Lab Class Week 3.a
Learning to use JSON and Ajax

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

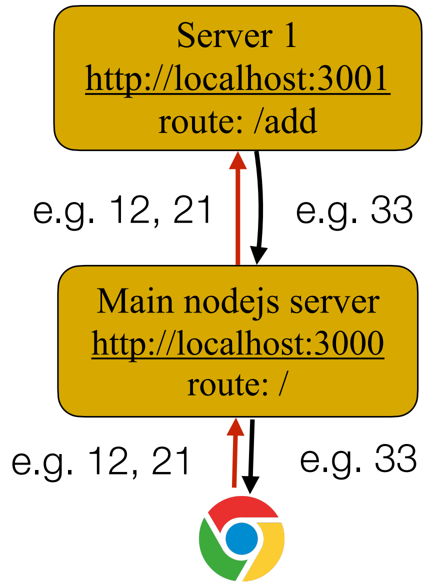
3 exercises today

- Learning to use AJAX for async browser communication
- Learning to use Axios for both client and server
- Learning to use socket.io
  - example: a chat system

Using JQuery for Ajax

- Revisit the exercise that created a constellation of servers that added two numbers from a form:
  - the client receives an EJS file with a form taking two integers
  - the client posts the numbers to the main node sever
  - the main node server receives the two numbers from the browser and sends them to the supporting server
  - The supporting server will sum the two numbers and will return the sum to the main server in json format
  - The main server will serve the EJS file again with the form but will change the title into the result of the sum

Example



Server 1
http://localhost:3001
route: /add

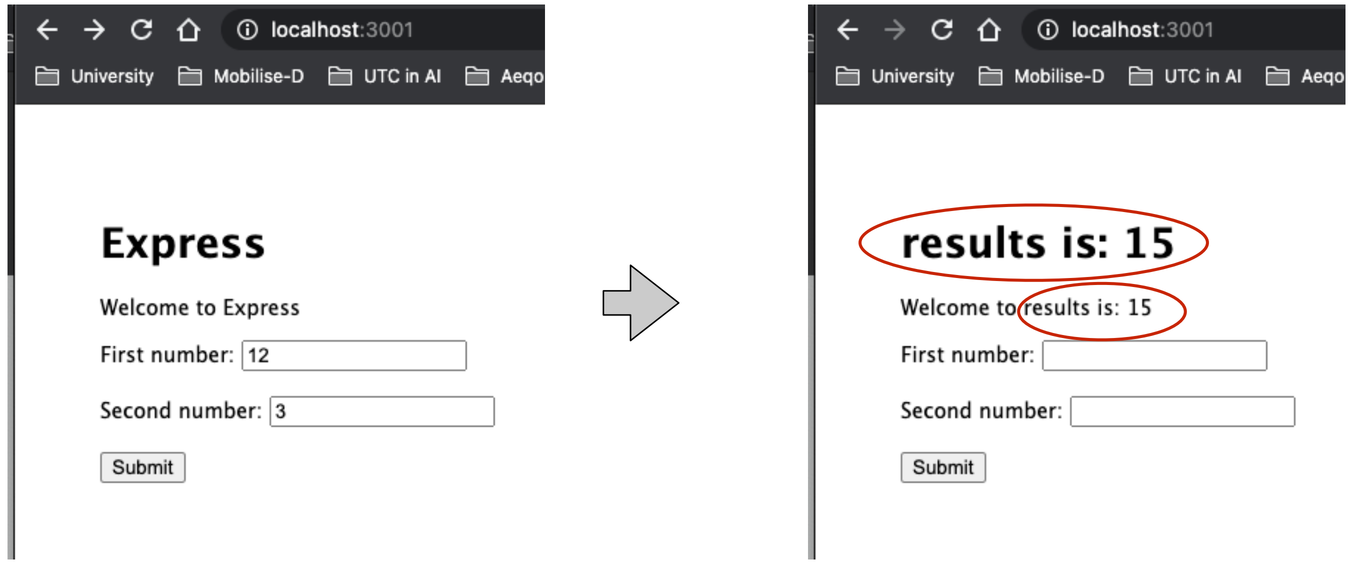
e.g. 12, 21

e.g. 33

Main nodejs server
http://localhost:3000
route: /

e.g. 12, 21

e.g. 33



Express

Welcome to Express

First number: 12

Second number: 3

Submit

results is: 15

Welcome to results is: 15

First number:

Second number:

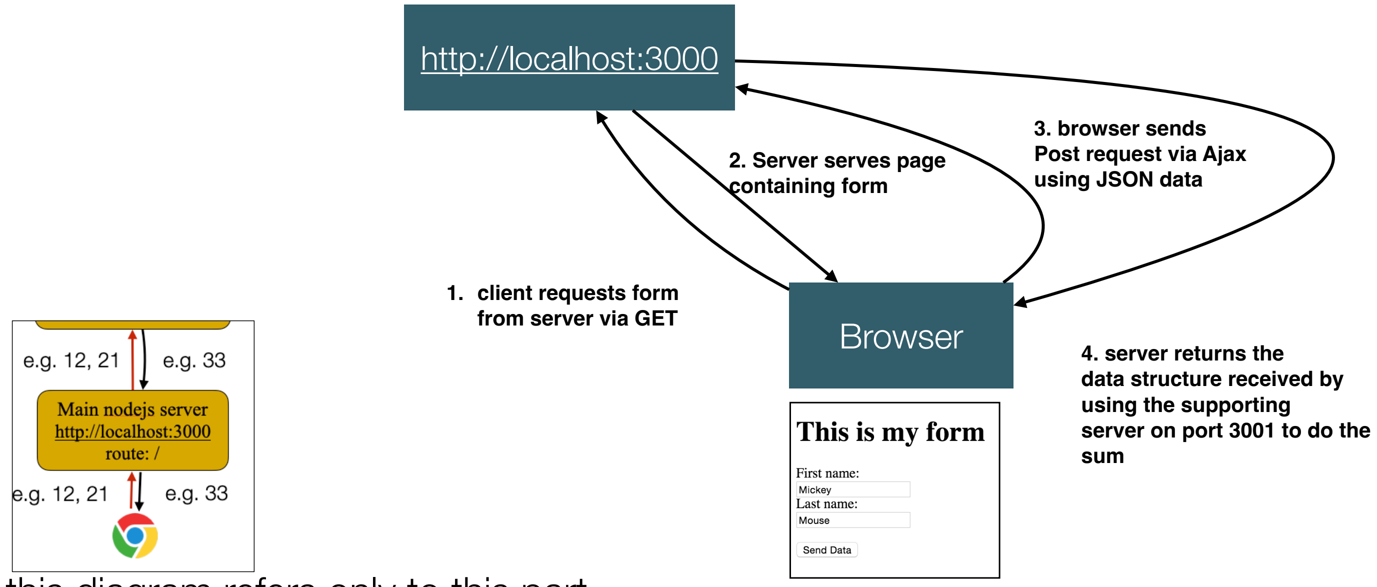
Submit

Using Axios

- In week 2 we implemented that using:
  - a standard form
  - the fetch library
- Today you are asked to implement the same exercise but:
  - the form should feed into a call via Ajax/JQuery
  - The form is provided in the starting point
- The dependent server (the one adding the two numbers) will not change
  - it is provided as a starting point

How would you do it?

Note: make sure that the server runs on port 3000 (set it in bin/www)


http://localhost:3000
2. Server serves page containing form
3. browser sends Post request via Ajax using JSON data
1. client requests form from server via GET
Browser
4. server returns the data structure received by using the supporting server on port 3001 to do the sum
e.g. 12, 21
e.g. 33
Main nodejs server
http://localhost:3000
route: /
e.g. 12, 21
e.g. 33

This is my form

First name: Mickey

Last name: Mouse

Send Data

this diagram refers only to this part

The form

- Javascript will have to intercept the form
  - remove

```
<form action="/" method="post">
```

  - and insert:
    - Jquery and javascripts loads

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script
<script src="/javascripts/index.js"></script>
```

    - and set the form as

```
<form id="xForm">
```

    - modify the body to load the init function

```
<body onload="init()">
```

- Declare a javascript file in public/javascripts;
  - call it index.js
- Insert the init function

```
function init(){
    const form = document.getElementById('xForm');
    form.onsubmit = onSubmit;
}
```

- declare the on submit function

```
/**
 * called when the submit button is pressed
 * @param event the submission event
 */
function onSubmit(event) {
    // The .serializeArray() method creates a JavaScript array of objects
    // https://api.jquery.com/serializearray/
    const formArray= $("form").serializeArray();
    const data={};
    for (let index in formArray){
        data[formArray[index].name]= formArray[index].value;
    }
    // const data = JSON.stringify($(this).serializeArray());
    sendAjaxQuery('/', data);
   // prevent the form from reloading the page (normal behaviour for forms)
    event.preventDefault()
}
```

Declare the Ajax function

```
function sendAjaxQuery(url, data) {
    $.ajax({
        url: url ,
        data: JSON.stringify(data),
        contentType: 'application/json',
        dataType: 'json',
        type: 'POST',
        success: function (dataR) {
            // no need to JSON parse the result, as we are using
            // dataType:json, so JQuery knows it and unpacks the
            // object for us before returning it
            // in order to have the object printed by alert
            // we need to JSON.stringify the object
            document.getElementById('results').innerHTML= "The result is: "+JSON.stringify(dataR);
        },
        error: function (response) {
            // the error structure we passed is in the field responseText
            // it is a string, even if we returned as JSON
            // if you want o unpack it you must do:
            // const dataR= JSON.parse(response.responseText)
            alert (response.responseText);
        }     });}
```

Exercise 3.d
Axios on both Client and Server

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

Using Axios

- This is the same exercise as 2.a but we use
  - Axios on teh client instead of JQuery
  - Axios on the server instead of fetch

Replacing JQuery

- add teh dependency on Axios in the ejs file

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

- Remove the JQuery call and insert the axios call
  - Solution in the next slide but try to work it out yourself

Solution for the client

```
function sendAjaxQuery(url, data) {
    axios.post(url , data)
        .then (function (dataR) {
            document.getElementById('results').innerHTML= "The result is: "+JSON.stringify(dataR.data);
        })
        .catch( function (response) {
            alert (response.toJSON());
        })
}
```

-

Replacing Fetch

- Axios is simpler to use than Fetch
  - remove this part

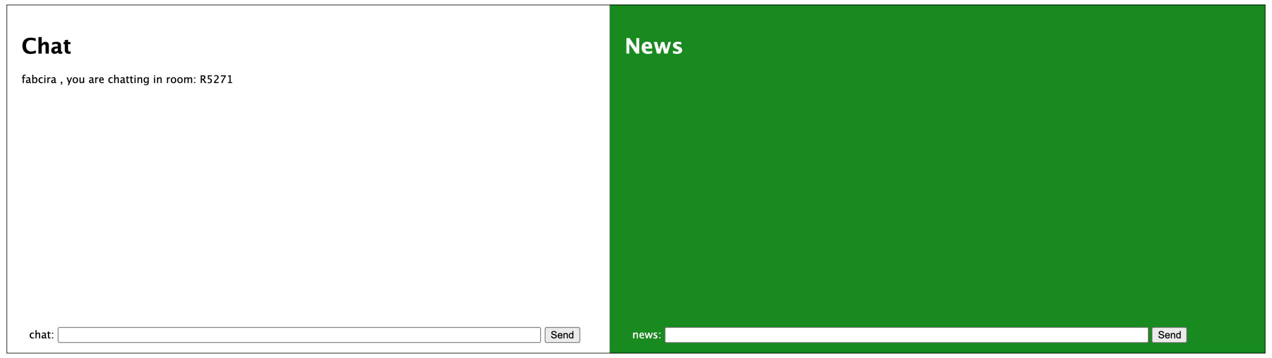
```
fetch('http://localhost:3001/add', {
    method: 'post',
    body: JSON.stringify({firstNumber: firstNo, secondNumber: secondNo}),
    headers: {'Content-Type': 'application/json'},
})
```

  - with the corresponding Axios call
- Solution in the next slide but try to work it out yourself using the lecture slides

Solution: the final route

```
router.route('/')
    .get (function(req, res) {
      res.render('index', {title: 'Express'});
    })

    .post(function  (req, res) {
        let firstNo = req.body.no1;
        let secondNo = req.body.no2;
        if (isNaN(firstNo) || isNaN(secondNo)) {
            res.setHeader('Content-Type', 'application/json');
            res.status(403).json({error: 403, reason: 'One of the numbers is invalid'});
        } else {
            axios.post('http://localhost:3000/add',
                {firstNumber: firstNo, secondNumber: secondNo})
                .then(json =>
                    res.json(json.data.result))
                .catch(err => {
                    res.setHeader('Content-Type', 'application/json');
                    res.status(403).json(err)
                })
        }
    });
```



Lab Class Week 3.b
Learning to use socket.io

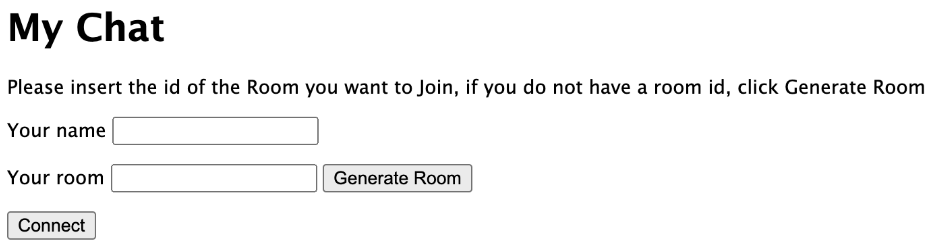
Professor Fabio Ciravegna
f.ciravegna@shef.ac.uk

Exercise 2

- In this exercise we will see how to build a chat system using socket.io
- The exercise is divided into two parts
  - Inspecting an existing chat system to understand how socket.io works
  - Adding a namespace to the chat system
    - which will require to define a new chat system similar to the one provided
- Provided:
  - the code of an implemented chat system for you to inspect and understand
  - a new version of the code above modified to support namespaces
    - to use as starting point to add the new namespace

The base chat system

- It implements a basic chat
- The interface has just one page
  - Animations will modify the page to provide user support
- Initially the user is asked for their name and the name of the room they want to join
  - if they do not have a room yet, they can generate a new name



My Chat

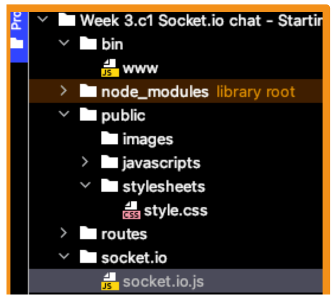
Please insert the id of the Room you want to Join, if you do not have a room id, click Generate Room

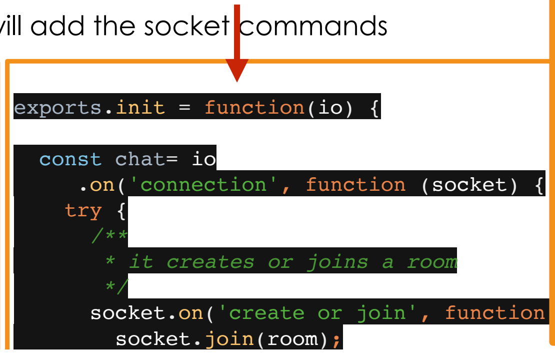
Your name

Your room

Installing socket.io

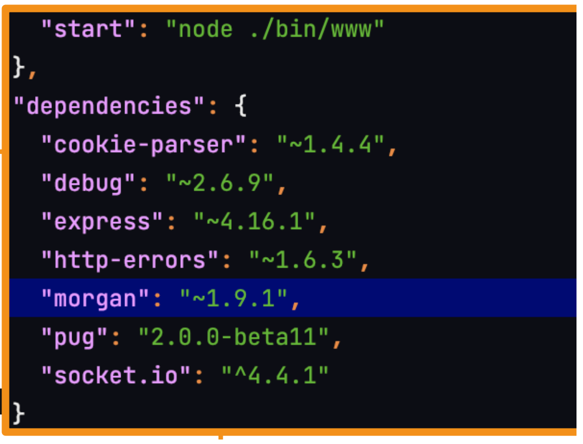
- open package.json
  - go to the bottom and start typing "socket.io", "^ (then select the top version)
    - do not forget to add the comma to the previous line!!!
    - the part "^X.X.X" will be highlighted. Right click and select "run rpm install"
- Then add folder called socket.io
  - and create a JS file called socket.io.js
    - where you will add the socket commands





```
exports.init = function(io) {

  const chat= io
      .on('connection', function (socket) {
    try {
      /**
       * it creates or joins a room
       */
        socket.on('create or join', function
          socket.join(room);
```



```
  "start": "node ./bin/www"
},
"dependencies": {
  "cookie-parser": "~1.4.4",
  "debug": "~2.6.9",
  "express": "~4.16.1",
  "http-errors": "~1.6.3",
  "morgan": "~1.9.1",
  "pug": "2.0.0-beta11",
  "socket.io": "^4.4.1"
}
```

Server side

- Declare socket.io at the end of bin/www

```
const io = require('socket.io')(server, {
  pingTimeout: 60000,
});
var socket_module = require('../socket.io/socket-io');
socket_module.init(io, app);
```

socket.io operations

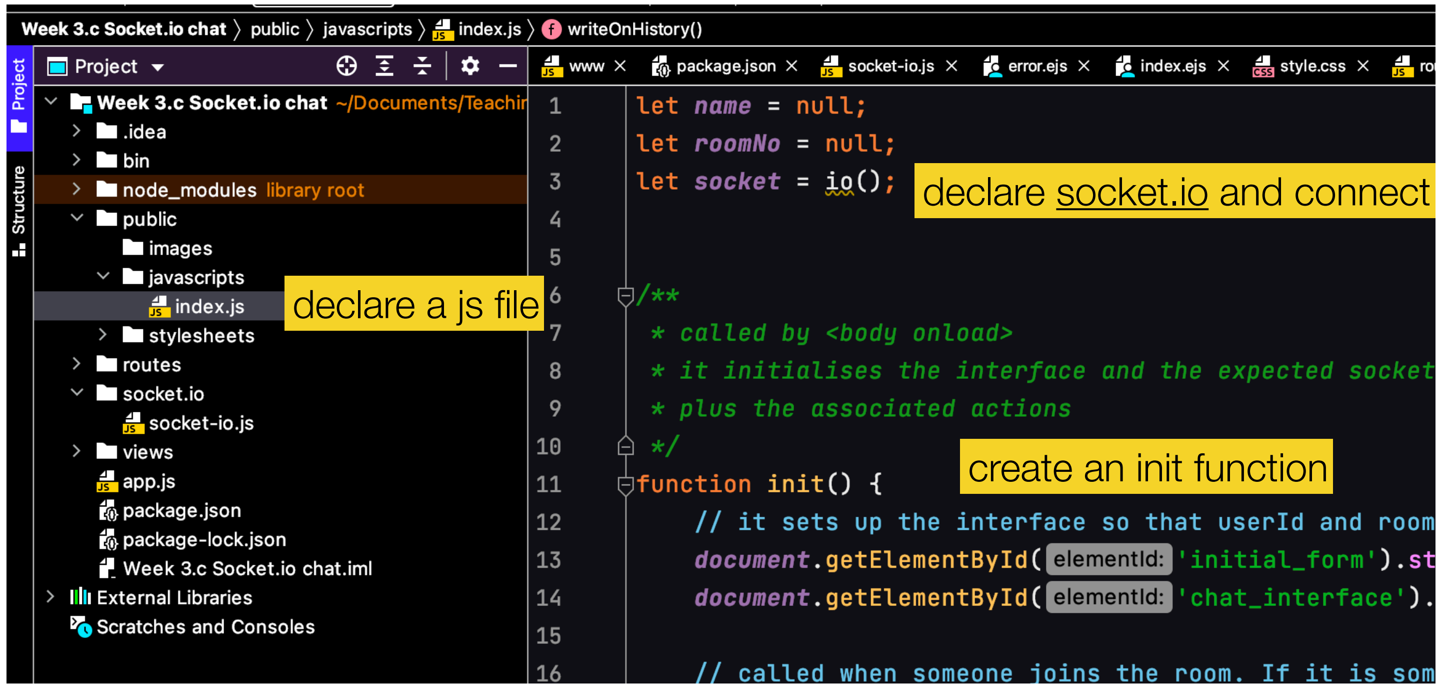
- In socket.io/socket.io.js We define two operations:
  - ‘create or join’ called when a room is joined
  - ‘joined’ called when someone joins the room
  - ‘chat’ called when someone sends a message
- Each of them receive at least a room and a user name
  - and will write to all participants in the room (including the sender)
    - using io.sockets.to(room).emit()

Declare operations in socket.io.js

operations server side

```
exports.init = function(io) {
  io.sockets.on('connection', function (socket) {
    try {
      /** it creates or joins a room  */
      socket.on('create or join', function (room, userId) {
        socket.join(room);
        io.sockets.to(room).emit('joined', room, userId);
      });
     socket.on('chat', function (room, userId, chatText) {
        io.sockets.to(room).emit('chat', room, userId, chatText);
      });
     socket.on('disconnect', function(){
        console.log('someone disconnected');
      });
    } catch (e) {  }});}
```

Client side socket.io



declare a js file

declare socket.io and connect

create an init function

```
let name = null;
let roomNo = null;
let socket = io();


/**
 * called by <body onload>
 * it initialises the interface and the expected socket
 * plus the associated actions
 */
function init() {
    // it sets up the interface so that userId and room
    document.getElementById(elementId: 'initial_form').st
    document.getElementById(elementId: 'chat_interface').

    // called when someone joins the room. If it is som
```

Joining a room

view/index.ejs

```
<form onsubmit="return false;">
    <p><label for="name"> Your name </label>
        <input type="text" id="name" name="name">
    </p>
    <p>
        <label for="roomNo"> Your room </label>
        <input type="text" id="roomNo" name="roomNo">
        <button id="roomNoGenerator" onclick="generateRoom()">Generate Room</but
    </p>
    <button id="connect" onclick="connectToRoom()">Connect</button>
</form>
```

javascripts/index.js

```
function connectToRoom() {
    roomNo = document.getElementById('roomNo').value;
    name = document.getElementById('name').value;
    if (!name) name = 'Unknown-' + Math.random();
    socket.emit('create or join', roomNo, name);
}
```

client side: javascripts/index.js

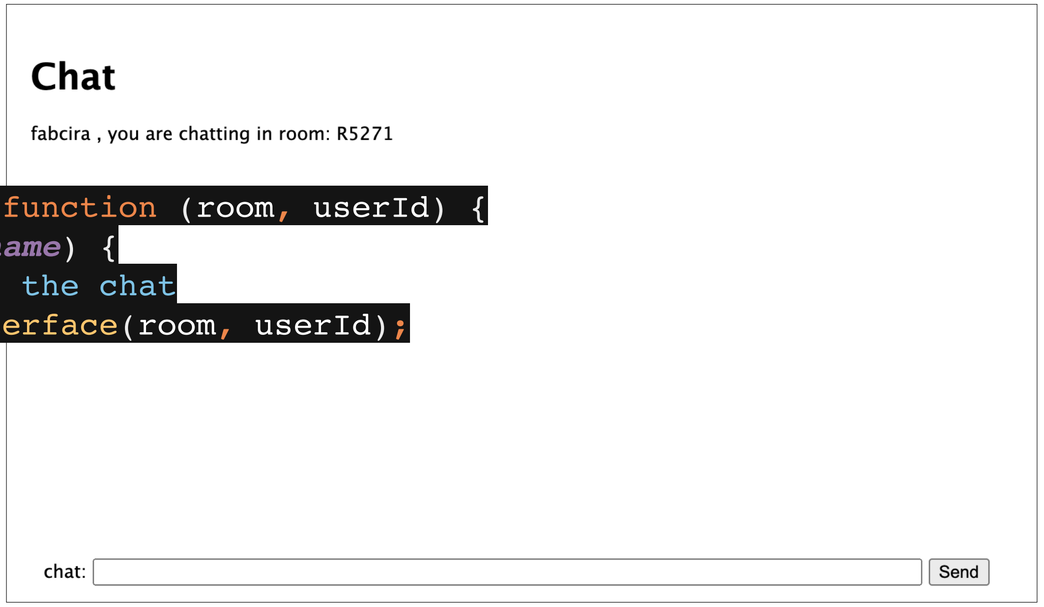
```
let name = null;
let roomNo = null;
let socket = io();
function init() {
    // it sets up the interface so that userId and room are selected
    document.getElementById('initial_form').style.display = 'block';
    document.getElementById('chat_interface').style.display = 'none';
    // called when someone joins the room. If it is someone else it notifies
    // the joining of the room in the chat
    socket.on('joined', function (room, userId) {
        if (userId === name) {
            // if we have joined, we show the chat interface
            hideLoginInterface(room, userId);
        } else {
            // notifies that someone has joined the room
            writeOnHistory('<b>'+userId+'</b>' + ' joined room ' + room);
        }});
    // called when a message is received
    socket.on('chat', function (room, userId, chatText) {
        let who = userId
        if (userId === name) who = 'Me';
        writeOnHistory('<b>' + who + ':</b> ' + chatText);});
```

receiving a joined message

receiving a chat message

The base system

- When a room is joined the chat system will appear as follows

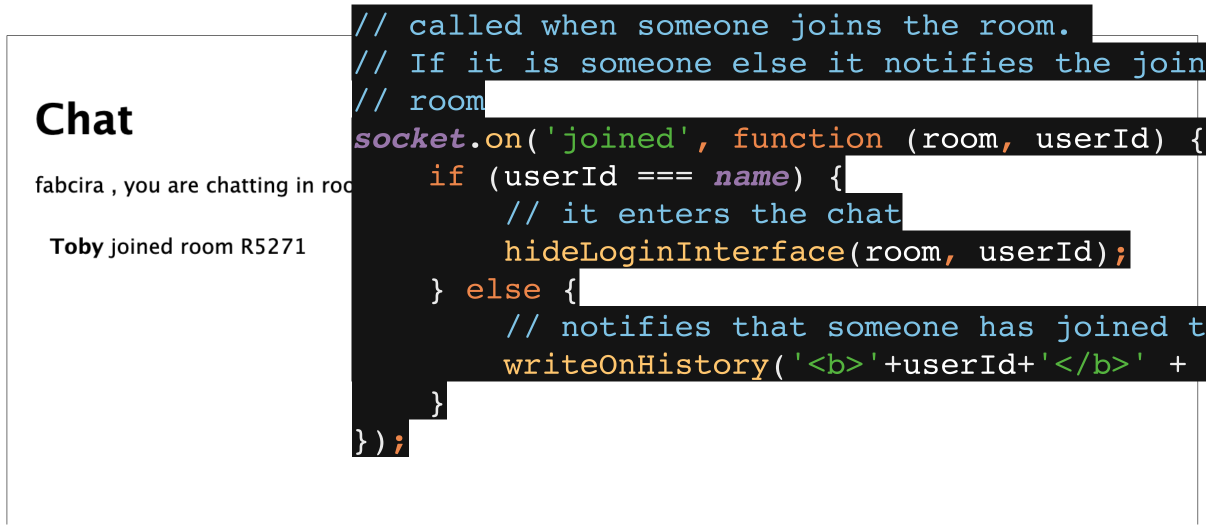


Chat

fabcira , you are chatting in room: R5271

```
socket.on('joined', function (room, userId) {
    if (userId === name) {
        // it enters the chat
        hideLoginInterface(room, userId);
…
});
```

- When someone else joins the room, the participants in the room are notified (function writeOnHistory)



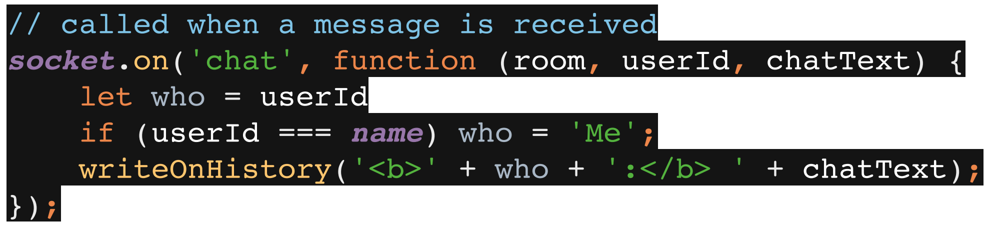
Chat

fabcira , you are chatting in roo

Toby joined room R5271

```
// called when someone joins the room.
// If it is someone else it notifies the joining of the
// room
socket.on('joined', function (room, userId) {
    if (userId === name) {
        // it enters the chat
        hideLoginInterface(room, userId);
    } else {
        // notifies that someone has joined the room
        writeOnHistory('<b>'+userId+'</b>' + ' joined roo
    }
});
```

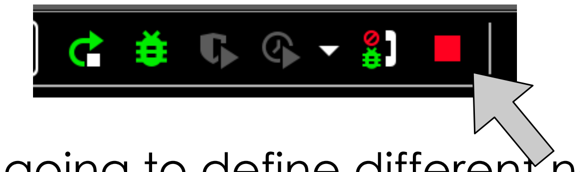
- When posting a sentence, this is shown in the history. The name of the sender is shown
  - e.g. Toby: hello!
- If it was sent by us, our name will be replaced by “Me:”
  - e.g. Me: hello!



```
// called when a message is received
socket.on('chat', function (room, userId, chatText) {
    let who = userId
    if (userId === name) who = 'Me';
    writeOnHistory('<b>' + who + ':</b> ' + chatText);
});
```

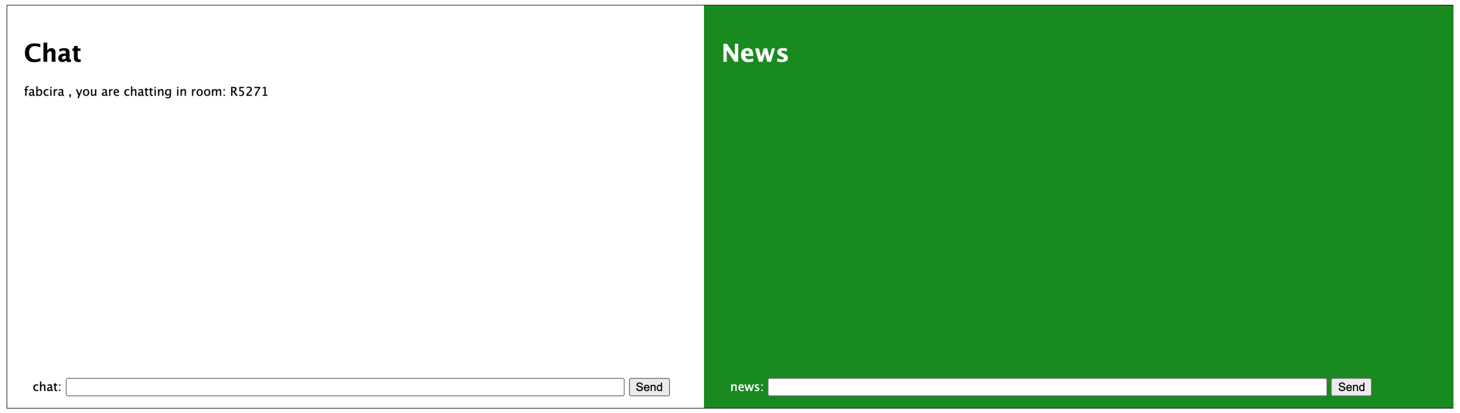
Moving to the next stage

- Make sure to understand the code
- Stop the server (red square close to the start server triangle)



- We are now going to define different name spaces where we will post on different channels
- Open the project
  - **Week 3.c1 Socket.io chat - Starting point for Solution**
  - Run the server
    - if you get a message saying that the port is already in use, you have not stopped the previous server (see above)

- the new chat has split screen with two channels corresponding to two namespaces
  - /chat and /news

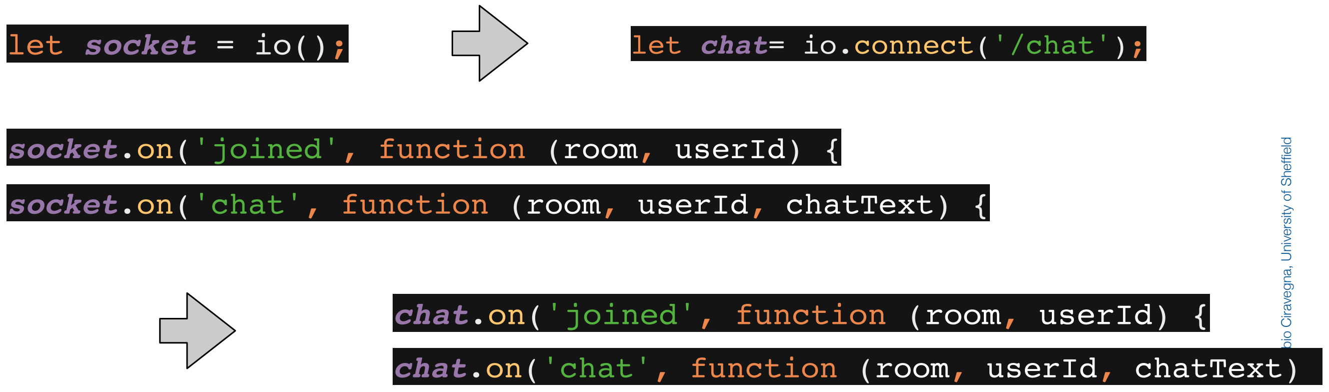


Chat

fabcira , you are chatting in room: R5271

News

- The system works as before but now the chat is executed in a new name space called /chat
  - The client side changes slightly by defining the /chat name space and use it instead of the variable socket:



```
let socket = io();
```

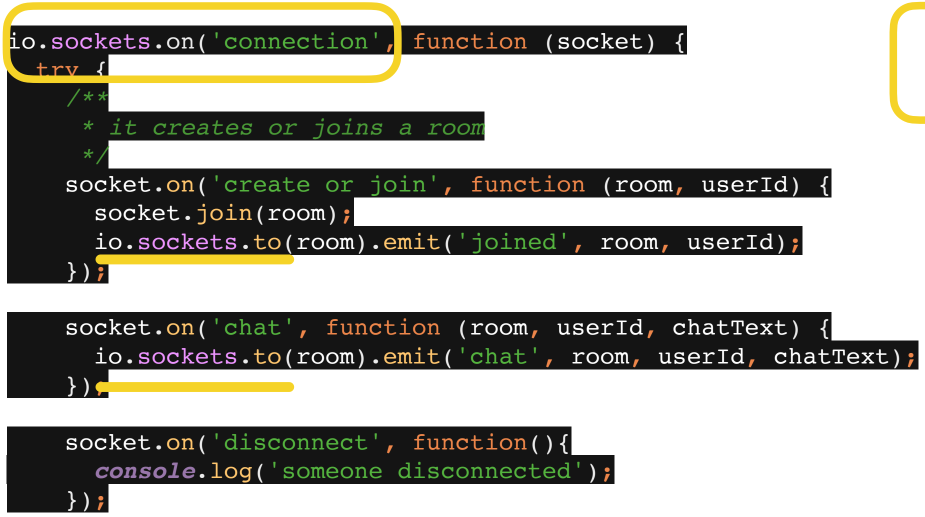
```
let chat= io.connect('/chat');
```

```
socket.on('joined', function (room, userId) {
socket.on('chat', function (room, userId, chatText) {
```

```
chat.on('joined', function (room, userId) {
chat.on('chat', function (room, userId, chatText)
```

Server side

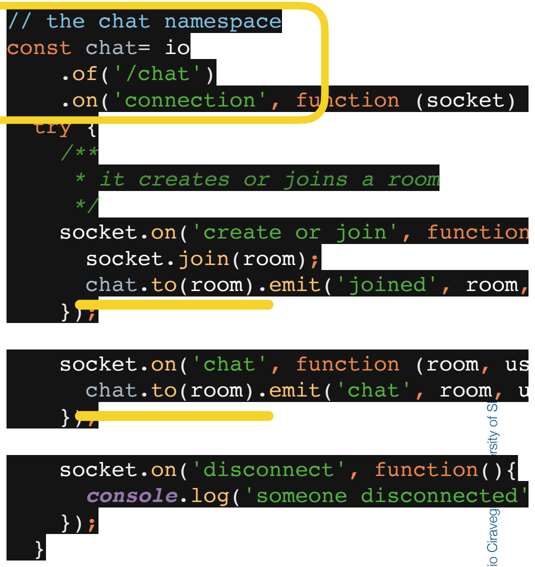
- The server side changes by declaring the same operations now defined in a name space



```
io.sockets.on('connection', function (socket) {
  try {
    /**
     * it creates or joins a room
     */
    socket.on('create or join', function (room, userId) {
      socket.join(room);
      io.sockets.to(room).emit('joined', room, userId);
    });

    socket.on('chat', function (room, userId, chatText) {
      io.sockets.to(room).emit('chat', room, userId, chatText);
    });

    socket.on('disconnect', function(){
      console.log('someone disconnected');
    });
```



```
// the chat namespace
const chat= io
    .of('/chat')
    .on('connection', function (socket)
  try {
    /**
     * it creates or joins a room
     */
    socket.on('create or join', function
      socket.join(room);
      chat.to(room).emit('joined', room,
    });

    socket.on('chat', function (room, us
      chat.to(room).emit('chat', room, u
    });

    socket.on('disconnect', function(){
      console.log('someone disconnected'
    });
  }
```

The Exercise

- Create the routes for the /news name space
- It will have the same operations as /chat both on client and on the server
  - I have left a few @todos in the code to guide you
- Hints:
  - declare namespaces and operations in socket.io.js
    - see @todo
  - declare namespaces and operations in javascripts/index.js
    - start from the function initChatSocket() (see @todo)
  - I have already defined a stub function called sendNewsText()
    - which will receive the text typed in the news form

Useful Editor tips

- To inspect where a function or variable is used or defined:
  - click on its name in the editor and hit **Command-b** on a Mac or **Control-b** on Windows
    - try it on sendNewsText now
    - if you keep hitting the key you will move between definition and uses
- To search across the entire project use Control-SHIFT-F on a Mac
  - not sure about Windows: check under Edit > Find > Find in Files
  - Try it now to search for all the occurrences of @todo

In the solution

- I have added one feature to the /news channel in the solution:
  - The news are are not copied to the author's history
    - This is to showcase the use of

```
socket.broadcast.to(room).emit(...);
```

    - which sends a message to all the participants except the originating one
    - as opposed to using

```
chat.to(room).emit(...);
```

    - which sends the message to everybody including the author
  - Note the difference: the latter
  - uses the namespace (**chat.**), while broadcast uses the socket received as parameter (**socket.**)