

Web Programming

Lecture 10 – Server Side

Shamsa Abid

Creating RESTful for a Library

- Hands-on example

S. N.	URI	HTTP Method	POST body	Result
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.

Objectives

- Some concepts
- Creating and configuring Express projects
- Some best practices for app development
- Setting up an MVC environment
- Adding Twitter Bootstrap for layout
- Publishing to a live URL, using Git and Heroku

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, which are executed when the route is matched.
- Route definition takes the following structure:
 - `app.METHOD(PATH, HANDLER)`

Routing

- The routing methods can have more than one callback function as arguments.
- With multiple callback functions, it is important to provide *next* as an argument to the callback function and then call `next()` within the body of the function to hand off control to the next callback.

Route handlers

A single callback function can handle a route. For example:

```
app.get('/example/a', function (req, res) {  
  res.send('Hello from A!')  
})
```

More than one callback function can handle a route (make sure you specify the next object).

```
app.get('/example/b', function (req, res, next) {  
  console.log('the response will be sent by the next function ...')  
  next()  
}, function (req, res) {  
  res.send('Hello from B!')  
})
```

Route handlers

An array of callback functions can handle a route. For example:

```
var cb0 = function (req, res, next) {  
  console.log('CB0')  
  next()  
}  
  
var cb1 = function (req, res, next) {  
  console.log('CB1')  
  next()  
}  
  
var cb2 = function (req, res) {  
  res.send('Hello from C!')  
}  
  
app.get('/example/c', [cb0, cb1, cb2])
```

Response methods

- The methods on the response object (res) send a response to the client, and terminate the request-response cycle.
- If none of these methods are called from a route handler, the client request will be left hanging.

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

app.route()

- You can create chainable route handlers for a route path by using app.route()

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

express.Router

- A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a “mini-app”.
- The following example creates a router as a module, loads a middleware function in it, defines some routes, and mounts the router module on a path in the main app.

```
var express = require('express')
var router = express.Router()

// define the home page route
router.get('/', function (req, res) {
  res.send('Birds home page')
})

// define the about route
router.get('/about', function (req, res) {
  res.send('About birds')
})

module.exports = router
```

Then, load the router module in the app:

```
var birds = require('./birds')

// ...

app.use('/birds', birds)
```

Writing middleware for use in Express apps

- **Middleware** functions are functions that have access to the [request object](#) (req), the [response object](#) (res), and the next function in the application's request-response cycle.
- The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.
- Middleware functions can perform the following tasks:
 - Execute any code.
 - Make changes to the request and the response objects.
 - End the request-response cycle.
 - Call the next middleware in the stack.

Writing middleware for use in Express apps

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

HTTP method for which the middleware function applies

Path (route) for which the middleware function applies.

The middleware function.

```
app.get('/', function(req, res, next) {  
  next();  
})
```

Callback argument to the middleware function, called "next"

```
app.listen(3000);
```

HTTP response argument to the middleware function, called "res"

HTTP request argument to the middleware function, called "req"

Using Middleware


- An Express application is essentially a series of middleware function calls.
- **Application-level middleware**
 - Bind application-level middleware to an instance of the [app object](#) by using the `app.use()` and `app.METHOD()` functions, where `METHOD` is the HTTP method of the request that the middleware function handles (such as `GET`, `PUT`, or `POST`) in lowercase.
 - This example shows a middleware function with no mount path. The function is executed every time the app receives a request.

```
var app = express()


app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
})
```

Using Middleware

- This example shows a middleware function mounted on the /user/:id path. The function is executed for any type of HTTP request on the /user/:id path.
- This example shows a route and its handler function (middleware system). The function handles GET requests to the /user/:idpath.



```
app.use('/user/:id', function (req, res, next) {  
  console.log('Request Type:', req.method)  
  next()  
})
```



```
app.get('/user/:id', function (req, res, next) {  
  res.send('USER')  
})
```

Using Middleware

- Here is an example of loading a series of middleware functions at a mount point, with a mount path. It illustrates a middleware sub-stack that prints request info for any type of HTTP request to the /user/:id path.

```
app.use('/user/:id', function (req, res, next) {  
  console.log('Request URL:', req.originalUrl)  
  next()  
}, function (req, res, next) {  
  console.log('Request Type:', req.method)  
  next()  
})
```

Router-level middleware

- Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of `express.Router()`.
- `var router = express.Router()`
- Load router-level middleware by using the `router.use()` and `router.METHOD()` functions.
- To skip the rest of the router's middleware functions, call `next('router')` to pass control back out of the router instance.

Router-level middleware

```
// a middleware sub-stack shows request info for any type of HTTP request to the /user/:id path
router.use('/user/:id', function (req, res, next) {
  console.log('Request URL:', req.originalUrl)
  next()
}, function (req, res, next) {
  console.log('Request Type:', req.method)
  next()
})

// a middleware sub-stack that handles GET requests to the /user/:id path
router.get('/user/:id', function (req, res, next) {
  // if the user ID is 0, skip to the next router
  if (req.params.id === '0') next('route')
  // otherwise pass control to the next middleware function in this stack
  else next()
}, function (req, res, next) {
  // render a regular page
  res.render('regular')
})
```

Router-level middleware

```
// handler for the /user/:id path, which renders a special page
router.get('/user/:id', function (req, res, next) {
  console.log(req.params.id)
  res.render('special')
})

// mount the router on the app
app.use('/', router)
```

Create an Express project

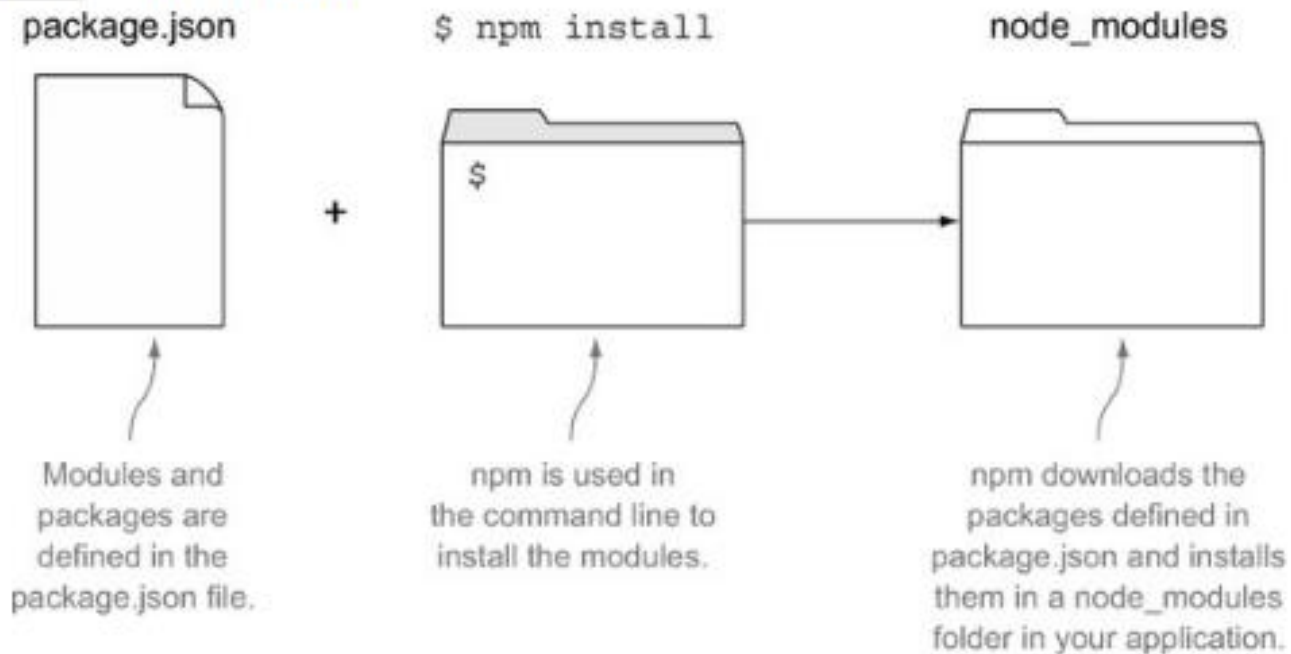
- `npm install -g express-generator`
- `express Loc8r --view=pug`
- `Cd Loc8r`
- `Npm install`
- `npm start`
- load <http://localhost:3000/> in your browser to access the app

Example package.json file in a new Express project

```
{
  "name": "application-name",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.9.0",
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "morgan": "~1.3.0",
    "serve-favicon": "~2.1.3",
    "debug": "~2.0.0",
    "jade": "~1.6.0"
  }
}
```

Npm install

Figure 3.2. The npm modules defined in a package.json file are downloaded and installed into the application's node_modules folder when you run the `npm install` terminal command.



PUG vs HTML

```
#banner.page-header
```

```
h1 My page
```

```
p.lead Welcome to my page
```

```
<div id="banner" class="page-header">
```

```
<h1>My page</h1>
```

```
<p class="lead">Welcome to my page</p>
```

```
</div>
```

Template engine

- A ***template engine*** enables you to use static template files in your application.
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page.
- Some popular template engines that work with Express are [Pug](#), [Mustache](#), and [EJS](#).

INTRODUCTION TO PUG

- What is Pug? It's a template engine for server-side Node.js applications.
- Express is capable of handling server-side template engines. Template engines allow us to add data to a view, and generate HTML dynamically.
- Pug is a new name for an old thing. It's *Jade 2.0*.

Launching app

. Landing page for a barebones Express project

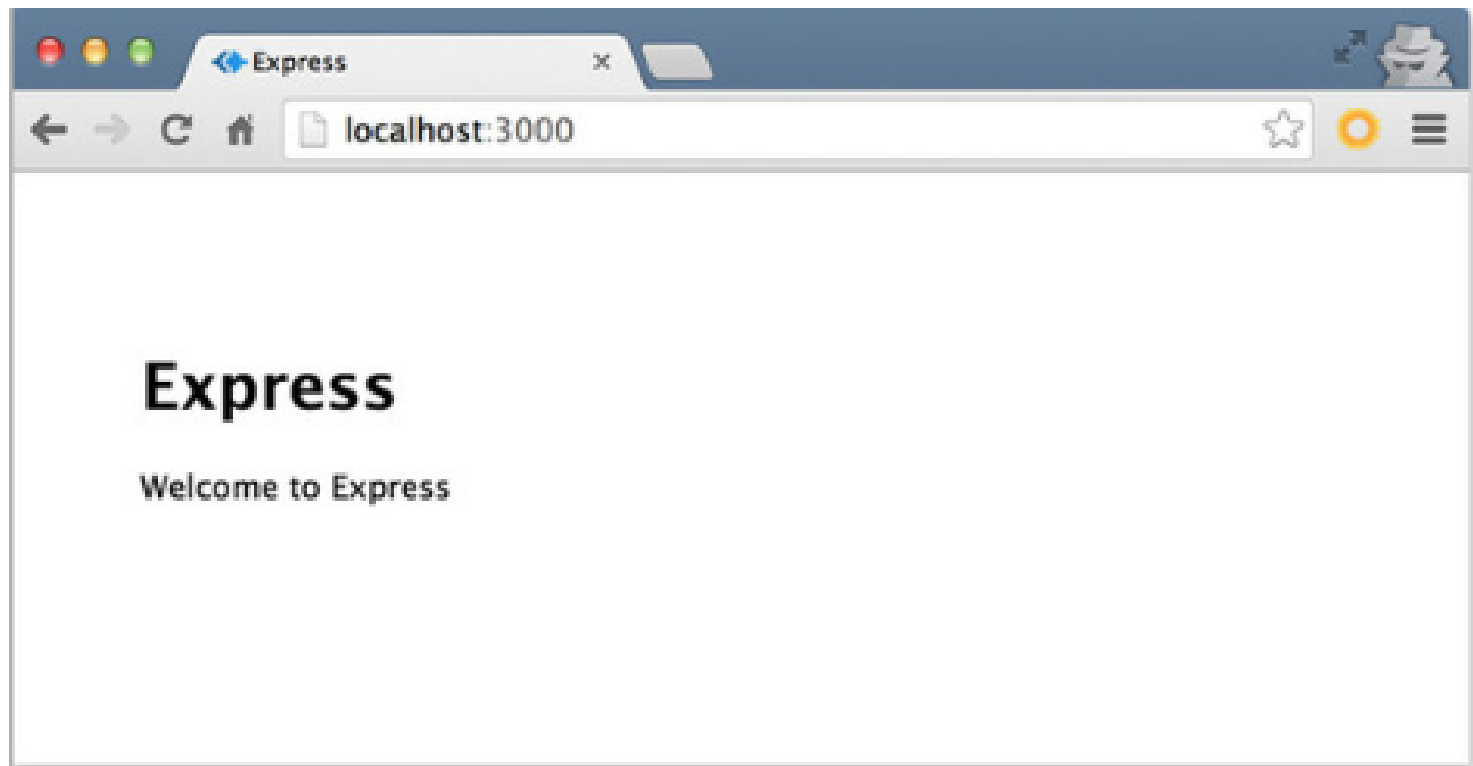
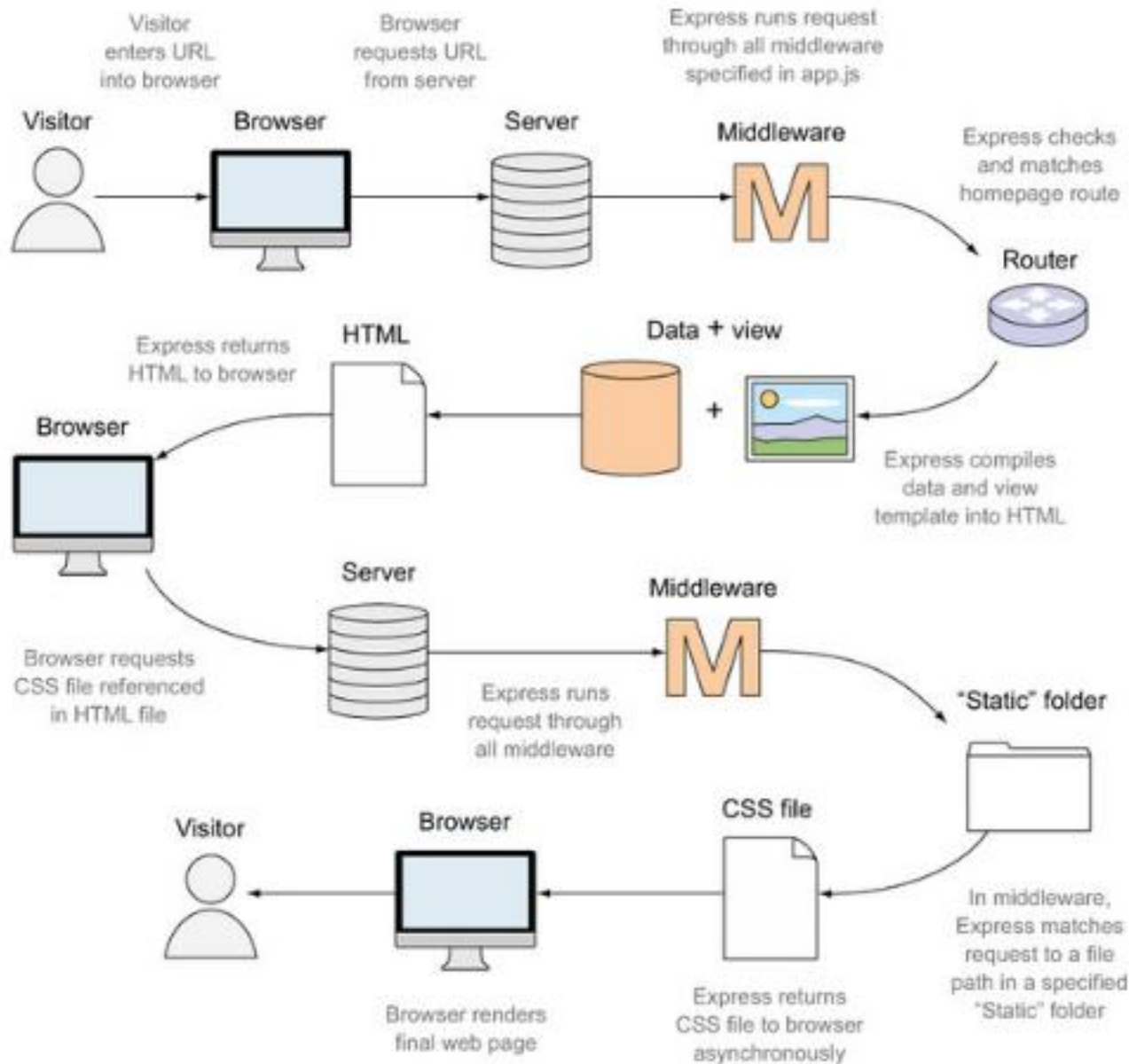


Figure 3.4. The key interactions and processes that Express goes through when responding to the request for the default landing page. The HTML page is processed by Node to compile data and a view template, and the CSS file is served asynchronously from a static folder.

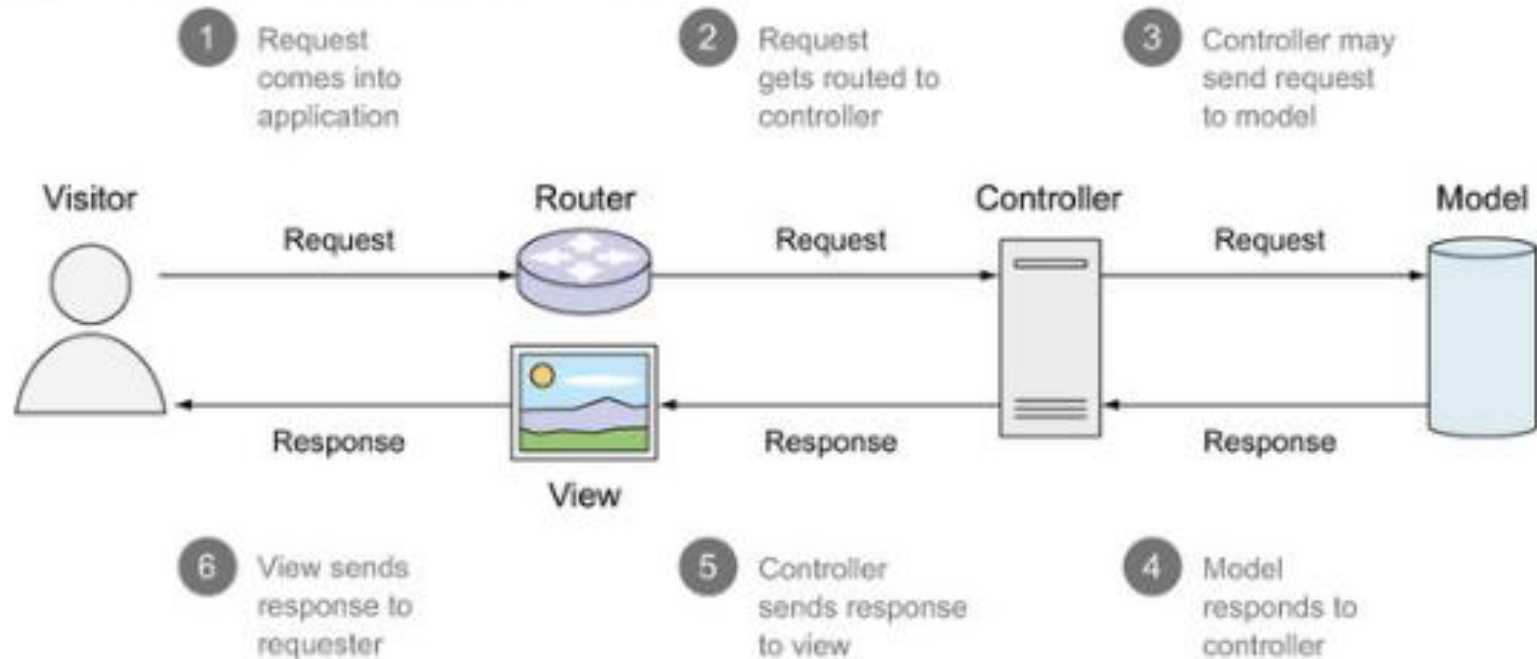


AUTOMATICALLY RESTARTING YOUR APPLICATION WITH NODEMON

- `npm install --save-dev nodemon`
- Add a line in scripts in package.json file
- `"devstart": "nodemon ./bin/www"`
- `npm run devstart`

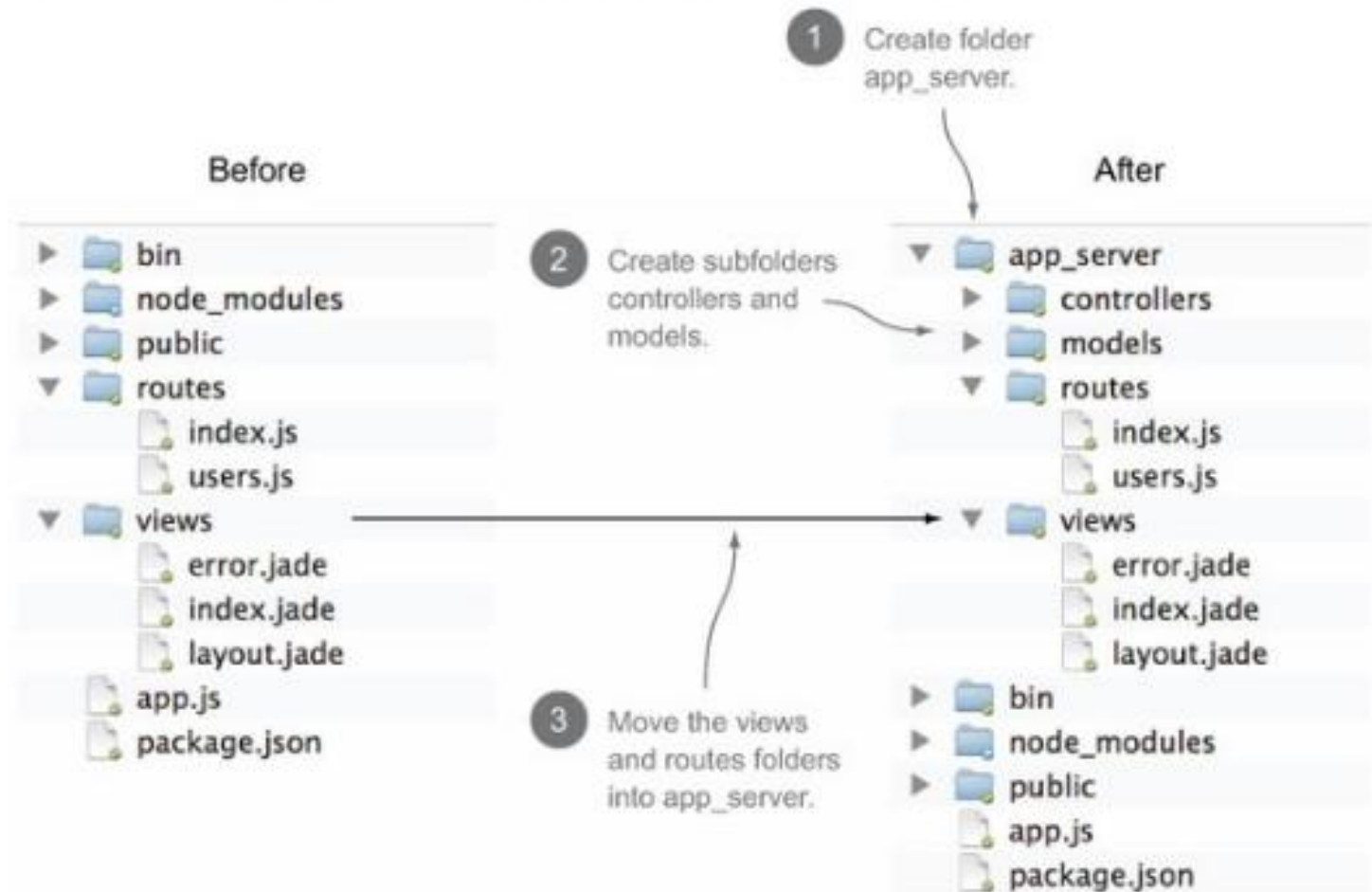
A birds eye view of MVC

Request-response flow of a basic MVC architecture



Changing the folder structure

Changing the folder structure of an Express project into an MVC architecture



Update app.js: Using the new views and routes folders

- Old
 - `app.set('views', path.join(__dirname, 'views'));`
- New
 - `app.set('views', path.join(__dirname, 'app_server', 'views'));`
- Old
 - `var routes = require('./routes/index');`
 - `var users = require('./routes/users');`
- New
 - `var routes = require('./app_server/routes/index');`
 - `var users = require('./app_server/routes/users');`

Understanding route definition

Look at index.js in routes folder

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

Taking the controller code out of the route: step 1

```
var homepageController = function (req, res) {  
  res.render('index', { title: 'Express' });  
};  
  
/* GET home page. */  
router.get('/', homepageController);
```


UNDERSTANDING RES.RENDER

JavaScript object containing
data for template to use

```
res.render('index', {title:'express'});
```

Name of template file to use—in
this case referencing index.jade

```
graph TD; A[JavaScript object containing data for template to use] --- B[{title:'express'}]; B --- C[res.render('index', {title:'express'});]; C --- D['index']; D --- E[Name of template file to use—in this case referencing index.jade];
```

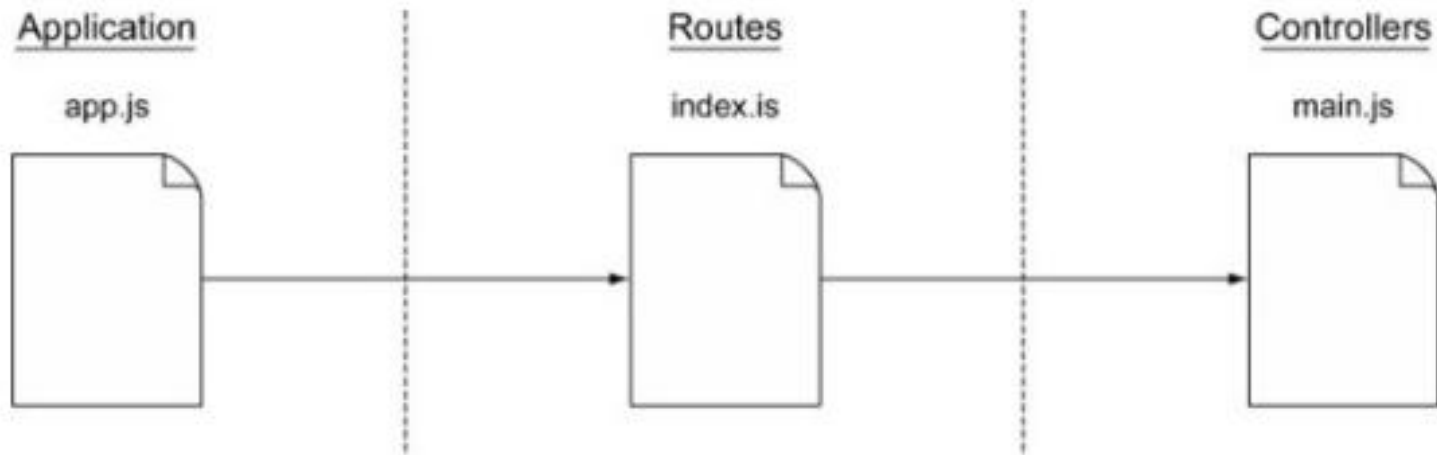
Setting up the homepage controller in app_server/controllers/main.js

```
/* GET home page */  
module.exports.index = function(req, res){  
  res.render('index', { title: 'Express' });  
};
```

Updating the routes file to use external controllers

```
var express = require('express');  
var router = express.Router();  
var ctrlMain = require('../controllers/main');  
/* GET home page. */  
router.get('/', ctrlMain.index);  
module.exports = router;
```

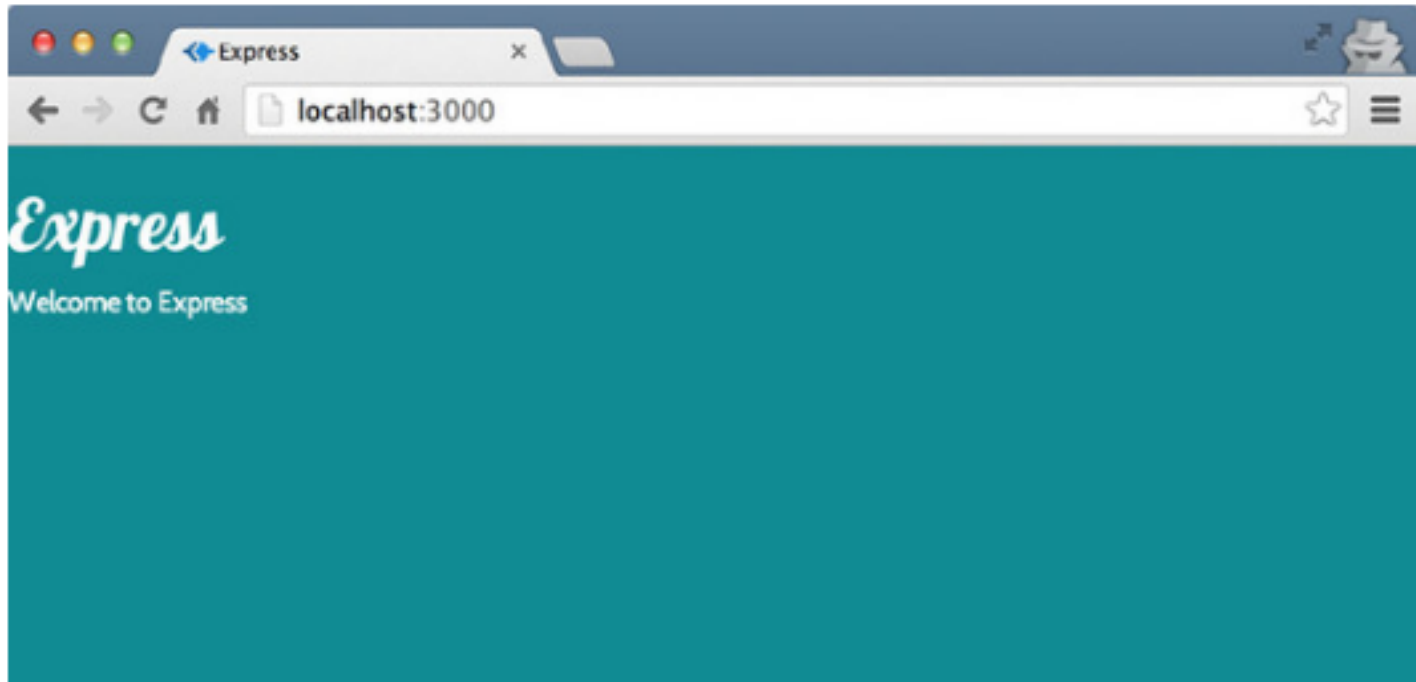
Separating the controller logic from the route definitions



Updated layout.pug including Bootstrap references

```
doctype html
html
  head
    meta(name='viewport', content='width=device-width, initial-scale=1')
    title= title
    link(rel='stylesheet', href='/bootstrap/css/amelia.bootstrap.css')
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
    script(src='/javascripts/jquery-1.11.1.min.js')
    script(src='/bootstrap/js/bootstrap.min.js')
```

Bootstrap theme having an effect on the default Express index page



Making it live on Heroku

- **Adding an engines section to package.json**

```
{
  "name": "Loc8r",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "engines": {
    "node": "~4.2.1",
    "npm": "~2.2.0"
  },
  "dependencies": {
    "express": "~4.9.0",
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "morgan": "~1.3.0",
    "serve-favicon": "~2.1.3",
    "debug": "~2.0.0",
    "jade": "~1.6.0"
  }
}
```

Create a Procfile on root folder and add the text

web: npm run devstart

Signup for Heroku

npm install -g heroku

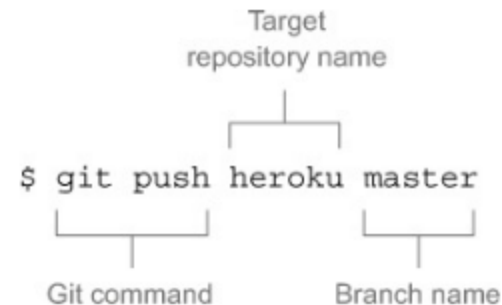
heroku login

Create .gitignore file and write node_modules in it

git init

git add .

git commit -m "First commit"



Project Assignment 1: Deadline (28th Sep 2018 11:59 pm)

- Make your project live on Heroku
- You have to submit the heroku link similar to this one for your project page
- <https://obscure-anchorage-73743.herokuapp.com/>
- There should at least be one route defined for your root path
- You can set the heading to your project title in the view (you can do more if you want, add some images for trial etc.)
- You should implement MVC architecture

References

- <https://flaviocopes.com/pug/#meta-tags>
- <https://devcenter.heroku.com/articles/deploying-nodejs#declare-app-dependencies>
- [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express Nodejs/skeleton website](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website)
- <https://expressjs.com/en/4x/api.html#express>
- <https://expressjs.com/en/guide/routing.html>