COM3504/6504
Lab Class Week 1

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

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

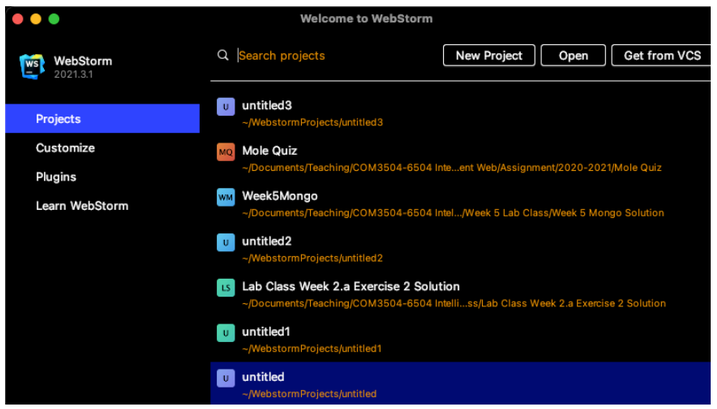
Important!

- 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

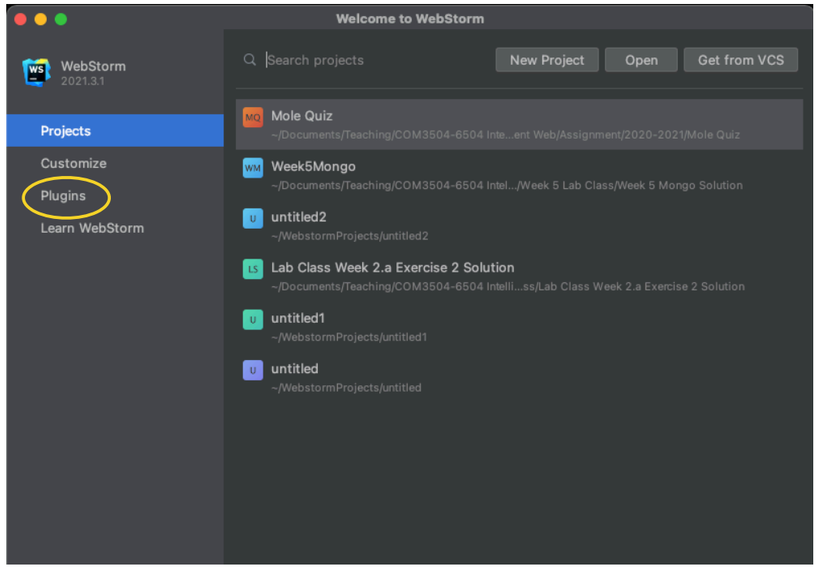
Creating an Express Project in WebStorm

Using WebStorm

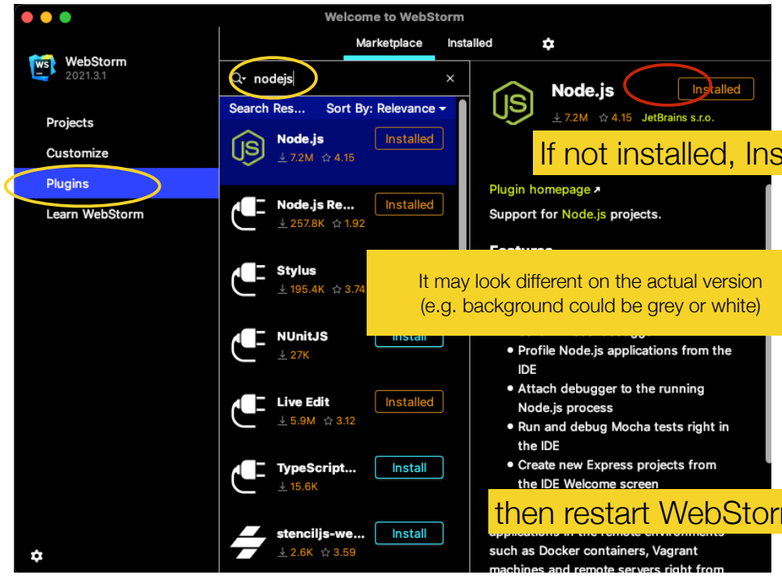
- In Windows Go to the Window menu and search
  - WebStorm
- On a Mac go to Applications and select WebStorm
- This is what you will see:
- you should have node already installed
  - to be sure: click on Plugins



Not on Lab computers: Install Node Plugin



Not needed on lab computers!!

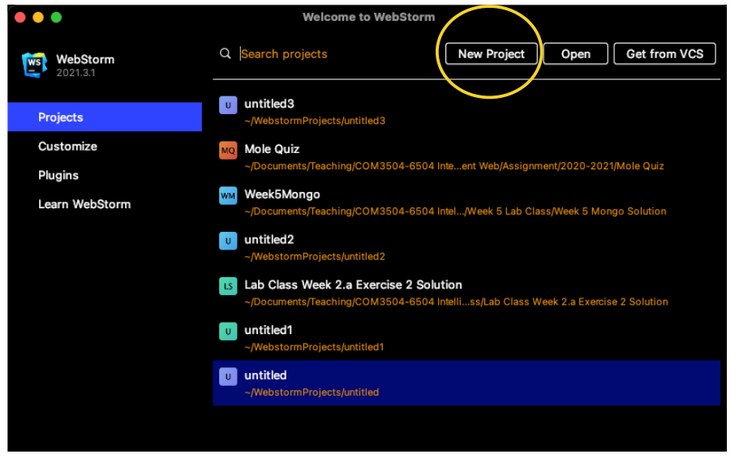


If not installed, Install it

It may look different on the actual version (e.g. background could be grey or white)

then restart WebStorm

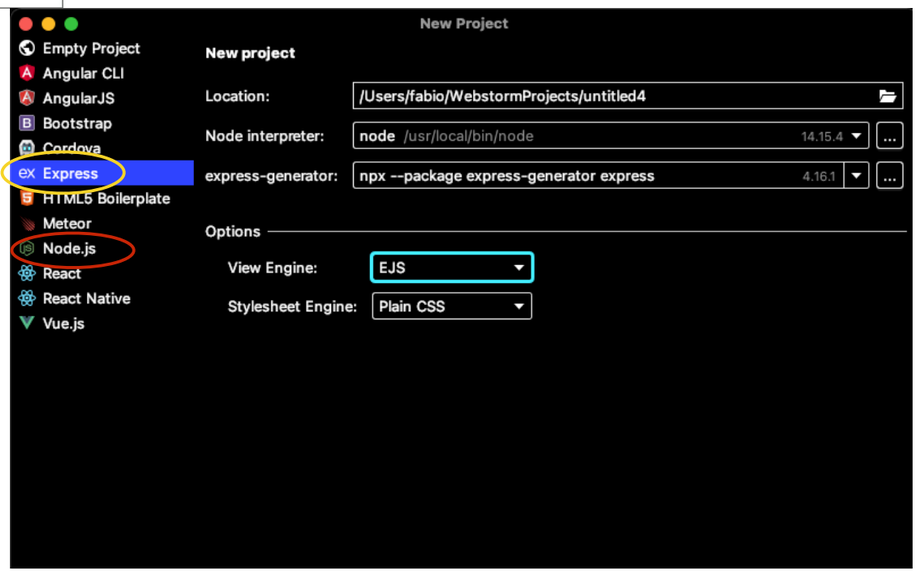
After installing and restarting, create a new project

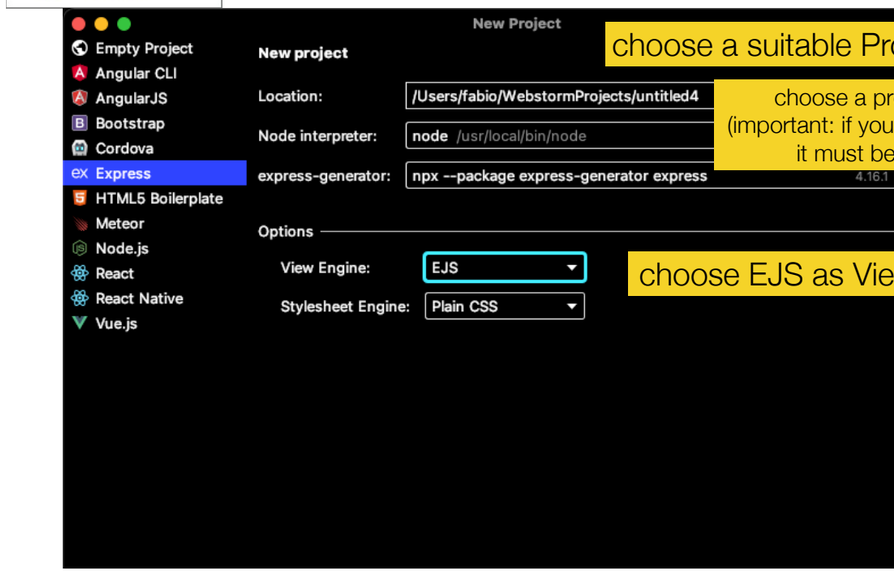


Select Express

Yes

No!!!





choose a suitable Project Name

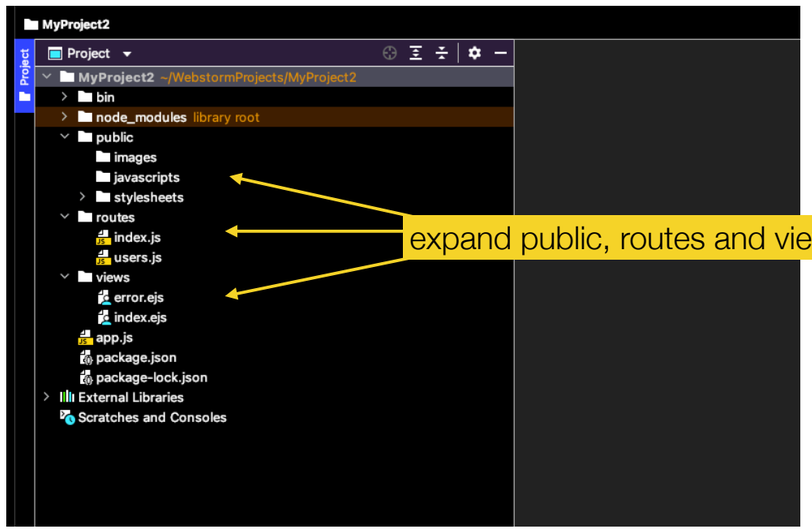
choose a project location folder
(important: if you are using lab computers
it must be on your U drive)

choose EJS as View Engine

Here is your project

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

Here is your project (it may take a while)



expand public, routes and views

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/

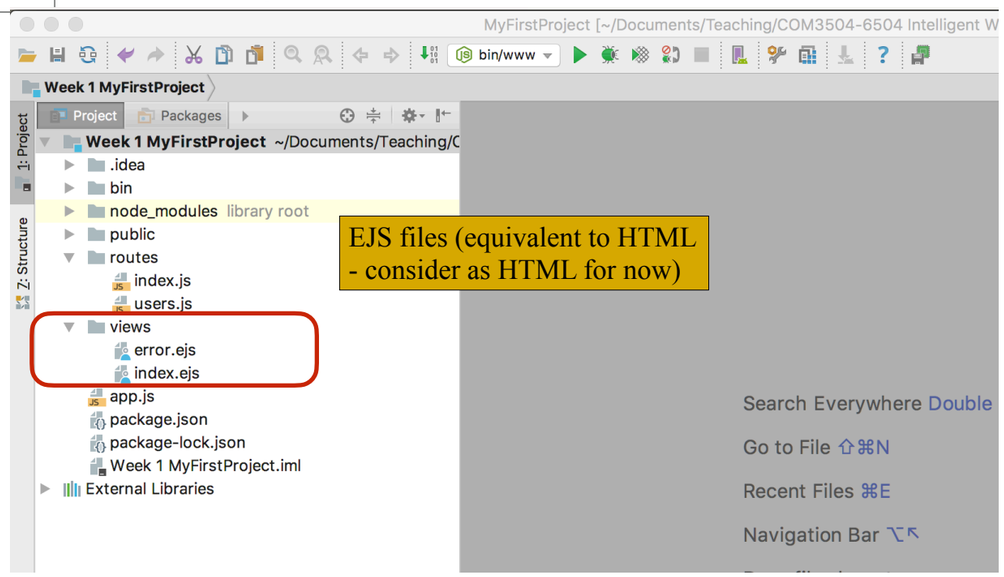
- in e.g. routes/users.js

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

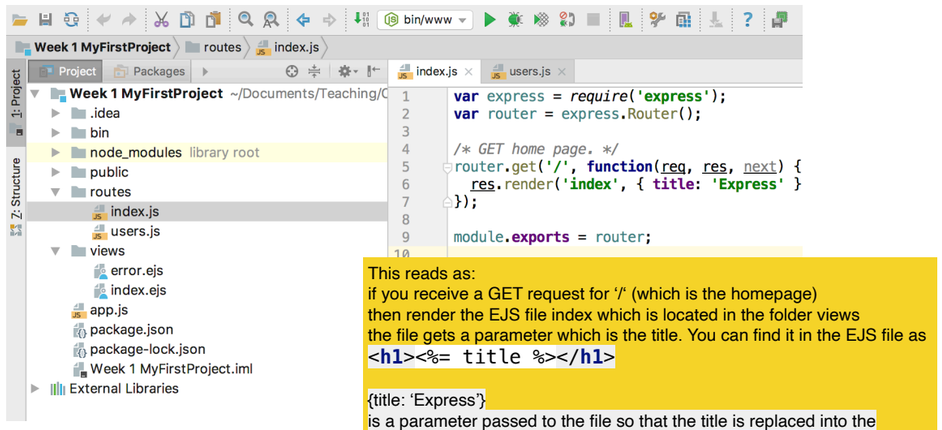
- this will respond to http://localhost:3000/users/

The client side

EJS Template files



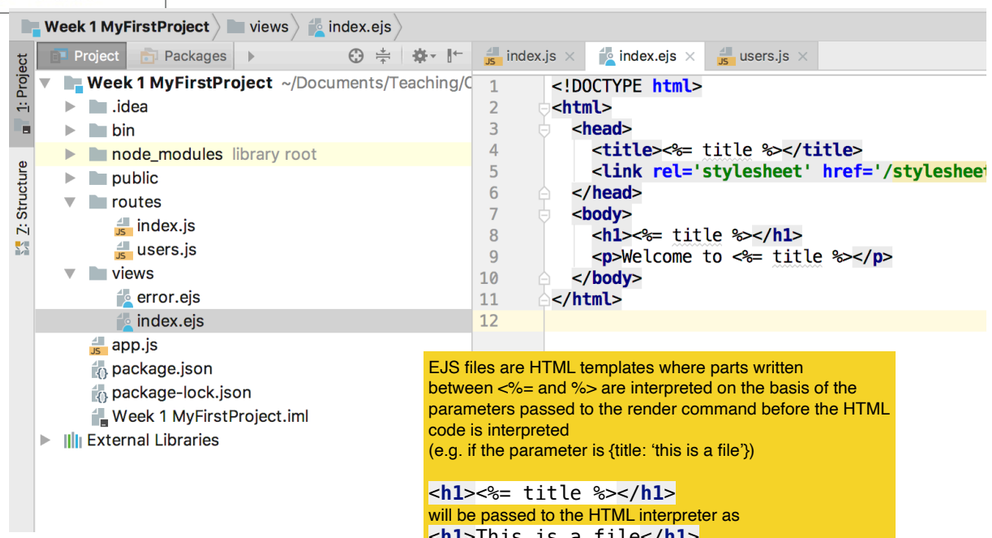
Serving EJS Template files in routes



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

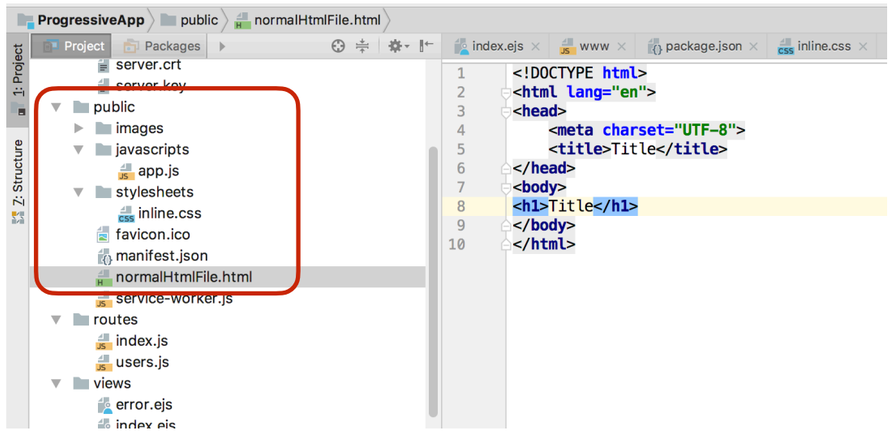


EJS files are HTML templates where parts written between <%= and %> are interpreted on the basis of the parameters passed to the render command before the HTML code is interpreted
(e.g. if the parameter is {title: 'this is a file'})

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

How to serve static files

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



Serving static files

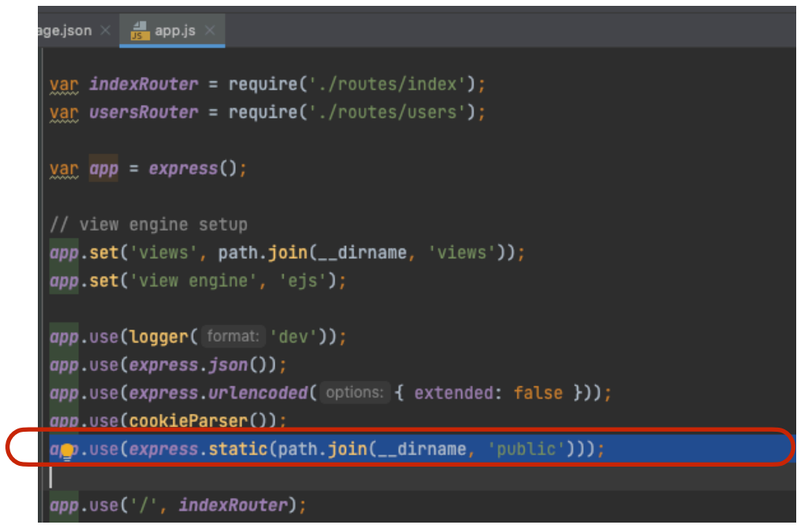
http://expressjs.com/starter/static-files.html

- 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 you 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')));
```


Where to declare the middleware

- add this line in app.js



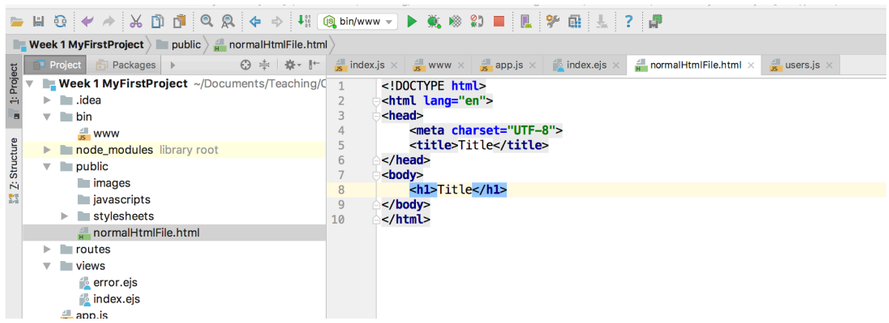
Static Files (ctd)

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

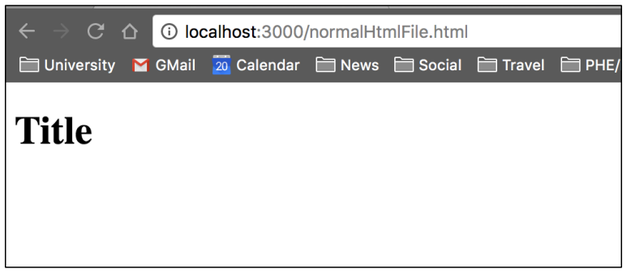
- http://localhost:3000/images/kitten.jpg
- http://localhost:3000/css/style.css
- http://localhost:3000/js/app.js
- http://localhost:3000/images/bg.png
- http://localhost:3000/hello.html

Task: create a file under public

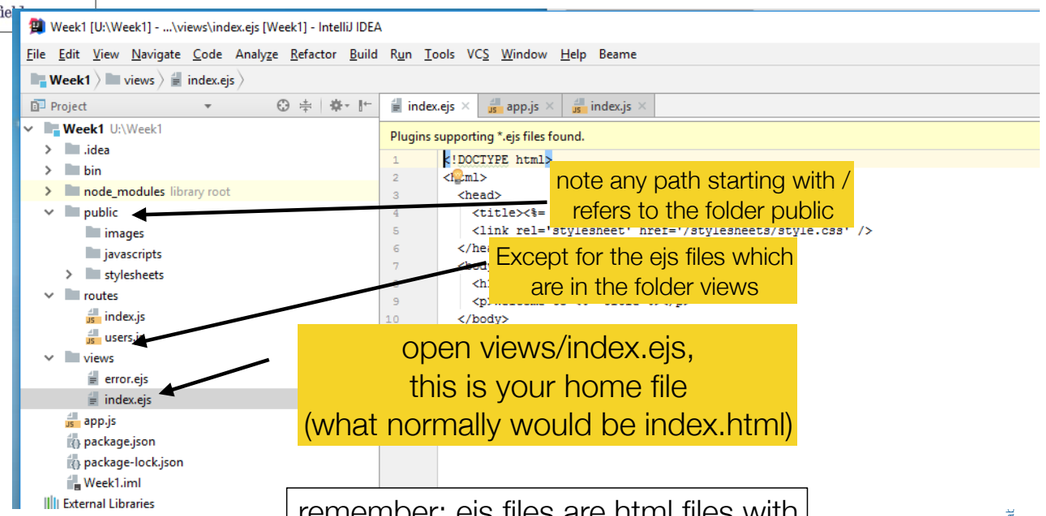
- Right click on the public folder and choose 'new'. Choose new HTML file



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

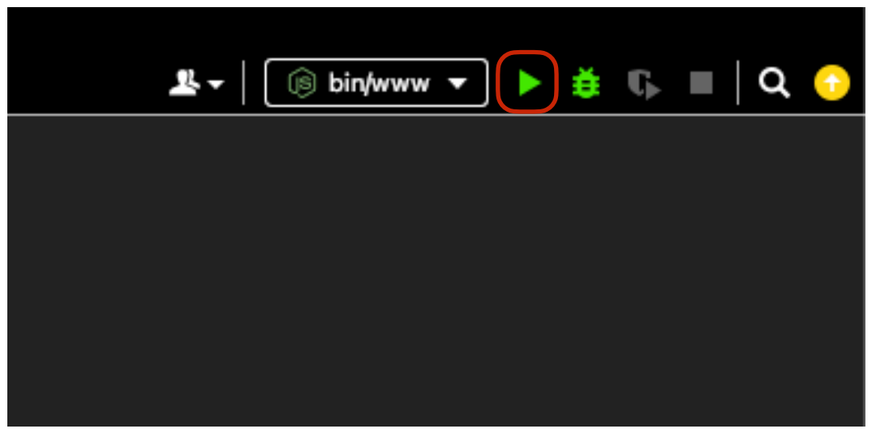


NOTE: FOR THE MODULE YOU ARE **REQUIRED TO USE CHROME**



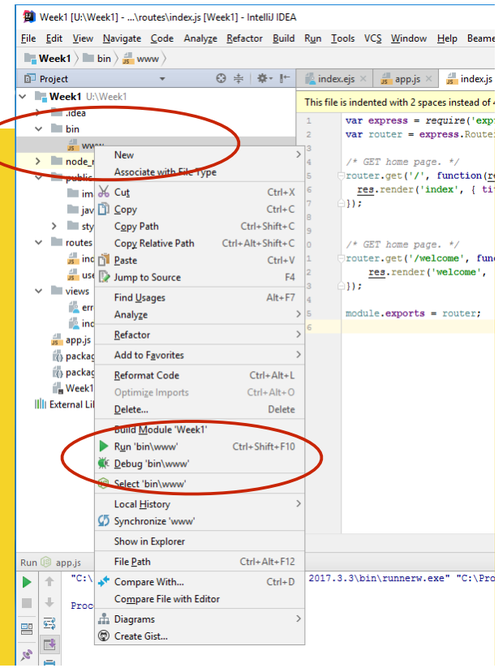
To run the server

- top right of screen
  - click on the green arrow

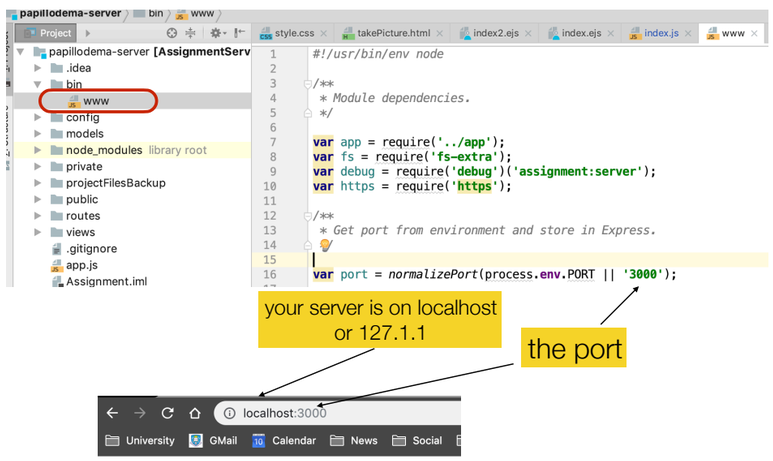


if the arrow is unelectable

- Right Click on bin>www
- Choose Run



Running the client



what is a port?

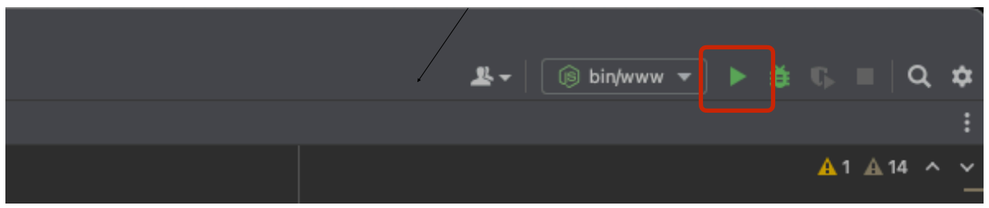
- Ports are an old concept from when servers had physical cables entering ports
  - you could contact a hardware server through a specific entry point, i.e. a port
    - Nowadays computers have just fibre optic entering them but the concept of ports has been kept
      - Ports are entry points to the 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');`

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



Week 1 Lab Class Exercise 1

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

Exercise one

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

Exercise 1

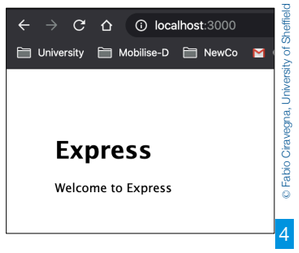
- This exercise is designed to make sure that you understand how to use
  - routes and
  - ejs files
  - in a GET using Express

- Open the project you have created

Open the client

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

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



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' });   }
```


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

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

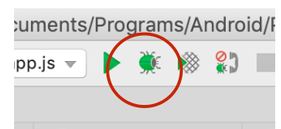
COM3504 is a cool module

- 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



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

Questions?

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

Debugging node.js

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

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

Debugging node.js

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

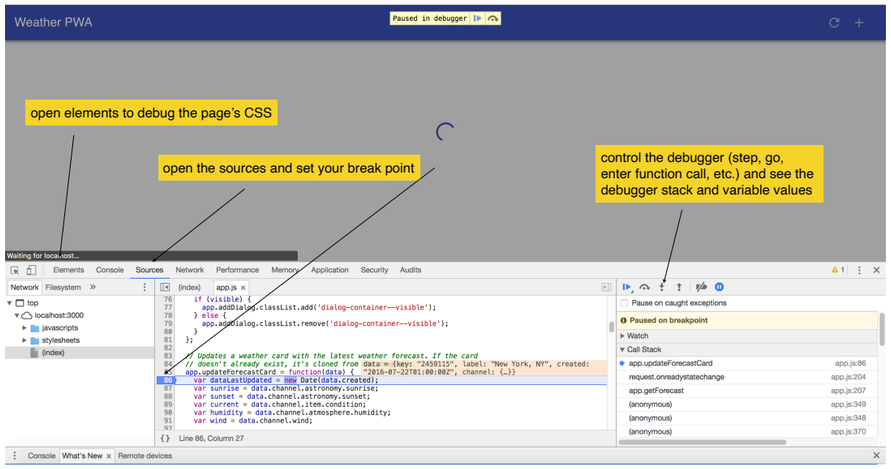
Debugging HTML/Javascript in

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

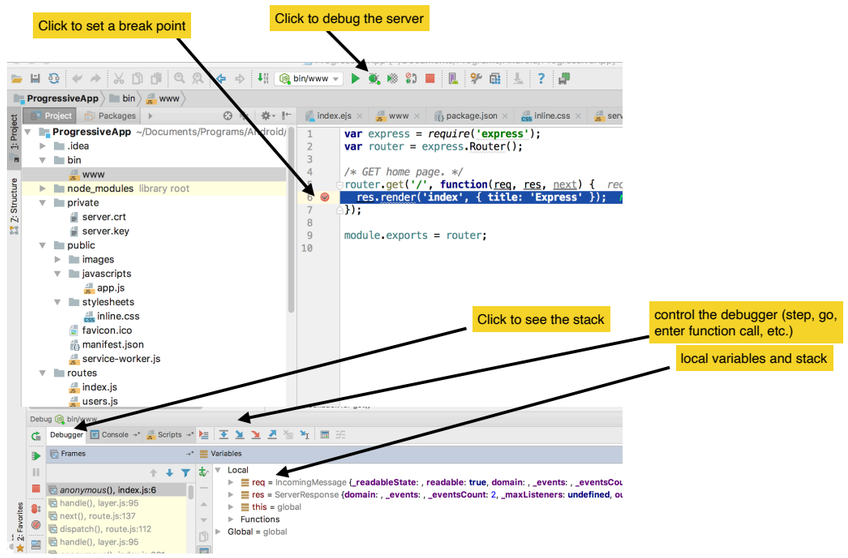
- We will be using IntelliJ
  - if you have used AndroidStudio or WebStorm, it is the same product with different flavours
- There are two parts in any client/server architecture:
  - the client (e.g. a browser): this can be debugged with Chrome
  - the server (the nodejs server), this is to be debugged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

Chrome debugging (client)

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



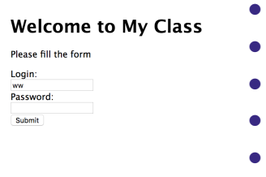
Debugging the server



Debugging client & server

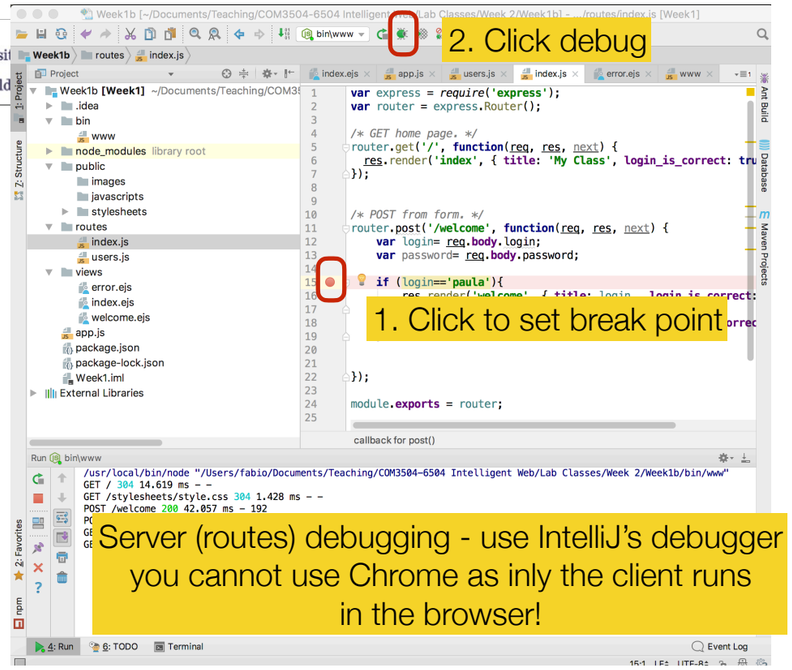
- While you develop, try to debug the client server architecture so to identify errors
  - 1. put a break into Chrome to check that the function *checkForErrors* works appropriately

Break points in client side



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

Click on the line number to set up a break point



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/

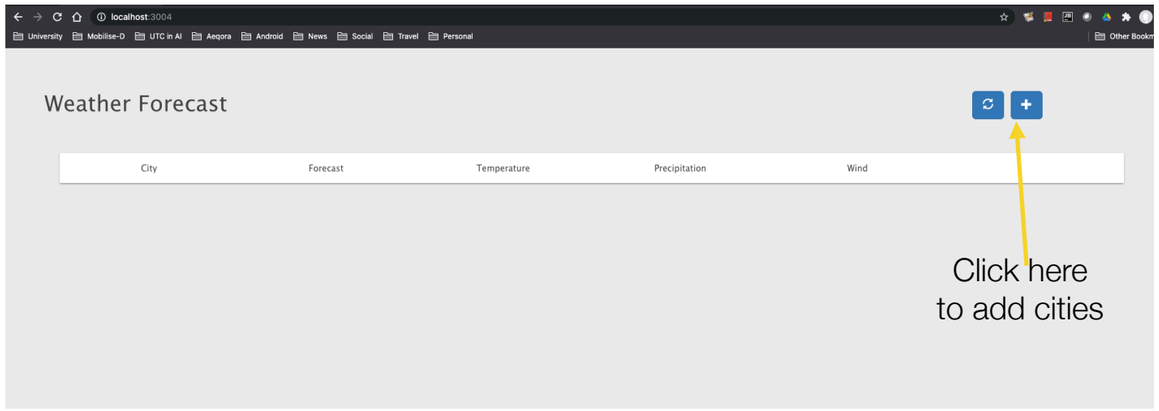
Pease note

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)

- 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

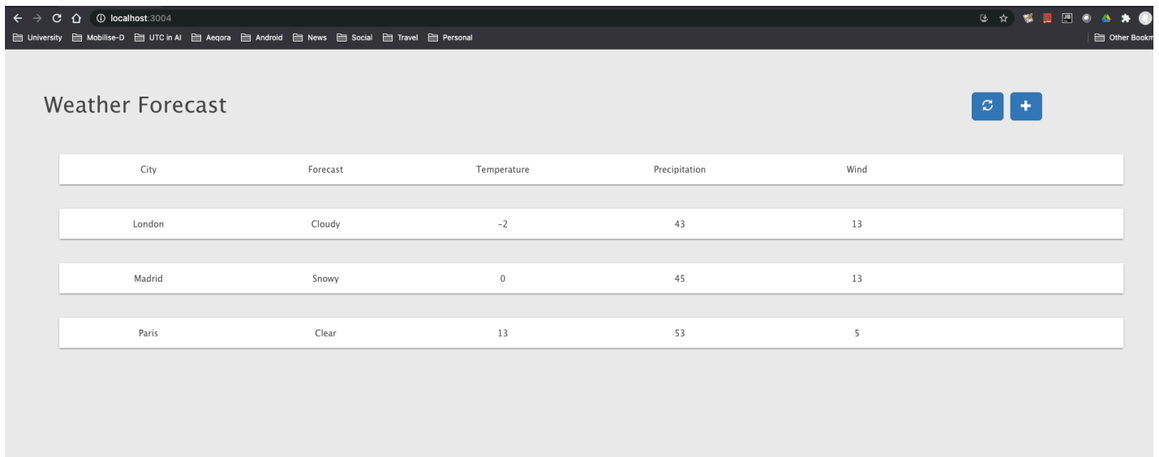
- Run the nodeJS project in IntelliJ
  - right click on bin/www
  - select run
- Open the browser on http://localhost:3004

this is what you will see



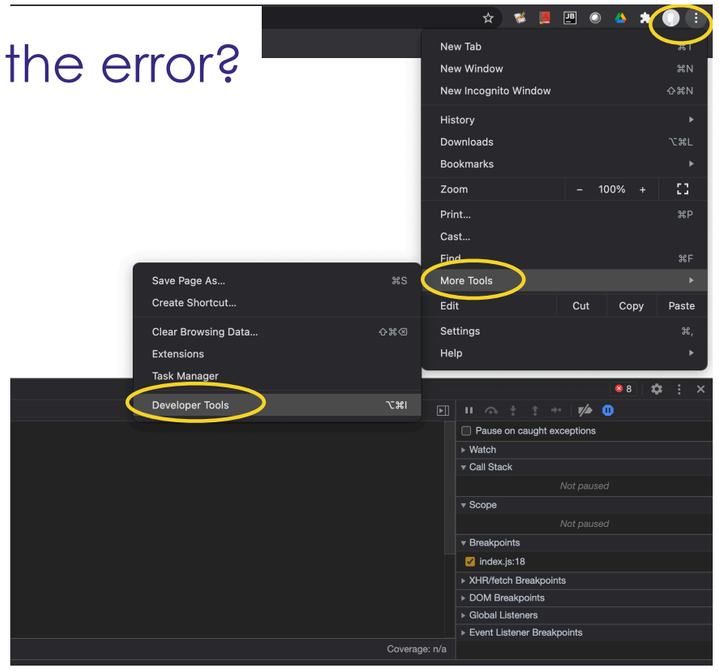
What you should see after adding some cities

- If there was no error



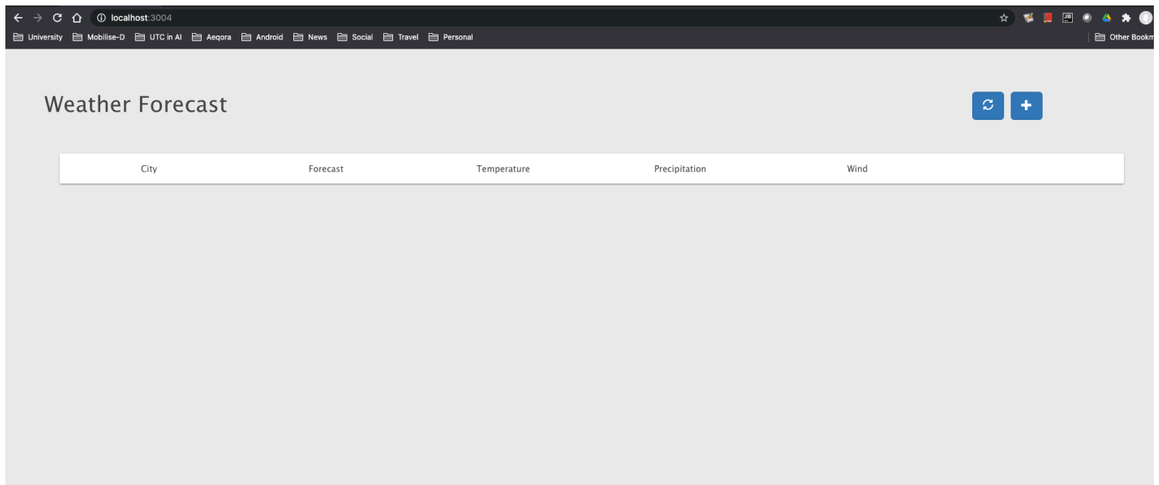
Where is the error?

- 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



Instead you will still see this

- No cities are added



The solution

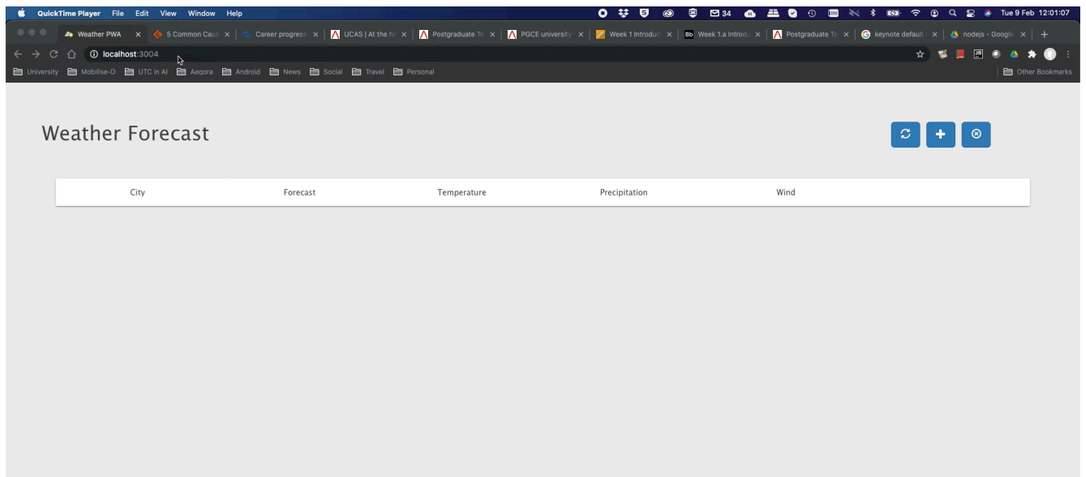
- Go to the function retrieveAllCitiesData
- Remove the following three lines

```
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

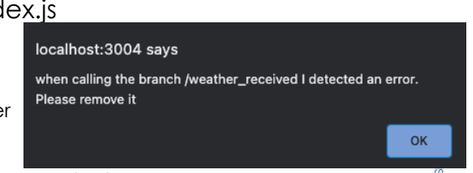
Video: how to debug the client side error

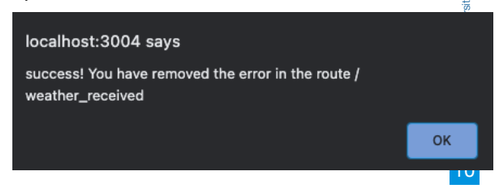
- To run the video, please go to the Solution folder on Blackboard



The server-side error

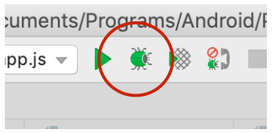
- When the client-side error is fixed, check the error on the server side
  - reload the page and you should see the correct cities and temperatures
  - click on the blue button with a cross in a circle (top right of screen)
    - an alert will tell you there is an error on the route /weather_received
  - 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



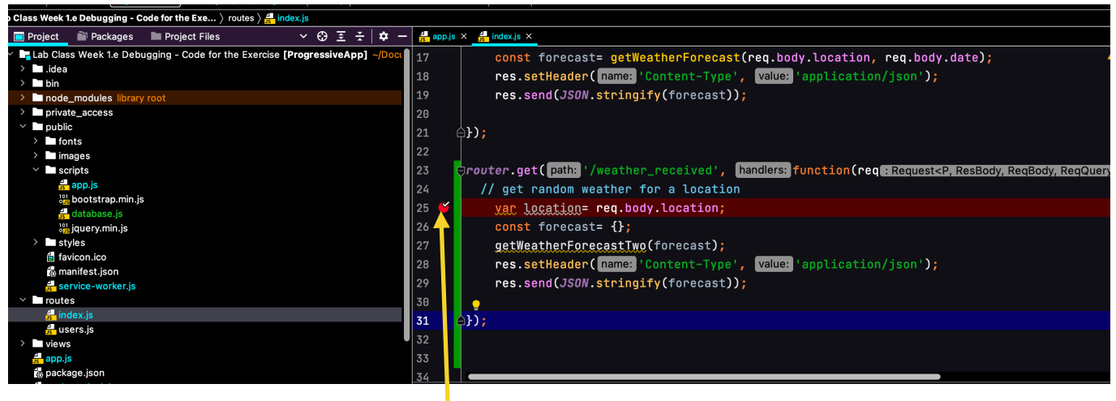


How to debug the Server

- 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



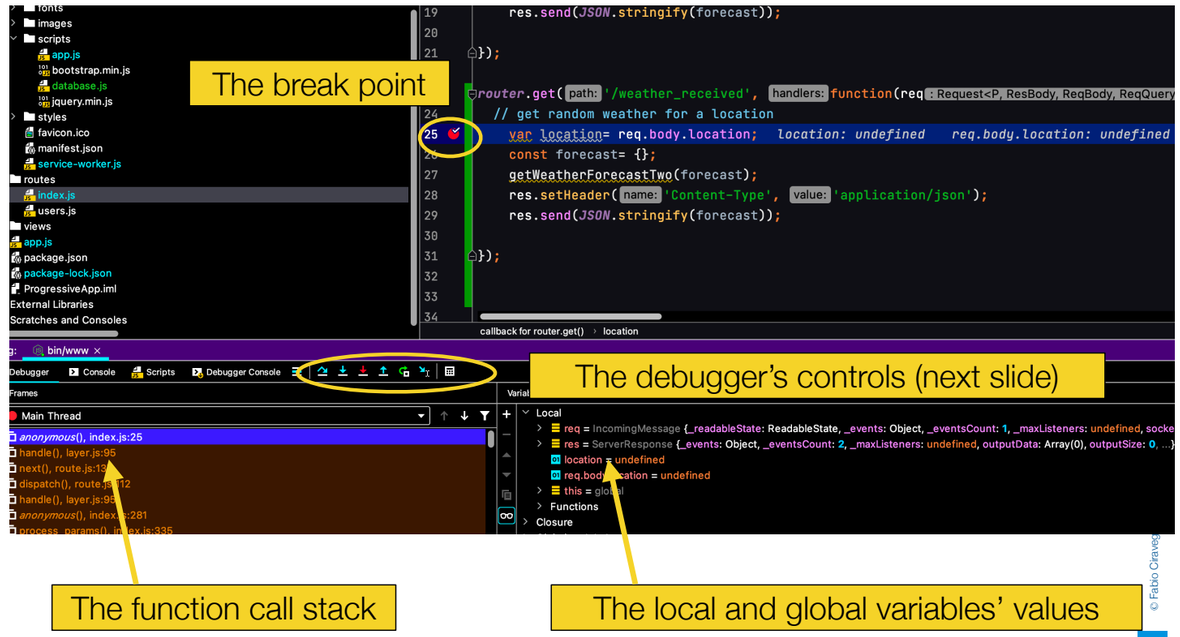
How to set a break point



Click here

The debugger

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



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

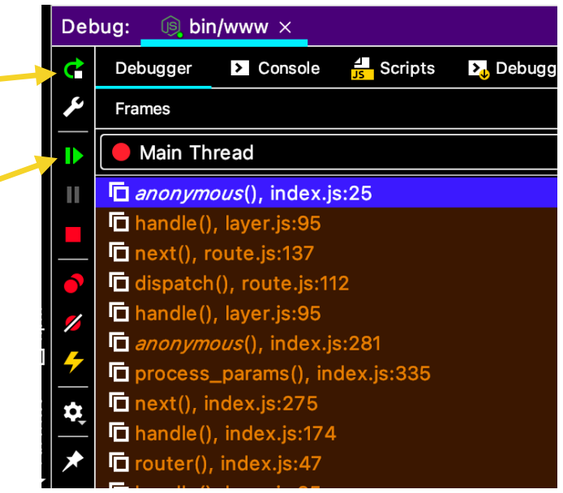
The stepper into a function - if the current line contains a function call, use this button to step into the function's code

The stepper out a function - if you entered are debugging a function, it will step out of the function and continue debugging the calling function

More controls (bottom left)

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



Solution

- comment out the following two lines

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

- the run the server again
- it should work

Video: How to debug the server side error

- To see the video, go to the Solution folder