Quick Guide: How to use Handlebars.

Part 1 – Setup your Project

* We can do this in 1 of 2 ways.
  + Manually
    - Create all your folders to look something like this.

A screenshot of a computer

Description automatically generated

* Using express generator
  + Install express generator: npm install express generator
  + Run express generator: npx express generator --hbs <folder\_name>

**Note: I will be using the Manual option when working with this document**

Part 2 – Making our Express server to use handlebars.

Add our packages:

In the root directory:

* npm init -y
* npm install express nodemon express-handlebars
* Add this code into the server file (index.js)

Here is our base file for rendering for handlebars

const express = require("express");

const exphbs = require("express-handlebars");

const app = express();

// app.engine(file\_extension, engine\_use(directory))

app.engine(

  ".hbs",

  exphbs.engine({

    extname: ".hbs",

    defaultLayout: "main",

    layoutsDir: \_\_dirname + "/views/layouts/",

    partialsDir: \_\_dirname + "/views/partials/",

  })

);

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

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

  res.send("home page for handlebars")

})

const PORT = 8000;

app.listen(PORT, () => {

  console.log(`http://localhost:${PORT}`);

});

**Lets replace the homepage res.send method with a handlebar page**. We will need to use res.render(). The render call works by calling onto a file\_name, and in our case, we want to also send data to fill in the placeholders.

// replaced information

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

  // find the home.hbs file, and fill in the information

  res.render("home", {

    title: "Home Page",

    message: "Welcome to Handlebars with Express!",

  });

});

Next, we need to create the handlebars files we will be using for this project. A base html file looks something like this:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Simple HTML Page</title>

</head>

<body>

    <h1>Hello, World!</h1>

    <p>This is a simple HTML file.</p>

</body>

</html>

We can break down this page into 3 sections, this will also go along with your future projects.

* Doctype + html header, the head call, the body call, and maybe a footer.

Now, instead of having to rewrite everything in each file, we can break them down into components and load them in as we please. You will see this a lot when you go into the next course.

So, we will need to breakdown the top http into 2 or 3 sections:

Main file, hosting the Doctype, html, head, body tags like this:

/views/layouts/**main.hbs**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>{{title}}</title>

</head>

<body>

  {{> header}}

  {{{body}}}

</body>

</html>

This file doesn’t contain any of the content we seek, but a placeholder for the content we want to put in later.

We will create a second file to handle filling in the content for the header portion:

File: /views/partials/ **header.hbs**

<header>

  <h1>{{title}}</h1>

</header>

Next:

We want to create a handlebars file that will render using the information we have made so far, replacing the body content from the main.hbs file

/views/**home.hbs**

<div>

  <h2>{{message}}</h2>

  <p>This is the home page.</p>

</div>

Since we have moved all the content out of the home file, we can now focus on writing the information inside the body, the important content. This file will have a placeholder for an incoming message and replace it when it is time to render the content.

Back to the res.render:

// replaced information

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

  // find the home.hbs file, and fill in the information

  res.render("home", {

    title: "Home Page",

    message: "Welcome to Handlebars with Express!",

  });

});

We render our the home.hbs file, and we replace the title and message with the content here.

The title is stored in the header file, and the message is stored in the home file.

Part 3 – handlebars helpers

Add these into your server file:

// if

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

  res.render("if", { user: sampleData.user });

});

// unless

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

  res.render("unless", { condition: sampleData.condition });

});

// each

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

  res.render("each", { users: sampleData.users });

});

Let’s go and make some content for the information

Here is some sample data: place it into the server file

// Sample data

const sampleData = {

  user: { name: "John Doe", email: "john@example.com" },

  users: [

    { name: "Alice", email: "alice@example.com" },

    { name: "Bob", email: "bob@example.com" },

  ],

  condition: false,

};

Lets make hbs files for if, unless and each

/views:

If.hbs

{{#if user}}

  <p>Hello, {{user.name}}!</p>

{{else}}

  <p>Please log in.</p>

{{/if}}

Unless.hbs

{{#unless user}}

  <p>Please log in.</p>

  {{else}}

  <p>Welcome to our webpage.</p>

{{/unless}}

Each.hbs

<h2>User List</h2>

<ul>

  {{#each users}}

    <li>{{this.name}} - {{this.email}}</li>

  {{/each}}

</ul>

Part 4 – Custom Helpers

The use for custom helpers is to achieve tasks that aren’t setup as a default from using handlebars, or creating tools you can use for any operation you need for your work.

In order to use our custom helper, we had to declare it in our engine call.

app.engine(

  ".hbs",

  exphbs.engine({

    extname: ".hbs",

    defaultLayout: "main",

    layoutsDir: \_\_dirname + "/views/layouts/",

    partialsDir: \_\_dirname + "/views/partials/",

**helpers: customHelpers**

  })

);

From this, we must create a folder to hold any helpers we create. So, inside the views folder, create a new folder called helpers – that covers the “helpers” tag.

Here is a small example for us to use:

Route:

app.get("/custom-helper-example", (req, res) => {

  const sampleDataCH = {

    name: 'John Doe',

    birthday: '1990-01-01',

    message: 'Hello, Custom Helpers!'

  };

  res.render("chExample", sampleDataCH);

});

Custom helper file (IT’S A JAVASCRIPT FILE, adding functionality):

Views/helpers/ **customHelpers.js**

module.exports = {

  uppercase: (str) => {

    return str.toUpperCase();

  },

  lowercase: (str) => {

    return str.toLowerCase();

  },

  formatDate: (date) => {

    const options = { year: "numeric", month: "long", day: "numeric" };

    return new Date(date).toLocaleDateString(undefined, options);

  },

};

Essentially, we have made 3 functions here, uppercase, lowercase and formatData. When a hbs file calls for any of these functions, it will attempt to run the code inside that function block. Example down below:

Hbs file:

Views/chExample.hbs

<h1>{{uppercase message}}</h1>

<p>Name in lowercase: {{lowercase name}}</p>

<p>Birthday: {{formatDate birthday}}</p>

Function call, alongside the data call