UNIVERSITÀ DI TORINO

Routing and Express

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

UNIVERSITÀ DI TORINO

Learning Objectives

- You will be introduced to Express as a way to create servers in an easy way
- You will learn
  - the spirit and syntax of Express
  - how to create routes in a server
    - i.e. URL paths the server responds to such as
      - http://myserver.com/index OR
      - http://myserver.com/another_route
  - how to create a project in WebStorm
  - how to access any parameter sent by a client via the body
  - how to access the data contained in the http request

Express - A Minimalistic Web Framework

- While Node.js provides the runtime environment,
  - Express is a minimal and flexible web application framework that runs on top of Node.js
  - Express simplifies the process of building robust and scalable web applications by providing a set of essential features and a clean structure for organising code
- Key Features of Express
  - Routing:
    - Express allows you to define routes for handling HTTP requests and specifying the associated logic for each route. This makes it easy to create RESTful APIs and web services.
  - Middleware:
    - Middleware functions in Express can be used for various purposes, such as authentication, logging, and error handling. They are crucial for structuring web applications.
  - Template Engines:
    - Express supports various template engines like EJS and Pug, making it simple to render dynamic HTML pages.

Node+Express

- Node.js and Express are popular choices in web development for several reasons:
  - Single Language:
    - Developers can use JavaScript for both client-side and server-side programming, reducing context-switching and streamlining development.
  - Performance:
    - Node.js's non-blocking I/O and event-driven architecture lead to high performance and responsiveness in applications.
  - Scalability:
    - Node.js and Express are well-suited for building scalable and real-time applications, making them ideal for modern web development needs.
  - Large Ecosystem:
    - The Node Package Manager (NPM) boasts a vast library of packages and modules, providing solutions for a wide range of development challenges.

UNIVERSITÀ DI TORINO

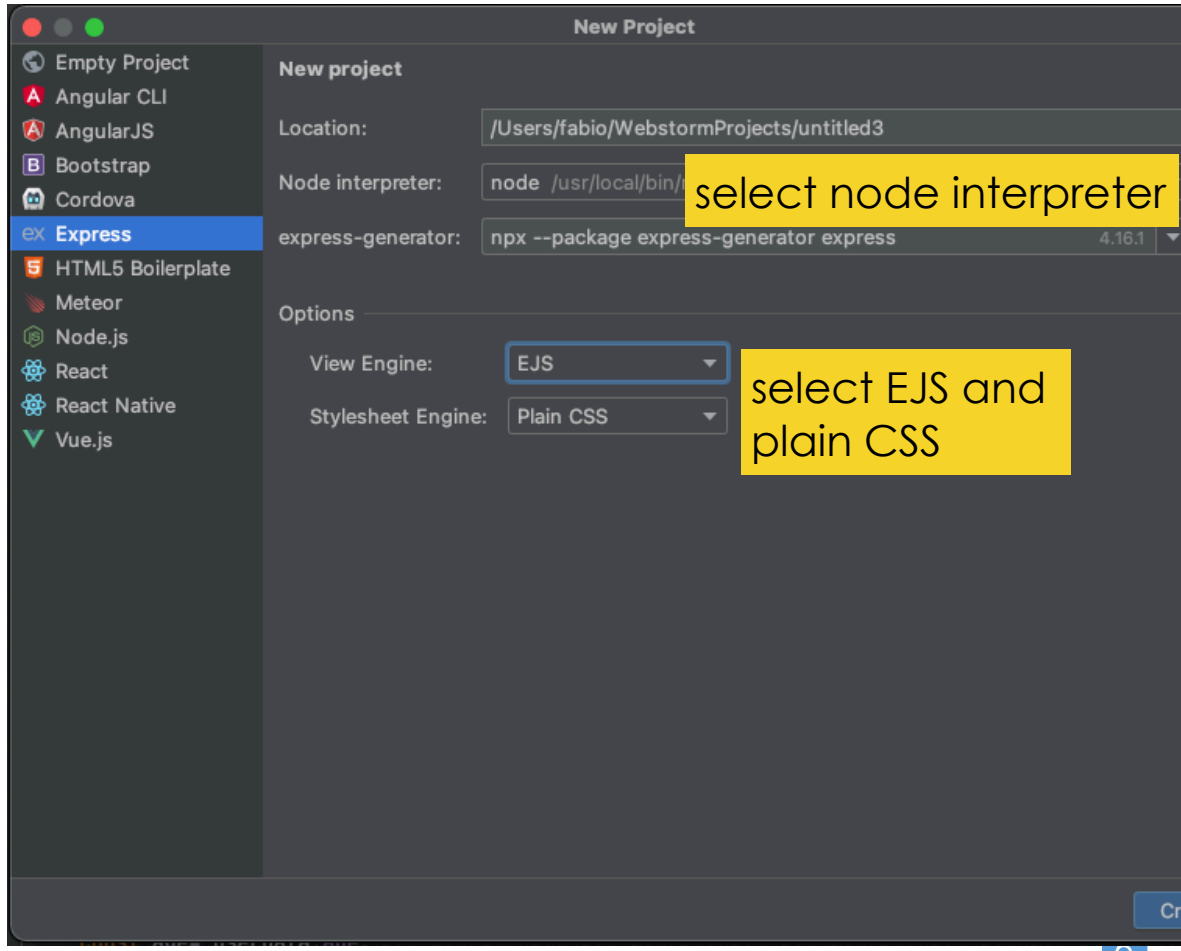
What we will see

- How to create a server using Express and NodeJ
- How to declare routes in Express
  - e.g. to make sure that your server recognises a request to the address http://localhost:3000/index
- How to respond to a client
- Express in IntelliJ for your lab

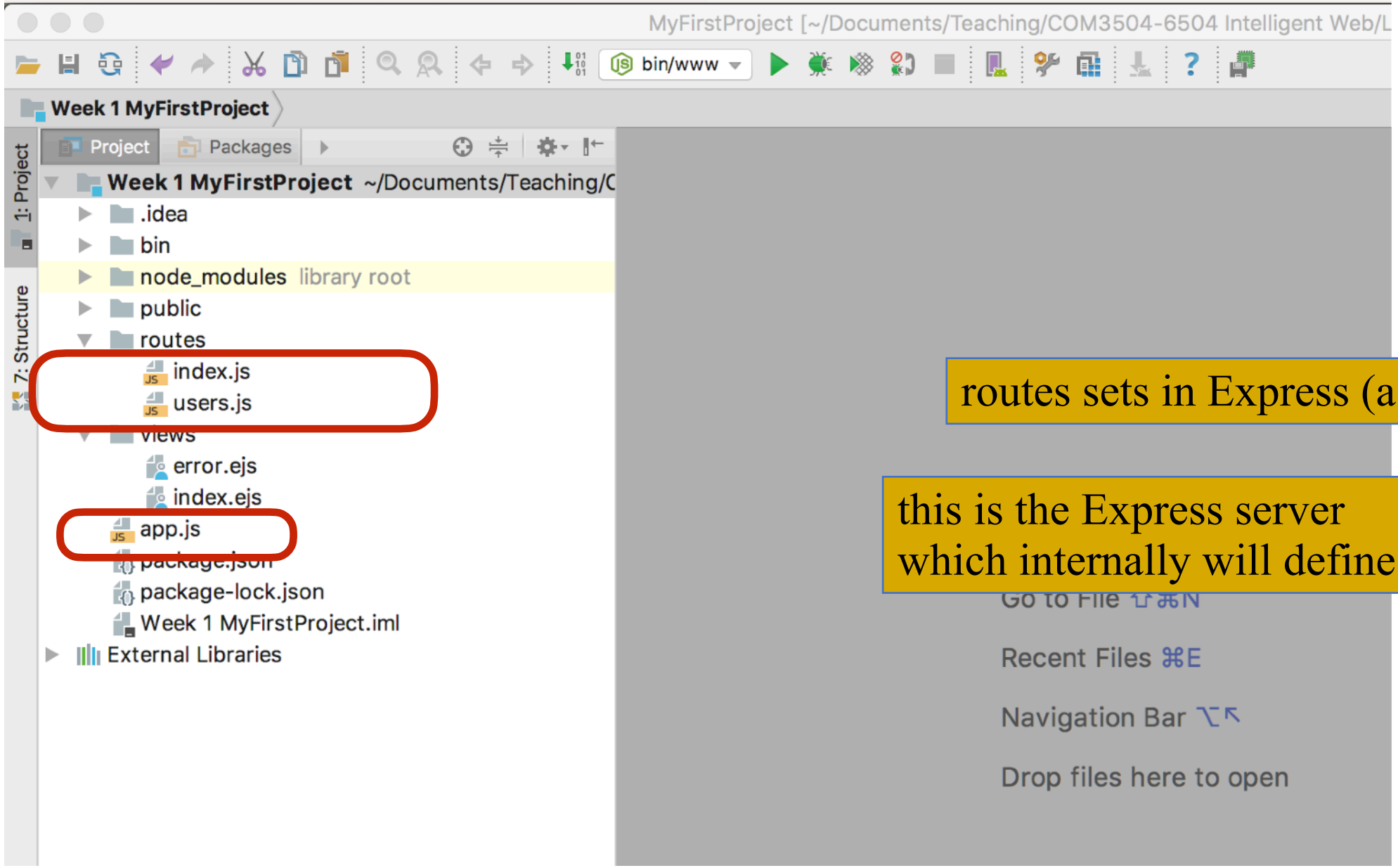
UNIVERSITÀ DI TORINO

How to create a project in Webstorm

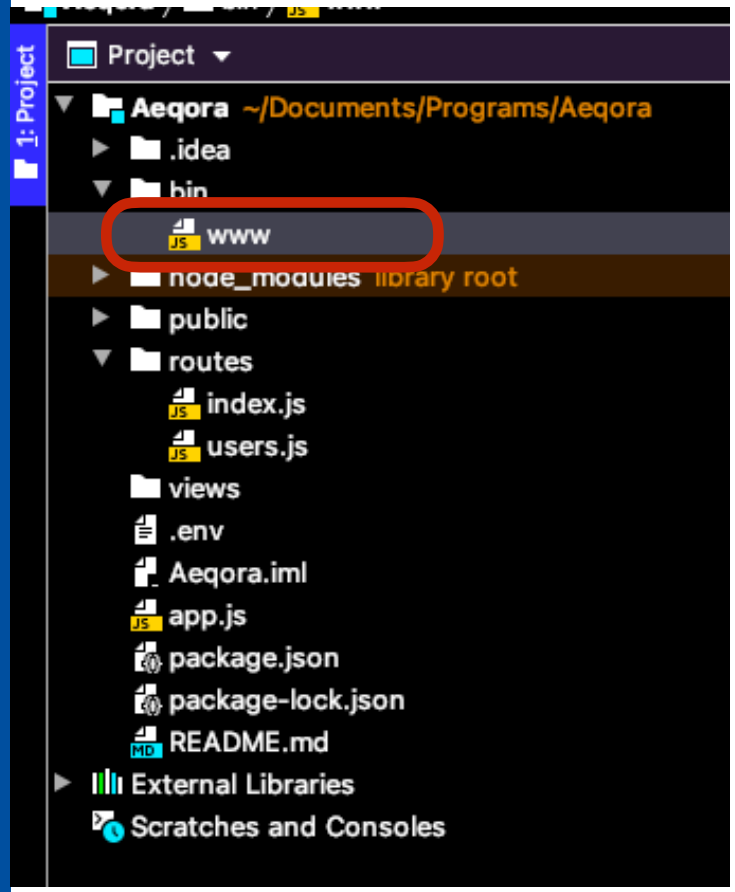
- WebStorm provides an excellent plugin for node projects
- To create a project:
  - file>new>project
  - if you do not see Express then you need to install node
  - go to plugin -> install under Settings/Preference Plugins



The server side in Express



The server and the port




```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var https = require('http');
var fs = require('fs-extra');


/**
 * Get port from environment and store in Express.
 */

var port = normalizePort( val: process.env.PORT || '8095');
app.set('port', port);

 /*
 Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
```

sets the server port

it creates the server

UNIVERSITÀ DI TORINO

Routing

- Routing refers to determining how an application responds to a client request to a particular endpoint,
  - which is a URI (or path) and
  - a specific HTTP request method (GET, POST, and so on)
- Each route can have one or more handler functions

Nodejs: Get Routing to /

```
var http = require('http');
var url= require('url');
var server = http.createServer(function (request, response) {
    var pathname = url.parse(req.url).pathname;
    if ((pathname=='/')&& (request.method == 'GET')) {
    response.end('Hello World');
    }});
server.listen(3000);
```

The app starts a server and listens on port 3000 for connection.
It will respond with "Hello World!" for requests to the homepage.
For every other path, it will respond with a 404 Not Found.

this is the standard node.js. We will not use it. We will use Express

UNIVERSITÀ DI TORINO

Express

http://expressjs.com/

- Node.js is great but most of its functions are rather verbose
- Express
  - A minimal and flexible node.js web application framework with a robust set of features for web applications
  - With a myriad of HTTP utility methods and middleware at your disposal
    - Creating a robust API is quick and easy
  - A thin layer of fundamental web application features, without obscuring Node features

Routes

- In Express, routes are used to define the behaviour of your application
  - when a specific URL is requested by a client
  - Routes are responsible for mapping URLs to functions that handle the incoming HTTP requests
    - and generate the appropriate responses.
  - They help organise your code, making it easier to manage and maintain your web application.

the route

http://localhost:3000/index

the route is
- **/:** the root directory
- **index**: the command or file

Route Anatomy: HTTP Method, Path, and Callback

- A route in Express consists of three main components:
- HTTP Method:
  - This specifies the type of HTTP request that the route should respond to. Common HTTP methods include GET, POST, PUT, DELETE, and others. Each method is associated with a specific type of action (e.g., retrieving data with GET or creating data with POST).
- Path:
  - The path is a string representing the URL route that the route should match. It can contain static parts (e.g., "/products") and dynamic parts (e.g., "/products/:id"), where placeholders like :id capture values from the URL for use in the route's callback function.
- Callback Function:
  - The callback function, also known as a route handler, is executed when a matching route is accessed.
    - It takes two arguments:
      - the request (req) object, which contains information about the incoming request, and
      - the response (res) object, which is used to send the response back to the client.
    - The callback function defines the logic for processing the request and generating the response.

Declaring a Server In Express

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('I received a request on / !')
});


app.get('/hello', function (req, res) {
  res.send(`I received a request on /hello!`)
})
```

/ is a route

/hello is a route

Every time we receive a GET for "/" or for "/hello", then send back the appropriate string
the client will have requested something like
http://your.server.address:3000/hello
we will work on a server local to your computer, so the address is
http://localhost:3000/hello

Another example

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

// Define a route for the homepage (GET request)
app.get('/', (req, res) => {
  res.send('Welcome to the homepage!');
});

// Define a route for a product detail page with a dynamic parameter (GET request)
app.get('/products/:id', (req, res) => {
  const productId = req.params.id; // Access the dynamic parameter
  res.send(`Product ID: ${productId}`);
});

// Define a route for handling form submissions (POST request)
app.post('/submit', (req, res) => {
  // Process the submitted data
  res.send('Form submitted successfully!');
});

// Start the Express server
app.listen(3000, () => {
  console.log('Server is listening on port 3000');
});
```

UNIVERSITÀ DI TORINO

Route Order

- The order in which routes are defined matters in Express
- Routes are evaluated from top to bottom,
  - and the first matching route is executed
- Therefore, more specific routes (e.g., routes with dynamic parameters)
  - should be defined after more general ones to avoid conflicts

UNIVERSITÀ DI TORINO

Routing in Express

- Route definition:

```
app.METHOD(PATH, CALLBACK)
```

  - app is an instance of express retrieved using *express()*
  - METHOD is an HTTP request method (POST, GET)
  - PATH is a path on the server,
  - HANDLER is the callback function executed when the route is matched

Routing Examples

METHOD

```
// respond with "Hello World!" on the homepage
app.get('/', function (req, res) {
  res.send('Hello World!');
})

// accept POST request on the homepage
app.post('/', function (req, res) {
  res.send('Got a POST request');
})
```

PATH

CALLBACK or HANDLER

- app is an instance of express
- is an HTTP request method (POST, GET)
- PATH is a path on the server,
- HANDLER is the callback function executed when the route

WebStorm

```
/* GET home page. */
router.get( path: '/index',  handlers: function(req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<Re
  res.render( view: 'index',  options: { title: 'My Form' });
});

router.post( path: '/index', character.getAge);


/* GET home page. */
router.get( path: '/insert',  handlers: function(req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<R
  res.render( view: 'insert',  options: { title: 'My Form' });
});

router.post( path: '/insert', character.insert);
```

Note the helpful parameter descriptions

Route Paths

- Route paths define the endpoints at which requests can be made to.
  - e.g. '/', or '/users/' …
- In express they can be:
  - strings
  - string patterns
  - regular expressions.
- Note!
  - Query strings are not a part of the route path.
    - In http://localhost/index.html**?index=34**
    - **?index=34** is **not** part of the route path

UNIVERSITÀ DI TORINO

Examples

```
// with match request to the root
app.get('/', function (req, res) {
  res.send('root requested')
})

// will match requests to /about
app.get('/about', function (req, res) {
  res.send('about requested')
})

// will match request to /random.html
app.get('/random.html', function (req, res) {
  res.send('random.html requested')
})
```

Defining routes

- Route handlers for a single route path can be created using

```
app.route('/book')

  .get(function(req, res) {
    res.send('Get a random book');
  })

  .post(function(req, res) {
    res.send('Add a book');
  })
```

- This reduce redundancy and typos.

app.all

- Special routing method not derived from any HTTP method
  - Express will compile it into two separated requests: GET+POST with identical callback
- Used for representing all request methods.

```
// respond with "Hello World!" to all type of
// requests (post, get, etc.) on the homepage

app.all('/', function (req, res, next) {
  res.send('Got a request');
})
```

Getting the request headers

http://nodejs.org/api/http.html

- Getting the request headers (e.g. user-agent)

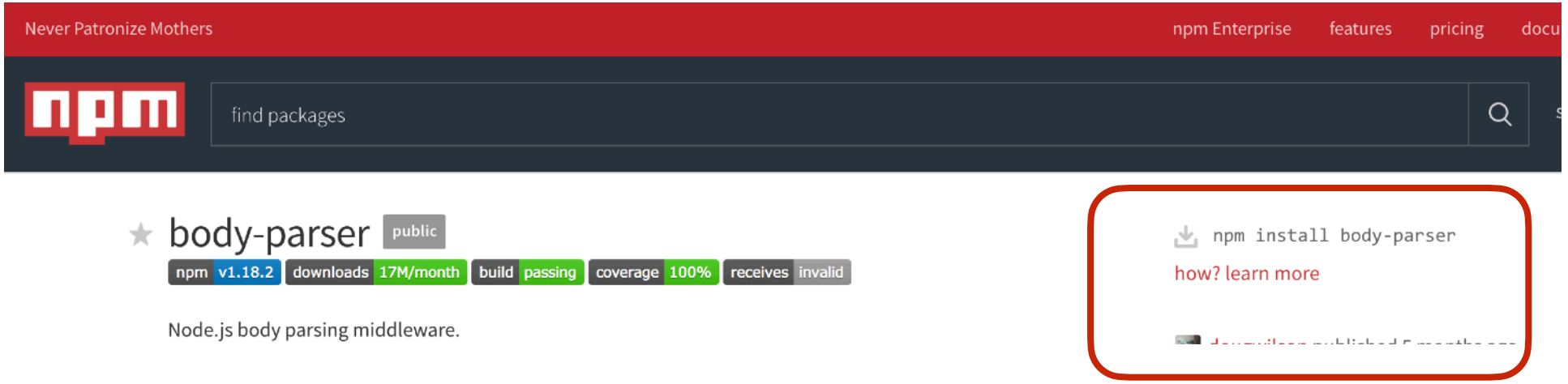
```
HTTP message headers are represented by an object like this:

{ 'content-length': '123',
  'content-type': 'text/plain',
  'connection': 'keep-alive',
  'host': 'mysite.com',
  'accept': '*/*' }
```

- a value is requested as
  - **request.headers**['user-agent']
- this enables to access the https protocol parameters (and hence e.g. to the request metadata - e.g. to check if the request comes from a Chrome browser)
- please note!
  - all fields are lowercase, values are not modified

Body-parser

- It enable accessing the parameters provided in the request



NPM modules always give you the command to use to install the module (use that on a command line)

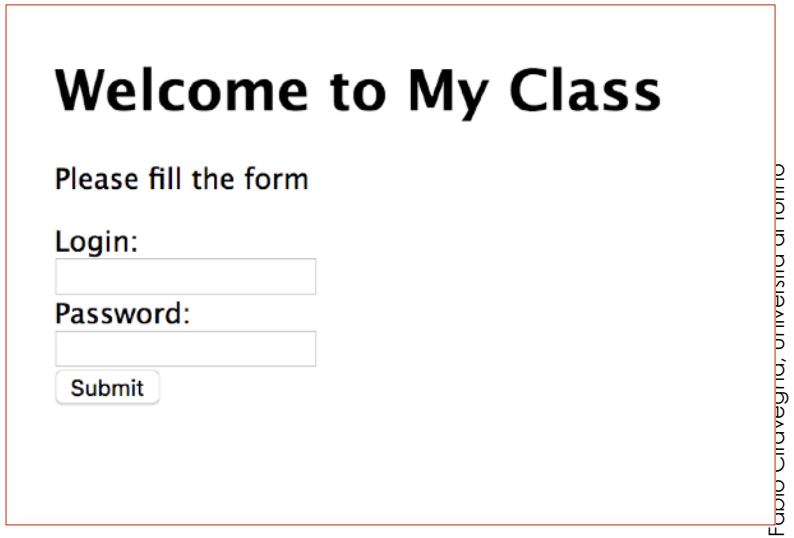
How to use body-parser

To respond to a form pointing to /index (see the next lab)

```
…

app.post('/index', function(req, res, next) {
    var login= req.body.login;
    var password= req.body.password;

…
}
```



the form fields become fields in the request.**body** field
In general, any data sent from the client will be inserted into the request's body field

We will see what **router.post** is. For now just check how to access the parameters **login and password** from a form

GET request: Accessing the parameters

// suppose GET request, ex: http://localhost/?name=Tom

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

app.get('/', function(req, res) {
      var queryData = req.body;
     res.writeHead(200, {"Content-Type": "text/plain"});
       // if parameter is provided
      if (queryData.name) {
        response.end('Hello ' + queryData.name + '\n');
       } else {
        response.end("Hello World\n");
      }
  }
});
```

Returning results

- this returns code 200 (ok) to the client
  - by writing it into the header of the HTTPS response

```
result.writeHead(200, {'Content-Type': 'text/plain'});
```

- it sends back the strings to the client
- it closes the communication

```
result.end('Hello World\n');
```

- It is equivalent to declaring .writeHead(200) + .end(…)

```
result.send('Hello World\n');
```

returning results must be the last instruction in each route. No further instructions are allowed after then. If you forget to return the results to the client, the server hangs and stops responding to all users

Defining global variables

either lasting until you turn off the server

or lasting for the duration of the specific client request

app.locals

You can set and access them at any time in the server or its routes

app.locals

The `app.locals` object is a JavaScript object, and its properties are local variables within the application.

```
app.locals.title
// => 'My App'

app.locals.email
// => 'me@myapp.com'
```

Once set, the value of `app.locals` properties persist throughout the life of the application, in contrast with res.locals properties that are valid only for the lifetime of the request.

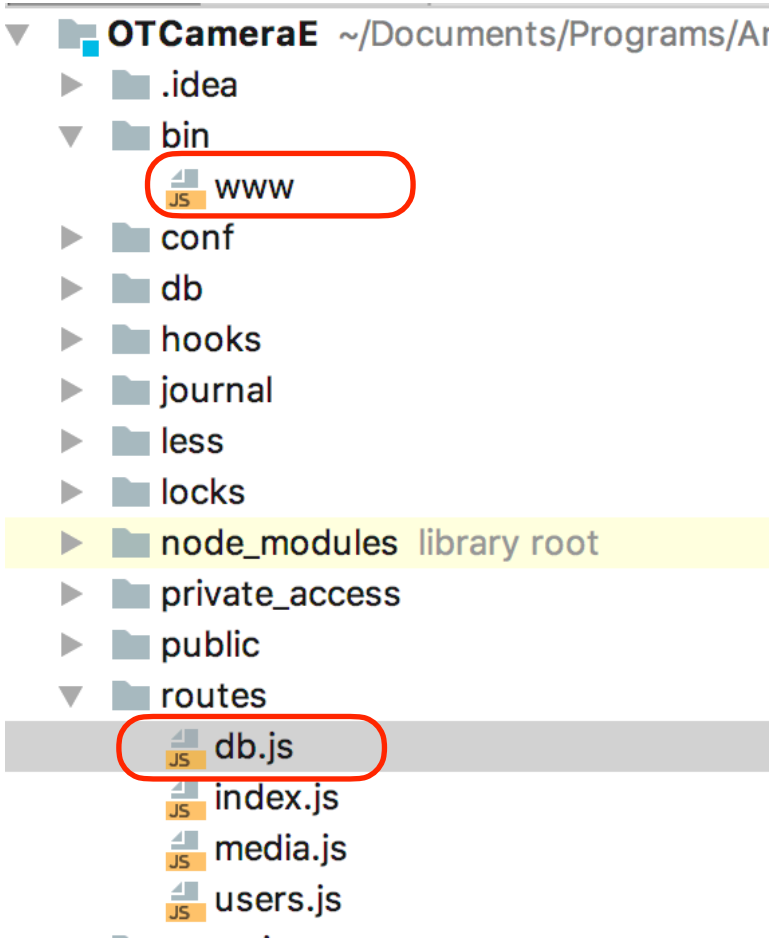
You can accesss local variables in templates rendered within the application. This is useful for providing helper functions to templates, as well as app-level data. Note, however, that you cannot access local variables in middleware.

```
app.locals.title = 'My App';
app.locals.strftime = require('strftime');
app.locals.email = 'me@myapp.com';
```

- **app.locals** variables persist for the lifetime of the server
  - so they are **shared by all user requests**
- **res.locals** for the lifetime of the request
  - so the value is valid **only for the specific user request** — remember http is memoryless

why you want to use it?

- app.locals is typically used to pass data across routes and from the app to the routes



db route

```
router.post('/find_clients_from_remote', function (req, res) {
    console.log('find_clients_from_remote: ');
    var body = req.body;
    Database.getClient(body, function (err, dataX) {
        if (err || dataX.length == 0) {
            res.writeHead(400);
            res.end(err || 'user not found');
        } else {
            dataX = dataX[0];
            console.log('found client ' + JSON.stringify(dataX));
            var cidX = dataX._id.toString();
            console.log('cidx ' + cidX);

            // if cid is inexistent it means it is not available
            if (isUndefined(cidX)) {
                console.log('cidx (NOT passed test) ' + cidX);
                res.writeHead(400);
                err = {status: 400, statusText: 'user not found'};
                res.end(JSON.stringify(err));
                return;
            }
            console.log('cidx (Passed test) ' + cidX);
            var npo = req.app.locals.getNewRoom();
            console.log(JSON.stringify(npo));
            var data = {
                cid: cidX.
```

ignore this part, it is irrelevant

app locals is retrieved using the request object

bin/www

```
function getNewRoom() {
    var ctrlRoomId = Math.floor((Math.random() * 10000) + 1);
    var useridCitizen = Math.floor((Math.random() * 10000) + 1);
    var roomid = Math.floor((Math.random() * 100000) + 1);
    var password = Math.floor((Math.random() * 100000) + 1);
    app.locals.allowedCtrlRooms["CR" + ctrlRoomId] = "R" + roomid;
    app.locals.allowedCitRooms["CT" + useridCitizen] = "R" + roomid;
    app.locals.validPorts["R" + roomid] = "P" + password;
    return ([roomid, ctrlRoomId, useridCitizen, password]);
}

app.locals.getNewRoom= getNewRoom;
```

ignore this part, it is irrelevant

note: it is not assigned using the request object

Request parameters

http://expressjs.com/4x/api.html

req.hostname

Contains the hostname from the "Host" HTTP header.

```
// Host: "example.com:3000"
req.hostname
// => "example.com"
```

req.ip

The remote IP address of the request.

If the `trust proxy` is setting enabled, it is the upstream address; see Express behind proxies for more information.

```
req.ip
// => "127.0.0.1"
```

These are used to access the HTTP request

UNIVERSITÀ DI TORINO

req.path

Contains the path part of the request URL.

```
// example.com/users?sort=desc
req.path
// => "/users"
```

req.protocol

The request protocol string, "http" or "https" when requested with TLS. When the "trust proxy" setting trusts the socket address, the value of the "X-Forwarded-Proto" header ("http" or "https") field will be trusted and used if present.

```
req.protocol
// => "http"
```

req.query

An object containing a property for each query string parameter in the route. If there is no query string, it is the empty object, `{}`.

```
// GET /search?q=tobi+ferret
req.query.q
// => "tobi ferret"

// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse
req.query.order
// => "desc"
```

UNIVERSITÀ DI TORINO

req.accepts(types)

Checks if the specified content types are acceptable, based on the request's `Accept` HTTP header field. The method returns the best match, or if none of the specified content types is acceptable, returns `undefined` (in which case, the application should respond with `406 "Not Acceptable"`).

The `type` value may be a single MIME type string (such as "application/json"), an extension name such as "json", a comma-delimited list, or an array. For a list or array, the method returns the ***best*** match (if any).

```
// Accept: text/html
req.accepts('html');
// => "html"

// Accept: text/*, application/json
req.accepts('html');
// => "html"
req.accepts('text/html');
// => "text/html"
req.accepts('json, text');
// => "json"
req.accepts('application/json');
// => "application/json"

// Accept: text/*, application/json
req.accepts('image/png');
req.accepts('png');
// => undefined

// Accept: text/*;q=.5, application/json
req.accepts(['html', 'json']);
req.accepts('html, json');
// => "json"
```

UNIVERSITÀ DI TORINO

req.get(field)

Returns the specified HTTP request header field (case-insensitive match). The `Referrer` and `Referer` fields are interchangeable.

```
req.get('Content-Type');
// => "text/plain"

req.get('content-type');
// => "text/plain"

req.get('Something');
// => undefined
```

Aliased as `req.header(field)`.

req.is(type)

Returns `true` if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the `type` parameter. Returns `false` otherwise.

```
// With Content-Type: text/html; charset=utf-8
req.is('html');
req.is('text/html');
req.is('text/*');
// => true

// When Content-Type is application/json
req.is('json');
req.is('application/json');
req.is('application/*');
// => true
```

Getting parameters (POST)

http://stackoverflow.com/questions/5710358/how-to-get-pos

```
.. in your route

// assuming POST: name=foo&color=red      <-- URL encoding
// OR POST: {"name":"foo","color":"red"}  <-- JSON encoding

app.post('/test-page', function(req, res) {
    var name = req.body.name,
        color = req.body.color;
});
```

Response object

res.attachment([filename])

Sets the HTTP response `Content-Disposition` header field to "attachment". If a `filename` is given, then it sets the Content-Type based on the extension name via `res.type()`, and sets the `Content-Disposition` "filename=" parameter.

```
res.attachment();
// Content-Disposition: attachment

res.attachment('path/to/logo.png');
// Content-Disposition: attachment; filename="logo.png"
// Content-Type: image/png
```

res.end([data] [, encoding])

Ends the response process. Inherited from Node's http.ServerResponse.

Use to quickly end the response without any data. If you need to respond with data, instead use methods such as res.send() and res.json().

```
res.end();
res.status(404).end();
```

res.json([body])

Sends a JSON response. This method is identical to `res.send()` with an object or array as the parameter. However, you can use it to convert other values to JSON, such as `null`, and `undefined`. (although these are technically not valid JSON).

```
res.json(null)
res.json({ user: 'tobi' })
res.status(500).json({ error: 'message' })
```

Response methods

The methods on the response object (`res`) in the following table can send a response to the client and terminate the request response cycle. If none of them is called from a route handler, the client request will be left hanging.

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

UNIVERSITÀ DI TORINO

Response: redirection

res.redirect([status,] path)

> Express passes the specified URL string as-is to the browser in the `Location` header, without any validation or manipulation, except in case of `back`.
>
> Browsers take the responsibility of deriving the intended URL from the current URL or the referring URL, and the URL specified in the `Location` header; and redirect the user accordingly.

Redirects to the URL dervied from the specified `path`, with specified HTTP status code `status`. If you don't specify `status`, the status code defaults to "302 "Found".

```
res.redirect('/foo/bar');
res.redirect('http://example.com');
res.redirect(301, 'http://example.com');
res.redirect('../login');
```

Redirects can be a fully-qualified URL for redirecting to a different site:

```
res.redirect('http://google.com');
```

Redirects can be relative to the root of the host name. For example, if the application is on `http://example.com/admin/post/new`, the following would redirect to the URL `http://example.com/admin`:

```
res.redirect('/admin');
```

A `back` redirection redirects the request back to the referer, defaulting to `/` when the referer is missing.

```
res.redirect('back');
```

note! useful for login requests!!!

UNIVERSITÀ DI TORINO

res.send([body])

Sends the HTTP response.

The `body` parameter can be a `Buffer` object, a `String`, an object, or an `Array`. For example:

```
res.send(new Buffer('whoop'));
res.send({ some: 'json' });
res.send('<p>some html</p>');
res.status(404).send('Sorry, we cannot find that!');
res.status(500).send({ error: 'something blew up' });
```

This method performs many useful tasks for simple non-streaming responses: For example, it automatically assigns the `Content-Length` HTTP response header field (unless previously defined) and provides automatic HEAD and HTTP cache freshness support.

When the parameter is a `Buffer` object, the method sets the `Content-Type` response header field to "application/octet-stream", unless previously defined as shown below:

```
res.set('Content-Type', 'text/html');
res.send(new Buffer('<p>some html</p>'));
```

When the parameter is a `String`, the method sets the `Content-Type` to "text/html":

```
res.send('<p>some html</p>');
```

When the parameter is an `Array` or `Object`, Express responds with the JSON representation:

```
res.send({ user: 'tobi' });
res.send([1,2,3]);
```

UNIVERSITÀ DI TORINO

res.sendStatus(statusCode)

Set the response HTTP status code to `statusCode` and send its string representation as the response body.

```
res.sendStatus(200); // equivalent to res.status(200).send('OK')
res.sendStatus(403); // equivalent to res.status(403).send('Forbidden')
res.sendStatus(404); // equivalent to res.status(404).send('Not Found')
res.sendStatus(500); // equivalent to res.status(500).send('Internal Server Error')
```

If an unsupported status code is specified, the HTTP status is still set to `statusCode` and the string version of the code is sent as the response body.

```
res.sendStatus(2000); // equivalent to res.status(2000).send('2000')
```

UNIVERSITÀ DI TORINO

res.set(field [, value])

Sets the response's HTTP header `field` to `value`. To set multiple fields at once, pass an object as the parameter.

```
res.set('Content-Type', 'text/plain');

res.set({
  'Content-Type': 'text/plain',
  'Content-Length': '123',
  'ETag': '12345'
})
```

Aliased as `res.header(field [, value])`.

res.status(code)

Use this method to set the HTTP status for the response. It is a chainable alias of Node's response.statusCode.

```
res.status(403).end();
res.status(400).send('Bad Request');
res.status(404).sendFile('/absolute/path/to/404.png');
```

res.type(type)

Sets the `Content-Type` HTTP header to the MIME type as determined by mime.lookup() for the specified `type`. If `type` contains the "/" character, then it sets the `Content-Type` to `type`.

```
res.type('.html');              // => 'text/html'
res.type('html');               // => 'text/html'
res.type('json');               // => 'application/json'
res.type('application/json');   // => 'application/json'
res.type('png');                // => image/png:
```

UNIVERSITÀ DI TORINO

Questions?

© Prof. Fabio Cravegna, Università di Torino