In this tutorial we will create an express application that uses handlebars to render its views on the server-side

1. Create a new node project  
   **npm init -y**
2. Install the following npm packages for production  
   **npm i -S express morgan body-parser express-handlebars**
3. Install the following npm packages for development  
   **npm i -D nodemon**
4. In package.json - modify the scripts block to the following  
   "scripts": {  
    "start": "nodemon app.js"  
    }
5. Create the following folder structure within your project root folder  
     
   **my\_hbs\_project  
    |\_routes  
    |\_views  
    |\_layouts  
    |\_partials**
6. Create an **app.js** file in the root folder of your project
7. Write a minimum express app to get started

**const** express **=** require('express')

**const** app **=** express()

app.get('/', (req, res) => {

res.send('Hello World')

})

app.listen(3000, ()=> {

console.log('Express server listening on port 3000')

})

1. Right-click on your root folder and choose “Git Bash here” from the context menu to open Git Bash command-line window
2. Run **npm start** in the command-line to start your application in nodemon
3. In your browser window - navigate to <http://localhost:3000/>
4. Now that you have a running express app, let’s modify the app.js to render a template instead of returning a string

Add the following lines at the top of your page to import the additional middleware we will be using  
   
 **const** bodyParser = require('body-parser')  
 **const** morgan = require('morgan')  
 **const** exp\_hbs = require('express-handlebars')

1. After importing the necessary libraries let’s use them  
   For body-parser and morgan write these lines after the app instance was created

app.use(bodyParser.urlencoded({ extended: false }))

app.use(bodyParser.json())

app.use(morgan('dev'))

1. Straight after write the following to set 'express-handlebars' middleware as a rendering engine

**const** hbs\_options = {

extname : '.hbs',

layoutsDir : 'views/layouts/',

partialsDir : 'views/partials/',

defaultLayout : 'main'

}

app.engine('hbs', exp\_hbs(hbs\_options));

app.set('view engine', 'hbs');

1. Next - let’s modify the root routing function we have to use the new templating engine we have just setup  
   Change the function to the following, and notice we are using res.render() to render a template we will soon create.

app.get('/', (req, res) => {

res.render('homepage')

})

1. In order for this to work, first we need a main layout as we configured in the options of section 13 to be the default Layout.   
   A layout is the html skeleton in which we embed the templating we want to render  
   This separation exists to enable us to use the same html skeleton for many templates and avoid redundancy.
2. Within the layouts folder create a new file and name it **main.hbs**Then write the following lines in it (you can use emmet…)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Express hbs templating example</title>

</head>

<body>

{{{body}}}

</body>

</html>

1. The triple curly-braces wrapping the body keyword is telling handlebars where to embed the template that is being rendered.
2. Finally - let’s create the homepage template that will be embedded in the main layout  
   In the views folder, **create** a file named **homepage.hbs** and write the following lines within it

<div class="main">

<h2>Express Handlebars Example</h2>

<button>GET STARTED!!</button>

</div>

1. Refresh your browser window and see the changes  
   While this is a nice starting point, we will obviously want to add styling, interactivity and embed images in our page  
   To do that let’s add the following folders to our root project folder  
     
    **|\_public  
    |\_css  
    |\_images  
    |\_js**
2. Within these folders we will store static assets out client-side will need   
   So to be able to use these static assets, we need to add another line of middleware in **app.js** that will allow requests to pull those files from the public folder…   
   Add the following line to the middleware configuration section within our **app.js** file, just before the routing section…

app.use(express.static(\_\_dirname + '/public'))

1. Create a **main.css** file within the **css** folder and write some styles to center your content in the middle of the page both horizontally and vertically.
2. For those styles to be effective, we will need to link the **main.css** file to our html.  
   Link the **main.css** file to our main layout which we created in the **main.hbs** file in the layouts folder.  
   In the head section add the following line.

<link rel="stylesheet" href="/css/main.css">

1. Refresh your browser and inspect the changes
2. Awesome!  
   Now it’s time to add a partial  
   Create a file named share\_panel.hbs in the partials folder and write the following lines in it

<div class="share-panel">

<div class="icons">

<i class="fa fa-facebook-official"></i>

<i class="fa fa-twitter-square"></i>

<i class="fa fa-instagram"></i>

<i class="fa fa-linkedin-square"></i>

</div>

</div>

1. Notice that this partial is using **font-awesome** icons, so naturally we will need to link the font-awesome css file in the head section. Go to the **main.hbs** layout file and include the following in the head section

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.6.3/css/font-awesome.min.css">

1. Finally - modify the homepage template to include this partial by adding the following line under the button

{{>share\_panel}}

1. Next we will want to add some client-side Javascript interactivity to our web-app  
   Let’s add the following line to the end of the **homepage.hbs** template

<script src="/js/homepage.js"></script>

1. Obviously, as the path suggests, we will create a client-site **homepage.js** file under **/public/js** folder  
   For now, all we want is to see we have a connected script   
   so let’s write within this file just an alert statement within a self-executing function

(function(){

alert('homepage client side js!!')

})()

1. Refresh the page in the browser to make sure we have connected the client-side script…
2. Let’s delete the alert statement and define a click event handler to the button that will redirect the browser to the following url <http://localhost:3000/users>Obviously for this to work, you will need to add an id of **btn** to the button in the template

(function(){

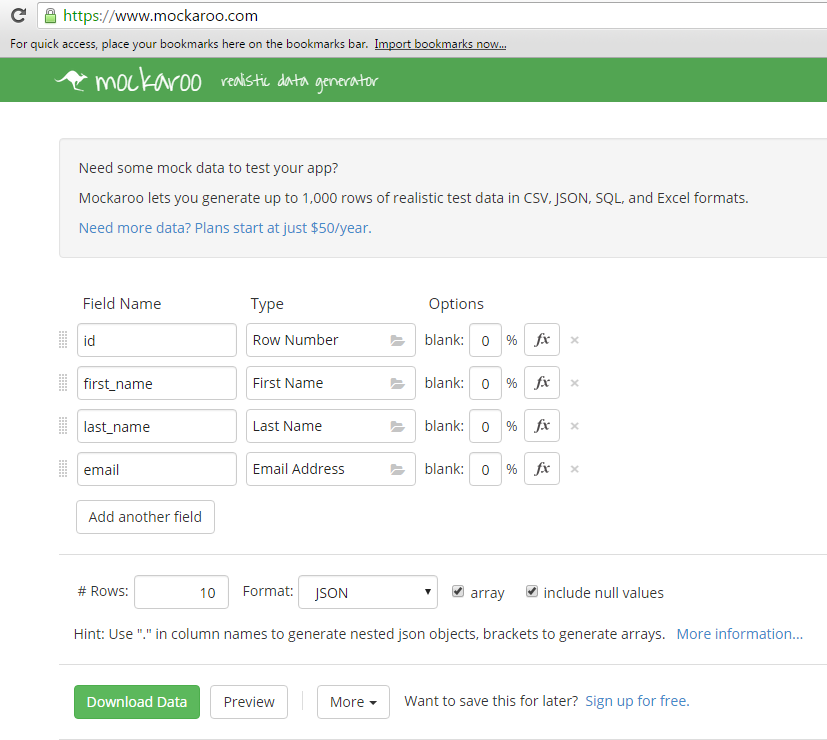
document.getElementById('btn').addEventListener('click',(event)=>{

window.location.href = '/users'

})

})()

1. In the next section we will want to inject some data to our templates and enable dynamic content to be rendered…  
   To keep it simple, let’s create a MOCK data containing a list of users Go to [www.mockaroo.com](http://www.mockaroo.com) and create an array of 10 users containing the following fields in json format  
   id, first\_name, last\_name, email



1. Click the ‘preview’ button in the mockaroo website and copy the content…  
   
2. Create a new folder under the project root folder and name it **mocks**
3. In it create a file named **MOCK\_USERS.json**
4. In the mock file you have just created write

{

“Users”: **PASTE-YOUR-MOCK-DATA-HERE**

}