



COM3504/6504

Lab Class Week 1

Professor Fabio Ciravegna
Department of Computer Science
University of Sheffield
f.ciravegna@shef.ac.uk



Important!

- Very important:
 - if you do not finish the exercise today,
 - make sure to finish it over the coming week
 - from next week we will build on this
 - if you have not completed the exercises you will struggle
- Also
 - use today as a test of your Javascript and HTML knowledge

© Fabio Ciravegna, University of Sheffield

3



Plan for today

- We have two hours and there is a lot to go through
- Learning Objectives
 - Learn to use WebStorm (our development environment)
 - Create your first nodeJs server
 - Learn to get a file
 - Learn to get an EJS file with parameters
 - Learn to post a form
- You are expected to know HTML and Javascript as a starting point

© Fabio Ciravegna, University of Sheffield

© Fabio Ciravegna, University of Sheffield

2



Creating an Express Project in WebStorm

© Fabio Ciravegna, University of Sheffield



Using WebStorm

- In Windows Go to the Window menu and search

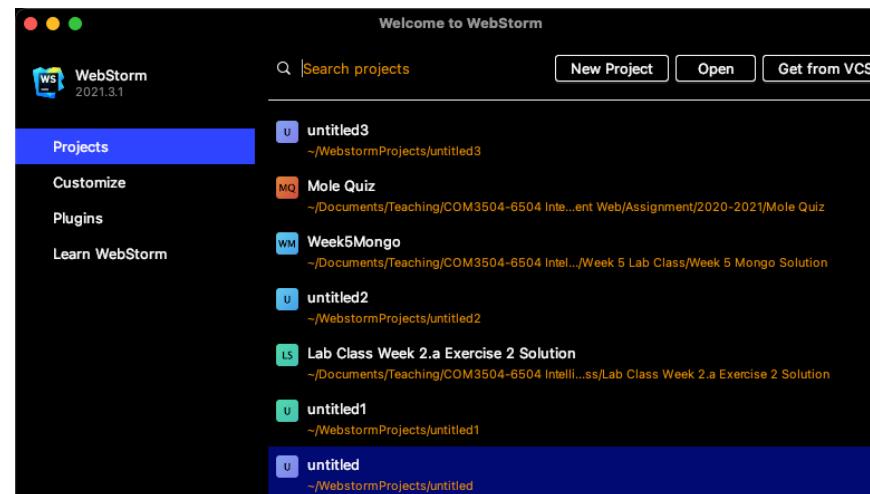
- WebStorm

- On a Mac go to Applications and select WebStorm

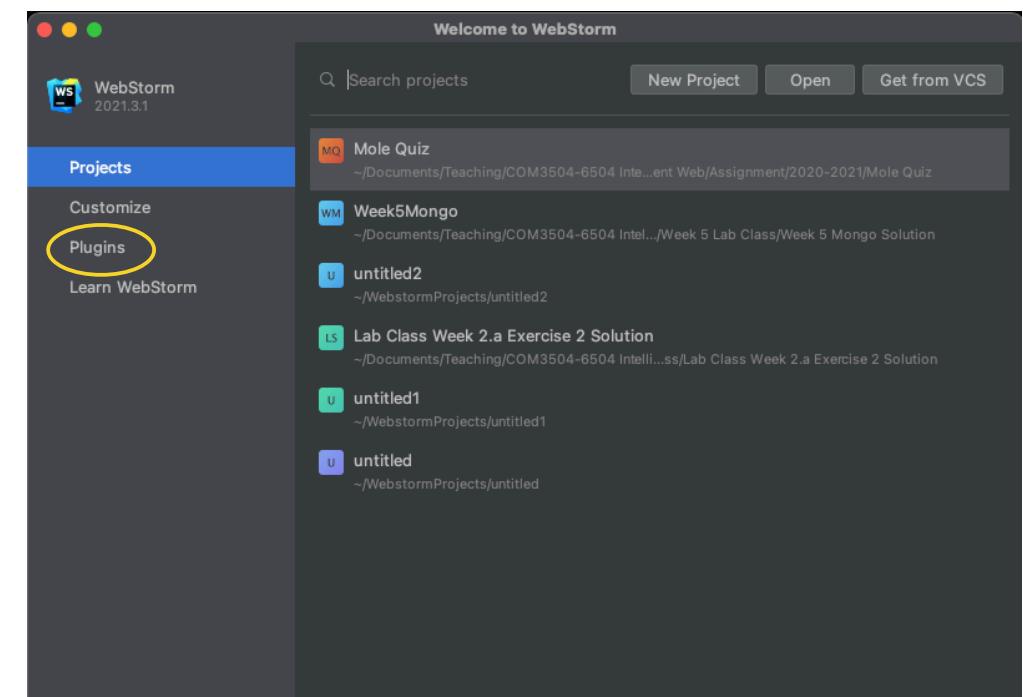
- This is what you will see:

- you should have node already installed

- to be sure: click on Plugins



Not on Lab computers: Install Node Plugin

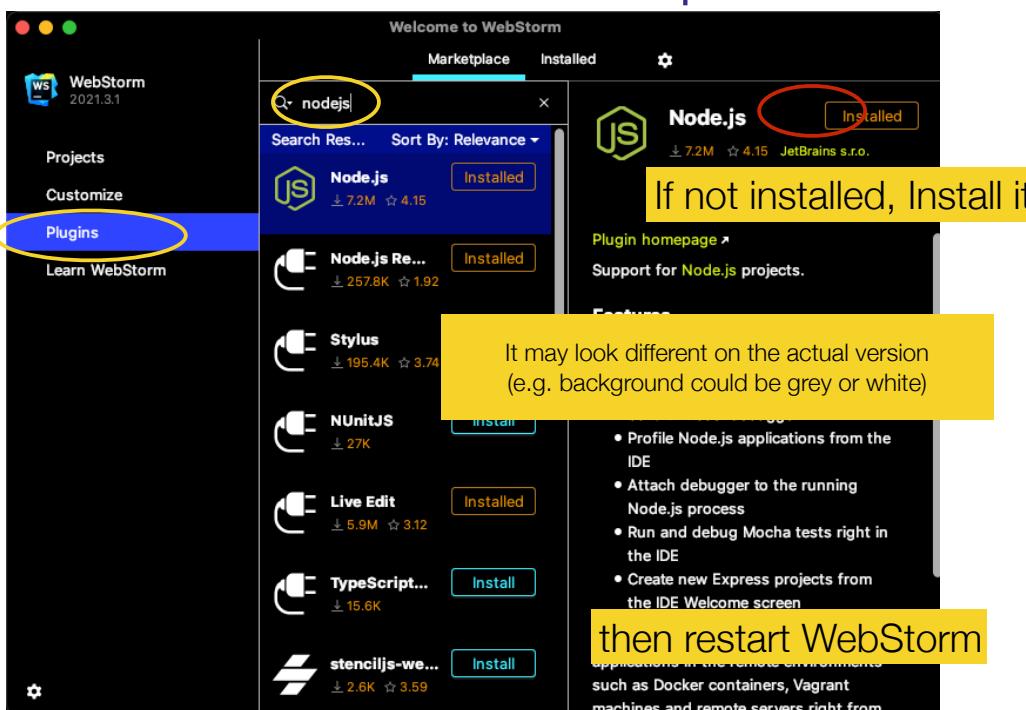


© Fabio Cravagna, University of Sheffield

6



Not needed on lab computers!!

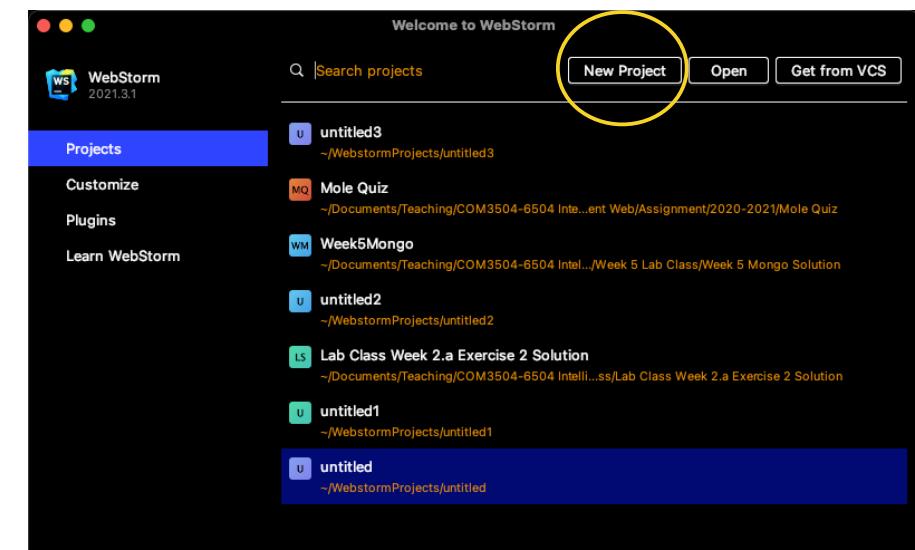


© Fabio Cravagna, University of Sheffield

7



After installing and restarting, create a new project



© Fabio Cravagna, University of Sheffield

8

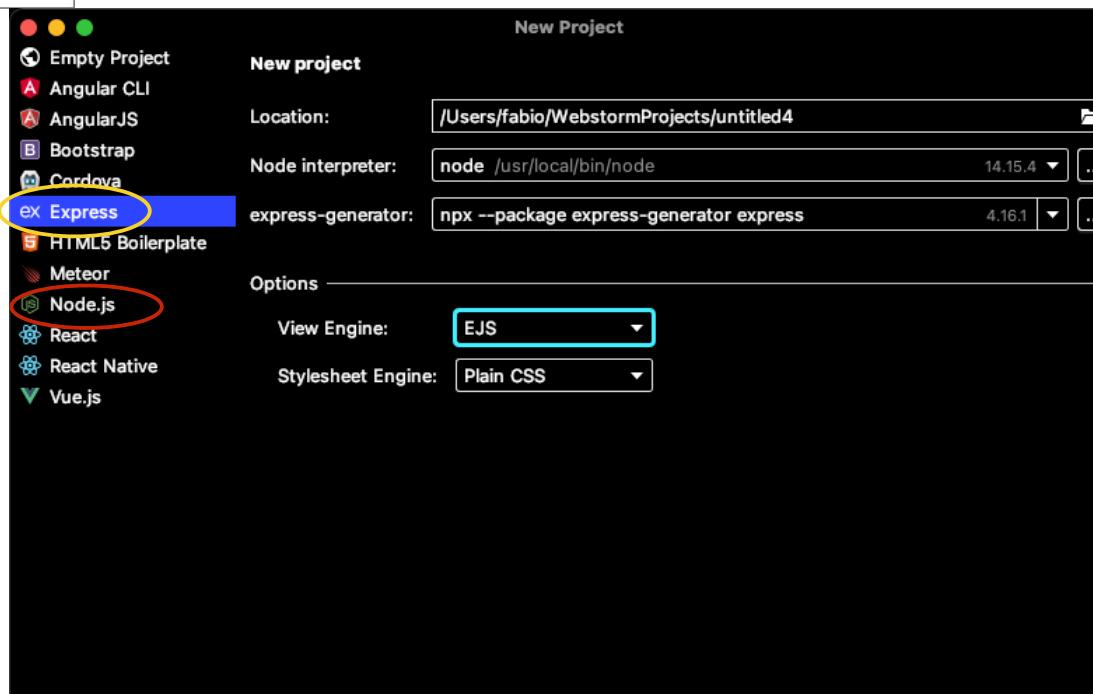


The
University
Of
Sheffield.

Select Express

Yes

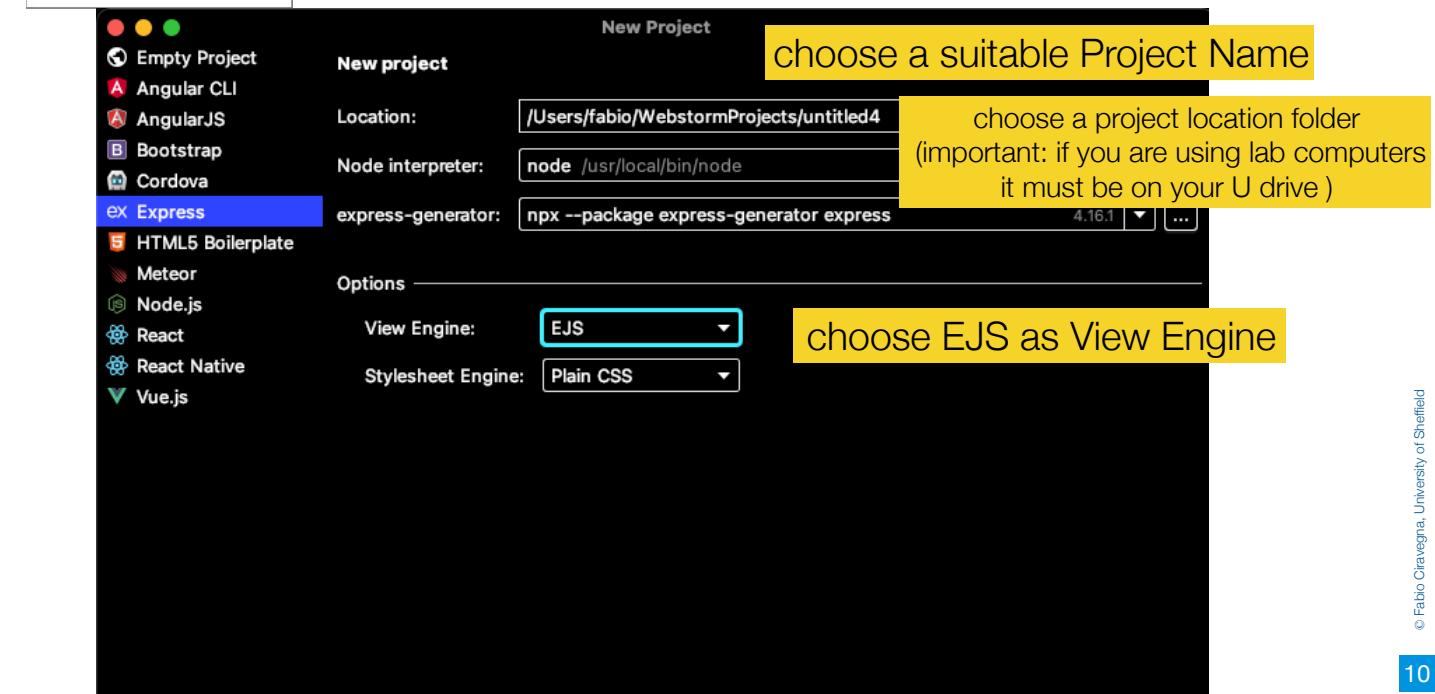
No!!!



9



The
University
Of
Sheffield.



© Fabio Cravagna, University of Sheffield

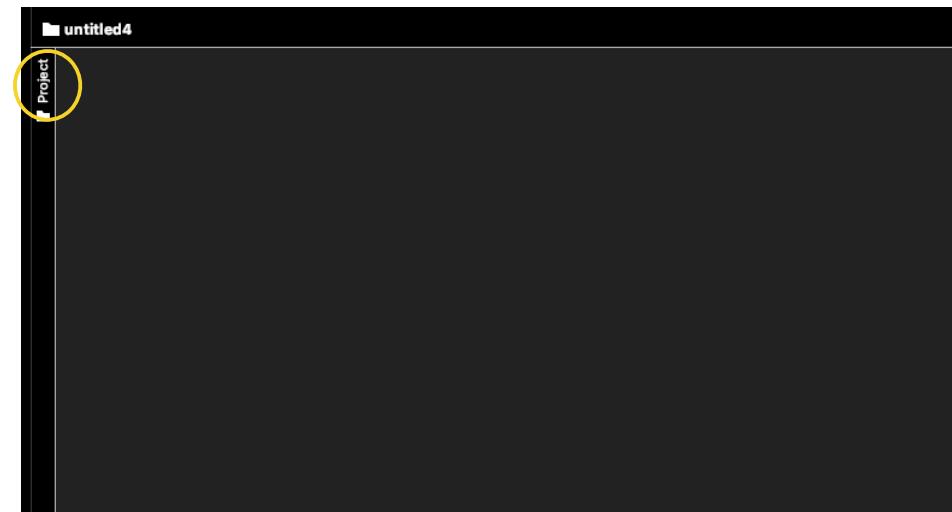
10



The
University
Of
Sheffield.

Here is your project

- If you see a missing left panel, click on Project
 - (vertical writing top left)

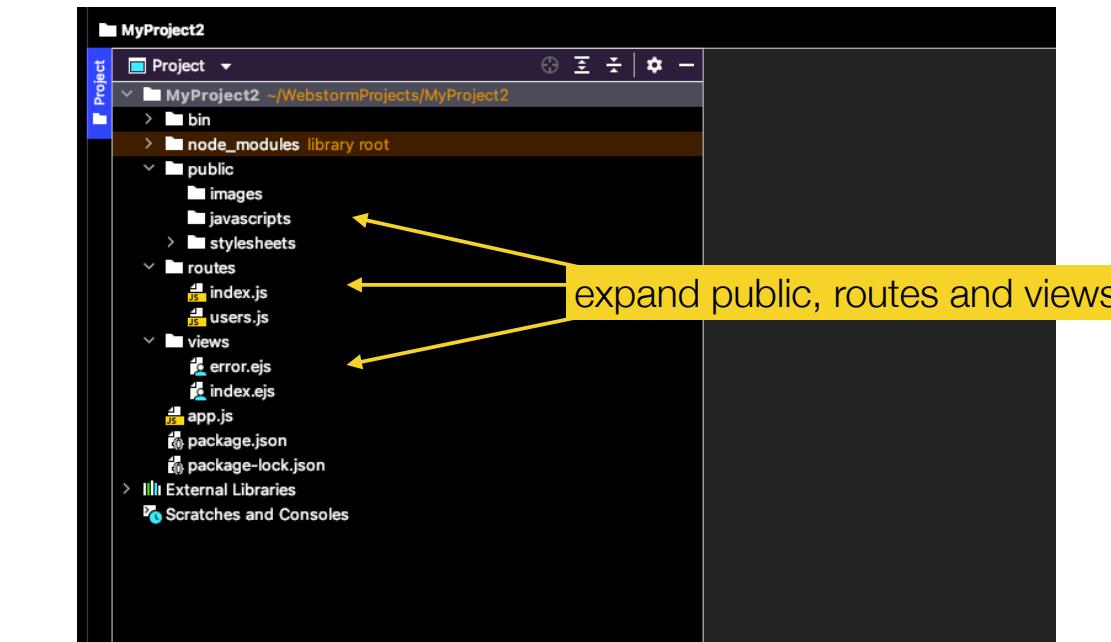


11



The
University
Of
Sheffield.

Here is your project (it may take a while)



© Fabio Cravagna, University of Sheffield

12



In IntelliJ

- in app.js

```
var users = require('./routes/users');
...
app.use('/users', users);
```



- what are the routes files?
- the default index.js responds to paths that follow /
- users.js responds to path that follow the path /users/; that is declared here
- if you added app.use('/whatever', whatever); in app.js
 - and a file called whatever.js under routes
 - all the routes there will respond to /whatever/

- in e.g. routes/users.js

```
/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('whatever');
});
```

• this will respond to http://localhost:3000/users/

© Fabio Cravagna, University of Sheffield

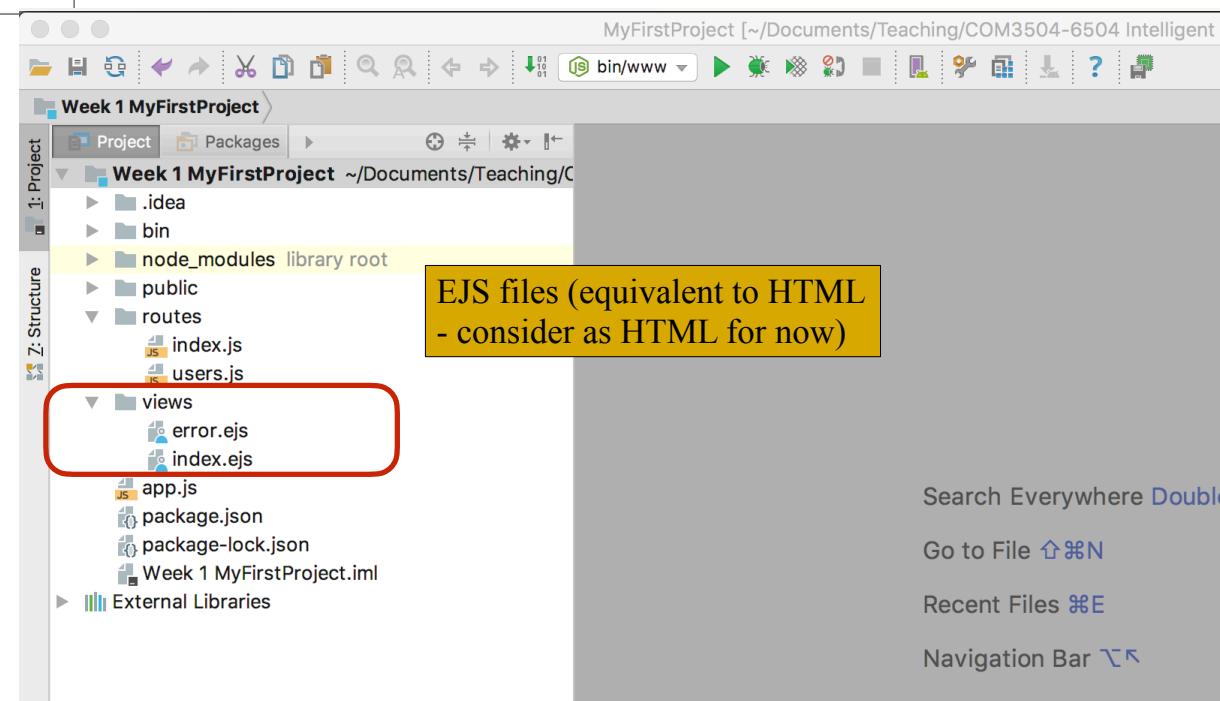
13



The client side



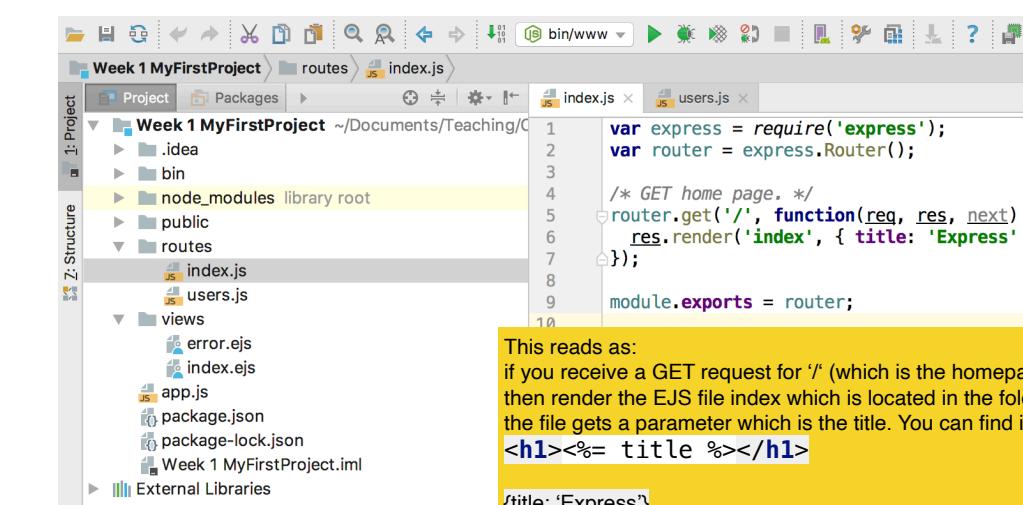
EJS Template files



Search Everywhere Double
Go to File ⌘N
Recent Files ⌘E
Navigation Bar ↕



Serving EJS Template files in routes



This reads as:
if you receive a GET request for '/' (which is the homepage)
then render the EJS file index which is located in the folder views
the file gets a parameter which is the title. You can find it in the EJS file as
<h1><%= title %></h1>

{title: 'Express'}
is a parameter passed to the file so that the title is replaced into the
code BEFORE the html code is interpreted

It is a way to make the html sensitive to parameters provided by the server

© Fabio Cravagna, University of Sheffield

© Fabio Cravagna, University of Sheffield

16



The
University
Of
Sheffield.

```

<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css'>
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>

```

EJS files are HTML templates where parts written between <%= and %> are interpreted on the basis of the parameters passed to the render command before the HTML code is interpreted (e.g. if the parameter is {title: 'this is a file'})

<h1><%= title %></h1>
will be passed to the HTML interpreter as
<h1>This is a file</h1>

17



How to serve static files

- If no special rendering is needed, you can insert HTML files under the public directory

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    <h1>Title</h1>
  </body>
</html>

```

© Fabio Cravagna, University of Sheffield

18



Serving static files

<http://expressjs.com/starter/static-files.html>

- Serving static files is accomplished with the help of a built-in middleware in Express
 - express.static.
- In app.js just the name of the directory where you keep your static files
- Note: WebStorm does it for you - the official name of the folder is **public**
- If you want to change the name of the folder to **public_files** (not suggested) insert the following line in app.js

```
app.use(express.static(path.join(__dirname, 'public_files')));
```

© Fabio Cravagna, University of Sheffield

19



Where to declare the middleware

- add this line in app.js

```

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

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger({ format: 'dev' }));
app.use(express.json());
app.use(express.urlencoded({ options: { extended: false } }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use('/', indexRouter);

```

© Fabio Cravagna, University of Sheffield

20



Static Files (ctd)

- Now, you will be able to load ALL files under the public directory:

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

© Fabio Oravagna, University of Sheffield

21



Task: create a file under public

- Right click on the public folder and choose 'new'. Choose new HTML file

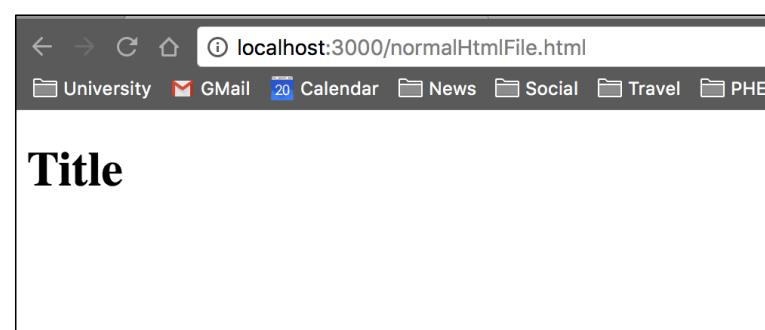
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<h1>Title</h1>
</body>
</html>
```

© Fabio Oravagna, University of Sheffield

22



Open Chrome: go to <http://localhost:3000/normalHtmlFile.html>



NOTE: FOR THE MODULE YOU ARE **REQUIRED TO USE CHROME**

© Fabio Oravagna, University of Sheffield

23



The University
Of
Sheffield

Plugins supporting *.ejs files found.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
<h1>Title</h1>
<p>This is a test page</p>
</body>
</html>
```

note any path starting with / refers to the folder public

Except for the ejs files which are in the folder views

open views/index.ejs, this is your home file (what normally would be index.html)

remember: ejs files are html files with parameters passed by the server!

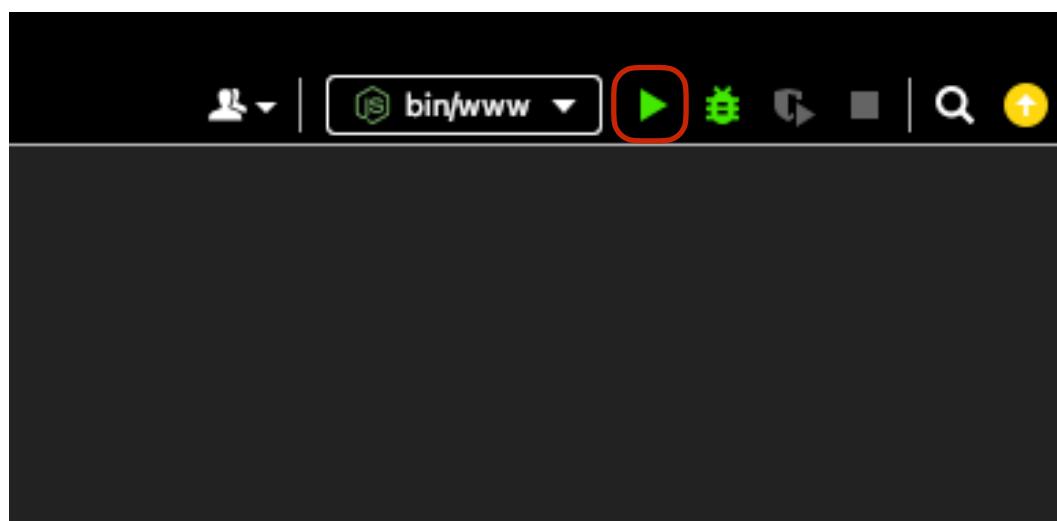
© Fat

24



To run the server

- top right of screen
- click on the green arrow



25



Running the client

```

papillodema-server bin www
Project > papillodema-server [AssignmentServer]
  > idea
  > bin
    > www
      > config
      > models
      > node_modules library root
      > private
      > projectFilesBackup
      > public
      > routes
      > views
        > .gitignore
        > app.js
        > Assignment.html

1  #!/usr/bin/env node
2
3  /**
4   * Module dependencies.
5   */
6
7  var app = require('../app');
8  var fs = require('fs-extra');
9  var debug = require('debug')('assignment:server');
10 var https = require('https');

11 /**
12  * Get port from environment and store in Express.
13 */
14
15 var port = normalizePort(process.env.PORT || '3000');

16

```

your server is on localhost or 127.1.1

the port

localhost:3000

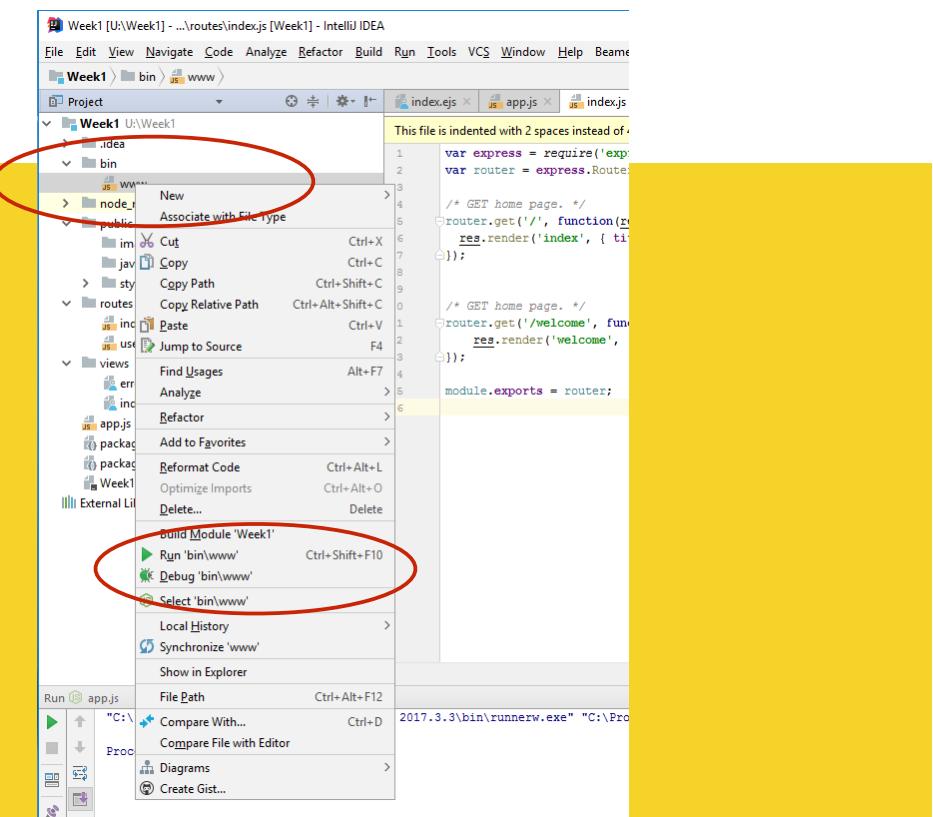
© Fabio Cravagna, University of Sheffield

27



- if the arrow is unelectable

- Right Click on bin>www
- Choose Run



26

© Fabio Cravagna, University of Sheffield



what is a port?

- Ports are an old concept from when servers had physical cables entering ports
- you could contact a hardware server through a specific entry point, i.e. a port
 - Nowadays computers have just fibre optic entering them but the concept of ports has been kept
 - Ports are entry points to the physical server
 - You can only have one process (e.g. your node server) running on one port
 - If you try to run a server when another one is running you will get an error telling you that the port is taken
 - If so, either stop the server on that port or run your process on a different port by changing the value 3000 in bin/www
 - var port = normalizePort(process.env.PORT || '3000');

28

© Fabio Cravagna, University of Sheffield



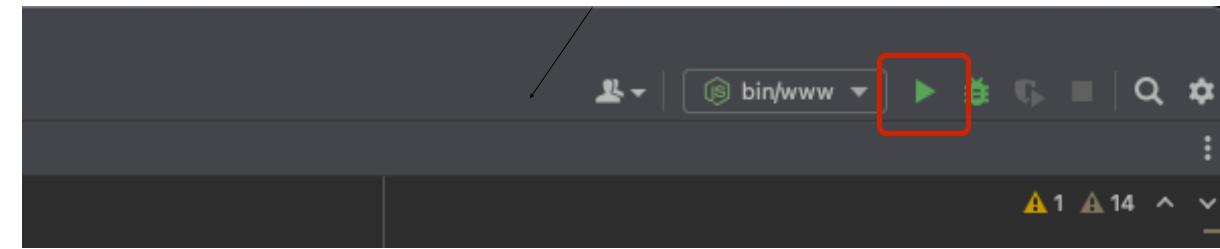
Ports (ctd)

- Ports have values 1-65535 are available, and ports in range 1-1023 are the privileged ones: an application needs to be run as root in order to listen to these ports
 - Suggestion: use ports 3000-3004 or 8080 (standard port) 8090-8092
- If you use 8080 you can omit the port. i.e.
http://localhost defaults to http://localhost:8080



Changes?

- Note: changes to the code have different effects:
 - changes to the **client**
 - i.e. in the Views and Public directories
 - require reloading the page in the browser
 - changes to the **server** (node js)
 - require restarting the server from IntelliJ





The
University
Of
Sheffield.

Week 1 Lab Class Exercise 1

Prof Fabio Ciravegna
f.ciravegna@shef.ac.uk

© Fabio Ciravegna, University of Sheffield



The
University
Of
Sheffield.

Exercise 1

- This exercise is designed to make sure that you understand how to use
 - routes and
 - ejs files
- in a GET using Express
- Open the project you have created

© Fabio Ciravegna, University of Sheffield

3



Exercise one

- Familiarise yourself with the Express structure also using the lecture slides
- then start working on the exercises in the next slides

© Fabio Ciravegna, University of Sheffield

2

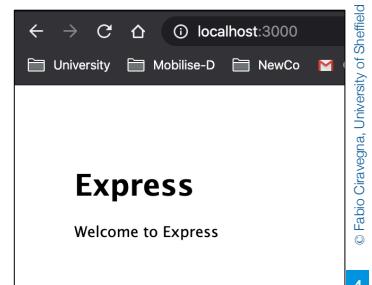


Open the client

- run the node interpreter (right click on bin/www or



- open the browser on http://localhost:3000



© Fabio Ciravegna, University of Sheffield

4



The
University
Of
Sheffield.

Learning to manipulate .ejs files

- Your exercise is to modify views/index.ejs so that it looks like

Welcome to My Class

Please click [getting the other file](#)

- Please note: the term **My Class** should be passed as parameter to the ejs file, so using the notation
`<h1> Welcome to <%= title %> </h1>`
- and having the route in routes/index.js modified as

```
router.get('/', function(req, res, next) {  
    res.render('index', { title: 'My Class' });  
})
```

© Fabio Cravagna, University of Sheffield

5

Following the link

Welcome to My Class

Please click [getting the other file](#)

- insert a link into the html of the ejs file so to provide a link to the route
 - /welcome
 - i.e. the html should be
`getting the other file`

6



The
University
Of
Sheffield.

Responding to /welcome

- Create a new route path in **routes/index.js**

- so that when you click the link, a file called welcome.ejs is rendered which will be shown as

Welcome to COM3504

COM3504 is a cool module

© Fabio Cravagna, University of Sheffield

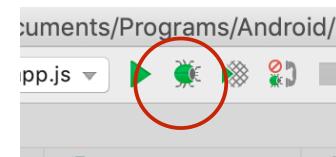
7

- Note: the string COM3504 should be changeable from server side to COM6504 without changing the HTML of the file welcome.ejs (i.e. it is a parameter as well)

How would you do it?

- Try to think: the solution is in the next slides but try to come up with a solution before checking it
- Hint 1: create an ejs view called `welcome.ejs` under `views`
- Hint 2: you must add a new route path for a GET which returns
 - the rendering of `welcome.ejs` with 'COM3504' as parameter

Note! remember to reload the server every time you change the routes files



© Fabio Cravagna, University of Sheffield

8



Solution

- This is the route path that you introduce in routes/index.js:

```
router.get('/welcome', function (req, res) {  
    res.render('welcome', { title: 'COM3504' });  
});
```

- The HTML should be straightforward
 - the complete solution will be published in teh next hours

© Fabio Cravegna, University of Sheffield

9



Questions?

You may wan to try and created a third file and link to it in the welcome file

© Fabio Cravegna, University of Sheffield



The
University
Of
Sheffield.

Debugging node.js

Prof. Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

© Fabio Ciravegna, University of Sheffield

3



The
University
Of
Sheffield.

Learning Objectives

- You will learn to debug a client server architecture where the client is a browser and the server is an Express server
 - how to debug the server using WebStorm
 - how to debug the client (browser) using Chrome

© Fabio Ciravegna, University of Sheffield

© Fabio Ciravegna, University of Sheffield

2



The
University
Of
Sheffield.

Debugging node.js

- Node.js does not run in a browser
 - it runs on a server

© Fabio Ciravegna, University of Sheffield

3



The
University
Of
Sheffield.

Debugging HTML/Javascript in

<https://developer.chrome.com/devtools/docs/javascript-debugging>

- We will be using IntelliJ
 - if you have used AndroidStudio or WebStorm, it is the same product with different flavours
- There are two parts in any client/server architecture:
 - the client (e.g. a browser): this can be debugged with Chrome
 - the server (the nodejs server), this is to be debugged with IntelliJ

© Fabio Ciravegna, University of Sheffield

4



Chrome debugging (client)

- open from right menu (or view>developers>javascript console on a Mac)

1

Debugging the server

The screenshot shows the Android Studio interface with the following components highlighted:

- Project Structure:** Shows the `ProgressiveApp` project structure with `node_modules` as the library root.
- Code Editor:** Displays `index.js` with code related to Express.js. A red circle indicates a breakpoint at line 6. Annotations include:
 - "Click to set a break point" pointing to the red circle.
 - "Click to debug the server" pointing to the green play button icon in the toolbar.
- Debugger Tool Window:** Shows the `bin/www` frame with the `Local` variables panel open. Annotations include:
 - "Click to see the stack" pointing to the stack icon in the toolbar.
 - "control the debugger (step, enter function call, etc.)" pointing to the toolbar.
 - "local variables and stack" pointing to the `Variables` tab in the debugger window.

© Enbis-Gironde | University of Sheffield

6



Debugging client & server

- While you develop, try to debug the client server architecture so to identify errors
 - 1. put a break into Chrome to check that the function `checkForErrors` works appropriately

1

Break points in client side

- open Chrome
 - open the page `http://localhost:3000/`
 - open the developers tools
 - insert break point
 - reload page

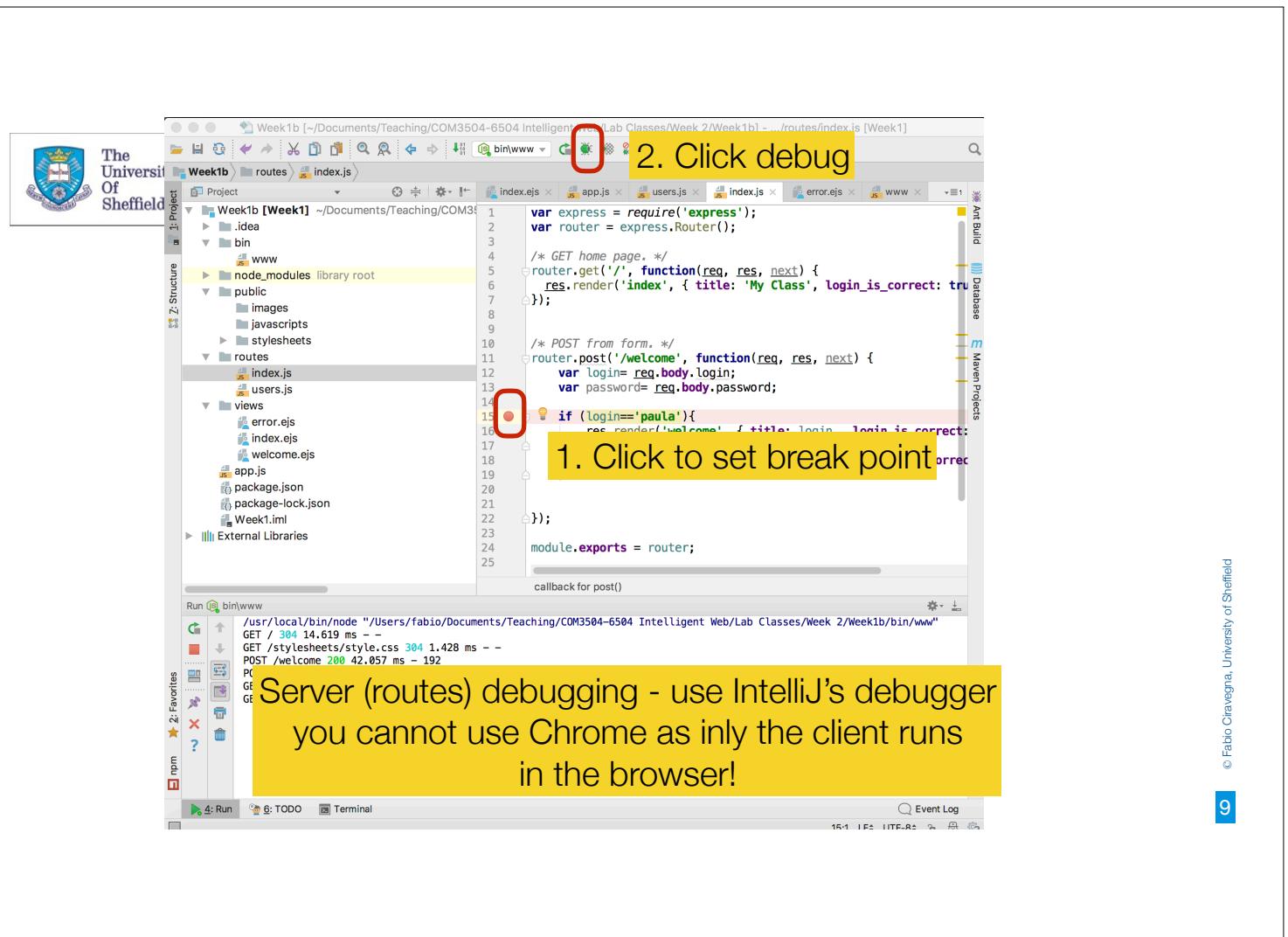
The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. On the left, the file structure is shown with 'javascripts/index.js' highlighted. The main pane displays the code of 'index.js'. A red box labeled '1.' highlights the 'Sources' tab. A red box labeled '2.' highlights the 'javascripts/index.js' file. A red box labeled '3.' highlights the number '8' at the start of line 8, which contains a conditional statement. The right sidebar shows developer settings like 'Pause on caught exceptions' and a list of breakpoints.

```
function checkForErrors(isLoginCorrect){  
  if (!isLoginCorrect){  
    alert('login or password is incorrect');  
  }  
}  
if (true){  
  login();  
}  
// Line 8: if (true){  
  login();  
}  
// Line 1, Column 1
```

Click on the line number to set up a break point

Fabio Ciravolo | University of Sheffield

9





Week 1 Debugging NodeJS Applications (with solution)

Professor Fabio Ciravegna
Department of Computer Science
University of Sheffield
<http://staffwww.dcs.shef.ac.uk/people/F.Ciravegna/>

© Fabio Ciravegna, University of Sheffield

Please note

My screenshots of WebStorm may look different from yours mostly in the colours) because I use the high contrast settings of WebStorm (set under Preferences)

© Fabio Ciravegna, University of Sheffield



- You are given one rather large application
 - of which you do not need to understand much
 - it is the exercise for week 4
 - there are errors in it and you have to fix them
 - Errors are
 - Some spurious lines in the Javascript client
 - that you have to identify with Chrome's debugger
 - Some javascript errors in the server
 - that you have to identify with the IntelliJ debugger
 - action:
 - again just remove those lines

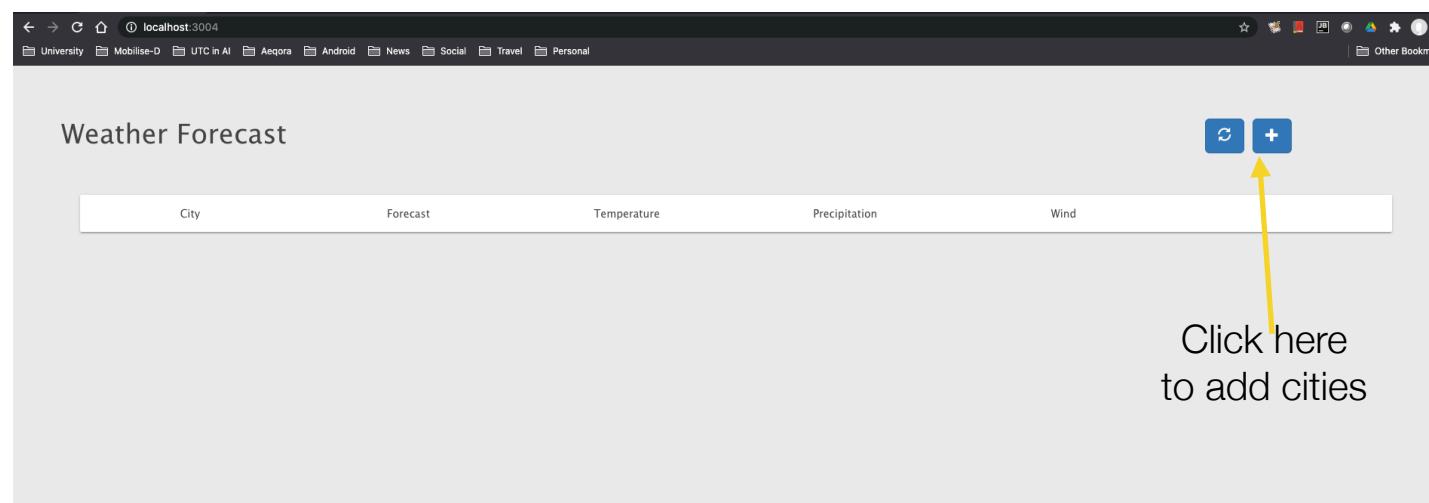
© Fabio Ciravegna, University of Sheffield

3



- Run the nodeJS project in IntelliJ
 - right click on bin/www
 - select run
- Open the browser on <http://localhost:3004>

this is what you will see



Click here
to add cities



What you should see after adding some cities

- If there was no error

City	Forecast	Temperature	Precipitation	Wind
London	Cloudy	-2	43	13
Madrid	Snowy	0	45	13
Paris	Clear	13	53	5

Instead you will still see this

- No cities are added

City	Forecast	Temperature	Precipitation	Wind

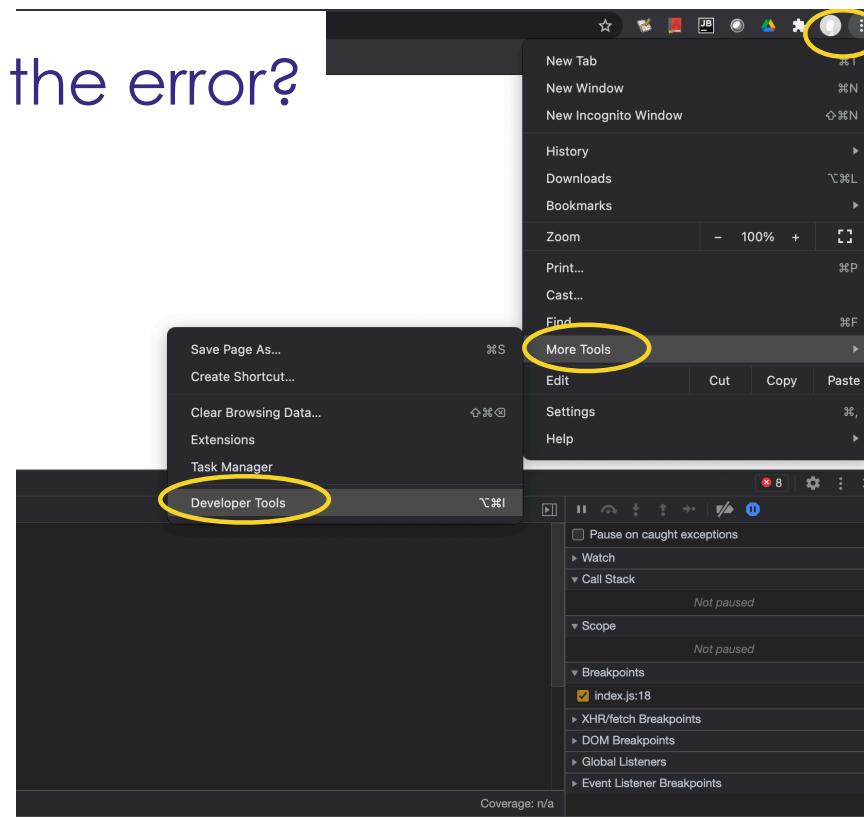


Where is the error?

- There is an error client side and an error server side

- Look first of all for the one client side. Use Chrome's debugger

- Once you have found the error, set a break point on the line before the error. Reload the page and inspect the value of the variable that causes the error



- See next slide to discover how to fix the error

- DO NOT LOOK AT THE NEXT SLIDE until you have identified the error yourself

The solution

- Go to the function retrieveAllCitiesData
- Remove the following three lines

```
let container=document.getElementById("container");
container.innerHTML = "Common JS Bugs and Errors";
if (!container) return;
```

- The error is due to the fact that there is not container element in the interface, so assigning its inner html causes an error



The
University
Of
Sheffield.

Video: how to debug the client side error

- To run the video, please go to the Solution folder on Blackboard

Weather Forecast

City	Forecast	Temperature	Precipitation	Wind

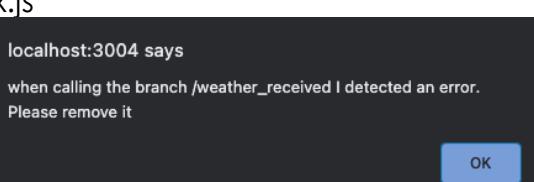
© Fabio Cravagna, University of Sheffield

9

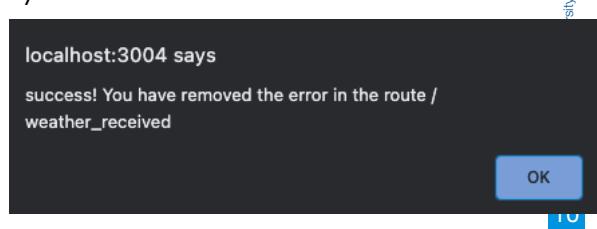


The server-side error

- When the client-side error is fixed, check the error on the server side
 - reload the page and you should see the correct cities and temperatures
 - click on the blue button with a cross in a circle (top right of screen)
 - an alert will tell you there is an error on the route /weather_received
 - identify the route /weather_received in routes/index.js
 - find the error by putting a break on the first line and stepping until the error shows up
 - it may either happen that you will lose control of the debugger
 - or the entire server will crash
 - the server will return an error that is intercepted by an alert
 - remove the offending line
 - reload the server
 - press the button again
 - The alert should tell you you have removed the error



University of S



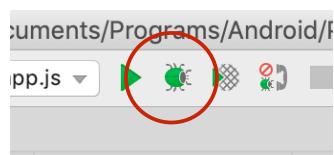
University of S



The
University
Of
Sheffield.

How to debug the Server

- Click on the green bug
- identify the error in IntelliJ's console
 - it will tell you that there is a 500 error on a specific route (which is the route that is called when clicking the button)
- Put a break point on the first line of the route that causes the error
- call the debugger
- understand why there is an error



© Fabio Cravagna, University of Sheffield

11



How to set a break point

```

const forecast= getWeatherForecast(req.body.location, req.body.date);
res.setHeader('Content-Type', 'application/json');
res.send(JSON.stringify(forecast));
}

router.get('/weather_received', function(req:Request<P, ResBody, ReqQuery>, res:Response<ResBody>){
  var location= req.body.location;
  const forecast= {};
  getWeatherForecastTwo(forecast);
  res.setHeader('Content-Type', 'application/json');
  res.send(JSON.stringify(forecast));
});

```

Click here

University of S

12



The
University
Of
Sheffield.

The debugger

- When the debugger stops at a break point you will see the debugger controls (next slide)

The break point

The debugger's controls (next slide)

The function call stack

The local and global variables' values

© Fabio Cravegna, University of Sheffield

13

14



The
University
Of
Sheffield.

The debugger's controls

The debugger view (default view)

The javascript console (where you will spot errors; this is where the output of console.log() instructions goes)

The stepper - used to move the debugger execution to the next line. If the current line is a function call, it will execute its code as a one block. It will not step into the function's code

The stepper into a function - if the current line contains a function call, use this button to step into the function's code

© Fabio Cravegna, University of Sheffield

15

it runs the debugger from the start (equivalent to clicking on the green bug at the top of the screen)

it moves the execution to the next break point. Useful when you are not interested in the current code and you want to proceed

© Fabio Cravegna, University of Sheffield

16



Solution

- comment out the following two lines

```
// var location= req.body.location;  
const forecast= {};  
// getWeatherForecastTwo(forecast);
```

- run the server again
- it should work

© Fabio Oravegna, University of Sheffield

17



Video: How to debug the server side error

- To see the video, go to the Solution folder

City	Forecast	Temperature	Precipitation	Wind
London	Overcast	1	95	18
Madrid	Clear	24	7	17
Paris	Cloudy	22	93	10

© Fabio Oravegna, University of Sheffield

8