UNIT-II

Building a Node Web Application

Creating and setting up a MEAN project: Creating an Express project, modifying Express for MVC, Importing Bootstrap for quick, responsive layouts, making it live on Heroku.

Building a static site with Node and Express: Defining the routes in Express, building basic controllers, creating some views, adding the rest of the views, taking the data out of the views and making them smarter.

Creating and setting up a MEAN project

- In the MEAN stack, Express is the Node web application framework. Together, Node.js and Express underpin the entire stack.
- Two things to focus
    1. Create the project and the encapsulating Express application that will house everything except the database.
    2. Set up the main Express application.

Loc8r

- We are going to build an application called Loc8r. This is going to be a location aware web application that will display listings near users and invite people to login and leave reviews.
- In the MEAN stack, Express is the Node web application framework. Together, Node and Express underpin the entire stack, so let's start here. In terms of building up the application architecture, Figure 3.1 shows where we'll be focusing in this chapter. We'll be doing two things:
  1. Creating the project and encapsulating the Express application that will house everything else except the database
  2. Setting up the main Express application

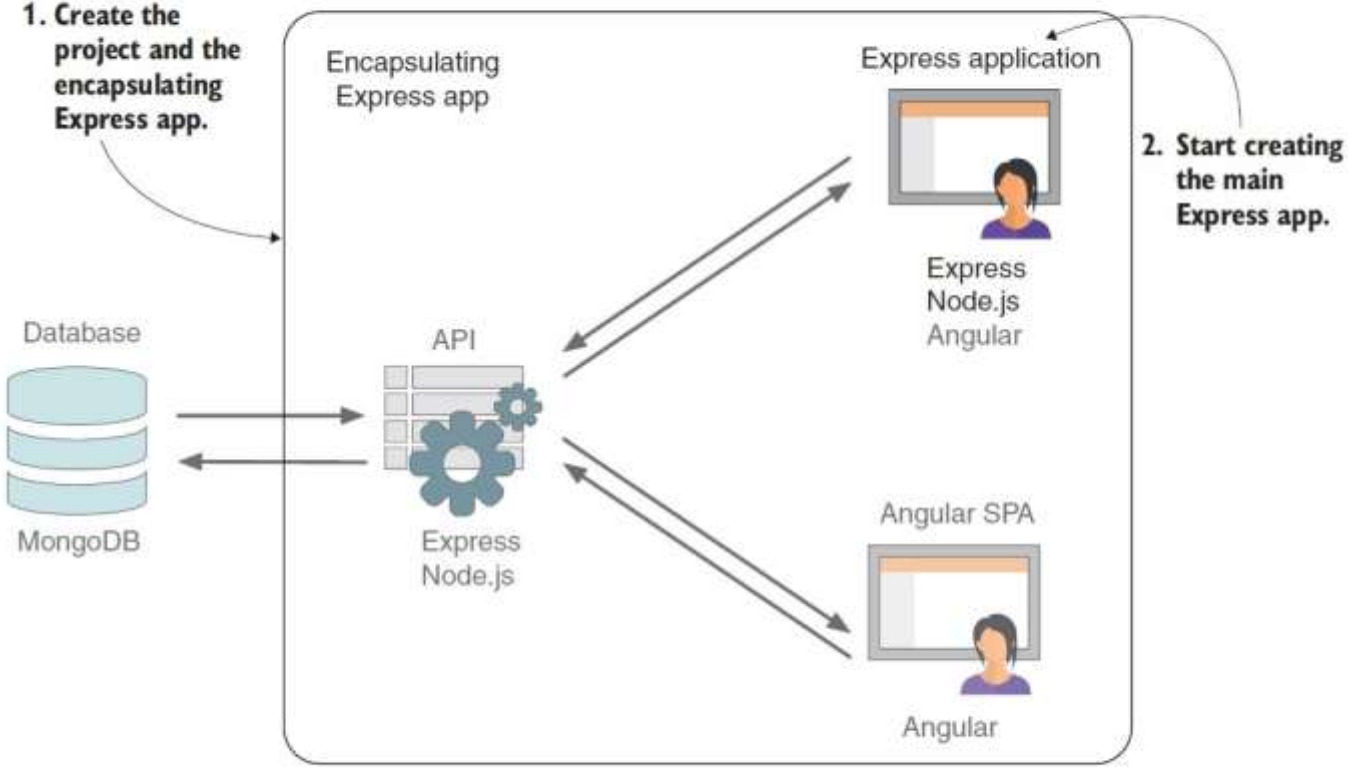


Figure 3.1 Creating the encapsulating Express application and starting to set up the main Express application

Brief look at Express, Node, and npm

- Express is a web application framework for Node. In basic terms, an Express application is a Node application that happens to use Express as the framework.
- npm is a package manager that gets installed when you install Node, which enables you to download Node modules or packages to extend the functionality of your application.

Defining packages with package.json

- In every Node application, you should have a file in the root folder of the application called package.json. This file can contain various metadata about a project, including the packages that it depends on to run.

Listing 3.1 Example package.json file in a new Express project

```
{
  "name": "application-name",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
```

Various metadata defining the application

Defining packages with package.json

```
  "dependencies":
    "body-parser": "~1.18.3",
    "cookie-parser": "~1.4.3",
    "debug": "~4.1.0",
    "express": "^4.16.4",
    "morgan": "^1.9.1",
    "pug": "^2.0.3",
    "serve-favicon": "~2.5.0"
  }
}
```

Package dependencies needed for the application to run

Working with dependency versions in package.json

- The name of each dependency is the version number that the application will use. Notice that they're prefixed with either a tilde (~) or a caret (^).
- Prefixing the whole version number with a ~ is like replacing the patch version with a wildcard, which means that the application will use the latest patch version available.
- Similarly, prefixing the version with a caret (^) is like replacing the minor version with a wildcard. This has become best practice because patches and minor versions should contain only fixes that won't have any effect on the application.
- But new major versions are released when a breaking change is made, so you want to avoid automatically using later versions of these in case the breaking change affects your application

Installing Node dependencies with npm

- Any Node application or module can have dependencies defined in a package.json file.
- Using a terminal prompt in the same folder as the package.json file, run the following command
  - $ npm install

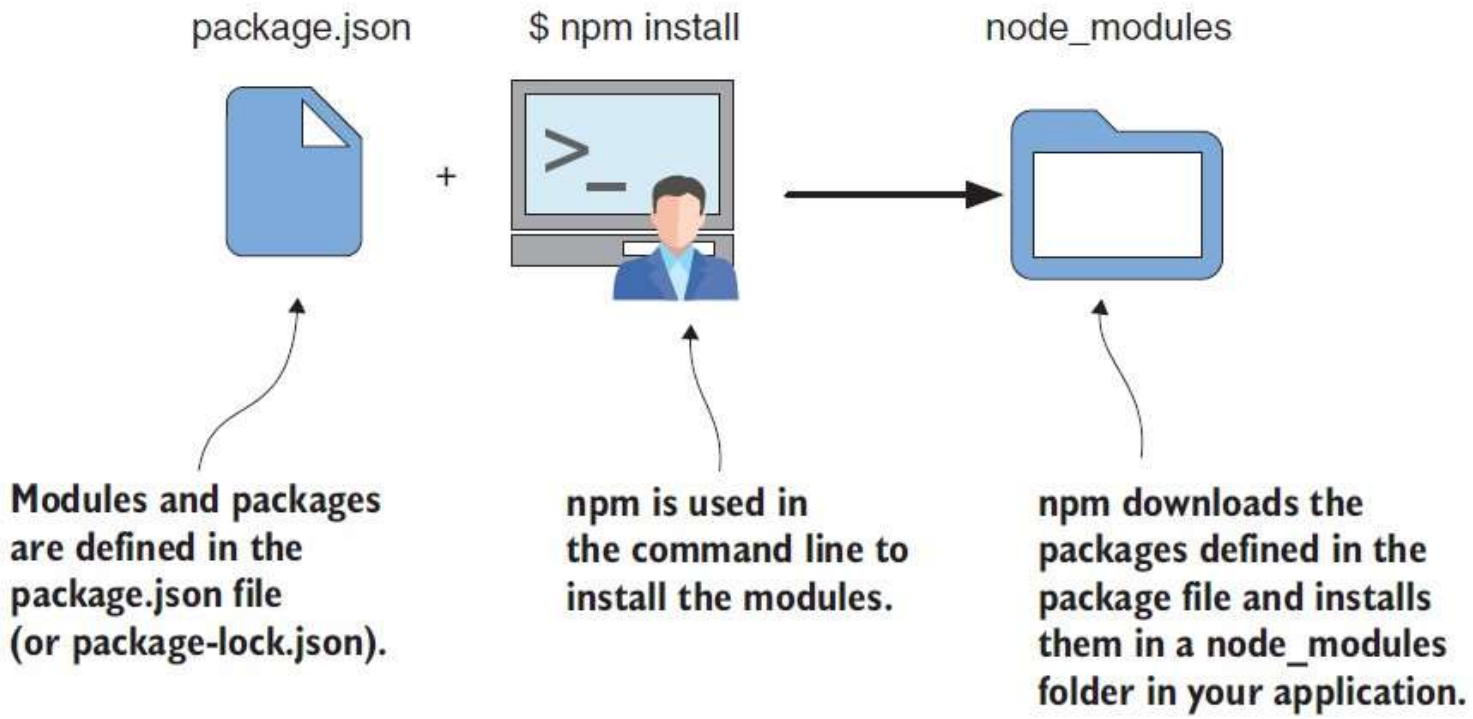


Figure 3.2 The npm modules defined in a package.json file are downloaded and installed in the application's node_modules folder when you run the `npm install` terminal command.

Adding More Packages To An Existing Project

- Using npm, it’s easy to add more packages to the application whenever you want. Find the name of the package you want to install, open a command prompt in the same folder as the package.json file, and then run a simple command like this:
  - $ npm install --save package-name
  - with this command, npm downloads and installs the new package in the node_modules folder.
  - The --save flag tells npm to add this package to the list of dependencies in the package.json file. As of npm version 5, the --save flag is no longer required, as NPM saves changes to the package.json file automatically.

Creating an Express project

- To create an Express project, you'll need to have five key things installed on your development machine:
  - Node and npm
  - The Express generator installed globally
  - Git
  - Heroku
  - Suitable command-line interface (CLI) or terminal

Creating an Express project

- To create an Express project, you'll need to have five key things installed on your development machine:
  - Node and npm
  - The Express generator installed globally
  - Git
  - Heroku
  - Suitable command-line interface (CLI) or terminal

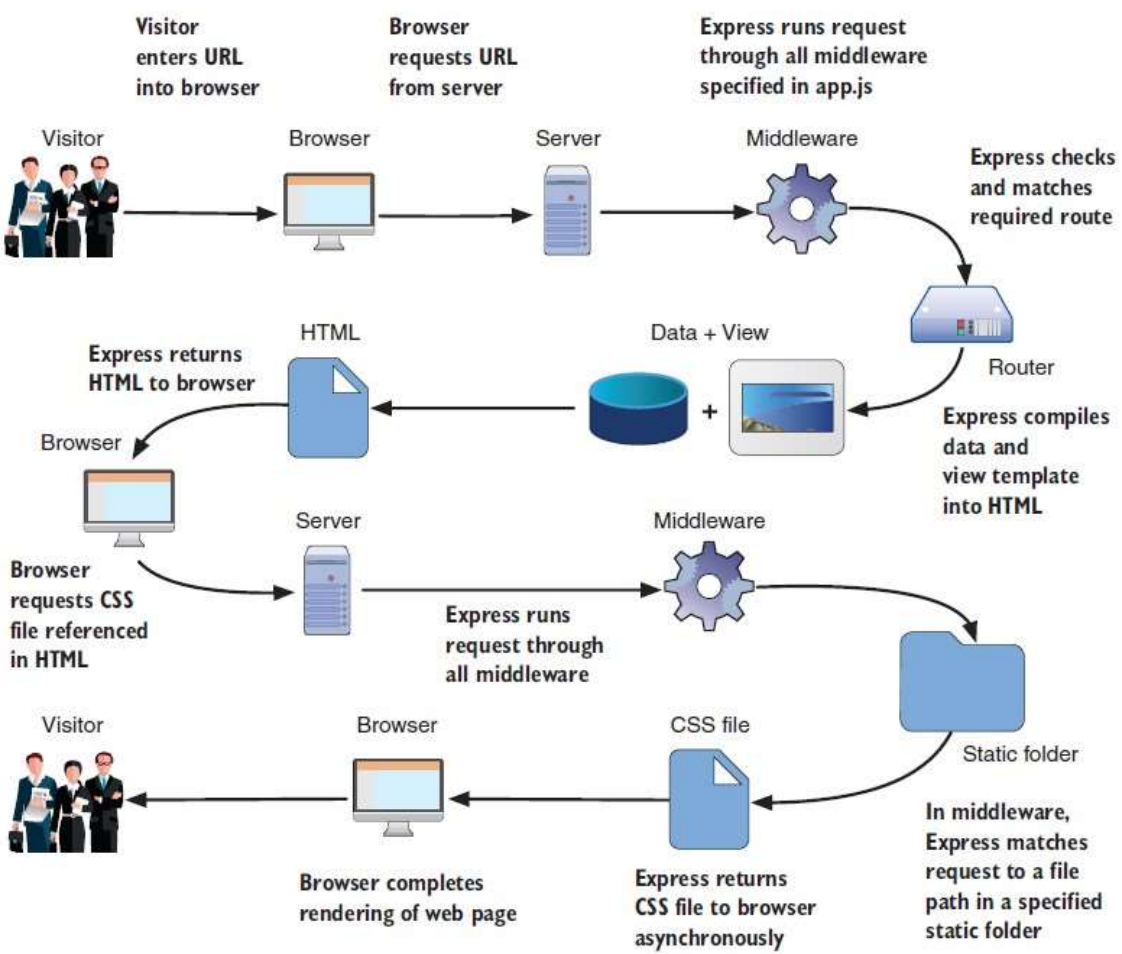


Figure 3.4 The key interactions and processes that Express goes through when responding to the request for the default landing page. The HTML page is processed by Node to compile data and a view template, and the CSS file is served asynchronously from a static folder.

Modifying Express for MVC

📌 What is MVC Architecture?

- MVC separates:
  - The **data** (Model)
  - The **display** (View)
  - The **application logic** (Controller)

⚙️ Development Advantage

- Fits well with **rapid prototype development**
- Allows focus on **one aspect at a time**

✅ Purpose of MVC

- Removes **tight coupling** between components
- Makes code more **maintainable** and **reusable**

🧩 MVC Scope in This Context

- **Whole books** are written on MVC nuances
- We'll keep it **high-level**
- Goal: Use **MVC with Express** to build the **Loc8r app**

A bird's-eye view of MVC

At a simple level, this loop in an MVC architecture works like this:

1. A request comes into the application.
2. The request gets routed to a controller.
3. The controller, if necessary, makes a request to the model.
4. The model responds to the controller.
5. The controller merges the view and the data to form a response.
6. The controller sends the generated response to the original requester.

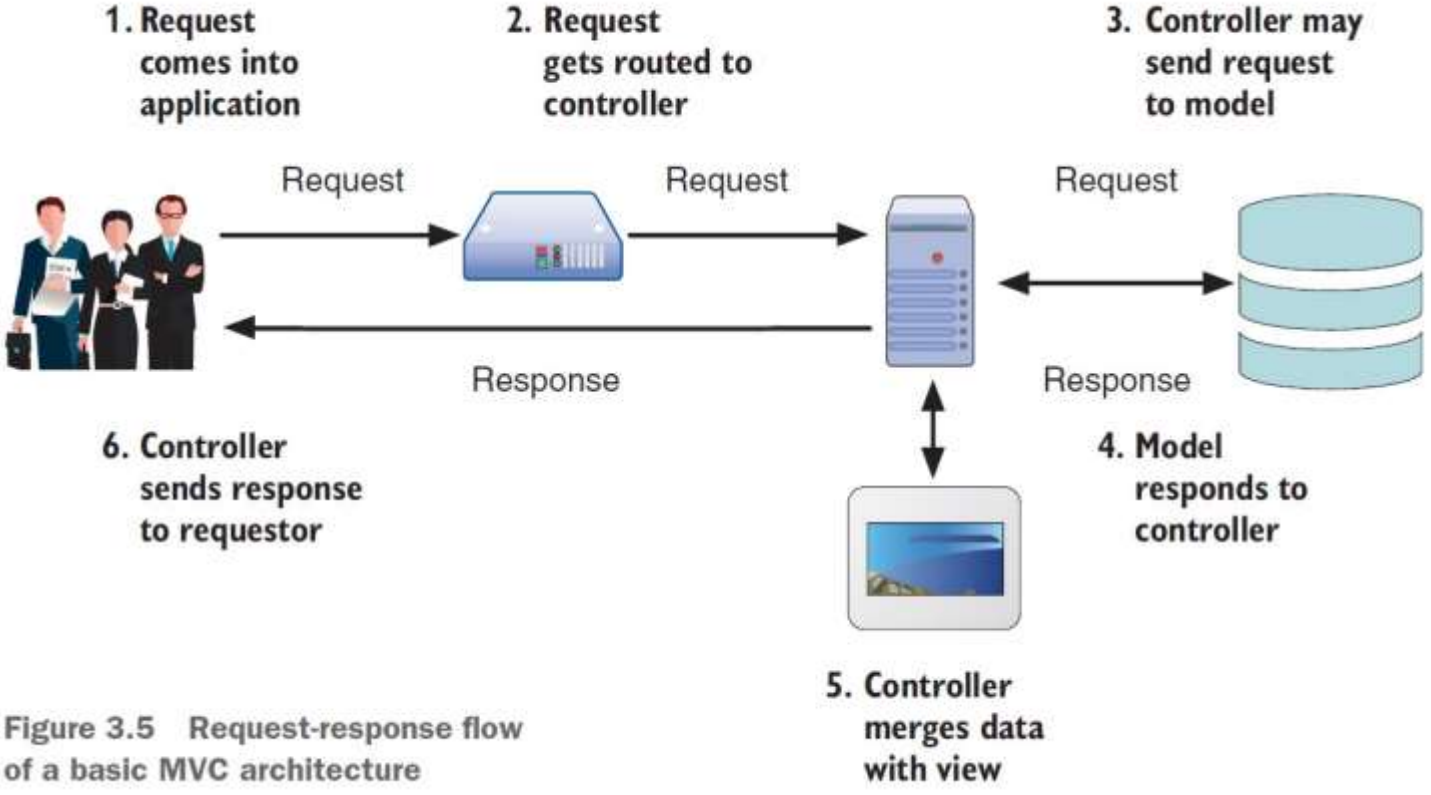


Figure 3.5 Request-response flow of a basic MVC architecture

Changing the folder structure

1. Create a new folder called app_server.
2. In app_server, create two new folders called models and controllers.
3. Move the views and routes folders from the root of the application into the app_server folder.

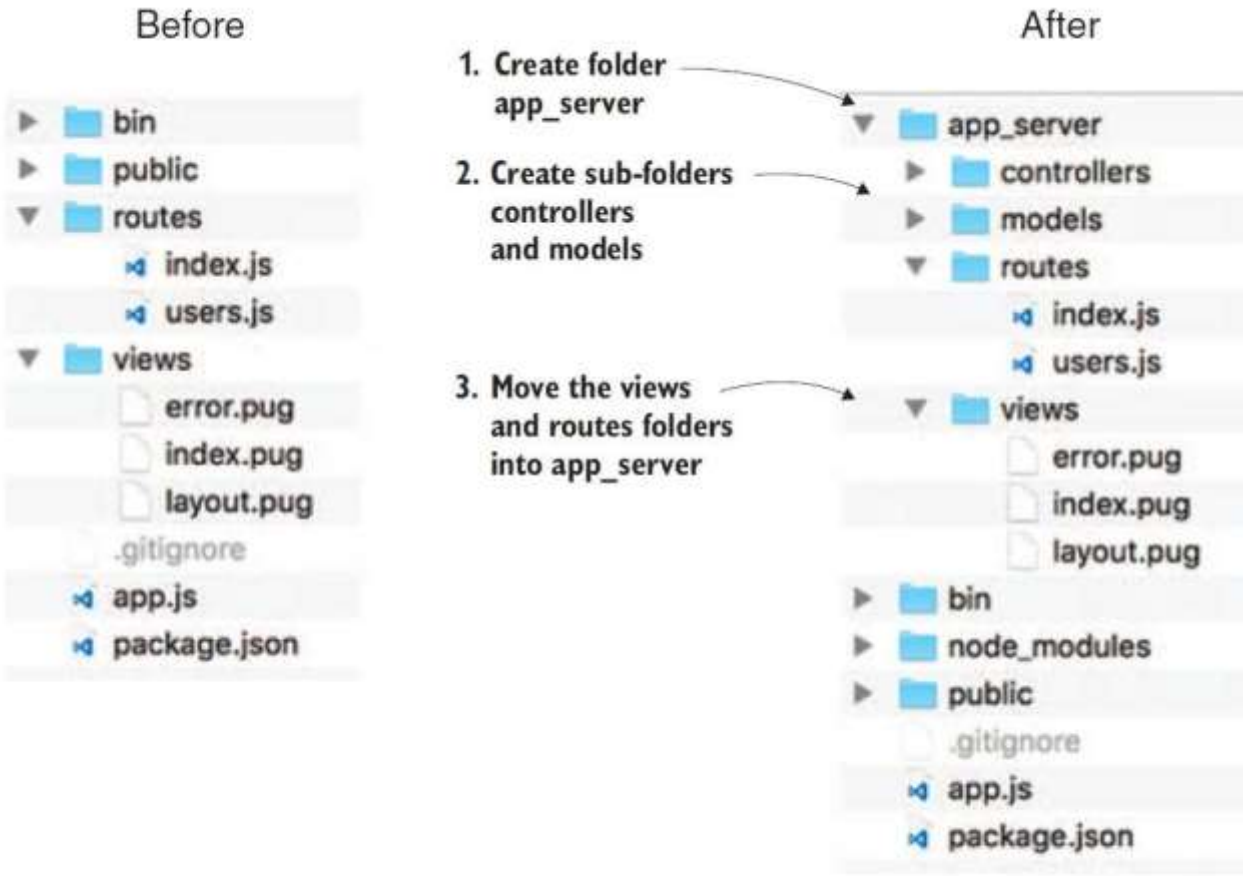


Figure 3.6 Changing the folder structure of an Express project into an MVC architecture

Using the views and routes relocated folders

USING THE NEW VIEWS FOLDER LOCATION

Express will be looking for /views, but it needs to look for /app_server/views. Changing the path is simple. In app.js, find the following line:

```
app.set('views', path.join(__dirname, 'views'));
```

Change it to the following (modifications in bold):

```
app.set('views', path.join(__dirname, 'app_server', 'views'));
```

Your application still won't work, because you've moved the routes, so tell Express about them too.

Using the views and routes relocated folders

USING THE NEW ROUTES FOLDER LOCATION

Express will be looking for /routes, but it needs to look for /app_server/routes. Changing this path is also simple. In app.js, find the following lines:

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

Change these lines to the following (modifications in bold):

```
const indexRouter = require('./app_server/routes/index');
const usersRouter = require('./app_server/routes/users');
```

Splitting controllers from routes

Splitting controllers from routes

- In a default Express setup, controllers are part of the routes, but Controllers should manage the application logic routing should map URL requests to controllers.

UNDERSTANDING ROUTE DEFINITION

Inside app_server/routes/index.js,

1. Router code
2. Controller code

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

1 Where the router looks for the URL

2 Controller content, albeit very basic right now

Controller code out of route

Listing 3.2 Taking the controller code out of the route: step 1

```
const homepageController = (req, res) => {
  res.render('index', { title: 'Express' });
};
/* GET homepage. */
router.get('/', homepageController);
```

Gives a name to the arrow function

Passes the name of the function through as a callback in the route definition

main.js file

- Create a new file called main.js in app_server/controllers.
- In this file, create and export a method called index, and use it to house the res.render code, as shown in the following listing.

Listing 3.3 Setting up the homepage controller in app_server/controllers/main.js

```
/* GET homepage */
const index = (req, res) => {
  res.render('index', { title: 'Express' });
};

module.exports = {
  index
};
```

Creates an index function

Includes controller code for the homepage

Exposes the index function as a method

app_server/routes/index.js

- Edit the bold content in index.js file

Listing 3.4 Updating the routes file to use external controllers

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

1 Requires the main controllers file

2 References the index method of the controllers in the route definition

Routing and Controller Architecture

- app.js requires routes/index.js, which in turn requires controllers/main.js.

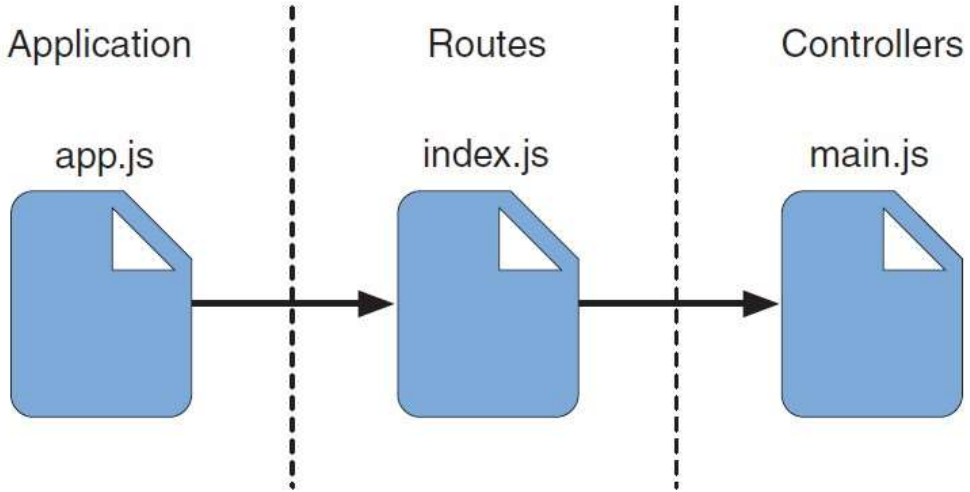
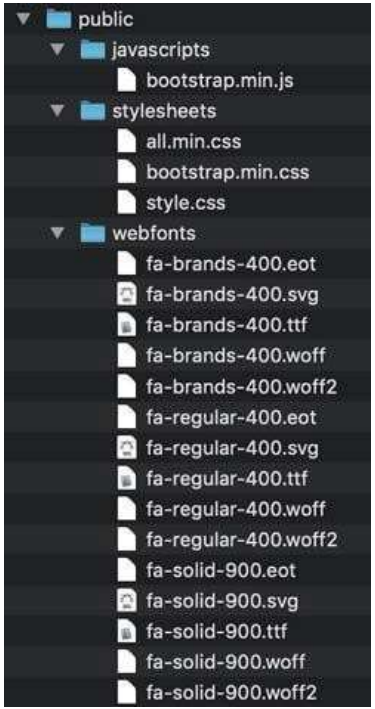


Figure 3.7 Separating the controller logic from the route definitions

Importing Bootstrap for quick, responsive layouts

Importing Bootstrap for quick, responsive layouts

- Loc8r application uses Twitter's Bootstrap framework to speed the development of a responsive design.
- adding some font icons and custom styles



1. Downloading Bootstrap and adding it to the application

- Instructions for downloading Bootstrap, downloading the font icons (by Font Awesome), creating a custom style, and adding the files to the project folder are in appendix B.
- Note that you use Bootstrap 4.1.

2. Twitter Bootstrap

- 📦 **Step 1:** Download the Bootstrap library files.
- 🗂️ **Step 2:** Unzip the downloaded file.
- 📁 **Step 3:** Place the unzipped files into your web application directory.

🔗 **Download Link:**

👉 https://getbootstrap.com

💡 **Important Note:**

- Always download the "**ready to use files**" (not the source code).
- The ZIP file contains two key folders:
  - `css` → contains stylesheets
  - `js` → contains JavaScript files

✅ Step 3: Organizing Bootstrap Files in Your Express App

Once you have **downloaded and unzipped** the Bootstrap library:

📁 Move the following files into your Express app's `public` folder:

1. **Copy** `bootstrap.min.css` →
   📁 `public/stylesheets/`
2. **Copy** `bootstrap.min.js` →
   📁 `public/js/` *(or* `public/javascripts/` *depe*

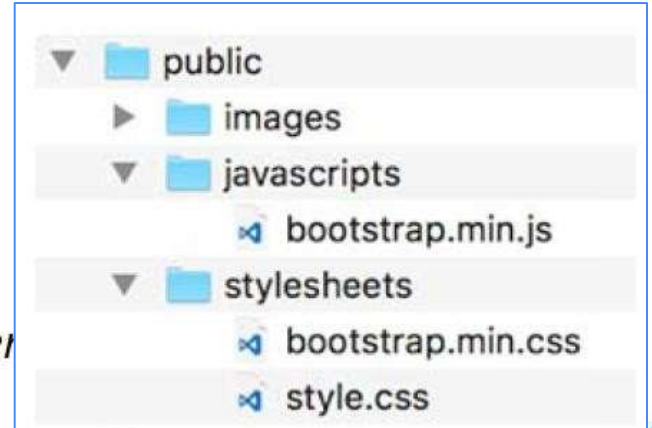


Figure B.1 The structure and contents of the public folder after Bootstrap has been added

Step 4: Add Custom Styles with style.css

- To add the custom styles, edit the style.css file in your public/stylesheets folder.

Note: Code available in Appendix B of the textbook

Listing B.1 Custom styles to give Loc8r a more distinctive look

```
@import url("//fonts.googleapis.com/css?family=Lobster|Cabin:400,700");

h1, h2, h3, h4, h5, h6 {
  font-family: 'Lobster', cursive;
}

legend {
  font-family: 'Lobster', cursive;
}

.navbar {
  background-color: #ad1d28;
  border-color: #911821;
}

.navbar-light .navbar-brand {
  font-family: 'Lobster', cursive;
```

```
  color: #fff;
}

.navbar-light .navbar-toggler {
  color: white;
  border-color: white;
}

.navbar-light .navbar-toggler-icon {
  background-image: url("data:image/svgxml;charset=utf8,%3Csvg
  ➥viewBox='0 0 30 30'xmlns='http://www.w3.org/2000/svg'%3E%
  ➥3Cpath stroke='white' stroke-width='2'stroke-linecap='round'
  ➥stroke-miterlimit='10' d='M4 7h22M4 15h22M4 23h22'/%3E%3C/svg%3E")
}

.navbar-light .navbar-nav .nav-link,
.navbar-light .navbar-nav .nav-link:focus,
.navbar-light .navbar-nav .nav-link:hover {
  color: white;
}

.card {
  background-color: #469ea8;
  padding: 1rem;
}

.card-primary {
  border-color: #a2ced3;
  margin-bottom: 0.5rem;
}
```

```
.banner {
  margin-top: 4em;
  border-bottom: 1px solid #469ea8;
  margin-bottom: 1.5em;
  padding-bottom: 0.5em;
}

.review-header {
  background-color: #31727a;
  padding-top: 0.5em;
  padding-bottom: 0.5em;
  margin-bottom: 0.5em;
}

.review {
  margin-right: -16px;
  margin-left: -16px;
  margin-bottom: 0.5em;
}

.badge-default, .btn-primary {
  background-color: #ad1d28;
  border-color: #911821;
}

h4 a, h4 a:hover {
  color: #fff;
}
```

```
h4 small {
  font-size: 60%;
  line-height: 200%;
  color: #aaa;
}

h1 small {
  color: #aaa;
}

.address {
  margin-bottom: 0.5rem;
}

.facilities span.badge {
  margin-right: 2px;
}

p {
  margin-bottom: 0.65rem;
}

a {
  color: rgba(255, 255, 255, 0.8)
}

a:hover {
  color:#fff
}

body {
  font-family: "Cabin", Arial, sans-serif;
  color: #fff;
  background-color: #108a93;
}
```

- To save you from typing all this code, you can get this file from the project repo on
- GitHub at https://github.com/cliveharber/gettingMean-2. It’s introduced in the
- chapter-04 branch.

✨ Font Awesome Integration (v5.2.0)

📥 Step 5: Download Font Awesome

- Go to: https://fontawesome.com/how-to-use/on-the-web/setup/hosting-font-awesome-yourself
- Click the **download** button.
- 📚 *This book uses version **5.2.0** of Font Awesome.*

📁 Step 6: Organize the Font Awesome Files

After unzipping, you'll find many folders. The two important ones are:

- `css/`
- `webfonts/`

🔧 **Steps to include in your Express app:**

1. 📁 **Copy** the entire `webfonts/` folder
   → into your `public/` directory.
2. 📄 **Copy** the `all.min.css` file
   → from the `css/` folder into `public/stylesheets/`

Font Awesome

5. First, head to https://fontawesome.com/how-to-use/on-the-web/setup/hostingfont- awesome-yourself, and click the download button to download the zip file.

- used version 5.2.0 in this book

6. The zip file contains loads of folders. The most important folders for this book are **css** and **webfonts.**

- When Font Awesome is downloaded and unzipped, follow these two steps:
  1. Copy the entire **webfonts/** folder into the **public folder** in your application.
  2. Copy the **all.min.css** file from the **css folder** into **public/stylesheets**.

- The public folder should look something like figure

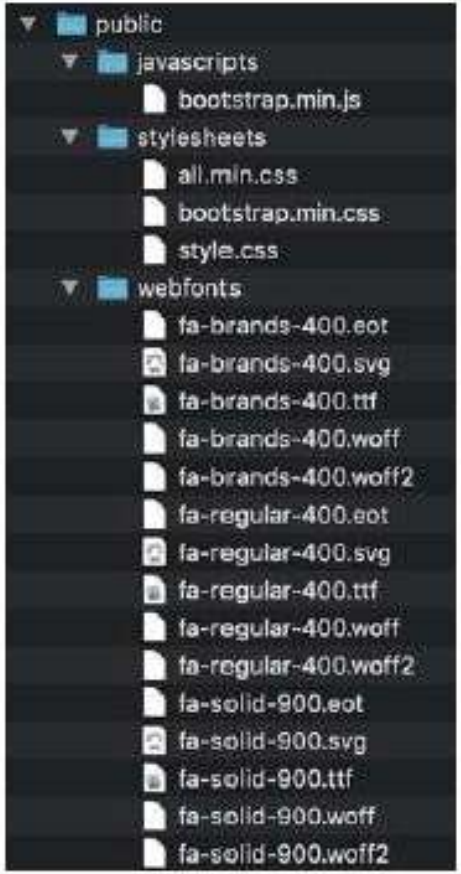


Figure 3.8 Structure of the public folder in the Express application after adding Bootstrap

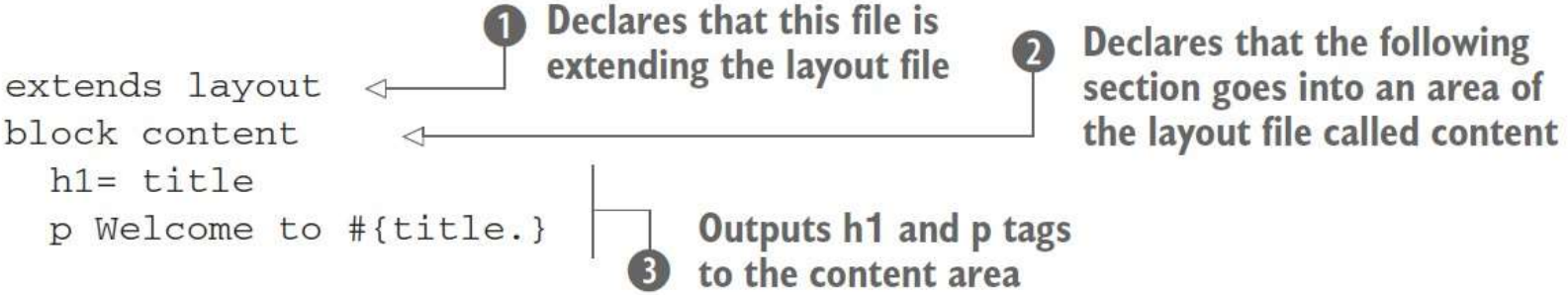
👷 Using Bootstrap in the Application

Step 7: Working with the Front-End (Pug Template)

📄 **File:** `index.pug`

📌 This file represents a **view** that extends a base layout and injects content into it.

Listing 3.5 The complete index.pug file



8. layout.jade file

Listing 3.6 Default layout.pug file

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

Empty named block can be used by other templates

✅ 9. ADDING BOOTSTRAP TO THE ENTIRE APPLICATION

🔧 In `layout.pug`, do these four things:

1. **Include Bootstrap and Font Awesome CSS files**
2. **Include the Bootstrap JavaScript file**
3. **Include jQuery and Popper.js** (required by Bootstrap)
4. **Add viewport `<meta>` tag** for responsive scaling on mobile devices

Example layout.pug With Bootstrap & Font Awesome Integration

- The CSS file and the viewport metadata should both be in the head of the document, and the two script files should be at the end of the body section.

Listing 3.7 Updated layout.pug including Bootstrap references

```
doctype html
html
  head
    meta(name='viewport', content='width=device-width,
      ➥initial-scale=1.0')
    title= title
    link(rel='stylesheet',  href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/all.min.css')
```

Sets the viewport metadata for better display on mobile devices

Includes Bootstrap and Font Awesome CSS

```
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
    script(src='https://code.jquery.com/jquery-3.3.1.slim.min.js',
    ➥integrity='sha384-
    ➥q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo',
    ➥crossorigin='anonymous')
    script(src='https://cdnjs.cloudflare.com/ajax/libs/
    ➥popper.js/1.14.3/umd/popper.min.js',integrity='sha384-
    ➥ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49',
    crossorigin='anonymous')
    script(src='/javascripts/bootstrap.min.js')
```

Brings in jQuery and Popper, needed by Bootstrap. Make sure that the script tags are all at the same indentation.

Brings in the Bootstrap JavaScript file

10. Re-run the code

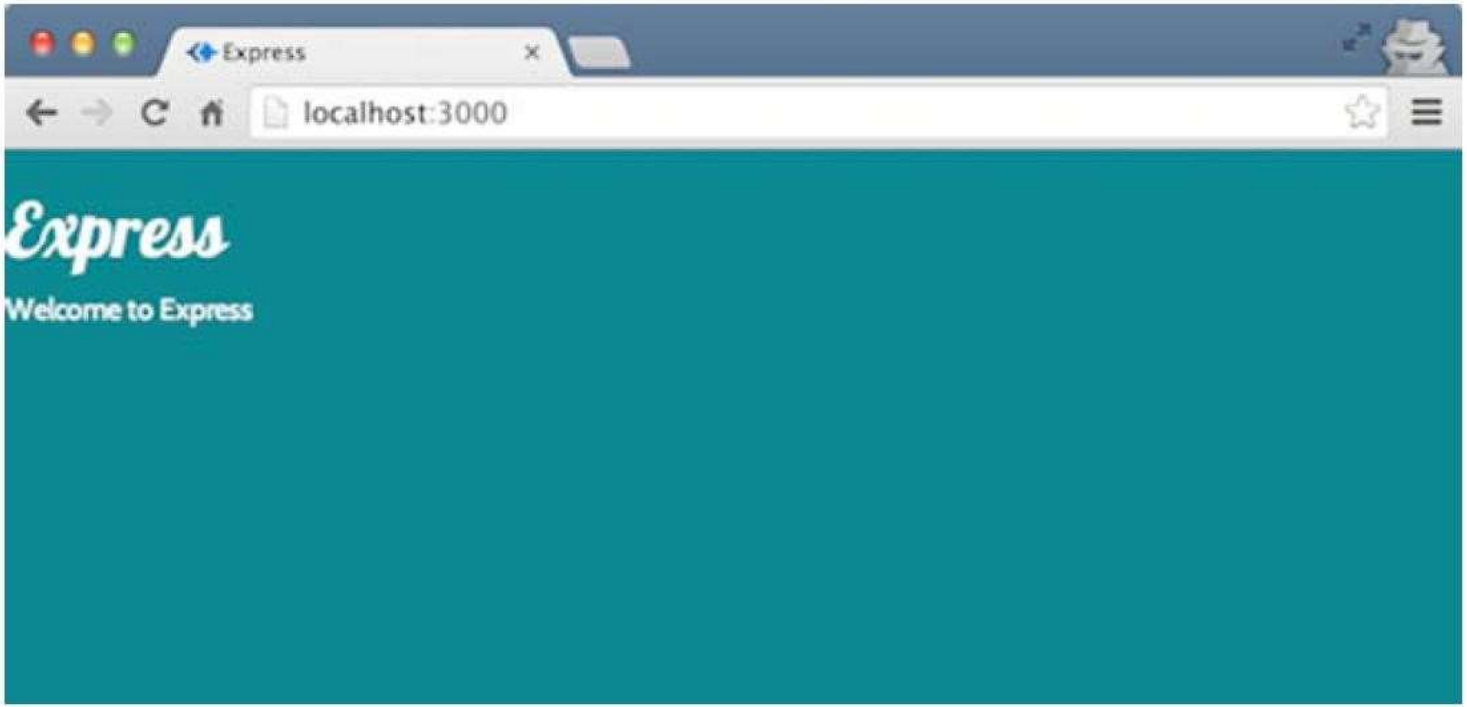


Figure 3.9 Bootstrap and your styles having an effect on the default Express index page

Making it live with render -Heroku

Making It Live on Heroku

- **Node apps can be tricky to deploy**, so it's best to get comfortable early.
- **Heroku** simplifies deployment by offering a Platform-as-a-Service (PaaS) environment.
- By deploying now, you:
  - Show progress to others easily.
  - Continuously update the live app as you build.
  - Avoid end-of-project deployment stress.

Pre-requisites

1. Node.js
2. Express
3. Git
4. The Heroku CLI

Downloadable links

1. https://nodejs.org/en/download/
2. npm install express-generator -g
3. https://git-scm.com/downloads
4. https://devcenter.heroku.com/articles/heroku-cli #install-the-heroku-cli

Steps to be followed for Deploying your app on the Heroku site

1. Register For Heroku and login in any browser
2. Download the Heroku CLI
   a. **$ heroku --version // type in your cmd or in terminal of your project**
   b. Verify if you installed successfully, if not, set the path in ENV
   c. O/P: **heroku/7.0.0 (darwin-x64) node-v8.0.0**
3. **Navigate to your project location in VS Code**
   a. **$ heroku login // type these in your project terminal**
   b. Complete the login process in the redirected web page or ignore if you login at initially

(**type the below codes** in your terminal of your project

1. **$ git init**
2. **$ heroku git:remote -a your-app-name**
3. **$ git add . (dot)**
4. **git commit -am "make it be better"**
5. After that you will see an error message naming **Please tell me who you are ?**
6. **$ git config --global user.email "your email id registered with heroku"**
7. **$ git commit -am "make it better"**
8. **$ git push heroku HEAD:master //** to push all your project files into Heroku Site
9. **$ heroku open**

Steps to be followed for Deploying your app on the Heroku site

1. Register For Heroku and login in any browser
2. Download the Heroku CLI
   a. **$ heroku --version** // **type in your cmd or in terminal of your project**
   b. Verify if you installed successfully, if not, set the path in ENV
   c. O/P: **heroku/7.0.0 (darwin-x64) node-v8.0.0**
3. **Navigate to your project location in VS Code**
   a. **$ heroku login** // **type these in your project terminal**
   b. Complete the login process in the redirected web page or ignore if you login at initially

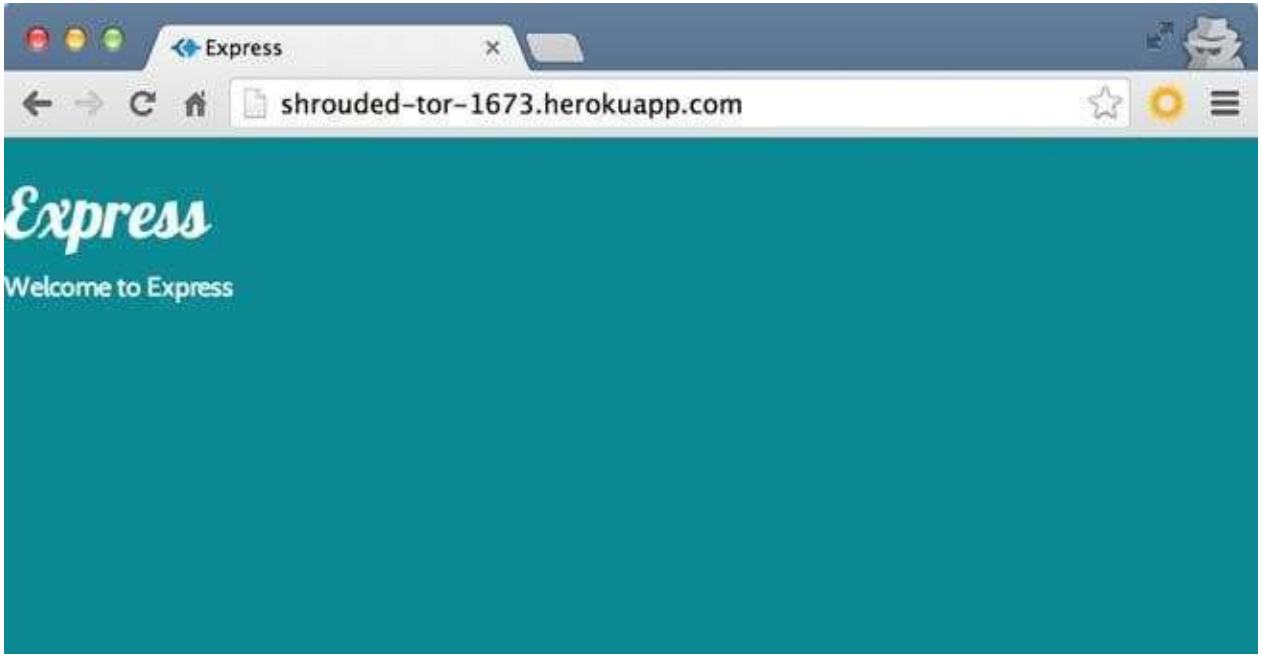
   (**type the below codes** in your terminal of your project

Steps to be followed for Deploying your app on the Heroku site

1. **$ git init**
2. **$ heroku git:remote -a your-app-name**
3. **$ git add . (dot)**
4. **git commit -am "make it be better”**
5. After that you will see an error message naming **Please tell me who you are ?**
6. **$ git config --global user.email “your email id registered with heroku”**
7. **$ git commit -m "make it better"**
8. **$ git push heroku HEAD:master //** to push all your project files into Heroku Site
9. **$ heroku open**

VIEWING THE APPLICATION ON A LIVE URL

- $ heroku open



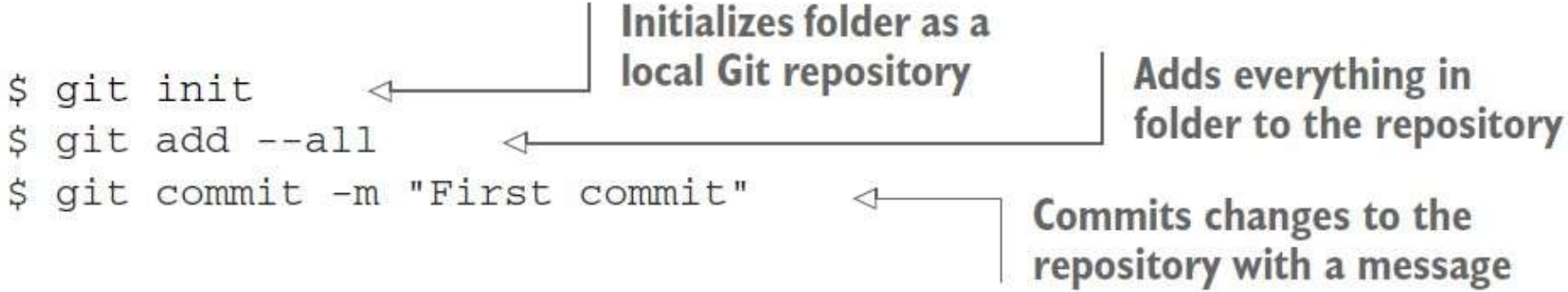
Pushing the site live using Git

STORING THE APPLICATION IN GIT

- The first action is storing the application in Git on your local machine. This process involves the following three steps:
  1. Initialise the application folder as a Git repository.
  2. Tell Git which files you want to add to the repository.
  3. Commit these changes to the repository.

Pushing the site live using Git

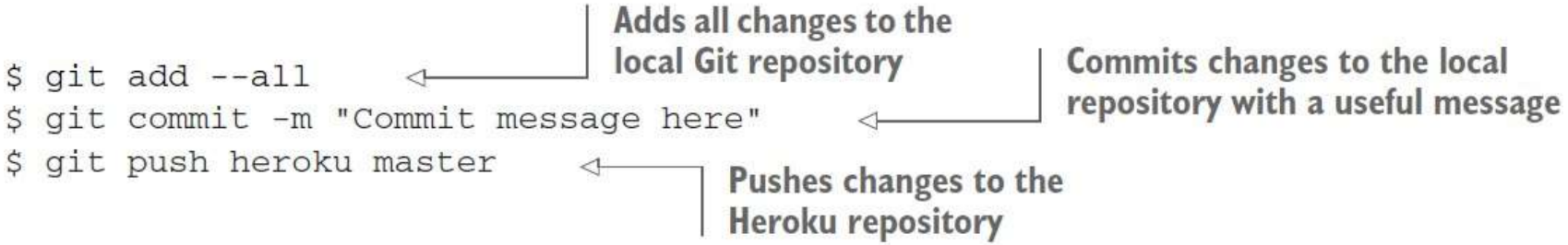
- Run the commands in the terminal



GIT CONFIG USERNAME AND EMAIL

- *git config --global user.email "you@example.com"*
- *git config --global user.name "Your Name“*
- *git push*

A SIMPLE UPDATE PROCESS



Pushing documents to Github steps:

```
echo "# MEAN2025" >> README.md

git init

git add README.md

git commit -m "first commit“

git config --global user.email "email id used for github login“

git config --global user.name "git user name"

git branch -M main

git remote add origin https://github.com/draukai/MEAN2025.git

git push -u origin main
```

Pushing documents to Github steps:

push an existing repository from the command line

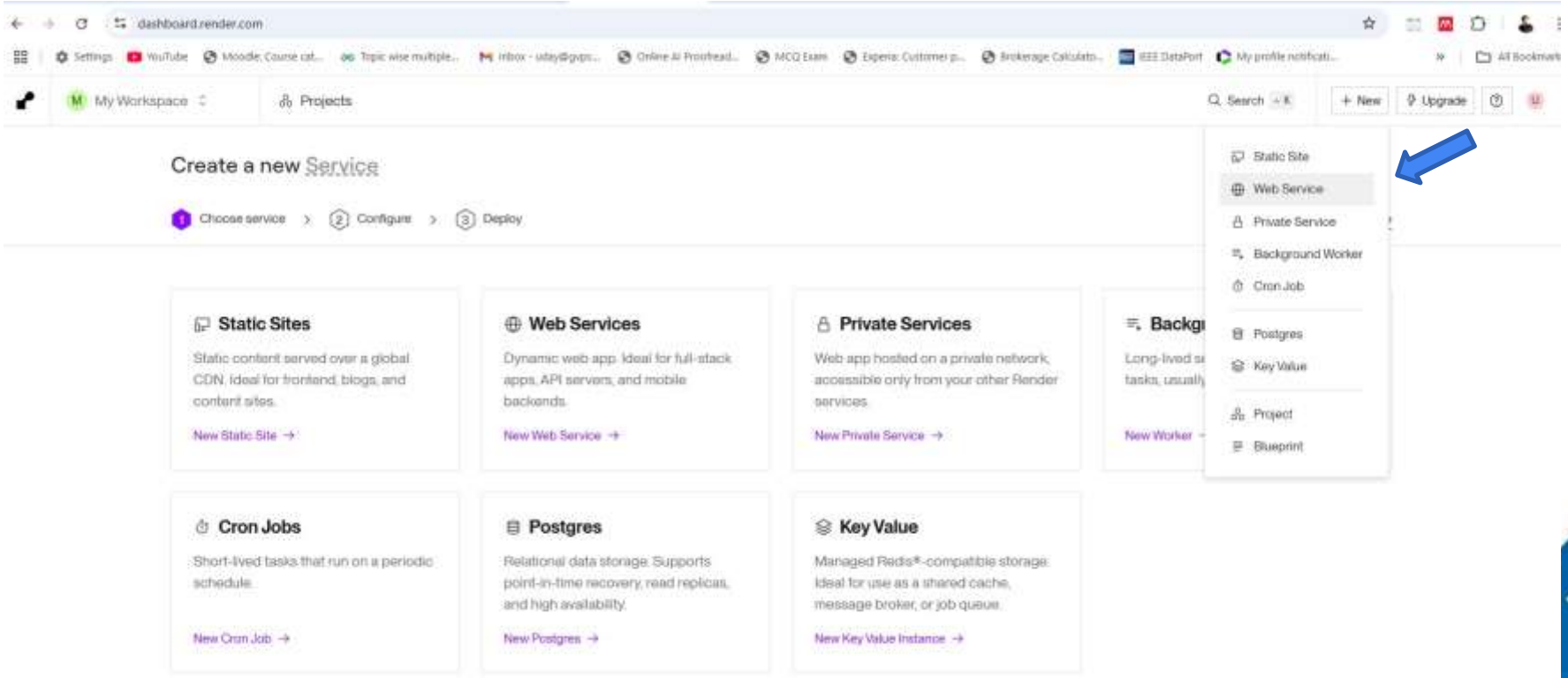
git remote add origin https://github.com/draukai/loc8r-c

git branch -M main

git push -u origin main

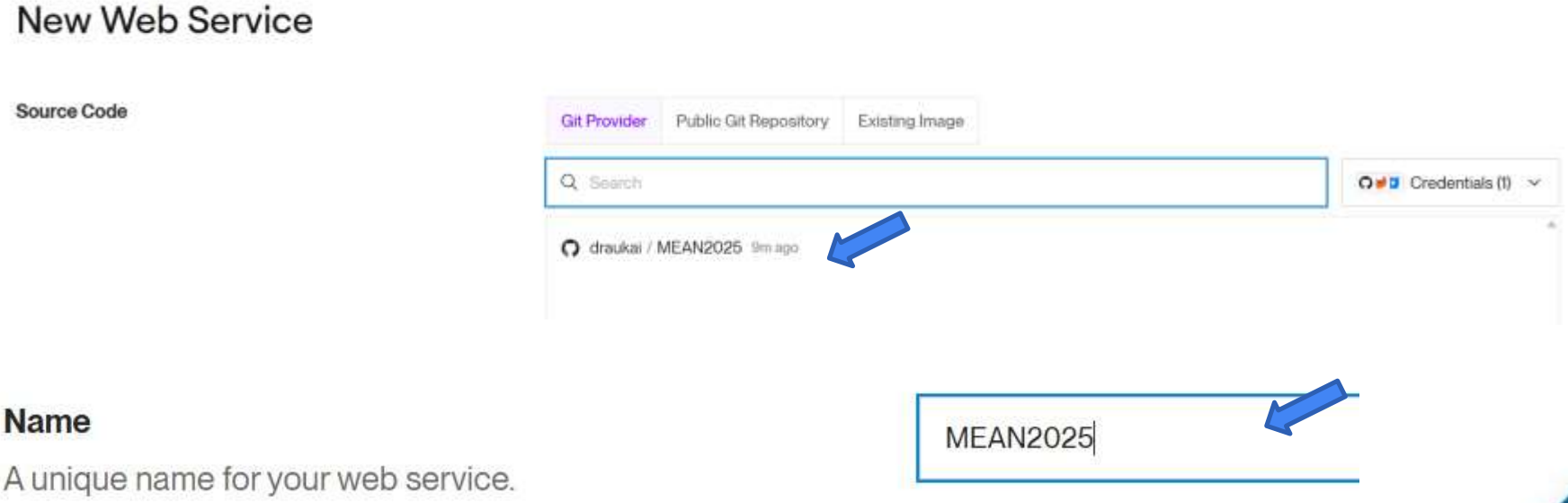
Open render.com and login through github

+Add new → Web service



Online Render

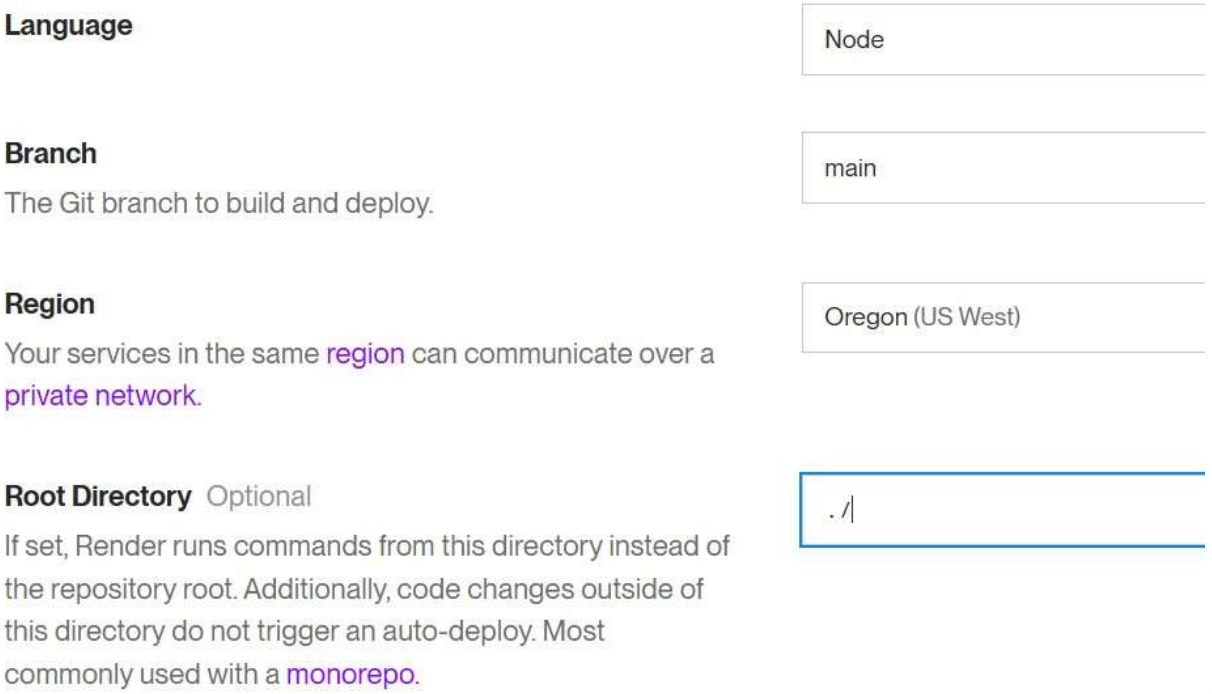
1. Select the git repository **2. Give the Name for the web service**



Online Render

3. Branch : main

4. Root Directory: ./



Online Render

5. Build Command: npm install

6. Start command: npm start

Build Command

Render runs this command to build your app before each deploy.

```
./ $ npm install
```

Start Command

Render runs this command to start your app with each deploy.

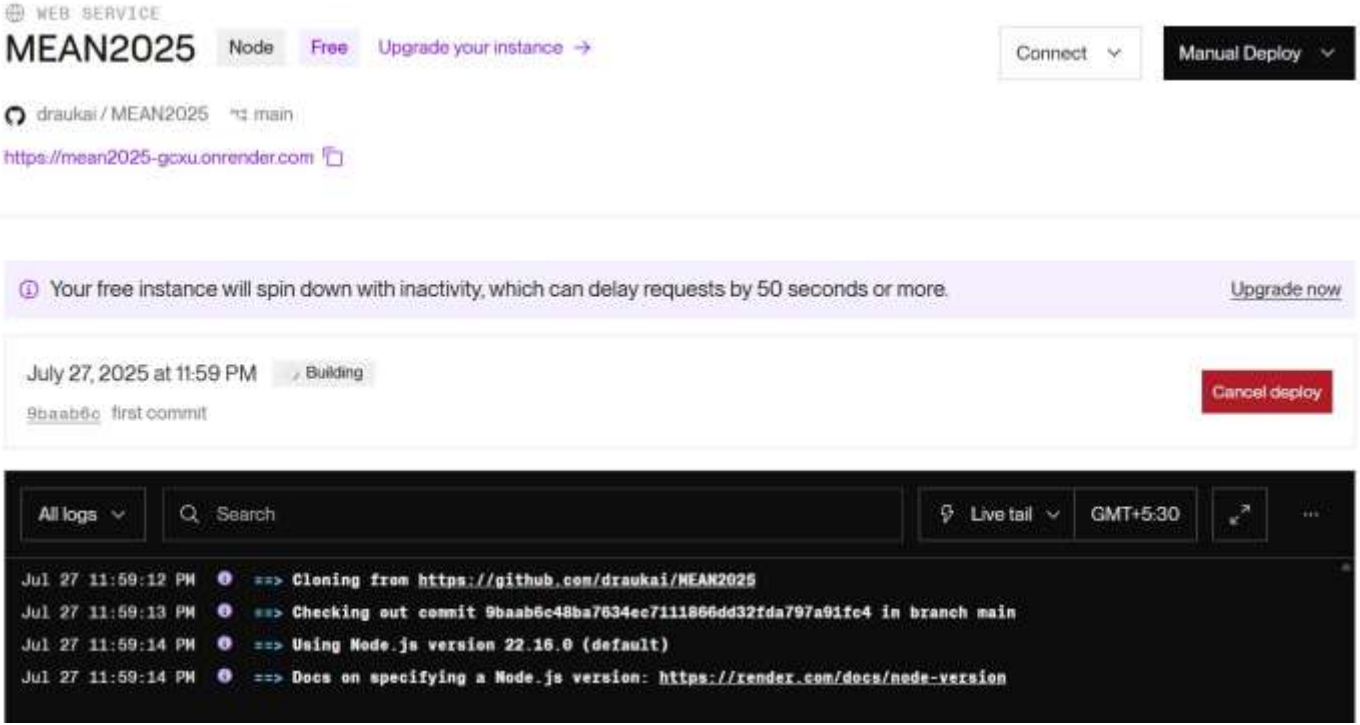
```
./ $ npm start
```

Instance Type

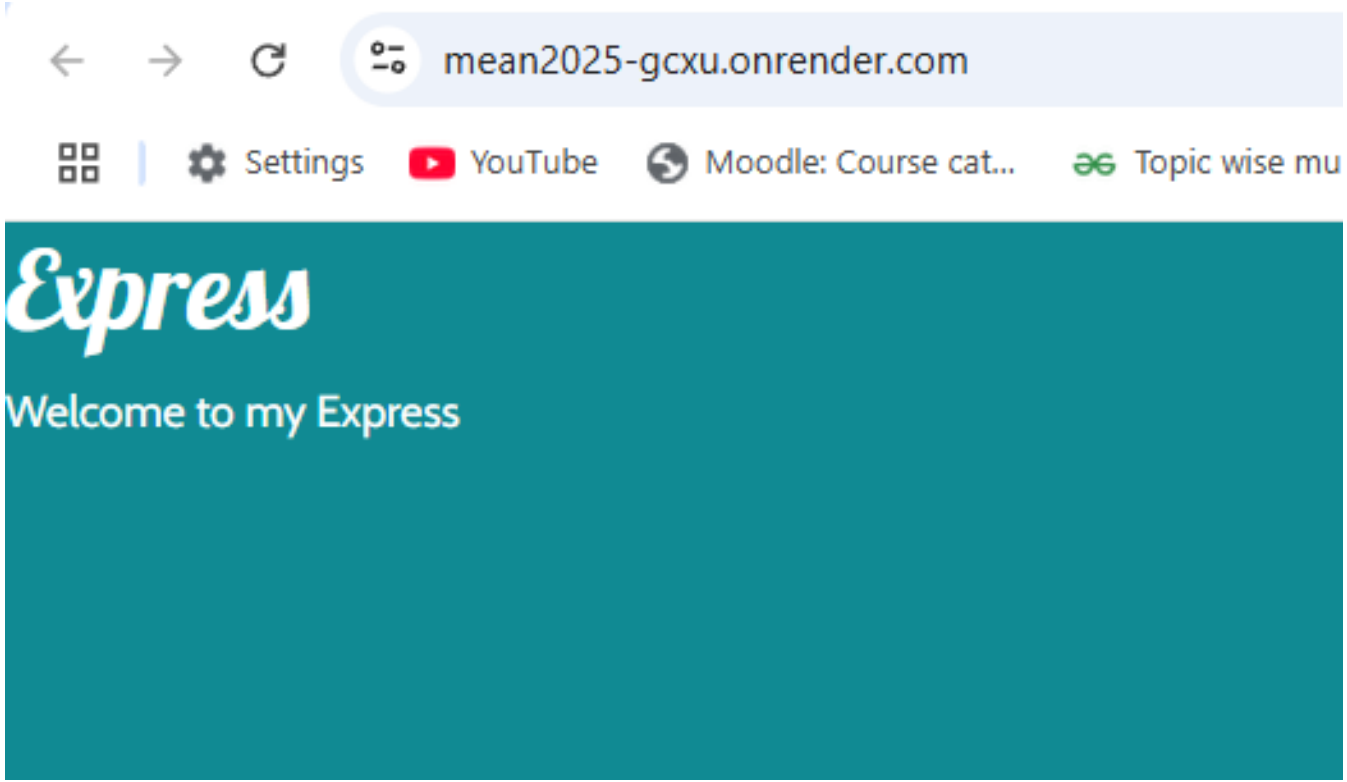
For hobby projects

Free	512 MB (RAM)
$0 / month	0.1 CPU

Online Render



Online Render



2. Building a static site with Node and Express

Using Express and Node to build a static site for testing views

- Two main steps are accomplished in this chapter, so two versions of the source code are available.
- The first version contains all the data in the views and represents the application
- The second version has the data in the controllers.

Using Express and Node to build a static site for testing views

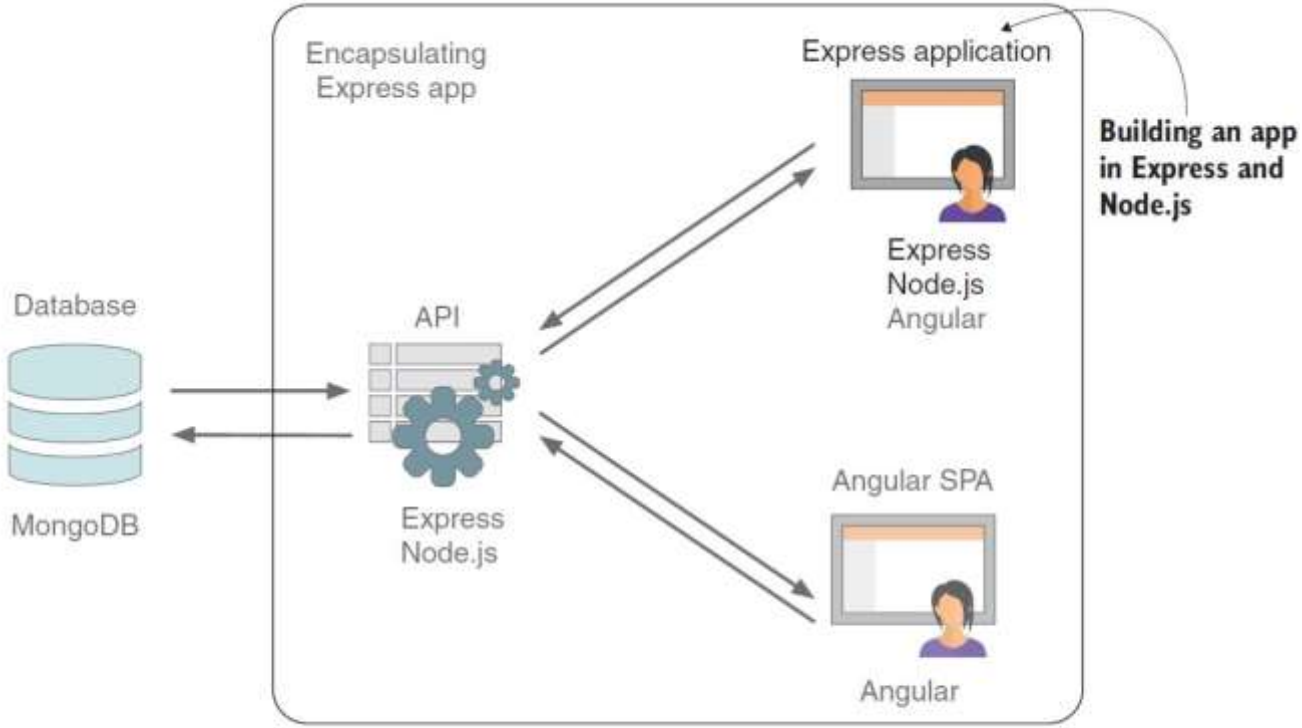
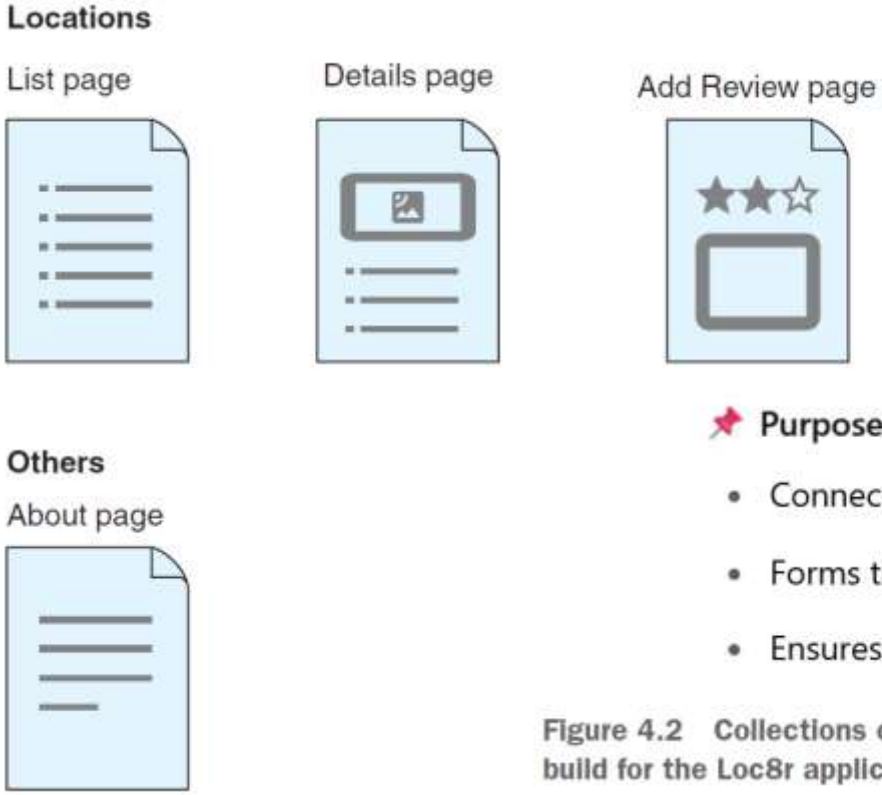


Figure 4.1 Using Express and Node to build a static site for testing views

Defining the routes in Express



📌 **Purpose of Routes**

- Connects **screens** of the application to **specific URL paths**
- Forms the basis of **Express routing**
- Ensures that each screen is accessible through a clear, structured URL

Figure 4.2 Collections of screens you'll build for the Loc8r application

Defining the routes in Express

ANURAG UNIVERSITY

Table 4.1 Defining a URL path, or route, for each of the screens in the prototype

Collection	Screen	URL path
Locations	List of locations (the homepage)	/
Locations	Location detail	/location
Locations	Location review form	/location/review/new
Others	About Loc8r	/about

Why Route Mapping Matters

- Helps organise pages logically before writing code
- Ensures a **standardised, scalable structure**
- Critical for linking the **frontend screens** to **backend logic**

Different controller files for different collections

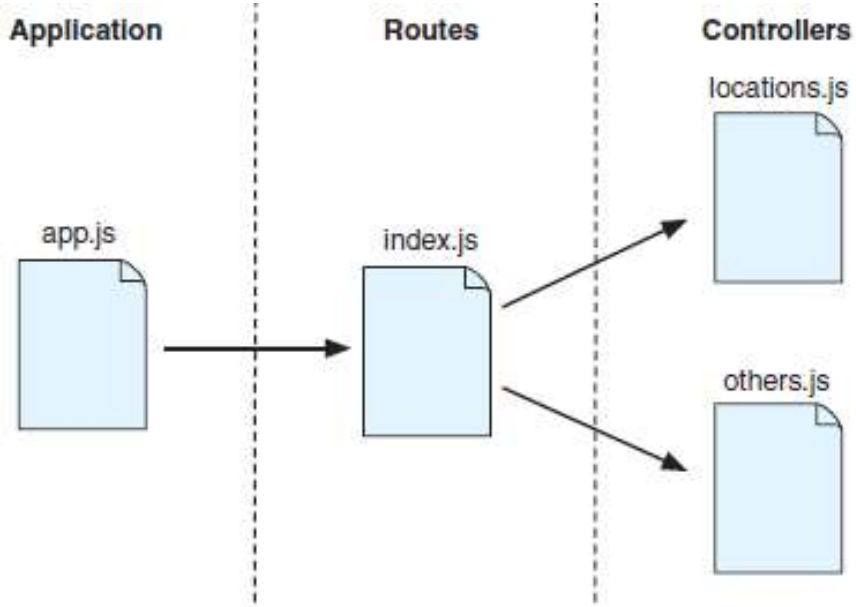


Figure 4.3 Proposed file architecture for routes and controllers in your application

📌 **Why Split Controller Files?**

- Controller logic was **moved out of route definitions**
- As the app grows, it's better **not to keep all controllers in one file**
- Logical approach: **split controllers by collections**

📦 **Collection-Based Organization**

- Based on defined collections:
  - Create separate controllers for **Locations** and **Others**
- Improves **scalability and maintainability**

🗂️ **Project Structure Concept**

- One **routes file**
- Multiple **controller files**, each mapped to a collection
- Promotes a **clean file-architecture approach**

🚀 **Starting Development**

- Use a single `routes/index.js` file
- Each controller manages logic for a **logical group of screens**
- Start by editing the **routes file first**, then move to controllers

REQUIRING THE CONTROLLER FILES

- These files will be called **locations.js** and **others.js**. These will be saved in **app_server/controllers**.
- In **index.js** you'll require both of these files and assign each to a relevant variable name, as shown in the following listing.

Listing 4.1 Requiring the controller files in app_server/routes/index.js

```
const express = require('express');
const router = express.Router();
const ctrlLocations = require('../controllers/locations');
const ctrlOthers = require('../controllers/others');
```

Replaces existing ctrlMain reference with two new requires

Now you have two variables that you can reference in the route definitions, containing different collections of routes.

SETTING UP THE ROUTES

- Define routes for **Locations** and **About** pages
- Each route maps a **URL** to a **controller function**
- Use require() to include:
  - locations.js
  - others.js
- All routes go inside **routes/index.js**

SETTING UP THE ROUTES

- routes/index.js

Listing 4.2 Defining the routes and mapping them to controllers

```
const express = require('express');
const router = express.Router();
const ctrlLocations =
     require('../controllers/locations');
const ctrlOthers = require('../controllers/others');

/* Locations pages */
router.get('/', ctrlLocations.homelist);
router.get('/location', ctrlLocations.locationInfo);
router.get('/location/review/new', ctrlLocations.addReview);

/* Other pages */
router.get('/about', ctrlOthers.about);

module.exports = router;
```

Requires controller files

Defines location routes and maps them to controller functions

Defines other routes

Building basic controllers

- ***Setting up controllers***
- You currently have one file: the main.js file in the controllers folder (in the app_server folder), which has a single function that's controlling the homepage.

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

You don't want a "main" controller file anymore, but you can use this one as a template. Start by renaming this file others.js.

Adding The Others Controllers

- Rename the existing `index` controller `about`;
- keep the same view template for now; and
- update the `title` property to something relevant.

others.js

Listing 4.3 Others controller file

```
/* GET 'about' page */
const about = (req, res) => {
  res.render('index', { title: 'About' });
};
module.exports = {
  about
};
```

Defines the route, using the same view template but changing the title to About

Updates the export to reflect the name change

ADDING THE LOCATIONS CONTROLLERS

In the controllers folder, create a file called locations.js, and create and export three basic controller functions: `homelist`, `locationInfo`, and `addReview`. The following listing shows how this file should look.

Listing 4.4 Locations controller file

```
/* GET 'home' page */
const homelist = (req, res) => {
  res.render('index', { title: 'Home' });
};

/* GET 'Location info' page */
const locationInfo = (req, res) => {
  res.render('index', { title: 'Location info' });
};

/* GET 'Add review' page */
const addReview = (req, res) => {
  res.render('index', { title: 'Add review' });
};
```

```
module.exports = {
  homelist,
  locationInfo,
  addReview
};
```

Testing the controllers and routes

$nodemon

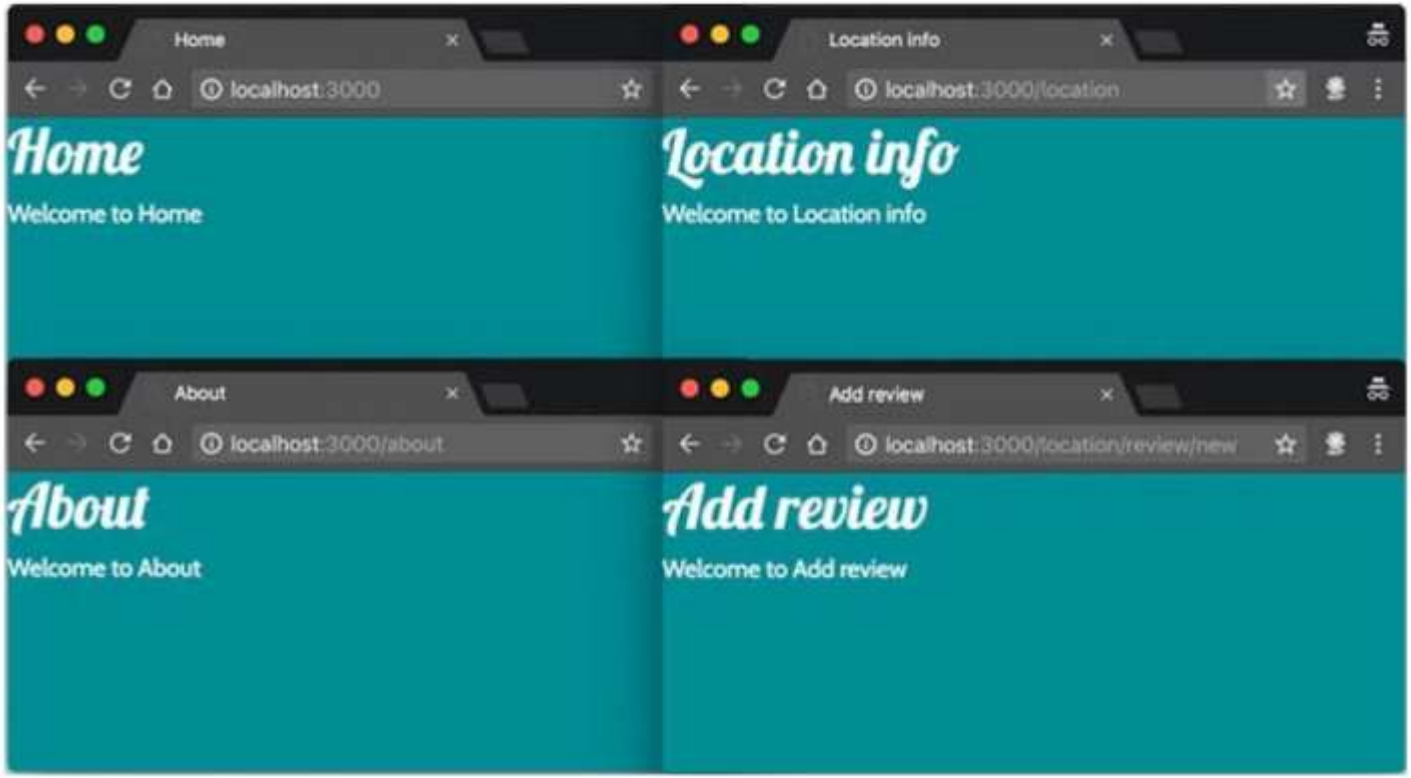


Figure 4.4 Screenshots of the four routes created so far, with different heading text coming through from the specific controllers associated with each route

3. Creating some views

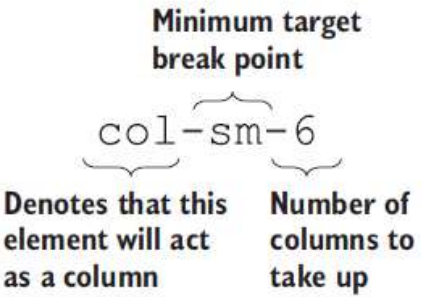
- **Pug/Jade** is the template engine you're using in Express
- **Bootstrap** is a front-end layout framework that makes it easy to build a responsive website that looks different on desktop and mobile devices.
- Bootstrap uses a **12-column grid**. No matter the size of the display you're using, there will always be these 12 columns.
- The **fundamental** concept of Bootstrap is that you can define how many columns an element uses, and this number can be different for different screen sizes.
- Bootstrap has various **CSS references** that let you target up to five different pixel width breakpoints for your layouts.

3. Creating some views

A look at Bootstrap

Table 4.2 Breakpoints that Bootstrap sets to target different types of devices

Breakpoint name	CSS reference	Example device	Width
Extra-small devices	`(none)`	Small phones	Fewer than 576
Small devices	`sm`	Smartphones	576 or more
Medium devices	`md`	Tablets	768 or more
Large devices	`lg`	Laptops	992 or more
Extra-large devices	`xl`	External monitors	1,200 or more



Example

- DIV ONE
- DIV TWO

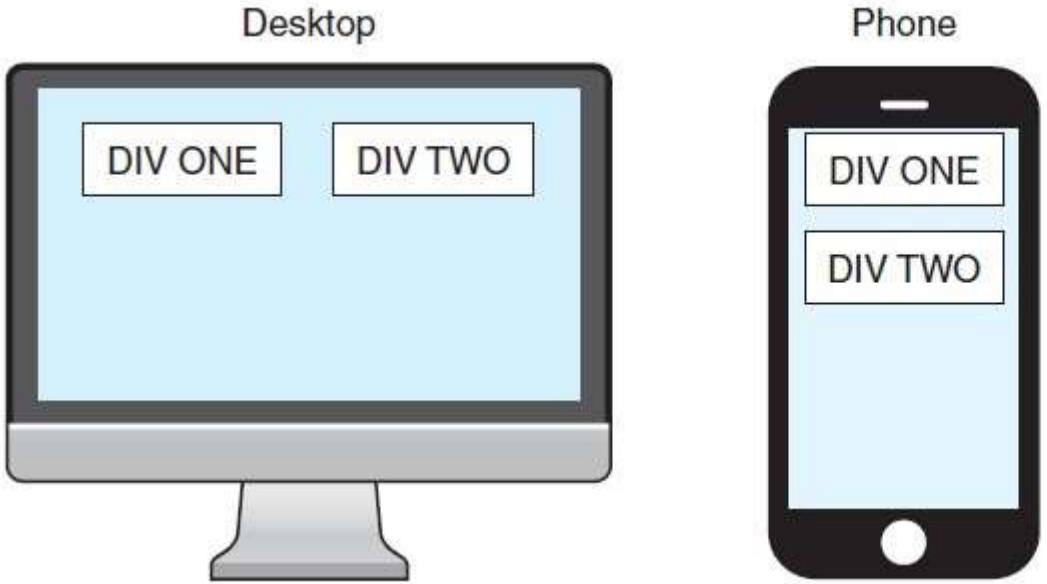


Figure 4.5 Bootstrap's responsive column system on a desktop and mobile device. CSS classes are used to determine the number of columns (out of 12) that each element should take up at different screen resolutions.

Setting up the HTML framework with Pug templates and Bootstrap

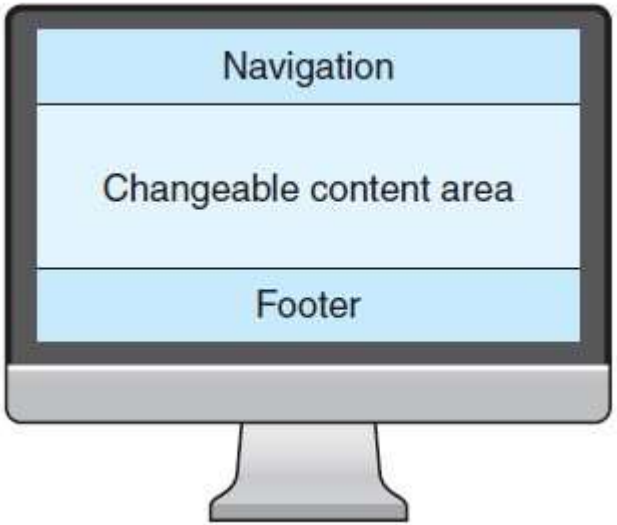


Figure 4.6 Basic structure of the reusable layout, comprising a standard navigation bar and footer with an extendable, changeable content area in between

views/layout.pug

```
doctype html
html
  head
    meta(name='viewport', content='width=device-width, initial-scale=1.0')
    title= title
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/all.min.css')
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    nav.navbar.fixed-top.navbar-expand-md.navbar-light
      .container
        a.navbar-brand(href='/') Loc8r
          button.navbar-toggler(type='button', data-toggle='collapse',
          ➥data-target='#navbarMain')
            span.navbar-toggler-icon
        #navbarMain.navbar-collapse.collapse
          ul.navbar-nav.mr-auto
            li.nav-item
              a.nav-link(href='/about/') About
```

Starting layout with a fixed navigation bar

Extendable content block is now wrapped in a container div

views/layout.pug

```
.container.content
  block content

  footer
    .row
      .col-12
        small &copy; Getting MEAN - Simon Holmes/Clive Harber 2018

script(src='https://code.jquery.com/jquery-3.3.1.slim.min.js',
➥integrity='sha384-
➥q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo',
➥crossorigin='anonymous')
script(src='https://cdnjs.cloudflare.com/ajax/libs/
➥popper.js/1.14.3/umd/popper.min.js' integrity='sha384- ZMP7rVo
➥]3mIykV+2+9J3UJ46jBk0WLaUAdn689a CwoqbBJiSnjAK/l8WvCWPIPm49',
➥crossorigin='anonymous')
script(src='/javascripts/bootstrap.min.js')
```

Simple copyright footer in the same container as the content block

After editing layout.pug

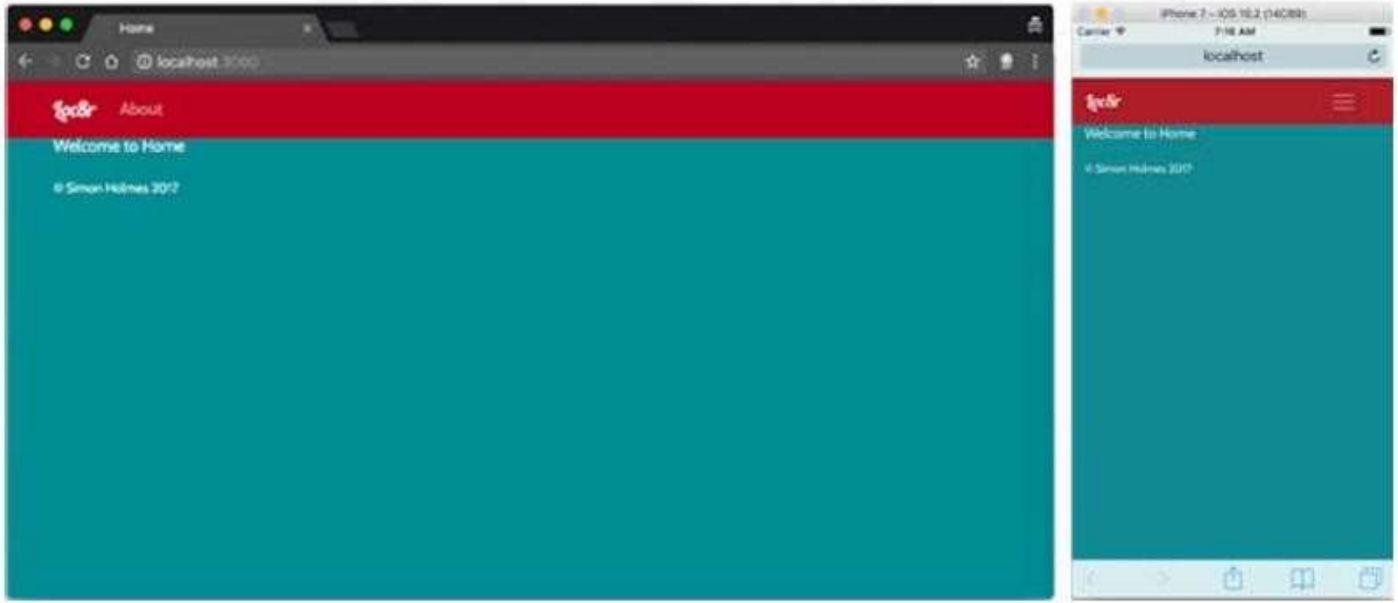


Figure 4.7 The homepage after the layout template has been set up. Bootstrap automatically collapsed the navigation on the small screen size of the phone. The navigation bar overlaps the content, but that problem will be fixed when the content layouts are created.

Building a template

DEFINING A LAYOUT

- The primary aim for the homepage is to display a list of locations. Each location needs to have a name, an address, the distance away from the user, user ratings, and a facilities list. You also want to add a header to the page and some text to put the list in context, so that users know what they're looking at when they visit.

Building a template

DEFINING A LAYOUT

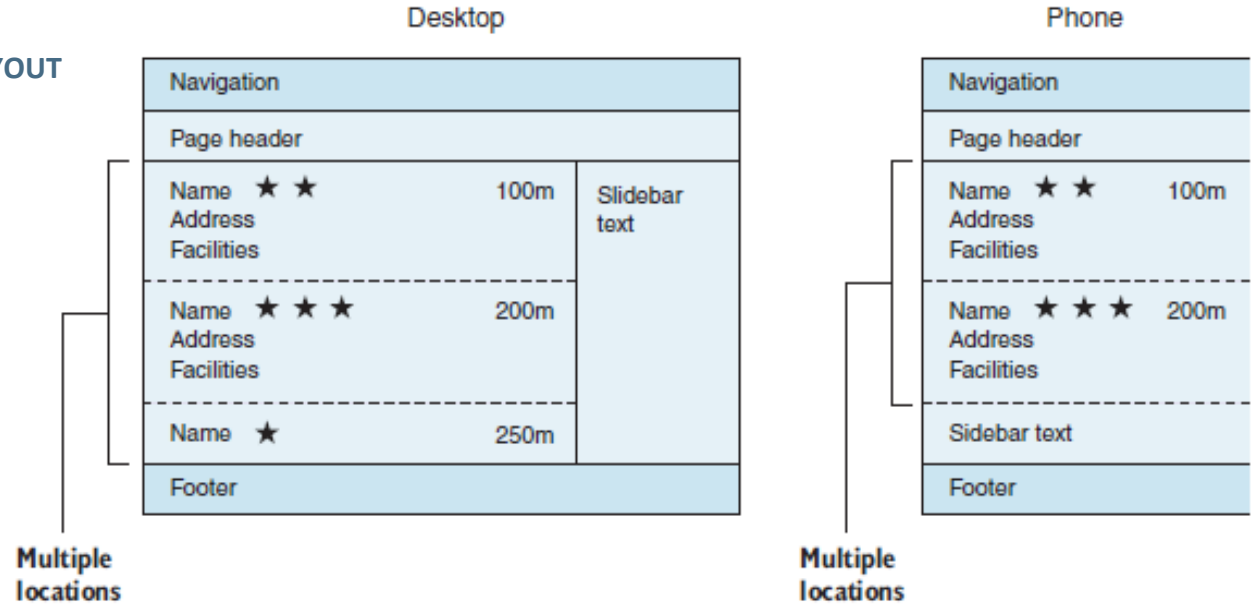


Figure 4.8 Desktop and mobile layout sketches for the homepage. Sketching the layouts for a page can give you a quick idea of what you're going to build without getting distracted by the intricacies of Adobe Photoshop or the technicalities of code.

SETTING UP THE VIEW AND THE CONTROLLER

- The first step is creating a new view file and linking it to the controller.
- In the app_server/views folder, make a copy of the **index.pug** view, and save it in the same folder as **locations-list.pug.**
- The second step is telling the controller for the homepage that you want to use this new view.
- The controller for the homepage is in the **locations.js** file in **app_server/controllers.**
- Update this file to change the view called by the homelist controller, as shown in the following code snippet (modifications in bold):

```
const homelist = (req, res) => {
  res.render('locations-list', { title: 'Home' });
};
```

CODING THE TEMPLATE: PAGE LAYOUT

At this point, you need to take a first stab at how many of the 12 Bootstrap columns you want each element to take up on different devices. The following code snippet shows the **layout of the three distinct areas** of the **Loc8r** List page in locations-list.pug:

```
.row.banner
  .col-12
    h1 Loc8r
      small  Find places to work with wifi near you!
.row
  .col-12.col-md-8
    p List area.
  .col-12.col-md-4
    p.lead Loc8r helps you find places to work when out and about.
```

Page header that fills the entire width of the screen

Container for the list of locations, spanning all 12 columns on extra-small and small devices, and 8 columns on medium devices and larger

Container for secondary or sidebar information, spanning all 12 columns on extra-small and small devices, and 4 columns on medium devices and larger

CODING THE TEMPLATE: LOCATIONS LIST

The following code snippet shows what you might come up with for a single location to replace the `p List area` placeholder in locations-list.pug:

```
.card
  .card-block

h4
  a(href="/location") Starcups
  small  
    i.fas.fa-star
    i.fas.fa-star
    i.fas.fa-star
    i.far.fa-star
    i.far.fa-star
  span.badge.badge-pill.badge-default.float-right 100m
p.address 125 High Street, Reading, RG6 1PS
.facilities
  span.badge.badge-warning Hot drinks
  span.badge.badge-warning Food
  span.badge.badge-warning Premium wifi
```

Creates a new Bootstrap card and card block to wrap the content

Name of the listing and a link to the location

Uses Font Awesome icons to output a star rating

Uses Bootstrap's badge helper class to display the distance away

Address of the location

Facilities of the location, output using Bootstrap's badge classes

CODING THE TEMPLATE: LOCATIONS LIST

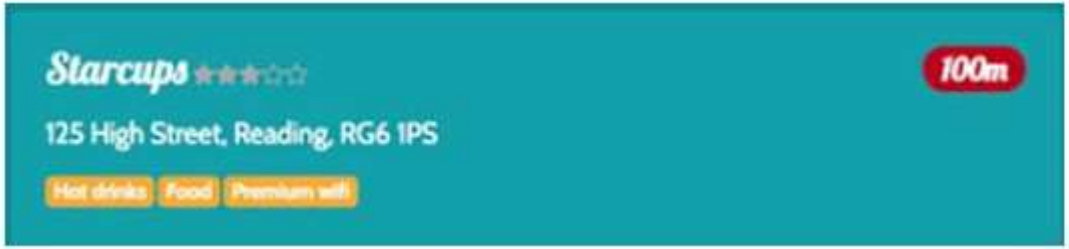


Figure 4.9 Onscreen rendering of a single location on the List page

Listing 4.6 Complete template for app_server/views/locations-list.pug

```
extends layout

block content
  .row.banner          <-------- Starts header area
    .col-12
      h1 Loc8r
        small  Find places to work with wifi near you!
```

CODING THE TEMPLATE: LOCATIONS LIST

```
.row
  .col-12.col-md-8
    .card
      .card-block
        h4
          a(href="/location") Starcups
          small  
            i.fas.fa-star
            i.fas.fa-star
            i.fas.fa-star
            i.far.fa-star
            i.far.fa-star
          span.badge.badge-pill.badge-default.float-right 100m
        p.address 125 High Street, Reading, RG6 1PS
        p.facilities
          span.badge.badge-warning Hot drinks
          span.badge.badge-warning Food
          span.badge.badge-warning Premium wifi
  .col-12.col-md-4
    p.lead Looking for wifi and a seat? Loc8r helps you find places to
    ➥work when out and about. Perhaps with coffee, cake or a pint?
    ➥Let Loc8r help you find the place you're looking for.
```

Starts responsive main listing column section

An individual listing; duplicates this section to create a list of multiple items

Sets up sidebar area and populates it with some content

CODING THE TEMPLATE: LOCATIONS LIST

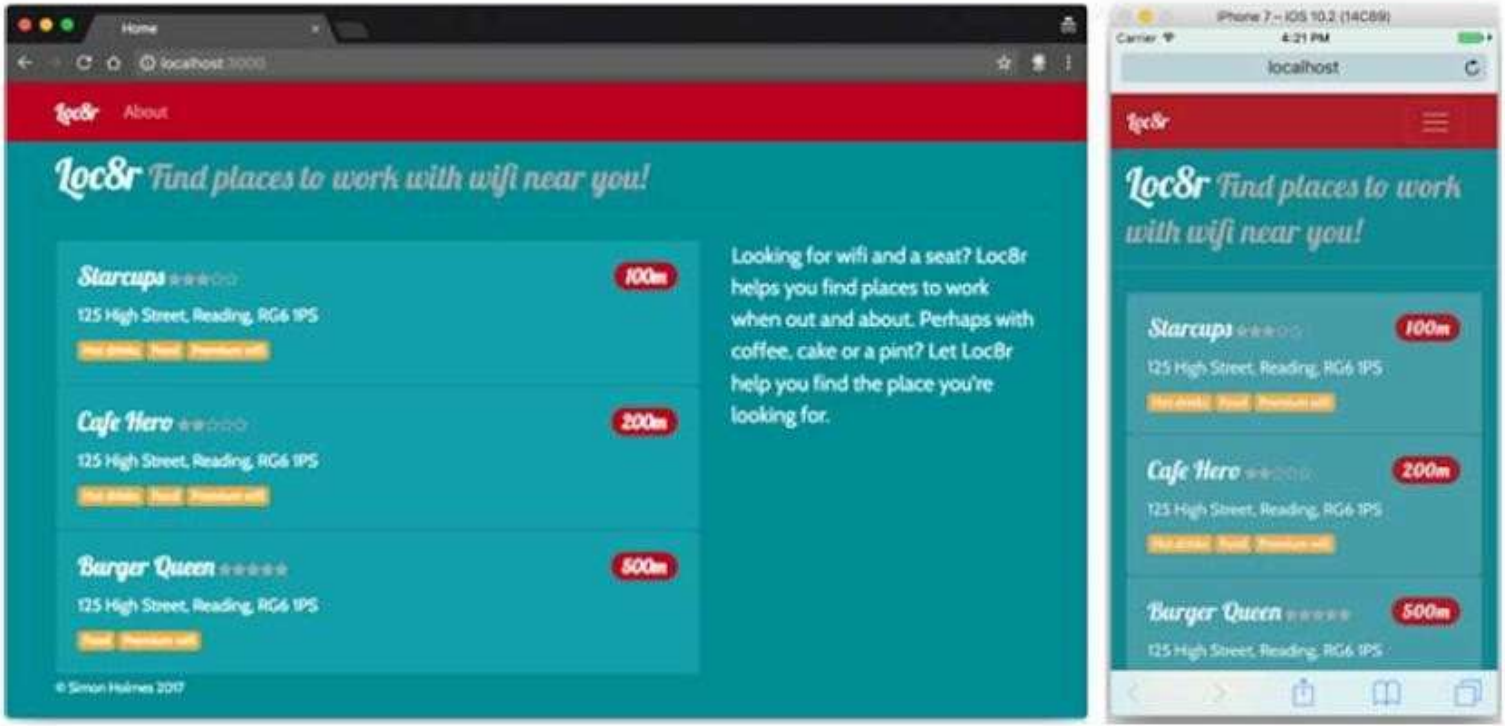


Figure 4.10 Responsive template for the homepage in action on different devices

4. Adding the rest of the views

- Details
- Add Review
- About

- Details page for an individual location.
- This page needs to display all the information about a location, including

Details page

- Name
- Address
- Rating
- Opening hours
- Facilities
- Location map
- Reviews, each with
  - Rating
  - Reviewer name
  - Review date
  - Review text
  - Button to add a new review
  - Text to set the context of the page

Preparation

- The first step is updating the controller for this page to use a different view. Look for the locationInfo controller in the locations.js file in app_server/controllers.
- Change the name of the view to location-info, as shown in the following code snippet:

```
const locationInfo = (req, res) => {
  res.render('location-info', { title: 'Location info' });
};
```

The next step is obtaining a key to access the Google Maps API. To get your keys, you need to sign up for an account, if you don't already have one, at the following address:

```
https://developers.google.com/maps/documentation/javascript/
    get-api-key?utm_source=geoblog&utm_medium=social&utm_campaign=
    2016-geo-na-website-gmedia-blogs-us-blogPost&utm_content=TBC
```

THE VIEW

- Create a new file in **app_server/views** and save it as **location-info.pug**. Remember that for the purposes of this stage in the prototype development, you're generating clickable pages with the data hardcoded directly into them.

Listing 4.7 View for the Details page, app_server/views/location-info.pug

```
extends layout

block content
  .row.banner
    .col-12                          Starts with page header
      h1 Starcups
  .row
    .col-12.col-lg-9                 Sets up nested responsive
      .row                           columns needed for the
        .col-12.col-md-6             template
          p.rating
            i.fas.fa-star
            i.fas.fa-star
            i.fas.fa-star
            i.far.fa-star
            i.far.fa-star
```

Figure 4.11 shows how this layout looks in a browser and on a mobile device.

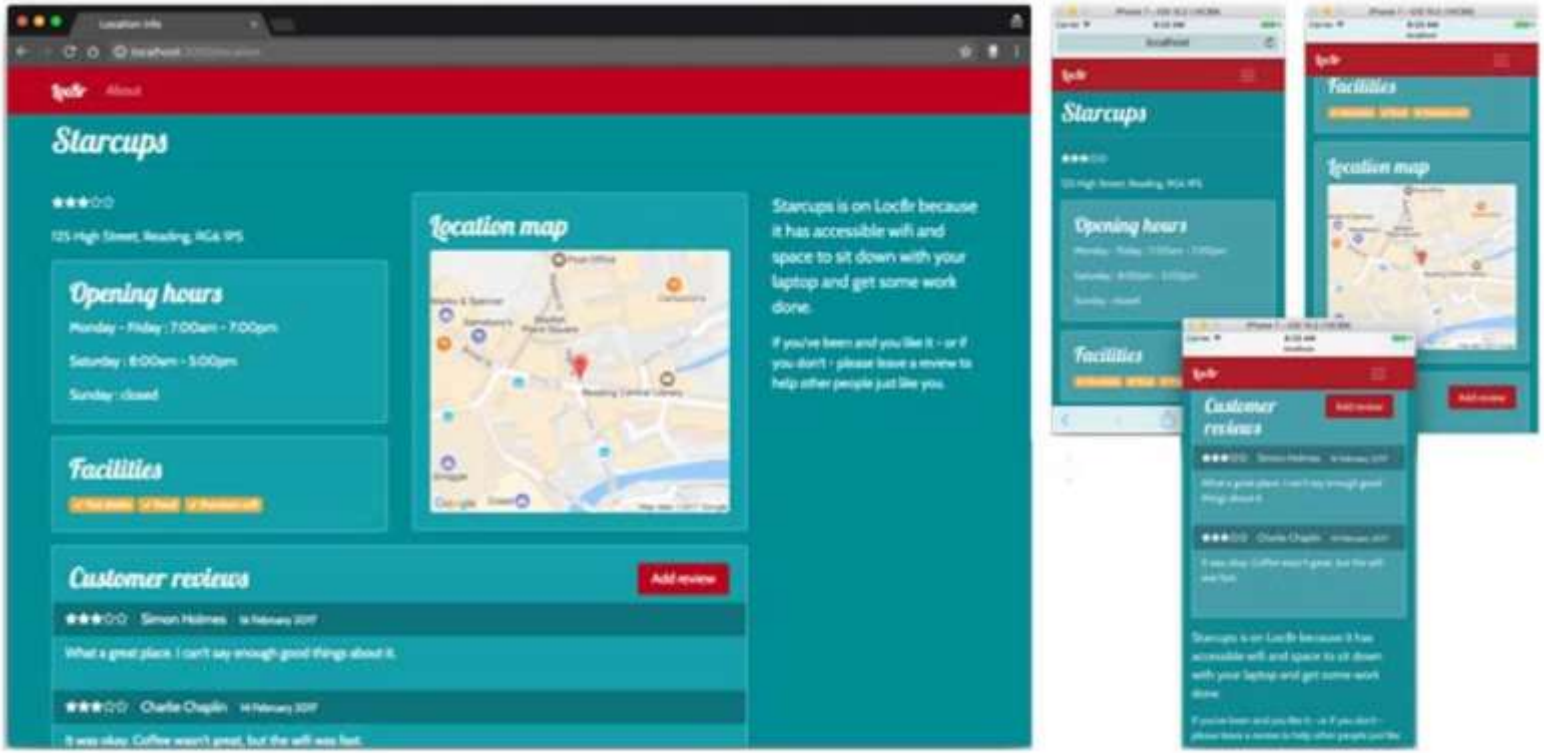


Figure 4.11 Details page layout on desktop and mobile devices

Adding the Review page

The first step is updating the controller to reference a new view. In app_server/controllers/locations.js, change the `addReview` controller to use the new view `location-review-form`, as in the following code snippet:

```
const addReview = (req, res) => {
  res.render('location-review-form', { title: 'Add review' });
};
```

The second step is creating the view itself. In the views folder **app_server/views**, create a new file called **location-review-form.pug**.

views/location-review-form.pug

Listing 4.8 View for the Add Review page, app_server/views/location-review-form.pug

```
extends layout

block content
  .row.banner
    .col-12
      h1 Review Starcups
  .row
    .col-12.col-md-8
      form(action="/location", method="get", role="form")
        .form-group.row
          label.col-10.col-sm-2.col-form-label(for="name") Name
          .col-12.col-sm-10
            input#name.form-control(name="name")
        .form-group.row
          label.col-10.col-sm-2.col-form-label(for="rating") Rating
          .col-12.col-sm-2
            select#rating.form-control.input-sm(name="rating")
              option 5
              option 4
              option 3
              option 2
              option 1
        .form-group.row
          label.col-sm-2.col-form-label(for="review") Review
          .col-sm-10
            textarea#review.form-control(name="review", rows="5")
        button.btn.btn-primary.float-right Add my review
    .col-12.col-md-4
```

Sets the form action to /location, and the method to get

Input field for reviewer to leave their name

Drop-down select box for rating 1 to 5

Text area for the text content of the review

Submit button for the form

Review page

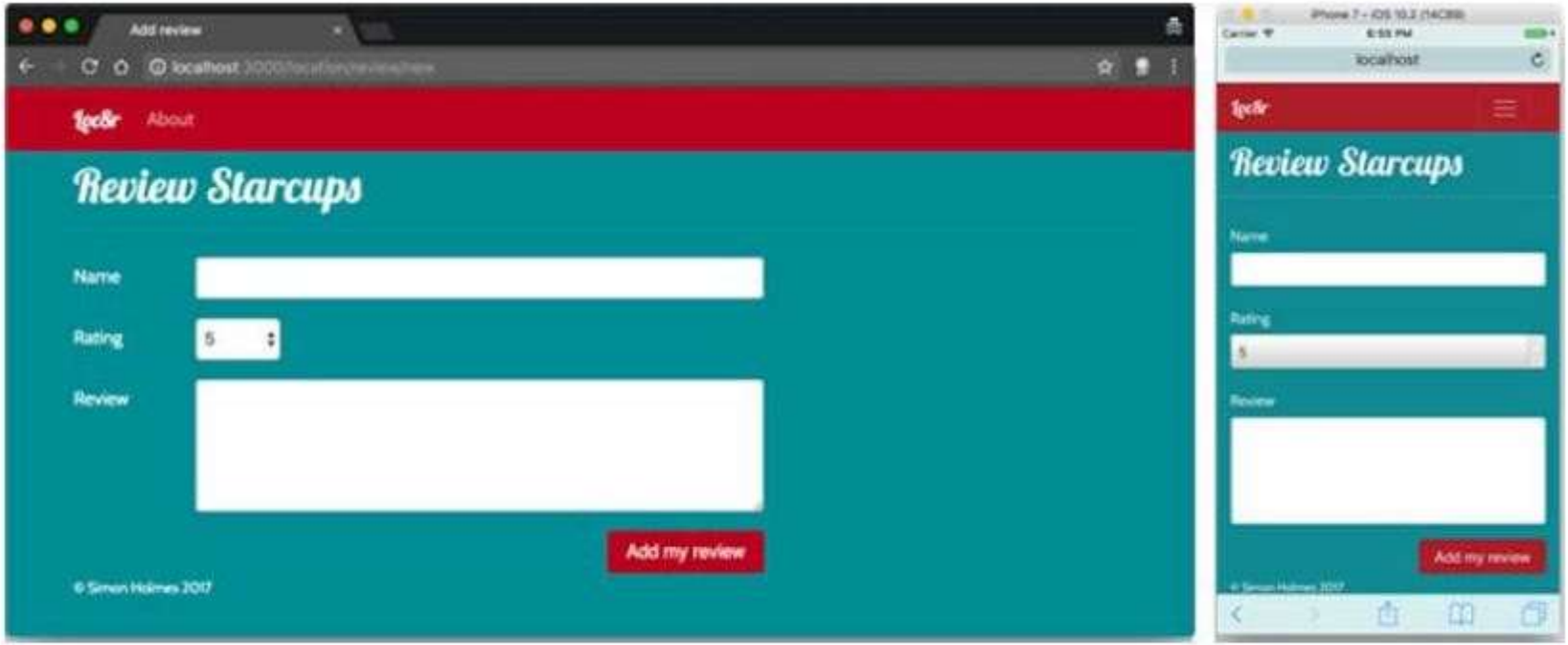


Figure 4.12 Complete Add Review page in desktop and mobile view

Adding the About page

- The controller for the About page is in the others.js file in app_server/controllers.
- You're looking for the controller called about, and you want to change the name of the view to **generic-text**, as in the following code snippet:

  const about = (req, res) => {
  *res.render('**generic-text**', { title: 'About' });*
  };

In terminal, you need to be in the root folder of the application. Then issue these commands:

```
$ git add --all
$ git commit –m "Adding the view templates"
$ git push heroku master
```

views/generic-text.pug

Listing 4.9 View for text-only pages: app_server/views/generic-text.pug

```
extends layout
block content
  .row.banner
    .col-12
      h1= title
  .row
    .col-12.col-lg-8
      p
        | Loc8r was created to help people find places to sit down and
        ➥get a bit of work done.
        | <br /><br />
        | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc
        ➥sed lorem ac nisi dignissim accumsan.
```

Use | to create lines of plain text within a <p> tag.

Localhost:3000

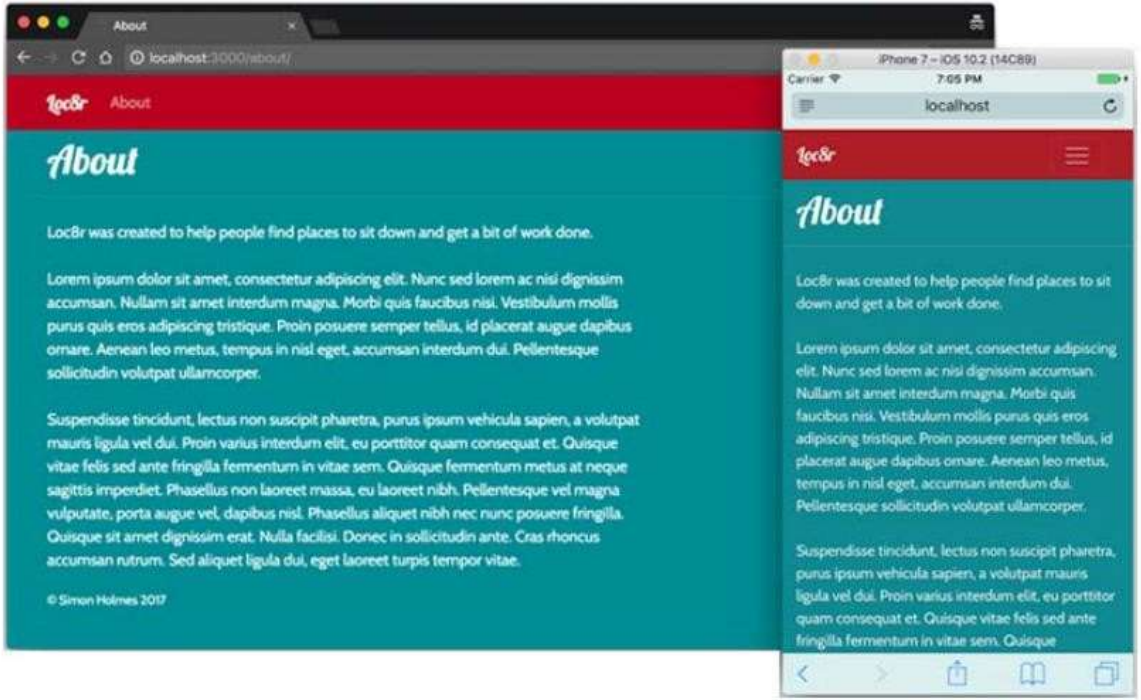


Figure 4.13 Generic text template rendering the About page

5. Taking the data out of the views and making them smarter

- A goal of the MVC architecture is to have views without content or data.
- Consider the **MVC architecture**:
  - the **model** holds the data;
  - the **controller** processes the data; and,
  - finally, the **view** renders the processed data.
- To make the views smarter and do what they're intended to do, you need to take the data and content out of the views and put it in the controllers.
- Update the Pug file to contain variables in place of the content and put the content as variables in the controller.
- Then the controller can pass these values into the view.

Taking the data out of the views and making them smarter

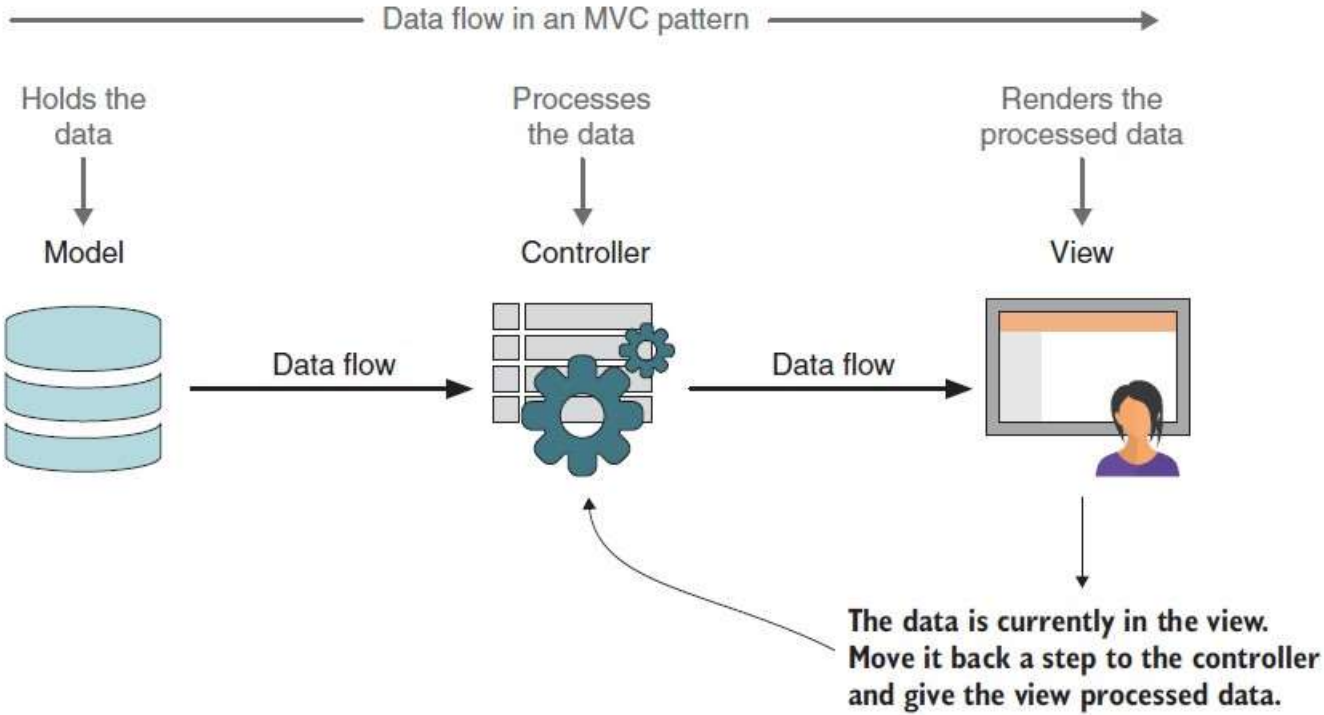


Figure 4.14 How the data should flow in an MVC pattern, from the model through the controller to the view. At this point in the prototype, your data is in the view, but you want to move it a step back into the controller.

Taking the data out of the views and making them smarter

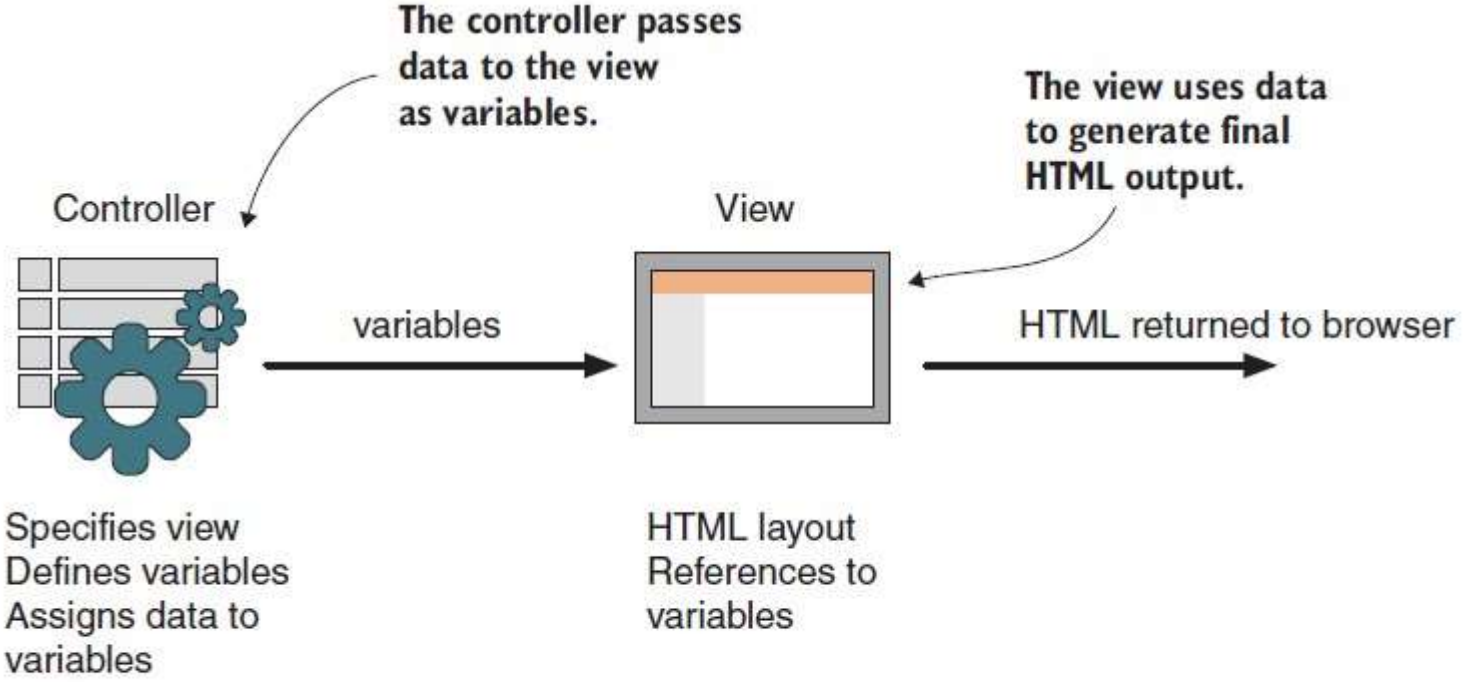


Figure 4.15 When the controller specifies the data, it passes the data to the view as variables; the view uses that data to generate the final HTML that's delivered to the user.

Moving data from the view to the controller

- Shift static content from the **locations-list.pug** view into the controller (**locations.js**).
- Current locations-list.pug file content

```
.row.banner
  .col-12
    h1 Loc8r
      small  Find places to work with wifi near you!
```

Large-font page title

Smaller-font strapline for page

The homepage controller currently looks like the following:

```
const homelist = (req, res) => {
  res.render('locations-list', { title: 'Home' });
};
```

Updating The Controller

Make these changes to the controller first, as follows (modifications in bold):

```
const homelist =  (req, res) => {
  res.render('locations-list', {
    title: 'Loc8r - find a place to work with wifi',
    pageHeader: {
      title: 'Loc8r',
      strapline: 'Find places to work with wifi near you!'
    }
  });
};
```

New nested pageHeader object containing properties for the title and the strapline of the page

- The **title** and the **strapline** are grouped within a **pageHeader** object.
- This approach is a good habit to get into and will make the controllers easier to update and maintain.

Updating The View

- To reference the page header strapline in the **locations-list.pug** view, use **pageHeader.strapline**.
- The following code snippet shows the page header section of the view (modifications in bold):

```
.row.banner
  .col-12
    h1= pageHeader.title
     small
      #{pageHeader.strapline}
```

= signifies that the following content is buffered code—in this case, a JavaScript object.

#{} delimiters are used to insert data into a specific place, such as part of a piece of text.

Dealing with complex, repeating data patterns

1. Controller: Send Array of Objects to the View
You've already done this for your locations. Your controller should look like:

Here, **each location is an object** with the fields you need:
name, address, rating, facilities (array), and distance.

The controller sends the array as locations to your Pug view.

```
const homelist = (req, res) => {
  res.render('locations-list', {
    title: 'Loc8r - find a place to work with wifi',
    pageHeader: {
      title: 'Loc8r',
      strapline: 'Find places to work with wifi near you!'
    },
    // Array of location objects
    locations: [
      {
        name: 'Starcups',
        address: '125 High Street, Reading, RG6 1PS',
        rating: 3,
        facilities: ['Hot drinks', 'Food', 'Premium wifi'],
        distance: '100m'
      },
      {
        name: 'Cafe Hero',
        address: '125 High Street, Reading, RG6 1PS',
        rating: 4,
        facilities: ['Hot drinks', 'Food', 'Premium wifi'],
        distance: '200m'
      },
      {
        name: 'Burger Queen',
        address: '125 High Street, Reading, RG6 1PS',
        rating: 2,
        facilities: ['Food', 'Premium wifi'],
        distance: '250m'
      }
    ]
  });
};
```

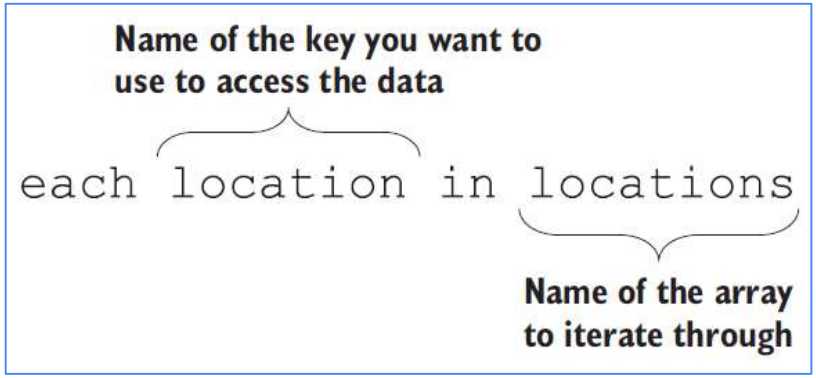
Dealing with complex, repeating data patterns

2. Pug View: Loop Through Array of Locations

```
extends layout

block content
  h1= pageHeader.title
  p= pageHeader.strapline

  // Loop through all locations
  each location in locations
    .card
      .card-block
        h4
          a(href="/location")= location.name
          //- Display the "distance" badge
          span.badge.badge-pill.badge-default.float-right= location.distance
        //- Display the address
        p.address= location.address
        //- Loop through facilities array inside this location
        .facilities
          each facility in location.facilities
            span.badge.badge-warning= facility
        //- (Ignore rating stars for now, or render statically as needed)
```



Manipulating the data and view with code

Show a 0–5 star rating using Font Awesome icons.

- rating = 3 → 3 solid stars, 2 empty stars
- **Solid**: i.fas.fa-star
- **Empty**: i.far.fa-star

```
small  
  - for (let i = 1; i <= location.rating; i++)
    i.fas.fa-star
  - for (let i = location.rating; i < 5; i++)
    i.far.fa-star
```

Figure 4.17 The Font Awesome star-rating system in action, showing a rating of three out of five stars

Using includes and mixins to create reusable layout components

Create a Mixin for Star Rating

Mixins are like reusable template functions in Pug. You define one for the star rating at the top of your file, or (better) in a shared include.

```
mixin outputRating(rating)
  - for (let i = 1; i <= rating; i++)
    i.fas.fa-star
  - for (let i = rating; i < 5; i++)
    i.far.fa-star
```

Defines mixin outputRating, expecting a single parameter rating

Uses the rating parameter inside for loops to output correct HTML

DEFINING PUG MIXINS

- A *mixin* in Pug is essentially a function. You can define a mixin at the top of your file and use it in multiple places.
- A mixin definition is straightforward: you define the name of the mixin, and then nest the content of it with indentation. The following code snippet shows a basic mixin definition:

```
mixin welcome
  p Welcome
```

- This definition outputs the Welcome text inside a <p> tag wherever it’s invoked

CALLING PUG MIXINS

- After defining the mixin, you'll want to use it, of course. The syntax for calling a mixin is to place a + before its name. If you have no parameters, such as the welcome mixin, this syntax looks like the following:

  +welcome

- This syntax calls the welcome mixin and outputs the text Welcome inside a <p> tag

```
small  
  - for (let i = 1; i <= location.rating; i++)
    i.fas.fa-star
  - for (let i = location.rating; i < 5; i++)
    i.far.fa-star
```

```
  h4
    a(href='/location')= location.name
+outputRating(location.rating)
```

USING INCLUDES IN PUG

- To allow your new mixin to be called from other Pug templates, you need to make it an include file, which is easy.
- Create a new file called **sharedHTMLfunctions.pug**, and paste the **outputRating** mixin definition into it, as follows:

```
mixin outputRating(rating)
  - for (let i = 1; i <= rating; i++)
    i.fas.fa-star
  - for (let i = rating; i < 5; i++)
    i.far.fa-star
```

Viewing the finished homepage

Listing 4.10 The `homelist` controller, passing hardcoded data to the view

```
const homelist = (req, res) => {
  res.render('locations-list', {
    title: 'Loc8r - find a place to work with wifi',
    pageHeader: {
      title: 'Loc8r',
      strapline: 'Find places to work with wifi near you!'
    },
```

Updates text for the HTML <title>

Adds text for the page header as two items inside an object

- Continue the remaining code from textbook

Listing 4.11 Final view for the homepage: app_server/views/locations-list.pug

```
extends layout
include _includes/sharedHTMLfunctions
block content
  .row.banner
    .col-12
      h1= pageHeader.title
        small  #{pageHeader.strapline}
  .row
    .col-12.col-md-8
      each location in locations
        .card
          .card-block
```

Brings in the external include file containing outputRating mixin

Outputs the page header text using different methods

Loops through the array of locations

- Continue the remaining code from textbook

Updating the rest of the views and controllers

After updates

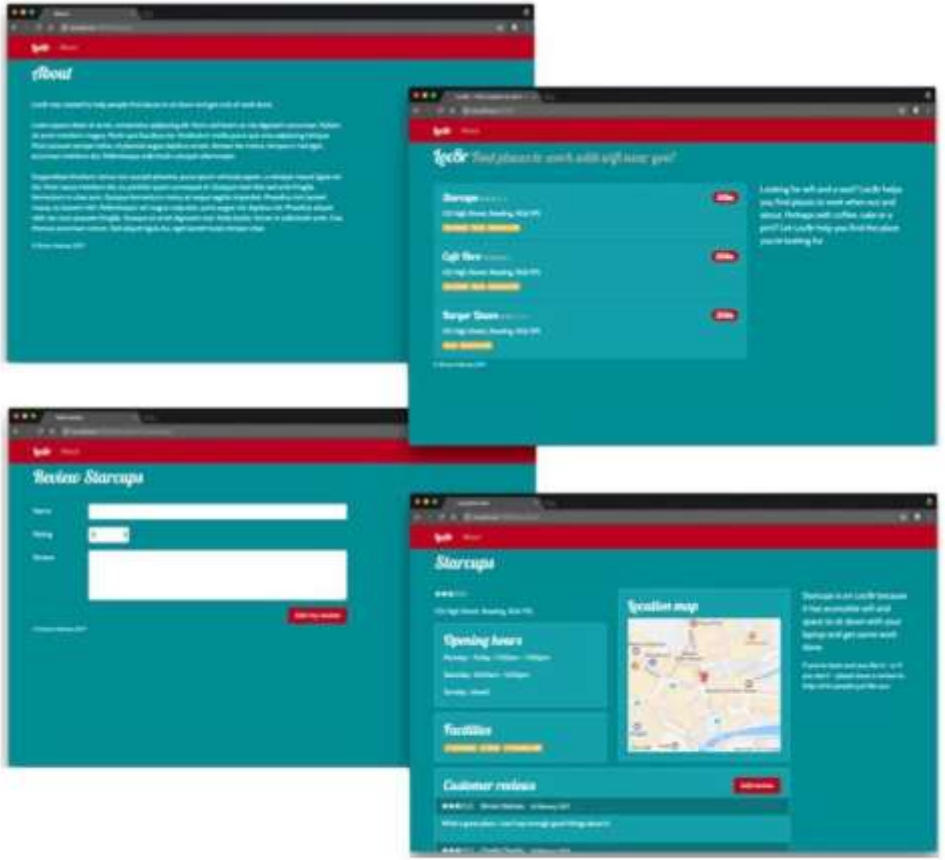


Figure 4.18 Screenshots of all four pages in the static prototype, using smart views and data hardcoded into the controllers

ANURAG UNIVERSITY

Thank you