Web Application Development (LTAT.05.004)

NODE.JS -I

MOHAMAD GHARIB
UNIVERSITY OF TARTU
Three-tier Architecture - Recap

Presentation tier — Application tier — Data tier
Three-tier Architecture in Web App - Recap
How the front-end works
How the front-end works

scheme://domain:port/path?query#fragment_id
http://www.example.com/index.html

We tell the browser to get something

Client
How the front-end works

scheme://domain:port/path?query#fragment_id
http://www.example.com/index.html

We tell the browser to get something

Client

request

Server
How the front-end works

scheme://domain:port/path?query#fragment_id

http://www.example.com/index.html

We tell the browser to get something

Client

request

Server

/index
/posts
/users
How the front-end works

scheme://domain:port/path?query#fragment_id
http://www.example.com/index.html

Client

We tell the browser to get something

request

Server

response

/index
/posts
/users

mongo DB

SQL
How the front-end works

Client

Browser parses the HTML file ($url) and fetches linked resources

Server

Request:

- /index
- /posts
- /users

Response:

mongo DB

SQL
How the front-end works
How the back-end works
How the back-end works

Listen on a **port** for a **Request**

Client

Server

```
/index
/posts
/users
```

mongo DB

SQL
How the back-end works

GET HTML Request
GET /echo HTTP/1.1
Host: ut.ee
Accept: text/html

Listen on a port for a Request

Client

Server

GET /index HTTP/1.1
Host: ut.ee
Accept: text/html

GET /posts HTTP/1.1
Host: ut.ee
Accept: text/html

GET /users HTTP/1.1
Host: ut.ee
Accept: text/html
How the back-end works

GET JSON Request
GET /echo/get/json HTTP/1.1
Host: ut.ee
Accept: application/json

Listen on a port for a Request

Client

Server
How the back-end works

Listen on a port for a Request
Respond by returning a resource

Client

Server

/m/index
/m/posts
/m/Users

mongo DB

SQL
How the back-end works

HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<head> <title>200 ok</title></head>
<body><h1>Hello, World!</h1></body>
</html>
How the back-end works

HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{  
  "data": "articles"
}

Client

Server

on a port for a Request
by returning a resource

mongo DB

/sql

/response

/index
/posts
/users
How the back-end works

HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1

<html>
  <head> 404 Not Found</title></head>
  <body>
    <h1>Not Found</h1>
  </body>
</html>
Localhost

127.0.0.1
Localhost

Client

http://localhost:8080/

We tell the browser to get something
Localhost

We tell the browser to get something

http://localhost:8080/

Client

/localhost

http://localhost:8080/

response

Server

127.0.0.1
/object

We tell the browser to get something

/localhost

http://localhost:8080/

response

Server

127.0.0.1
/object
Listen on a **port** for a **Request** and **Respond** by returning a resource.
Node.Js
Why Node.Js?

It's very fast.
It can professionally handle great number of concurrent requests.
It's written in JavaScript (the same code we used for the front-end).
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.
What is Node.js?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment.

Node.js runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js uses the single-threaded, non-blocking, and event-driven execution model.

<table>
<thead>
<tr>
<th>Threads</th>
<th>Asynchronous Event-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi threads can lock application.</td>
<td>Only one thread, which repeatedly fetches events.</td>
</tr>
<tr>
<td>Multithreaded server might block the request that involve multiple events.</td>
<td>Saves state and then goes on to process the next event</td>
</tr>
</tbody>
</table>
Node.Js, let's start ...
Using **Node.js**，we are able to use JavaScript code **outside** the browser, and we can use **consts**, **variables**, **arrays**, **objects**, **methods**, …

**Node.js** programs can be created using JavaScript files that have a filename extension of `.js`，and they can be executed with the **Node.js** executable.

To execute a **Node.js** program, run the **Node.js** executable for the given environment and pass the name of the file (with or without the `.js` extension) as the first argument.
In Node.js, we have the “global” object, just like the “Windows” object in JavaScript, which we can use its methods (e.g., setTimeout(), etc.)

We can use the “global” object methods (e.g., clearInterval(), setInterval(), setTimeout(), etc.)

You can access the methods of the global object using the dot notation (global.setInterval(..)), or you can just call its methods.
Node.js - Modules

Modules, in Node.js, are simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Modules are used to organise your code by splitting it into different files (modules), which we import when required.

We can use a module in another file by using the term require followed by the name of the module.
Node.js comes with several core modules (e.g., fs, http), which provides very useful methods (fs.readFile, http.createServer(..)).
The Node.js file system (fs) module allows dealing with the file system on the machine.

The fs module is responsible for all asynchronous or synchronous file I/O operations.

To include the file system (fs) module, we need to use the require() method.
The `fs.readFileSync()` method can be used to **synchronously read** a file.

The `fs.readFile()` method can be used to **asynchronously read** a file, and executes a **callback** function when the read operation completes.

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

//read a file Synchronously
// fs.readFileSync(filename, options)
const dataSync = fs.readFileSync('./file.txt', 'utf8');
console.log("File content:", dataSync);

//read a file Asynchronously
//fs.readFile(filename, options, callbackFun)

fs.readFile('./file.txt', 'utf8',
  function(err, dataAsync) {
    if (err) throw err;
    console.log("File content:", dataAsync);
  });
```
fs module – Writing a File

The `fs.writeFile()` method replaces the file content **if it exists**. If the file does **not exist**, a new file, containing the specified content, will be **created**.

```javascript
const fs = require('fs');
//write a file
//fs.writeFile(file, string, position, encoding, callback)
//file: a file descriptor the value returned by opening the file using the fs.open(), which creates an empty file.

fs.writeFile('./file2.txt', 'I have been modified', (err) => {
  if (err) return console.log(err);
  console.log("File content: modified");
});
```
The `fs.appendFile()` method appends the specified content at the end of the specified file.

```javascript
const fs = require('fs');
// update a file
// fs.appendFile(file, data[, options], callback )

fs.appendFile('file2.txt', '\n This text is added by appendFile().', function(err) {
  if (err) throw err;
  console.log('File content: updated!');
});
```
The **fs.unlink()** method can be used to **delete** the specified file.

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

// delete a file
// fs.unlink( file, callback )

fs.unlink('./file2.txt', (err) => {
  if (err) return console.log(err);
  console.log("File is deleted");
});
```
Creating a basic server
Creating a basic server

Listen on a port for a Request
Respond by returning a resource
Creating a basic server

- Create a server.js file.
- Include the http module.
- Create a server instance.
- Make the server listen to a port.

**Note:** if you want to change anything in the code, you need to stop the running server (Ctrl + c). Then, run it again to apply the changes.

**OR** you can use `nodemon`

```javascript
// http is a built-in module that allows Node.js to transfer data using HTTP.
const http = require('http');

const server = http.createServer((req, res) => {
  console.log('A request has arrived to the Server');
});

// The server.listen() method creates a listener on the specified port and path.
server.listen(3000, 'localhost', () => {
  console.log('I am listing for requests on port 3000');
});
```
Creating a basic server

- Create a server.js file.
- Include the http module.
- Create a server instance.
- Make the server listen to a port.

**Note:** if you want to change anything in the code, you need to stop the running server (Ctrl + c). Then, run it again to apply the changes.

**OR** you can use **nodemon**

```javascript
// http is a built-in module that allows Node.js to transfer data using HTTP.
const http = require('http');

const server = http.createServer((req, res) => {
  console.log('A request has arrived to the Server');
});

//The server.listen() method creates a listener on the specified port and path.
server.listen(3000, 'localhost', () => {
  console.log('I am listing for requests on port 3000');
});
```
Creating a basic server

- Create a server.js file.
- Include the http module.
- Create a server instance.
- Make the server listen to a port.

**Note:** if you want to change anything in the code, you need to stop the running server (Ctrl + c). Then, run it again to apply the changes.

**OR** you can use **nodemon**

```javascript
// http is a built-in module that allows Node.js to transfer data using HTTP.
const http = require('http');

const server = http.createServer((req, res) => {
  console.log('A request has arrived to the Server');
});

// The server.listen() method creates a listener on the specified port and path.
server.listen(3000, 'localhost', () => {
  console.log('I am listening for requests on port 3000');
});
```
The request object is passed as the first parameter on the request handler, and it has information about the user’s request.

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

const server = http.createServer((req, res) => {
    console.log(req);
    console.log(req.url);
    console.log(req.method);
});

server.listen(3000, 'localhost', () => {
    console.log('I am listing for requests on port 3000');
});
```
The request object is passed as the first parameter on the request handler, and it has information about the user’s request. 

Note: you need to send a request to the server (open http://localhost:3000/) to trigger this action.
The response object is passed as the second parameter on the request handler, and it can be used to send response to the client request.

Responding with a plain text

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

const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/plain');
  res.write('First response message!');
  res.end();
});

server.listen(3000, 'localhost', () => {
  console.log('I am listing for requests on port 3000');
});
```
The **response object** is passed as the **second parameter** on the request handler, and it can be used to send **response** to the client request.

### Responding with an HTML script

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

const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<h1>First HTML response message!</h1>');
  res.end();
});

server.listen(3000, 'localhost', () => {
  console.log('I am listing for requests on port 3000');
});
```
The Response Object

The response object is passed as the second parameter on the request handler, and it can be used to send response to the client request.

Responding with an HTML page

code snippet:
```javascript
const http = require('http');
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  fs.readFile('./views/index.html', (err, data) => {
    if (err) {
      console.log(err);
    } else {
      res.write(data);
      res.end();
    }
  });
});
server.listen(3000, 'localhost', () => {
  console.log('I am listing for requests on port 3000');
});
```
The Response Object

The **response object** is passed as the **second parameter** on the request handler, and it can be used to send **response** to the client request.

Responding with an HTML page
Creating basic routing
Creating basic routing

We know how to server one HTML page, but how we can server more than one?
Creating basic routing

We know how to server one HTML page, but how we can server more than one?

Almost in the exact same way as we serve one page, but using a route handler to serve the right page.
Creating basic routing

We know how to server one HTML page, but how we can server more than one?

Almost in the exact same way as we serve one page, but using a route handler to serve the right page.
Creating basic routing

We know how to server one HTML page, but how we can server more than one?

Almost in the exact same way as we serve one page, but using a route handler to serve the right page.
Creating basic routing

We know how to server one HTML page, but how we can server more than one?

Almost in the exact same way as we serve one page, but using a route handler to serve the right page.
Creating basic routing

We know how to server one HTML page, but how we can server more than one?

Almost in the exact same way as we serve one page, but using a route handler to serve the right page.
The Express Framework
The Express Framework

Express is a framework that enables us to professionally handle our routes, and also it makes the code much easier to read and modify.
The Express Framework

Express is a framework that enables us to professionally handle our routes, and also it makes the code much easier to read and modify.

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

// listen for requests on port 3000
app.listen(3000);

// app.get() is used to respond to Get requests, and it takes two arguments: 1- path, and 2- function that takes in request and response objects
app.get('/', (req, res) => {
    res.sendFile('./views/index.html', { root: __dirname});
});

......

app.use((req, res) => {
    res.status(404).sendFile('./views/404.html', { root: __dirname });
});
```

Note: there are .post, .put, .delete, and we will cover them later.
The Express Framework

Express is a framework that enables us to professionally handle our routes, and also it makes the code much easier to read and modify.

res.sendFile() is a method that can be used to send files as its name indicates. However, it takes the absolute not the relative path to the file. Therefore, you need to specify what is the root directory by using "__dirname"

Note: there are .post, .put, .delete, and we will cover them later

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

// listen for requests on port 3000
app.listen(3000);

//app.get() is used to respond to Get requests, and it takes two arguments: 1- path, and 2- function that takes in request and response objects
app.get('/', (req, res) => {
  res.sendFile('./views/index.html', { root: __dirname});
});

......

app.use((req, res) => {
  res.status(404).sendFile('./views/404.html ', { root: __dirname });
});
```
The Express Framework - Middleware

Client

Server

```
app.get('/', (req, res) => { .. }
app.get('/posts', (req, res) => { .. }
app.get('/contactus', (req, res) => { .. })
app.use((req, res) => {
    res.status(404).render('404');
});
```
The Express Framework - Middleware

Client

Server

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

app.get('/posts', (req, res) => {
  // ...
});

app.get('/contactus', (req, res) => {
  // ...
});

app.use((req, res) => {
  res.status(404).render('404');
});
```
The Express Framework - Middleware

Client

Server

```
app.get('/', (req, res) => { .. }
app.get('/posts', (req, res) => { .. }
app.get('/contact', (req, res) => { .. }
app.use((req, res) => {
    res.status(404).render('404');
});
```
The Express Framework - Middleware

Client

Server

- Request: `app.get('/', (req, res) => { ..})`
- Request: `app.get('/posts', (req, res) => { ..})`
- Request: `app.get('/contactus', (req, res) => { ..})`
- Request: `app.use((req, res) => { res.status(404).render('404'); });`
Middleware is a “functions” that execute (on the server) during the request-response cycle.

Middleware has access to both the request and the response objects object.

Middleware is executed during the window between when a server receives a request and when it sends a response.

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

// listen for requests on port 3000
app.listen(3000);

// new Middleware that will block the execution
app.use((req, res) => {
    console.log('a new request was made to the server');
});

app.get('/', (req, res) => {
    res.sendFile('./views/index.html', { root: __dirname });
});

......

app.use((req, res) => {
    res.status(404).sendFile('./views/404.html', { root: __dirname });
});
```
The Express Framework - Middleware

Client

Server

app.get('/', (req, res) => { .. }

app.use((req, res) => {
  console.log('a new request was made');
});

app.get('/posts', (req, res) => { .. }

app.get('/contactus', (req, res) => { .. }

app.use((req, res) => {
  res.status(404).render('404');
});
The Express Framework - Middleware

Client

Server

```javascript
app.get('/', (req, res) => { .. }
app.use((req, res) => {
  console.log('a new request was made');
});
app.get('/posts', (req, res) => { .. }
app.get('/contactus', (req, res) => { .. }
app.use((req, res) => {
  res.status(404).render('404');
});
```
The Express Framework - Middleware

Some custom made middlewares, check CORS (Cross-origin resource sharing)
https://blog.js scrambler.com/setting-up-5-useful-middlewares-for-an-express-api
Node Package Manager (npm)
Node.js Package Manager (npm)\(^1\) is a package manager for the JavaScript programming language. 

*npm* has its own command-line interface (CLI).

*npm* offers a registry that is an online database of public and paid-for private package.

*npm* provides a public package repository, a specification for building packages, and a command line tool for working with packages.

Why *npm* is very important: it allows adapting packages of code for your apps, or incorporating packages as they are.

---

\(^1\)[https://www.npmjs.com/]
Packages can be installed **locally** or **globally**.

Local packages are stored locally in a project, in the node_modules folder.

Global packages are stored **globally** on the **system**.

Local packages are available only within their specific project, and global packages are available system wide.
The first argument to the `npm` is the `command` to be executed.

Packages can be installed/uninstalled with the install/uninstall commands.

The `--global` flag installs/uninstalls the package `globally`, without it, packages are installed/uninstalled locally.

```
> npm install --global grunt-cli
> npm install grunt
> npm uninstall --global grunt-cli
> npm uninstall grunt
```
Node.js is a great environment for building back-end applications, which are **not I/O intensive**.

Working with Node.js involves many tools, frameworks for managing packages, debugging code, and packaging projects for deployment.

The Node Package Manager (NPM) is used to install very useful packages.
Node.Js - I

Try it yourself …
Setting up a Node.js project, and how we can creating basic server

There are information on the course wiki page that will guide you to set up a Node.js project, and installing nodemon and Express.

Clone the repo of the course material:

```
>> git clone -b lecture11-node1 https://github.com/M-Gharib/WAD2022.git
```

It contains the code we have covered today except the used Express, which will be covered in the lab.

The code mainly cover how we can create a basic server without the use of Express framework.
Install Nodemon\(^1\) is a tool that helps developing node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Express is a framework that enables us to professionally manage our routing requests, and also it makes the code much easier to read and modify.

Installation details are provided on the course Wiki.
Extra resources

Node.js Tutorial: https://www.w3schools.com/nodejs/

Introduction to Node.js: https://nodejs.dev/en/learn/

Node.js in Action - 1st Edition by Mike Cantelon, Marc Harter, TJ Holowaychuk, Nathan Rajlich
Thank You for your attention

Mohamad Gharib
mohamad.gharib@ut.ee