.get`** on line 5 tells express that an HTTP GET request to the path **`/`** shall execute the callback function **`(req, res)=> res.send('Server is up')`**

The First Express App, Cont’d

```
1  const express = require('express');
2  const app = express();
3  const port = 8001;
4
5  app.get('/', (req, res) =>
6    res.send('Server is up'));
7
8  app.listen(port, () =>
9    console.log(`Server running at ${port}!`));
```

- Line 6 calls the method **`send`** in the **`res`** object.

The First Express App, Cont'd

```
1  const express = require('express');
2  const app = express();
3  const port = 8001;
4
5  app.get('/', (req, res) =>
6    res.send('Server is up'));
7
8  app.listen(port, () =>
9    console.log(`Server running at ${port}!`));
```

- Line 6 calls the method **`send`** in the **`res`** object.
- The **`res`** object represents the HTTP response, the **`send`** method sends an HTTP response containing only the specified string.

The First Express App, Cont'd

```
1 const express = require('express');
2 const app = express();
3 const port = 8001;
4
5 app.get('/', (req, res) =>
6   res.send('Server is up'));
7
8 app.listen(port, () =>
9   console.log(`Server running at ${port}!`));
```

- The call to **`app.listen`** on line 8 makes the server listen for http requests on the specified port.

The First Express App, Cont'd

```
1  const express = require('express');
2  const app = express();
3  const port = 8001;
4
5  app.get('/', (req, res) =>
6    res.send('Server is up'));
7
8  app.listen(port, () =>
9    console.log(`Server running at ${port}!`));
```

- The call to **`app.listen`** on line 8 makes the server listen for http requests on the specified port.
- The callback function on line 9 is executed when the server has started.

A Word About Nodemon

- nodemon is a utility that monitors the files in a web application and restarts the application if any file changes.

A Word About Nodemon

- nodemon is a utility that monitors the files in a web application and restarts the application if any file changes.
- In **`package.json`**, the server is started by executing

  ```
  nodemon.js ./scripts/server
  ```

- Installation is as usual with npm, add nodemon to the dependency section in **`package.json`** and execute **`npm install`**.

The Express Objects

- The express framework consists of the four objects **`application`**, **`request`**, **`response`** and **`router`**.

The Express Objects

- The express framework consists of the four objects **`application`**, **`request`**, **`response`** and **`router`**.
    - Application represents the entire app, and is used to route HTTP requests to JavaScript functions.

The Express Objects

- The express framework consists of the four objects **`application`**, **`request`**, **`response`** and **`router`**.
  - Application represents the entire app, and is used to route HTTP requests to JavaScript functions.
  - Request represents the HTTP request, and is used to get information about the request.

The Express Objects

- The express framework consists of the four objects **`application`**, **`request`**, **`response`** and **`router`**.
  - Application represents the entire app, and is used to route HTTP requests to JavaScript functions.
  - Request represents the HTTP request, and is used to get information about the request.
  - Response represents the HTTP response, and is used to send a response.

The Express Objects

- The express framework consists of the four objects **`application`**, **`request`**, **`response`** and **`router`**.
    - Application represents the entire app, and is used to route HTTP requests to JavaScript functions.
    - Request represents the HTTP request, and is used to get information about the request.
    - Response represents the HTTP response, and is used to send a response.
    - Router is like a mini-application, which can contain its own HTTP request to JavaScript function mappings. The entire router object is mapped to a URL by the **`application`** object.

The `Application` Object, Routing

- We have seen how a HTTP GET was mapped to the root, `/`, by calling

```
app.get('/', (req, res) => //handle request)
```

The **Application** Object, Routing

- We have seen how a HTTP GET was mapped to the root, /, by calling

```
app.get('/', (req, res) => //handle request)
```

- Any request using HTTP GET, POST, PUT or DELETE, can be mapped to any URL in the same way.

The `Application` Object, Routing

- We have seen how a HTTP GET was mapped to the root, /, by calling

```
app.get('/', (req, res) => //handle request)
```

- Any request using HTTP GET, POST, PUT or DELETE, can be mapped to any URL in the same way.
- For example, a POST request is mapped to the URL **`/abc/def`** by calling

```
app.post('/abc/def', (req, res) => // ...)
```

The **`Application`** Object, Middleware

- A middleware function performs some action before a request is passed to its request handling function.

The **`Application`** Object, Middleware

- A middleware function performs some action before a request is passed to its request handling function.
- Typically used for non-functional requirements, for example logging or security.

The **Application** Object, Middleware, Cont'd

```
app.use((req, res, next) => {
  console.log(
      neq Date().toLocaleTimeString() + ': ' +
      req.method + ' ' + req.originalUrl +
      ' from ' + req.ip
  );
  next();
});
```

- A middleware function is registered by calling **app.use**. The example above registers a function that logs all HTTP requests.

The **Application** Object, Middleware, Cont'd

```
1 app.use((req, res, next) => {
2   console.log(
3       neq Date().toLocaleTimeString() + ': ' +
4       req.method + ' ' + req.originalUrl +
5       ' from ' + req.ip
6   );
7   next();
8 });
```

- A middleware function is registered by calling **`app.use`**. The example above registers a function that logs all HTTP requests.
- Note the call to **`next()`** on line 7. That is when the request handling function (or next middleware function) is invoked.

The **`Application`** Object, Middleware, Cont’d

```
1 app.use((req, res, next) => {
2   console.log(
3       neq Date().toLocaleTimeString() + ': ' +
4       req.method + ' ' + req.originalUrl +
5       ' from ' + req.ip
6   );
7   next();
8 });
```

- A middleware function is registered by calling **`app.use`**. The example above registers a function that logs all HTTP requests.
- Note the call to **`next()`** on line 7. That is when the request handling function (or next middleware function) is invoked.
- Without a call to **`next`**, the request handling is terminated when the middleware function returns.

The **`Request`** Object, Parameters

- The **`request`** object is used to get information about the request.

The **`Request`** Object, Parameters

- The **`request`** object is used to get information about the request.
- The logging middleware function above illustrated how to get for example the HTTP method and the URL of the request.

The **Request** Object, Parameters

- The **request** object is used to get information about the request.
- The logging middleware function above illustrated how to get for example the HTTP method and the URL of the request.
- When writing a REST API, it is common to include a resource id as a part of the URL. An HTTP GET to the URL **/user/2** means to read the suer with id 2.
- To extract such a parameter, the **request** object is used as below.

```
app.get('/user/:id', (req, res) =>
  res.send('The id is ' + req.params.id));
```

The **Request** Object, Body

- When the user is uploading a new state of a resource, for example by creating a new message in a chat, that new state might be in the request body.

The **`Request`** Object, Body

- When the user is uploading a new state of a resource, for example by creating a new message in a chat, that new state might be in the request body.
- Neither node itself, nor express, parses body data. We need an npm package that handles the parsing.

The **Request** Object, Body

- When the user is uploading a new state of a resource, for example by creating a new message in a chat, that new state might be in the request body.
- Neither node itself, nor express, parses body data. We need an npm package that handles the parsing.
- The sample application uses the package **`body-parser`**, as illustrated on the next slide.

The **Request** Object, Body, Cont'd

```
1  const bodyParser = require('body-parser');
2  app.use(bodyParser.json());
3
4  app.post('/msg', (req, res) =>
5    res.send('The body is ' + req.body.msg));
```

- Line one loads the body-parser package.

The `Request` Object, Body, Cont'd

```
1  const bodyParser = require('body-parser');
2  app.use(bodyParser.json());
3
4  app.post('/msg', (req, res) =>
5    res.send('The body is ' + req.body.msg));
```

- Line one loads the body-parser package.
- Line two registers a middleware function that can parse bodies in JSON format. The parser will add the parsed objects to the express **`request`** object.

The **Request** Object, Body, Cont'd

```
1  const bodyParser = require('body-parser');
2  app.use(bodyParser.json());
3
4  app.post('/msg', (req, res) =>
5    res.send('The body is ' + req.body.msg));
```

- Line one loads the body-parser package.
- Line two registers a middleware function that can parse bodies in JSON format. The parser will add the parsed objects to the express **request** object.
- If the request body contained an variable named **msg**, that object is contained in the parameter **req.body.msg**, on line five.

The **Response** Object

- The response object is used to send an HTTP response.

The **`Response`** Object

- The response object is used to send an HTTP response.
- As seen in many examples above, the **`send`** method is used to send a response.

The **`Response`** Object

- The response object is used to send an HTTP response.
- As seen in many examples above, the **`send`** method is used to send a response.
  - **`send`** performs many tasks, for example to set the **`Content-Length`** response header and cache-related headers.

The **`Response`** Object

- The response object is used to send an HTTP response.
- As seen in many examples above, the **`send`** method is used to send a response.
  - **`send`** performs many tasks, for example to set the **`Content-Length`** response header and cache-related headers.
  - The default status code is 200 (OK).

The **Response** Object

- The response object is used to send an HTTP response.
- As seen in many examples above, the **send** method is used to send a response.
  - **send** performs many tasks, for example to set the **Content-Length** response header and cache-related headers.
  - The default status code is 200 (OK).
- The HTTP status code can be changed by calling **res.status**.

The **Response** Object

- The response object is used to send an HTTP response.
- As seen in many examples above, the **send** method is used to send a response.
  - **send** performs many tasks, for example to set the **Content-Length** response header and cache-related headers.
  - The default status code is 200 (OK).
- The HTTP status code can be changed by calling **res.status**.
- If the response has no body, terminate the response by calling **end**.

```
app.get('/', //status is 404, body is empty
  (req, res) => res.status(404).end());
app.get('/', //status is 200, body is empty
  (req, res) => res.end());
```

The **Response** Object, Cont'd

- If the body shall be in JSON format, as is normally the case in a REST API, use the method **json** instead of **send**.

```
app.get('/',
        (req, res) =>
           res.json({param1: 'abc123'}));
```

- If the name of a JSON object property can not be hardcoded, use square bracket notation for a computed property name.

```
const propName = 'param1';
app.get('/',
        (req, res) =>
          res.json({[propName]: 'abc123'}));
```

The **Router** Object

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

const msgApi = express.Router();
msgApi.get('/:id', (req, res) =>
  res.send('returning msg ' + req.params.id)
);
msgApi.delete('/:id', (req, res) =>
  res.send('deleted msg ' + req.params.id)
);

app.use('/msg', msgApi);
```

- A router can be considered a mini-application, containing only middleware and request handlers.

The **Router** Object

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

const msgApi = express.Router();
msgApi.get('/:id', (req, res) =>
  res.send('returning msg ' + req.params.id)
);
msgApi.delete('/:id', (req, res) =>
  res.send('deleted msg ' + req.params.id)
);

app.use('/msg', msgApi);
```

- A router can be considered a mini-application, containing only middleware and request handlers.
- Can make the code easier to read, by grouping related request handlers.

The **Router** Object

```
 1  const express = require('express');
 2
 3  const msgApi = express.Router();
 4  msgApi.get('/:id', (req, res) =>
 5    res.send('returning msg ' + req.params.id)
 6  );
 7  msgApi.delete('/:id', (req, res) =>
 8    res.send('deleted msg ' + req.params.id)
 9  );
10
11  app.use('/msg', msgApi);
```

- A router can be considered a mini-application, containing only middleware and request handlers.
- Can make the code easier to read, by grouping related request handlers.
- The example above creates an API for message handling on lines 3-9, and adds the API to the application on line 11.

Section

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
    - Authentication (login)
    - Settings
    - Advertising
    - Shopping basket
- This is often solved with cookies.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket
- This is often solved with cookies.
- A cookie is a name/value pair passed between browser and server in the HTTP header.

Cookies

- HTTP is stateless. Still there are many reasons why it is useful for a server to identify the client.
  - Authentication (login)
  - Settings
  - Advertising
  - Shopping basket
- This is often solved with cookies.
- A cookie is a name/value pair passed between browser and server in the HTTP header.
- A cookie is only passed to the server from which it originated.

Cookies and REST

- Can we really use cookies in a REST API? Doesn't that break the requirement that a REST API must be stateless?

Cookies and REST

- Can we really use cookies in a REST API? Doesn't that break the requirement that a REST API must be stateless?
- How to handle authentication in a REST API has been debated a lot.

Cookies and REST

- Can we really use cookies in a REST API? Doesn't that break the requirement that a REST API must be stateless?
- How to handle authentication in a REST API has been debated a lot.
- The problem is that if there shall be no state stored on the server, then all state, including proof of authentication, must be included in each request.

Cookies and REST

- Can we really use cookies in a REST API? Doesn’t that break the requirement that a REST API must be stateless?
- How to handle authentication in a REST API has been debated a lot.
- The problem is that if there shall be no state stored on the server, then all state, including proof of authentication, must be included in each request.
  - Including the user’s credentials in each request brings a security risk, since then the credentials must be managed by the client.

Cookies and REST

- Can we really use cookies in a REST API? Doesn't that break the requirement that a REST API must be stateless?
- How to handle authentication in a REST API has been debated a lot.
- The problem is that if there shall be no state stored on the server, then all state, including proof of authentication, must be included in each request.
  - Including the user's credentials in each request brings a security risk, since then the credentials must be managed by the client.
  - There is good overview of this problem at `http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/`

Cookies and REST, Cont'd

- An often used compromise is to return a token identifying the user to a client that has authenticated.

Cookies and REST, Cont'd

- An often used compromise is to return a token identifying the user to a client that has authenticated.
- The client must then include that token in the **`Authorization`** HTTP header of each request.

Cookies and REST, Cont'd

- An often used compromise is to return a token identifying the user to a client that has authenticated.
- The client must then include that token in the `Authorization` HTTP header of each request.
- However, also a cookie is passed as an HTTP request header. Therefore, it is not worse, from a REST point of view, to pass the token in a cookie. Using a cookie facilitates client side handling.

Cookies and REST, Cont'd

- An often used compromise is to return a token identifying the user to a client that has authenticated.
- The client must then include that token in the `Authorization` HTTP header of each request.
- However, also a cookie is passed as an HTTP request header. Therefore, it is not worse, from a REST point of view, to pass the token in a cookie. Using a cookie facilitates client side handling.
  - This solution is very common, and is also used in the course.

Cookies and REST, Cont’d

- The argument against the solution on the previous slide is that we just reinvented HTTP sessions, and had to write the session management ourselves.

Cookies and REST, Cont'd

- The argument against the solution on the previous slide is that we just reinvented HTTP sessions, and had to write the session management ourselves.
- In reply, we can claim that we are now free to place the token also in the **`Authorization`** header, which is required in some cases.

Cookies and REST, Cont'd

- The argument against the solution on the previous slide is that we just reinvented HTTP sessions, and had to write the session management ourselves.
- In reply, we can claim that we are now free to place the token also in the `Authorization` header, which is required in some cases.
- We could also argue that we have better control over the "session" handling, we are for example free to choose format of the token.

Cookies and REST, Cont'd

- The argument against the solution on the previous slide is that we just reinvented HTTP sessions, and had to write the session management ourselves.
- In reply, we can claim that we are now free to place the token also in the `Authorization` header, which is required in some cases.
- We could also argue that we have better control over the "session" handling, we are for example free to choose format of the token.
- And so the debate continues...

To Set a Cookie

- Cookies are set with the **`cookie`** function of the express **`response`** object. Since cookies are HTTP headers, this function must be called before any output is sent.

To Set a Cookie

- Cookies are set with the **cookie** function of the express **response** object. Since cookies are HTTP headers, this function must be called before any output is sent.

```
res.cookie('name', 'value', {expires: 0});
```

To Set a Cookie

- Cookies are set with the **`cookie`** function of the express **`response`** object. Since cookies are HTTP headers, this function must be called before any output is sent.

```
res.cookie('name', 'value', {expires: 0});
```

- **`name`** and **`value`** is the cookie's name/value pair.

To Set a Cookie

- Cookies are set with the **cookie** function of the express **response** object. Since cookies are HTTP headers, this function must be called before any output is sent.

```
res.cookie('name', 'value', {expires: 0});
```

- **name** and **value** is the cookie's name/value pair.
- **expire** tells the instant in time when the cookie expires. Setting it to zero creates a session cookie, which means it should be discarded when the browser is closed.

To Set a Cookie

- Cookies are set with the **`cookie`** function of the express **`response`** object. Since cookies are HTTP headers, this function must be called before any output is sent.

```
res.cookie('name', 'value', {expires: 0});
```

- **`name`** and **`value`** is the cookie's name/value pair.
- **`expire`** tells the instant in time when the cookie expires. Setting it to zero creates a session cookie, which means it should be discarded when the browser is closed.
  - However, web browsers may use session restoring, which means session state, including cookies, is not discarded.

To Retrieve a Cookie

- Cookies are retrieved using the cookie-parser npm package.

To Retrieve a Cookie

- Cookies are retrieved using the cookie-parser npm package.
  - Registers a middleware function that retrieves all cookies in the HTTP request and stores them in the object **`request.cookies`**.

To Retrieve a Cookie

- Cookies are retrieved using the cookie-parser npm package.
  - Registers a middleware function that retrieves all cookies in the HTTP request and stores them in the object `request.cookies`.
- The following code registers the **`cookie-parser`** middleware.

```
const cookieParser =
  require('cookie-parser');
app.use(cookieParser());
```

To Retrieve a Cookie

- Cookies are retrieved using the cookie-parser npm package.
  - Registers a middleware function that retrieves all cookies in the HTTP request and stores them in the object `request.cookies`.
- The following code registers the `cookie-parser` middleware.

```
const cookieParser =
  require('cookie-parser');
app.use(cookieParser());
```

- The following code retrieves the cookies with the name **`cookieName`**.

```
const theCookie = req.cookies.cookieName;
```

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.
- If many servers set the same third party cookie, the third party server can track the user's surfing.

Third Party Cookies

- Cookies set by a server with a domain name different from the server's.
- If many servers set the same third party cookie, the third party server can track the user's surfing.
- Typically used for marketing.

Third Party Cookies

- Cookies set by a server with a domain name different from the server’s.
- If many servers set the same third party cookie, the third party server can track the user’s surfing.
- Typically used for marketing.
- There are many other ways, beside cookies, to identify a user for tracking purposes.
  - Flash, Silverlight and HTML5 storages
  - HTML5 canvas painting
  - content of caches and cache tags like Last-Modified or ETag
  - social networks
  - fingerprinting mechanisms like supported ciphersuites, DNS content, HTTP headers, plugins and fonts, clock drift, CPU and GPU benchmarks, network level information, user behavior

The EU Cookie Law

A person shall not store or gain access to information stored, in the terminal equipment of a subscriber or user unless the requirements of paragraph (2) are met.

(2) The requirements are that the subscriber or user of that terminal equipment

1. is provided with clear and comprehensive information about the purposes of the storage of, or access to, that information; and
2. has given their consent.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
  - Not relevant here.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
  - Not relevant here.
- The cookie is strictly necessary for the provision of an information society service requested by the subscriber or user.

Exceptions To The Law

- The cookie is for the sole purpose of carrying out the transmission of a communication over an electronic communications network.
    - Not relevant here.
- The cookie is strictly necessary for the provision of an information society service requested by the subscriber or user.
    - Applies to for example shopping baskets and authentication in an internet bank.

General Data Protection Regulation, GDPR

A data subject's (user's) personal data may be handled **only** if one of the following conditions apply. (From `https://gdpr.eu/what-is-gdpr/`)

- The data subject gave you specific, unambiguous consent to process the data. (e.g. **They have opted in to your marketing email list**.)

General Data Protection Regulation, GDPR

A data subject's (user's) personal data may be handled **only** if one of the following conditions apply. (From `https://gdpr.eu/what-is-gdpr/`)

- The data subject gave you specific, unambiguous consent to process the data. (e.g. **They have opted in to your marketing email list**.)
- Processing is necessary to execute or to prepare to enter into a contract to which the data subject is a party. (e.g. **You need to do a background check before leasing property to a prospective tenant**.)

General Data Protection Regulation, GDPR

A data subject's (user's) personal data may be handled **only** if one of the following conditions apply. (From `https://gdpr.eu/what-is-gdpr/`)

- The data subject gave you specific, unambiguous consent to process the data. (e.g. **They have opted in to your marketing email list**.)
- Processing is necessary to execute or to prepare to enter into a contract to which the data subject is a party. (e.g. **You need to do a background check before leasing property to a prospective tenant**.)
- You need to process it to comply with a legal obligation of yours. (e.g. **You receive an order from the court in your jurisdiction**.)

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) personal may be handled **only** if one of the following conditions apply. (Text from `https://gdpr.eu/what-is-gdpr/`)

- You need to process the data to **save somebody's life**. (e.g. Well, you'll probably know when this one applies.)

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) personal may be handled **only** if one of the following conditions apply. (Text from `https://gdpr.eu/what-is-gdpr/`)

- You need to process the data to **save somebody's life**. (e.g. Well, you'll probably know when this one applies.)
- Processing is necessary to perform a task in the public interest or to carry out some official function. (e.g. **You're a private garbage collection company**.)

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) personal may be handled **only** if one of the following conditions apply. (Text from `https://gdpr.eu/what-is-gdpr/`)

- You need to process the data to **save somebody's life**. (e.g. Well, you'll probably know when this one applies.)
- Processing is necessary to perform a task in the public interest or to carry out some official function. (e.g. **You're a private garbage collection company**.)
- You have a legitimate interest to process someone's personal data. This is the most flexible lawful basis, though the "fundamental rights and freedoms of the data subject" always override your interests.

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) consent to handle their private data (first condition above) must follow these rules. (Text from `https://gdpr.eu/what-is-gdpr/`)

- Consent must be "freely given, specific, informed and unambiguous."

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) consent to handle their private data (first condition above) must follow these rules. (Text from `https://gdpr.eu/what-is-gdpr/`)

- Consent must be "freely given, specific, informed and unambiguous."
- Requests for consent must be "clearly distinguishable from the other matters" and presented in "clear and plain language."

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) consent to handle their private data (first condition above) must follow these rules. (Text from `https://gdpr.eu/what-is-gdpr/`)

- Consent must be "freely given, specific, informed and unambiguous."
- Requests for consent must be "clearly distinguishable from the other matters" and presented in "clear and plain language."
- Data subjects can withdraw previously given consent whenever they want, and you have to honor their decision. You can't simply change the legal basis of the processing to one of the other justifications.

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) consent to handle their private data (first condition above) must follow these rules. (Text from `https://gdpr.eu/what-is-gdpr/`)

- Consent must be "freely given, specific, informed and unambiguous."
- Requests for consent must be "clearly distinguishable from the other matters" and presented in "clear and plain language."
- Data subjects can withdraw previously given consent whenever they want, and you have to honor their decision. You can't simply change the legal basis of the processing to one of the other justifications.
- Children under 13 can only give consent with permission from their parent.

General Data Protection Regulation, GDPR, Cont'd

A data subject's (user's) consent to handle their private data (first condition above) must follow these rules. (Text from `https://gdpr.eu/what-is-gdpr/`)

- Consent must be "freely given, specific, informed and unambiguous."
- Requests for consent must be "clearly distinguishable from the other matters" and presented in "clear and plain language."
- Data subjects can withdraw previously given consent whenever they want, and you have to honor their decision. You can't simply change the legal basis of the processing to one of the other justifications.
- Children under 13 can only give consent with permission from their parent.
- You need to keep documentary evidence of consent.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.
- It is not mandatory in any way to obey the users preferences.

Do Not Track Specification

- Do Not Track, DNT, is a W3C specification enabling the user to express preferences regarding tracking.
- Defines a HTTP header, and how to handle it on the server.
- It is not mandatory in any way to obey the users preferences.
- Must be implemented by server developer.

Question 1

Section

- Client-Side Instead of Server-Side Rendering
- A REST API
- The framework Express.js
- Cookies
- HTTP Sessions
- Authentication
- Scopes
- Other Npm Packages in the Chat Application
- Architecture

Sessions

- A session is the time span during which a particular client interacts with a particular server.

Sessions

- A session is the time span during which a particular client interacts with a particular server.
- The **`express-session`** npm package is used for session handling with express.

Sessions

- A session is the time span during which a particular client interacts with a particular server.
- The **`express-session`** npm package is used for session handling with express.
  - Creates and maintains a session tracking id (Unique ID, UID), for each visitor, and sets a cookie containing that id.

Sessions

- A session is the time span during which a particular client interacts with a particular server.
- The **`express-session`** npm package is used for session handling with express.
  - Creates and maintains a session tracking id (Unique ID, UID), for each visitor, and sets a cookie containing that id.
  - Uses the same session id to identify a user's session data on the server.

No Sessions in a REST API

- As explained in the sections on a REST API and on cookies, we shall not use sessions because REST requires the app to be stateless.

Section

Authentication

- To Authenticate means to prove the identity.

Authentication

- To Authenticate means to prove the identity.
- In this course, it is done by providing username and password.

Authentication

- To Authenticate means to prove the identity.
- In this course, it is done by providing username and password.
- Other methods are for example a digital certificate or two-factor authentication.

Authentication

- To Authenticate means to prove the identity.
- In this course, it is done by providing username and password.
- Other methods are for example a digital certificate or two-factor authentication.
- Once a user is authenticated, it is necessary to recognize requests from that particular user. Otherwise, the user would have to authenticate at each request.

Authentication

- To Authenticate means to prove the identity.
- In this course, it is done by providing username and password.
- Other methods are for example a digital certificate or two-factor authentication.
- Once a user is authenticated, it is necessary to recognize requests from that particular user. Otherwise, the user would have to authenticate at each request.
- Authenticating at each request brings a security risk, because then the password must somehow be managed by the client.

JSON Web Tokens, JWT

- JSON Web Tokens, JWT, is a standard maintained by the Internet Engineering Task Force (IETF)

JSON Web Tokens, JWT

- JSON Web Tokens, JWT, is a standard maintained by the Internet Engineering Task Force (IETF)
- A JWT is a signed set of key-value pairs.

JSON Web Tokens, JWT

- JSON Web Tokens, JWT, is a standard maintained by the Internet Engineering Task Force (IETF)
- A JWT is a signed set of key-value pairs.
- JWT guarantees integrity, the content of a JWT can not be altered without revealing that it is altered.

JSON Web Tokens, JWT

- JSON Web Tokens, JWT, is a standard maintained by the Internet Engineering Task Force (IETF)
- A JWT is a signed set of key-value pairs.
- JWT guarantees integrity, the content of a JWT can not be altered without revealing that it is altered.
- It is common solve the problem of recognizing an authenticated user by placing a JWT in a cookie.

The Anatomy of a JWT

- A JWT consists of three parts, header, payload and signature.

The Anatomy of a JWT

- A JWT consists of three parts, header, payload and signature.
- The header defines the type (which is JWT) and the algorithm used for signing.

The Anatomy of a JWT

- A JWT consists of three parts, header, payload and signature.
- The header defines the type (which is JWT) and the algorithm used for signing.
- The payload is the set of key-value pairs containing the JWT's data.

The Anatomy of a JWT

- A JWT consists of three parts, header, payload and signature.
- The header defines the type (which is JWT) and the algorithm used for signing.
- The payload is the set of key-value pairs containing the JWT's data.
- The signature is computed by a signing algorithm, and is based on the header, the payload, and a secret known only to the issuer and to those verifying the JWT.

The Anatomy of a JWT

- A JWT consists of three parts, header, payload and signature.
- The header defines the type (which is JWT) and the algorithm used for signing.
- The payload is the set of key-value pairs containing the JWT's data.
- The signature is computed by a signing algorithm, and is based on the header, the payload, and a secret known only to the issuer and to those verifying the JWT.
- To verify that the content of a JWT has not been altered, the signature is re-calculated and compared to the signature in the JWT.

Registered Claims

- A key-value pair in the JWT payload is called a claim.

Registered Claims

- A key-value pair in the JWT payload is called a claim.
- There are some registered claims, with reserved names, that are understood by all JWT implementations.

Registered Claims

- A key-value pair in the JWT payload is called a claim.
- There are some registered claims, with reserved names, that are understood by all JWT implementations.
- Using the registered claims is optional. The sample chat application uses two, **`iat`** and **`exp`**.

Registered Claims

- A key-value pair in the JWT payload is called a claim.
- There are some registered claims, with reserved names, that are understood by all JWT implementations.
- Using the registered claims is optional. The sample chat application uses two, **`iat`** and **`exp`**.
  - iat means issued at and tells when the JWT was created.

Registered Claims

- A key-value pair in the JWT payload is called a claim.
- There are some registered claims, with reserved names, that are understood by all JWT implementations.
- Using the registered claims is optional. The sample chat application uses two, **`iat`** and **`exp`**.
  - iat means issued at and tells when the JWT was created.
  - exp means expiration time and defines a time after which the JWT must not be accepted.

What to Include In the JWT

- A JWT shall be self-contained, meaning it shall contain all required information about the user whose identity it proves.

What to Include In the JWT

- A JWT shall be self-contained, meaning it shall contain all required information about the user whose identity it proves.
- In the sample chat application, this is the user's id and username.

To Create a JWT

- The sample application uses the npm package **`jsonwebtoken`** to create and verify JWTs.

To Create a JWT

- The sample application uses the npm package **`jsonwebtoken`** to create and verify JWTs.
- A JWT is created with the **`sign`** method, as below.

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

const jwtToken = jwt.sign(
  {id: user.id, username: user.username},
  'this is the secret used to sign the JWT',
  {
    expiresIn: '30 minutes',
  }
);
```

To Read a JWT

- A JWT is read with the **`verify`** method, as below.

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

const cookieContent = req.cookies.chatAuth;
const JWTPayload = jwt.verify(
  cookieContent,
  'this is the secret used to sign the JWT'
);
```

Is the User Really Logged In?

- As was seen above, the expiry time is short, just 30 minutes. That means the user is logged out every 30 minutes.

Is the User Really Logged In?

- As was seen above, the expiry time is short, just 30 minutes. That means the user is logged out every 30 minutes.
- The reason is that the JWT payload might become invalid, therefore a new JWT is created each 30 minutes.

Is the User Really Logged In?

- As was seen above, the expiry time is short, just 30 minutes. That means the user is logged out every 30 minutes.
- The reason is that the JWT payload might become invalid, therefore a new JWT is created each 30 minutes.
- To releave the user of having to log in again and again, the user’s real login expiry time is stored in the database. If the JWT expires, but the logout period in the database has not expired, a new JWT is created without forcing the user to re-login.

Is the User Really Logged In?

- As was seen above, the expiry time is short, just 30 minutes. That means the user is logged out every 30 minutes.
- The reason is that the JWT payload might become invalid, therefore a new JWT is created each 30 minutes.
- To releave the user of having to log in again and again, the user's real login expiry time is stored in the database. If the JWT expires, but the logout period in the database has not expired, a new JWT is created without forcing the user to re-login.
- This also makes it possible to log out the user programmatically, by setting the user's database status to **`not logged in`**.

But Isn't That an HTTP Session?

- It can easily be argued, as was done previously, that since we now anyway need a database call to verify the user's status, we have re-created some kind of HTTP session.

But Isn't That an HTTP Session?

- It can easily be argued, as was done previously, that since we now anyway need a database call to verify the user's status, we have re-created some kind of HTTP session.
- "Sure, but now we can ...", see the discussion on REST authentication above.

Section

Scopes in a Web Application

- The scope of an object in a web application can be a tricky issue, since the application executes an HTTP request, send an HTTP response, and then stops executing.

Scopes in a Web Application

- The scope of an object in a web application can be a tricky issue, since the application executes an HTTP request, send an HTTP response, and then stops executing.
- When it starts from the beginning with the next request, what remains of objects that were created during the previous request handling?

Scopes in a Web Application

- The scope of an object in a web application can be a tricky issue, since the application executes an HTTP request, send an HTTP response, and then stops executing.
- When it starts from the beginning with the next request, what remains of objects that were created during the previous request handling?
- An advantage of a REST application like the sample application, is that it has just two scopes, that are easy to understand.

Application Scope

- Everything created outside a request handling function has application scope.

Application Scope

- Everything created outside a request handling function has application scope.
  - Such objects live during the entire lifetime of the application.

Application Scope

- Everything created outside a request handling function has application scope.
  - Such objects live during the entire lifetime of the application.
  - They are shared by all users and all requests.

Request Scope

- Everything created in a request handling function has request scope.

Request Scope

- Everything created in a request handling function has request scope.
  - Such objects live only during a single HTTP request.

Request Scope

- Everything created in a request handling function has request scope.
  - Such objects live only during a single HTTP request.
  - They are not shared at all.

Other Scopes, Not Used Here

- Some scopes that might appear in a web application with more complicated architecture are

Other Scopes, Not Used Here

- Some scopes that might appear in a web application with more complicated architecture are
  - Session scope objects live during one HTTP session, and are shared by all requests from all windows of the same browser on the same computer.

Other Scopes, Not Used Here

- Some scopes that might appear in a web application with more complicated architecture are
  - Session scope objects live during one HTTP session, and are shared by all requests from all windows of the same browser on the same computer.
  - Tab scope objects live during the existence of one browser tab in one HTTP session, and are shared by all request from that tab.

Section

- Client-Side Instead of Server-Side Rendering
- A REST API
- The framework Express.js
- Cookies
- HTTP Sessions
- Authentication
- Scopes
- Other Npm Packages in the Chat Application
- Architecture

Error Handling with VError

`verror` contains quite a lot of functionality for error handling, but the sample application uses only the following.

Error Handling with VError

`verror` contains quite a lot of functionality for error handling, but the sample application uses only the following.

- Chains of exceptions, using a **`cause`** property the same way the **`rootCause`** property is used in Java.

Error Handling with VError

`verror` contains quite a lot of functionality for error handling, but the sample application uses only the following.

- Chains of exceptions, using a **`cause`** property the same way the **`rootCause`** property is used in Java.
- Different levels of information, by adding an **`info`** property which may contain an object with additional information about the exception.

No Configuration in Source Code

- All applications have configurations that vary depending on where and when the application is running.

No Configuration in Source Code

- All applications have configurations that vary depending on where and when the application is running.
  - Typical examples are a password of a database to which the app must connect, or the port number on which the app shall listen.

No Configuration in Source Code

- All applications have configurations that vary depending on where and when the application is running.
  - Typical examples are a password of a database to which the app must connect, or the port number on which the app shall listen.
- It is considered best practice to store such configuration settings in environment variables, and not in the source code of the app.

No Configuration in Source Code

- All applications have configurations that vary depending on where and when the application is running.
    - Typical examples are a password of a database to which the app must connect, or the port number on which the app shall listen.
- It is considered best practice to store such configuration settings in environment variables, and not in the source code of the app.
    - An environment variable is a named value, set in the operating system, that can be read by a processes on the computer.

No Configuration in Source Code

- All applications have configurations that vary depending on where and when the application is running.
  - Typical examples are a password of a database to which the app must connect, or the port number on which the app shall listen.
- It is considered best practice to store such configuration settings in environment variables, and not in the source code of the app.
  - An environment variable is a named value, set in the operating system, that can be read by a processes on the computer.
  - All operating systems handle environment variables.

No Configuration in Source Code, Cont'd

There are several reasons for this best practice.

No Configuration in Source Code, Cont'd

There are several reasons for this best practice.

- Most important, it helps avoid leakage of secrets. If a password is present in source code, there is a big risk it is revealed when the source code is made public.

No Configuration in Source Code, Cont'd

There are several reasons for this best practice.

- Most important, it helps avoid leakage of secrets. If a password is present in source code, there is a big risk it is revealed when the source code is made public.
- It is independent of languages and frameworks.

No Configuration in Source Code, Cont'd

There are several reasons for this best practice.

- Most important, it helps avoid leakage of secrets. If a password is present in source code, there is a big risk it is revealed when the source code is made public.
- It is independent of languages and frameworks.
- Configuration is clearly separated from the app itself.

But How to Set Environment Variables?

- While there are big advantages of following this best practice, it also brings a problem: How do we set env vars?

But How to Set Environment Variables?

- While there are big advantages of following this best practice, it also brings a problem: How do we set env vars?
- The appropriate settings must be stored somewhere, or they will very soon be forgotten.

But How to Set Environment Variables?

- While there are big advantages of following this best practice, it also brings a problem: How do we set env vars?
- The appropriate settings must be stored somewhere, or they will very soon be forgotten.
- The env var settings should be managed by some product completely independent of the app.

Configuration With Dotenv-Safe

- The sample application uses a compromise, in order to not force yet more products to be installed.

Configuration With Dotenv-Safe

- The sample application uses a compromise, in order to not force yet more products to be installed.
- Configuration is managed by the npm package **`dotenv-safe`**.

Configuration With Dotenv-Safe

- The sample application uses a compromise, in order to not force yet more products to be installed.
- Configuration is managed by the npm package **`dotenv-safe`**.
- Place all configuration in a file called **`.env`**, and it is loaded into the environment by **`dotenv-safe`**. All that is needed in the source code is to include the line

```
require('dotenv-safe').config();
```

Configuration With Dotenv-Safe

- The sample application uses a compromise, in order to not force yet more products to be installed.
- Configuration is managed by the npm package **`dotenv-safe`**.
- Place all configuration in a file called **`.env`**, and it is loaded into the environment by **`dotenv-safe`**. All that is needed in the source code is to include the line

```
require('dotenv-safe').config();
```

- This way there is no config at all in the source code, but unfortunately it is the app itself that creates environment variables.

Configuration With Dotenv-Safe, Cont'd

- Remember that the **`.env`** file must be kept strictly secret. It must for example be included in the **`.gitignore`** file, in order to not be checked in to any git repository.

Configuration With Dotenv-Safe, Cont'd

- Remember that the `.env` file must be kept strictly secret. It must for example be included in the `.gitignore` file, in order to not be checked in to any git repository.
- Copy `.env` to a file called `.env.example`, but do not include the value of any setting in that file.

Configuration With Dotenv-Safe, Cont'd

- Remember that the **`.env`** file must be kept strictly secret. It must for example be included in the **`.gitignore`** file, in order to not be checked in to any git repository.
- Copy **`.env`** to a file called **`.env.example`**, but do not include the value of any setting in that file.
  - **`.env.example`** now serves as documentation, illustrating which variables must be set.

Configuration With Dotenv-Safe, Cont'd

- Remember that the `.env` file must be kept strictly secret. It must for example be included in the `.gitignore` file, in order to not be checked in to any git repository.
- Copy `.env` to a file called `.env.example`, but do not include the value of any setting in that file.
    - `.env.example` now serves as documentation, illustrating which variables must be set.
    - Also, `dotenv-safe` will check that all variables in `.env.example` are set in `.env`, and throw an exception if some setting is forgotten.

Reading an Env Var

- Env vars are read in the application by calling **`process.env`** as below.

```
process.env.ENV_VAR_NAME
```

Reading an Env Var

- Env vars are read in the application by calling `process.env` as below.

```
process.env.ENV_VAR_NAME
```

- **`ENV_VAR_NAME`** shall be the name of the env var.

Serving Static Files

- The server must be able to deliver files in response to an HTTP GET request.

Serving Static Files

- The server must be able to deliver files in response to an HTTP GET request.
  - That is how the browser will load for example images and javascript files that shall be executed on the client.

Serving Static Files

- The server must be able to deliver files in response to an HTTP GET request.
  - That is how the browser will load for example images and javascript files that shall be executed on the client.
- Delivering static files is part of express, no extra package is needed.

Serving Static Files

- The server must be able to deliver files in response to an HTTP GET request.
  - That is how the browser will load for example images and javascript files that shall be executed on the client.
- Delivering static files is part of express, no extra package is needed.
  - To make express deliver files in the directory **`public`**, include the statement below.

```
app.use(express.static('public'));
```

Serving Static Files

- The server must be able to deliver files in response to an HTTP GET request.
  - That is how the browser will load for example images and javascript files that shall be executed on the client.
- Delivering static files is part of express, no extra package is needed.
  - To make express deliver files in the directory **`public`**, include the statement below.

    ```
    app.use(express.static('public'));
    ```

  - Note that **`public`** is now the root directory for the server's static files. An HTTP get request for **`/abc/def.png`** will make the server look for the file **`public/abc/def.png`**

Section

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.
- Low coupling, Objects and subsystems do not depend on each other more than necessary.

Remember Object Oriented Design?

- We want the code to be easy to modify and easy to understand. To achieve this we need (among other things):
- High Cohesion, Each class, method, etc has well-defined knowledge and a well-defined task.
- Low coupling, Objects and subsystems do not depend on each other more than necessary.
- Encapsulation, Objects and subsystems do not reveal their internals.

The File Structure

The server's root directory contains three things.

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.
- The directory **`src`**, which contains the JavaScript source code.

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.
- The directory **`src`**, which contains the JavaScript source code.
- All configuration files

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.
- The directory **`src`**, which contains the JavaScript source code.
- All configuration files
    - **`.env`** and **`.env.example`** for settings used in our own code

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.
- The directory **`src`**, which contains the JavaScript source code.
- All configuration files
  - **`.env`** and **`.env.example`** for settings used in our own code
  - **`package.json`** and **`package-lock.json`** for settings needed by npm.

The File Structure

The server's root directory contains three things.

- The directory **`public`**, which contains all static files.
- The directory **`src`**, which contains the JavaScript source code.
- All configuration files
  - **`.env`** and **`.env.example`** for settings used in our own code
  - **`package.json`** and **`package-lock.json`** for settings needed by npm.
  - It might be a better structure to place config files in a separate directory, **`config`**, but it is not obvious how to make npm look for **`package.json`** in another directory.

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).
- There are the following layers.

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).
- There are the following layers.
  - Api corresponds to an MVC view. It is not really a view, since no user interaction is handled by a REST API, but it handles requests and responses from the client.

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).
- There are the following layers.
  - Api corresponds to an MVC view. It is not really a view, since no user interaction is handled by a REST API, but it handles requests and responses from the client.
  - Controller is a typical MVC controller.

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).
- There are the following layers.
  - Api corresponds to an MVC view. It is not really a view, since no user interaction is handled by a REST API, but it handles requests and responses from the client.
  - Controller is a typical MVC controller.
  - The model consists only of DTOs, since this simple application has no behavior. It only moves data between client and datastore.

Layers

- The layer structure exists only in terms of directories, since there is no concept of grouping (like packages in Java).
- There are the following layers.
    - Api corresponds to an MVC view. It is not really a view, since no user interaction is handled by a REST API, but it handles requests and responses from the client.
    - Controller is a typical MVC controller.
    - The model consists only of DTOs, since this simple application has no behavior. It only moves data between client and datastore.
        - It could be argued that login is behavior that should be placed in a class **`User`** in the model.

Layers, Cont’d

- There are the following layers, continued.
  - Integration is responsible only for calling the datastore, since there are no other external systems.

Layers, Cont'd

- There are the following layers, continued.
  - Integration is responsible only for calling the datastore, since there are no other external systems.
  - Data contains a very simple datastore, consisting only of two arrays. It will be replaced by a real database later in the course.

More “packages”

- Besides the layers mentioned above, there are also the following source code directories.

More “packages”

- Besides the layers mentioned above, there are also the following source code directories.
    - The api layer contains the auth directory, which contains code related to authentication.

More “packages”

- Besides the layers mentioned above, there are also the following source code directories.
    - The api layer contains the auth directory, which contains code related to authentication.
        - The only class in **`auth`** is **`Authorization`**, which forms a procedural api (no state, only static methods).

More “packages”

- Besides the layers mentioned above, there are also the following source code directories.
  - The api layer contains the auth directory, which contains code related to authentication.
    - The only class in **`auth`** is **`Authorization`**, which forms a procedural api (no state, only static methods).
  - util, containing general utility classes, which in this case means only **`Logger`**.

More “packages”

- Besides the layers mentioned above, there are also the following source code directories.
  - The api layer contains the auth directory, which contains code related to authentication.
    - The only class in **`auth`** is **`Authorization`**, which forms a procedural api (no state, only static methods).
  - util, containing general utility classes, which in this case means only **`Logger`**.
  - It is convention to place the file **`server`**, which starts the entire application, directly in the source code root, which is **`src`** in this application.

The DAO Pattern

- The responsibility of a Database Access Object, DAO is to handle database calls. All code related to database calls shall be in a DAO.

The DAO Pattern

- The responsibility of a Database Access Object, DAO is to handle database calls. All code related to database calls shall be in a DAO.
- A DAO shall be located in the integration layer.

The DAO Pattern

- The responsibility of a Database Access Object, DAO is to handle database calls. All code related to database calls shall be in a DAO.
- A DAO shall be located in the integration layer.
- It shall not call any other layer, except the **`data`** layer, but can of course use DTOs.

The DAO Pattern

- The responsibility of a Database Access Object, DAO is to handle database calls. All code related to database calls shall be in a DAO.
- A DAO shall be located in the integration layer.
- It shall not call any other layer, except the `data` layer, but can of course use DTOs.
- It shall not contain business logic.

The DAO Pattern

- The responsibility of a Database Access Object, DAO is to handle database calls. All code related to database calls shall be in a DAO.
- A DAO shall be located in the integration layer.
- It shall not call any other layer, except the `data` layer, but can of course use DTOs.
- It shall not contain business logic.
- Its public interface meets the needs of client objects in controller and model, and is independent of the database.

DAO Example

```
class ChatDAO {
  findUserByUsername(username) {
    // Database-related code.
  }

  findUserById(id) {
    // Database-related code.
  }

  updateUser(user) {
    // Database-related code.
  }

  createMsg(msg, author) {
    // Database-related code.
  }

  findAllNotDeletedMsgs() {
    // Database-related code.
  }

  deleteMsg(id) {
    // Database-related code.
  }
}
```

Benefits of the DAO Pattern

- DAO provides high cohesion since all database access code is collected in the DAO, instead of being mixed with other code.

Benefits of the DAO Pattern

- DAO provides high cohesion since all database access code is collected in the DAO, instead of being mixed with other code.
- DAO provides encapsulation since no object outside the DAO will know the design of the database or database calls.

Benefits of the DAO Pattern

- DAO provides high cohesion since all database access code is collected in the DAO, instead of being mixed with other code.
- DAO provides encapsulation since no object outside the DAO will know the design of the database or database calls.
- It decouples other layers from the database, since the public interface of the DAO is adapted to the need of other layers, not to the database.

Classes or no Classes?

- Since JavaScript is not fully Object-Oriented, there is the question to what extent classes shall be used.

Classes or no Classes?

- Since JavaScript is not fully Object-Oriented, there is the question to what extent classes shall be used.
- For clarity, try to be consistent. Use either classes, function objects, or no objects at all. Try not to mix.

Classes or no Classes?

- Since JavaScript is not fully Object-Oriented, there is the question to what extent classes shall be used.
- For clarity, try to be consistent. Use either classes, function objects, or no objects at all. Try not to mix.
- The sample application is an effort to use classes as much as possible, at almost any cost.

Classes or no Classes?

- Since JavaScript is not fully Object-Oriented, there is the question to what extent classes shall be used.
- For clarity, try to be consistent. Use either classes, function objects, or no objects at all. Try not to mix.
- The sample application is an effort to use classes as much as possible, at almost any cost.
  - Increases cohesion and, only to some extent since class support is poor, reduces coupling and improves encapsulation.

Classes or no Classes?, Cont'd

- Create one file for each class, to improve readability and cohesion.

Classes or no Classes?, Cont'd

- Create one file for each class, to improve readability and cohesion.
- Where classes can not be used, it is still important that all code in the same file together performs only one task, which must have high cohesion.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.
- Some files require a bit more explanation. The first is **`server`**, whose task is to start the application.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.
- Some files require a bit more explanation. The first is **`server`**, whose task is to start the application.
  - Since a JavaScript program can not start executing in a method, the startup code in **`server`** is just a script executing from top to bottom.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.
- Some files require a bit more explanation. The first is **`server`**, whose task is to start the application.
  - Since a JavaScript program can not start executing in a method, the startup code in **`server`** is just a script executing from top to bottom.
  - Just like the **`main`** method in a Java program, it shall be as short as possible.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.
- Some files require a bit more explanation. The first is **`server`**, whose task is to start the application.
  - Since a JavaScript program can not start executing in a method, the startup code in **`server`** is just a script executing from top to bottom.
  - Just like the **`main`** method in a Java program, it shall be as short as possible.
  - It loads all required packages and all endpoints in the REST API.

The Startup Script

- All layers except **`api`** and the startup script, **`server`**, are quite straight-forward with only classes.
- Some files require a bit more explanation. The first is **`server`**, whose task is to start the application.
    - Since a JavaScript program can not start executing in a method, the startup code in **`server`** is just a script executing from top to bottom.
    - Just like the **`main`** method in a Java program, it shall be as short as possible.
    - It loads all required packages and all endpoints in the REST API.
    - When loading is done, it makes the server listen for requests.

Request Handlers

- It is not possible to route an HTTP request to a method in a class, but only to a function outside a class.

Request Handlers

- It is not possible to route an HTTP request to a method in a class, but only to a function outside a class.
- In order to still use the advantages of classes in the api layer, **`MsgApi`** and **`UserApi`** are classes with a method, **`registerHandlers`**, which defines the anonymous request handling functions.

Request Handlers

- It is not possible to route an HTTP request to a method in a class, but only to a function outside a class.
- In order to still use the advantages of classes in the api layer, **`MsgApi`** and **`UserApi`** are classes with a method, **`registerHandlers`**, which defines the anonymous request handling functions.
- Next question is, who calls **`registerHandlers`** in all api classes? It is not the task of **`server`** to know the internals of the **`api`** layer.

Request Handlers, Cont'd

- To encapsulate the request handling classes (**`MsgApi`** and **`UserApi`**), the file **`index`** is created in **`api`**.

Request Handlers, Cont'd

- To encapsulate the request handling classes (**`MsgApi`** and **`UserApi`**), the file **`index`** is created in **`api`**.
- Note that, with the statement below, **`server`** requires the **`api`** directory, not any class in that directory.

```
const reqHandlerLoader = require('./api');
```

Request Handlers, Cont’d

- To encapsulate the request handling classes (**`MsgApi`** and **`UserApi`**), the file **`index`** is created in **`api`**.
- Note that, with the statement below, **`server`** requires the **`api`** directory, not any class in that directory.

```
const reqHandlerLoader = require('./api');
```

- Requireing a directory means that the **`index.js`** file in that directory is executed.

Request Handlers, Cont'd

- **`index`** defines a class, **`RequestHandlerLoader`**, whose task is to call **`registerHandlers`** in all classes defining request handling functions.

Request Handlers, Cont'd

- **index** defines a class, **RequestHandlerLoader**, whose task is to call **registerHandlers** in all classes defining request handling functions.
- In the sample app, **MsgApi** and **UserApi** are passed explicitly to **RequestHandlerLoader**, with the statements below.

```
const loader = new RequestHandlerLoader();
loader.addRequestHandler(new UserApi());
loader.addRequestHandler(new MsgApi());
```

Request Handlers, Cont'd

- **`index`** defines a class, **`RequestHandlerLoader`**, whose task is to call **`registerHandlers`** in all classes defining request handling functions.
- In the sample app, **`MsgApi`** and **`UserApi`** are passed explicitly to **`RequestHandlerLoader`**, with the statements below.

```
const loader = new RequestHandlerLoader();
loader.addRequestHandler(new UserApi());
loader.addRequestHandler(new MsgApi());
```

- The code would be more flexible, but slightly more complicated, if **`RequestHandlerLoader`** instead listed all classes in the **`api`** directory, and assumed that for example all classes with names ending in **`Api`** contained request handling functions.