MVC

--- ft. ---

Handlebars
Sessions

MVC OverView

- What is MVC?
  MVC: Model-View-Controller is a design pattern that separates the data from the presentation. It is done by separating the data into the model and the presentation into the view. Controller is the glue between the model and the view.
- What is the difference between MVC and MVVM?

MVC:

- Models: the core data of your app
- Views: the UI components, such as your HTML layouts
- Controllers: the link between your models and views
  MVVM:
- Model: The data.
- View: The presentation.
- ViewModel: view-model is a class that contains the data and the logic.

MVC layout

- Handlebasr JS : View layer,
- Express JS : Controller layer,
- ORM AND SQL : Model layer

Adding data to views

- instead of fetching data we will use template engine to render our views with data
- this will help us to avoid fetching data than rendering it, instead we will do it on server side
- we will use Handlebasr.js template engine
- `npm install express-handlebars`
- add `const hbs = require('express-handlebars').create({})`
- add `app.engine('handlebars', hbs.engine)`

Adding views

- folder structure for views
  - views
    - main
      - layout
        - main.handlebars
    - home
      - index.handlebars
    - about
      - index.handlebars
    - contact
      - index.handlebars
    - error
      - index.handlebars

- Main layout is located in `views/layout/main.handlebars` all other views are located in `views/`

Handlebars syntax

- `{{{}}}`: for html
- `{{>}}`: for partials
- `{{}}`: for variables
- `{{#}}`: for loops `{{/}}`: for closing loops
- `{{#if}}`: for if statements `{{/if}}`: for closing if statements
- `{{#unless}}`: for unless statements `{{/unless}}`: for closing unless statements
- `{{#with}}`: for with statements `{{/with}}`: for closing with statements
- `{{#each}}`: for each statements `{{/each}}`: for closing each statements
- `{{someVar}}`: for variables

response.render()

- `response.render(view, {data})` .render will render the view with the data and send it to the client
- the view will automatically go in `main.handlebars` and `{{{body}}}` will be replaced with the rendered view
- `response.render(view, {data}, callback)` .render will render the view with the data and send it to the client and call the callback function

```
response.render('home/index', {
  title: 'Home',
  message: 'Welcome to my website'
})
```

{{#each }} and {{/each }}

- Within the `{{#each}}` block, Handlebars.js is smart enough to know that it's working with an object on each iteration, so it looks for the necessary properties.
- this is useful when you want to iterate over an array of objects
- to end the loop, use `{{/each}}`

```
{{#each projects as |project|}}
  <div class="project">
    <h3>{{project.name}}</h3>
    <p>{{project.description}}</p>
{{/each}}
```

{{#if }} and {{/if }}

- `{{#if}}` is used to render a block of code if the condition is true.
- The `{{#if}}` helper checks truthy values only, so you couldn't write something like `{{#if x === y}}`.
- `{{#if}}` is the same as `{{#unless}}` but with the opposite meaning.

```
{{#if isLoggedIn}}
  <p>You are logged in</p>
{{/if}}
{{#unless isLoggedIn}}
  <p>You are not logged in</p>
{{/unless}}
```

partials

- partials are reusable templates that can be used in any view, they are located in `views/partials/` and help reduce the amount of code that needs to be written.
- `{{>}}` is used to render a partial.
- to pass object to the partial, use `{{> partialName obj}}`

```
<div class="header">
  <h1>{{title}}</h1>
```

```
{{> header}}
{{> footer}}
```

this in handlebars

- `{{this}}` is used to reference the current object in the loop.

```
{{#each this}}
  <p>{{name}}</p>
{{/each}}
```

Custom helpers in handlebars

- `{{#helper}}` is used to define a custom helper.
- a helper funtion to to only get even ids

```
const getEvenIds = (array) => {
  return array.filter(item => item.id % 2 === 0)
}
{{#helper getEvenIds}}
  {{#each this}}
    <p>{{name}}</p>
  {{/each}}
{{/helper}}
```

Sessions

- Sessions allow our Express.js server to keep track of which user is making a request, and store useful data about them in memory.
- Cookies are used to store data in the browser
- `npm install express-session` - express-session is a middleware that allows us to use sessions in our Express.js server.
- `npm install connect-sessions-sequelize` - connect-sessions-sequelize is a library automatically stores the sessions created by express-session into our database.

create session store in database

```
const session = require('express-session')
const SequelizeStore = require('connect-session-sequelize')(session.Store)
const dbSess  = {
  store: new SequelizeStore({
    db: db,
    checkExpirationInterval: 15 * 60 * 1000, // 15 minutes.
    expiration: 24 * 60 * 60 * 1000 // 24 hours.
  }),
  secret: 'mySecretKey',
  resave: false,
  saveUninitialized: true
}
app.use(session(dbSess))
```

understanding the code

- `app.use(session(dbSess))` - this is the middleware that will be used to create the session.
- `store = new SequelizeStore` - this is the store that will be used to store the sessions in the database.
- `db: sequelize` - sequelize is the db instance that we will use to create the store.
- `checkExpirationInterval: 15 * 60 * 1000` - the store session checks for session expiration every 15 minutes.
- `expiration: 24 * 60 * 60 * 1000` - the store session expires after 24 hours.
- `secret: 'mySecretKey'` - the secret key used to sign the session.
- `resave: false` - the store session will not be resaved if it is not modified.

What is the difference between a session and a cookie?

- A session is a temporary storage of data that is created by the server and is destroyed when the browser is closed.
  - data is stored in the server memory, and is not stored on the user's computer.
  - session data is largers than a cookie.
  - server side sessions are more secure than client side sessions.
- A cookie is a piece of data that is stored in the browser and is sent back to the server with every request.
  - data stored in browser
  - sent back to server with every request

Session Store in Database

- when we have more than one session, we need to store the sessions in a database.
- where do we store the session info?
- currently the session are stored local to application
- we need to connect to a database to store the sessions.
- we will usee `connect-session-sequelize` to store the sessions in a database.

configuration

- `app.use(session(options))` - this is the middleware that will be used to create the session.
- `options` - this is an object that contains the session configuration.
  - `store` - takes in the store object that will be used to store the sessions in the database. which has the connection and db info
  - `secret` - the secret key used to sign the session. if secret is incorrect, the session will not be created.
  - `resave` - if true, the session will be saved back to the store even if it was not modified.
  - `saveUninitialized` - if true, the session will be saved back to the store even if it was not initialized.

cookie options

- `cookies:` - this is an object that contains the cookie configuration, will store the session id so every-time the user makes a request, the session id will be sent back to the server.
    - `maxAge` - the max age of the cookie in milliseconds.

req.session

- `req.session` - this is an object that contains the session data.
- update session data to count views;

Middle Ware

- `app.use(middleware)` - this is used to add middleware to the application.
- add a custom middleware to the application.
- `app.use(middleware)` - this is used to add middleware to the application.

```
app.use(function(req, res, next) {
  console.log('Request URL:', req.url);
  next();
}
```