MVC-based Web App

Outline

1. MVC-based Web App
2. View Template using Handlebars
3. Client-side vs. Server-side Rendering of Views

Web Client

Request

Response

Web Server

Frontend development

- HTML for page structure
- CSS for styling
- JavaScript for interaction
- AJAX for partial page updates (without reload)

Backend development

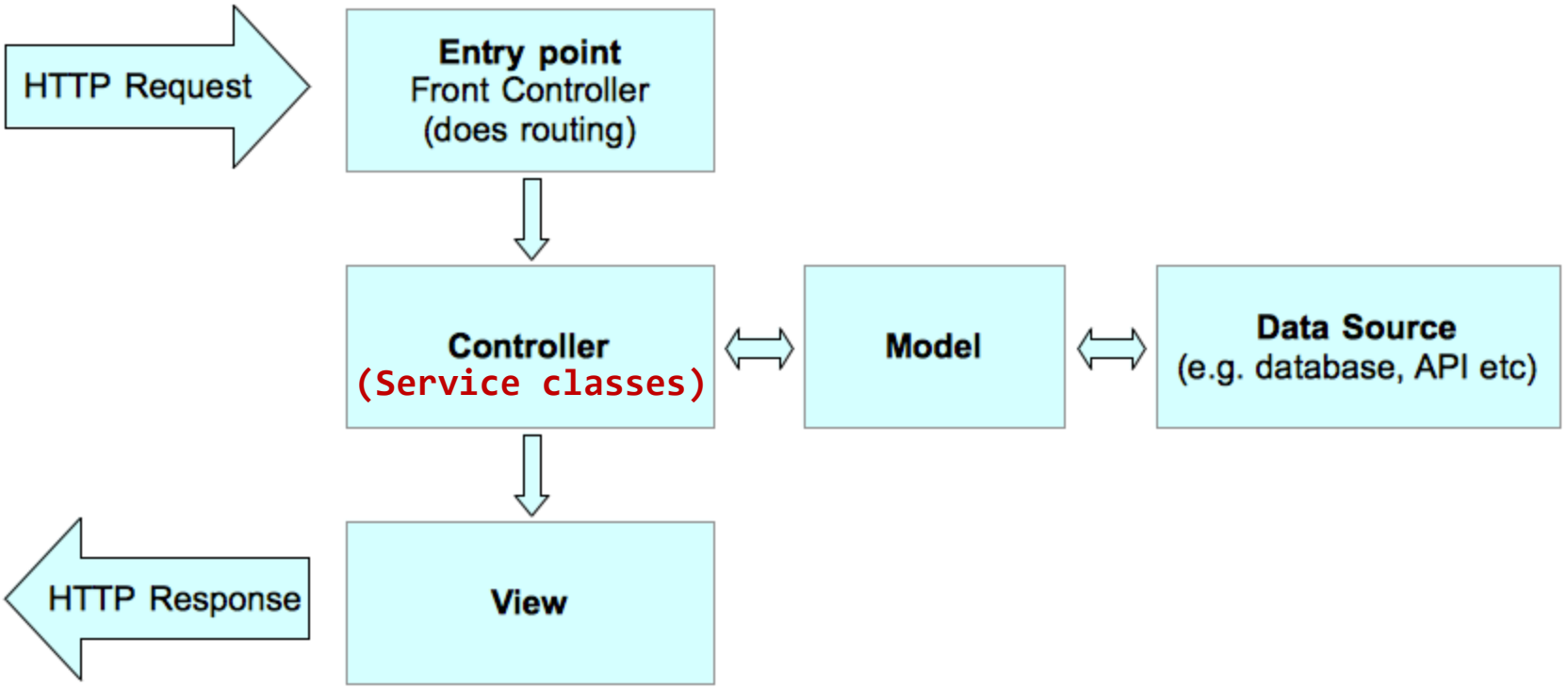
- Web API
- Views Template
- Data storage

express + node.js

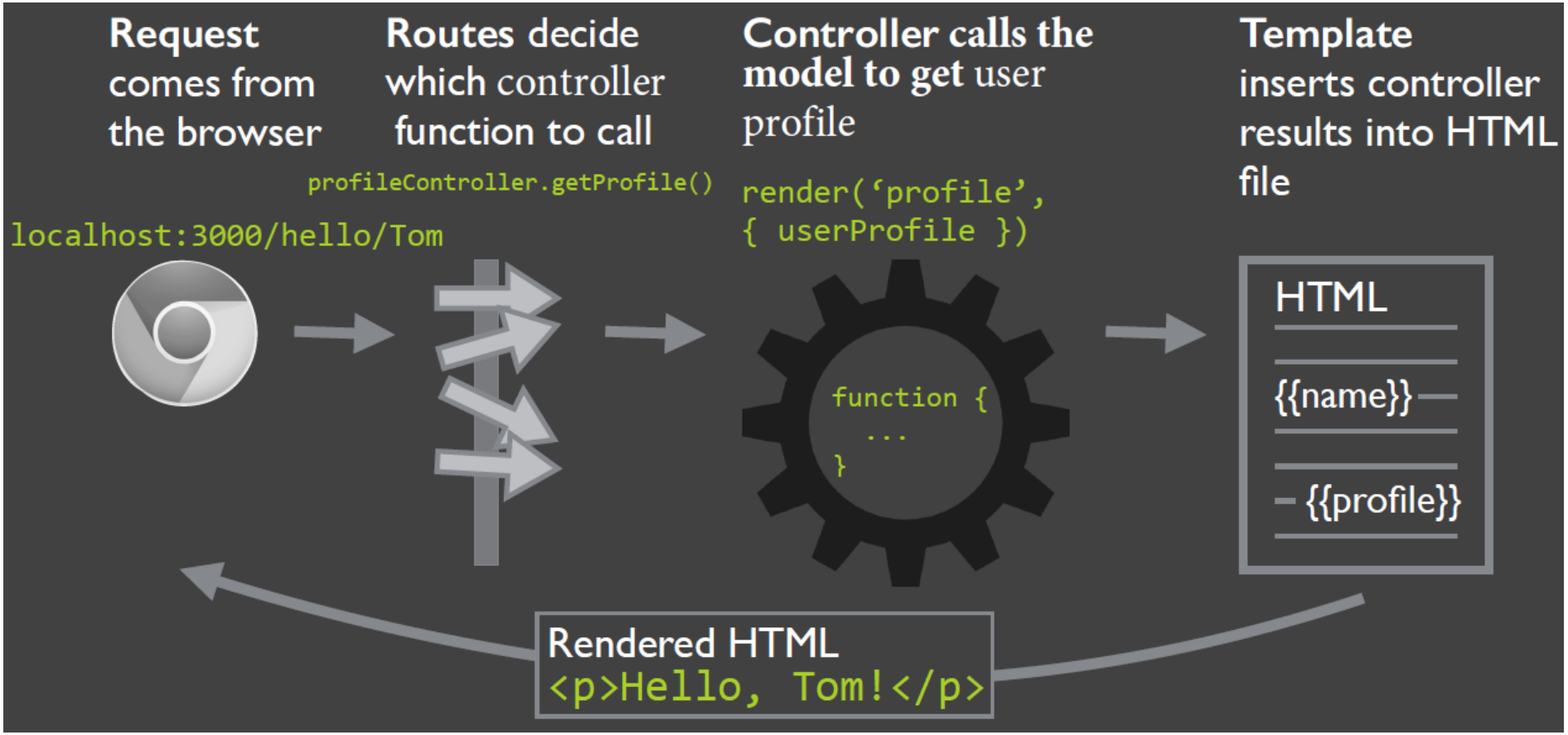
handlebars

mongoDB.

MVC-based Web App



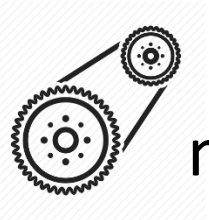
Interaction between App Modules



MVC-based Web application

Controller

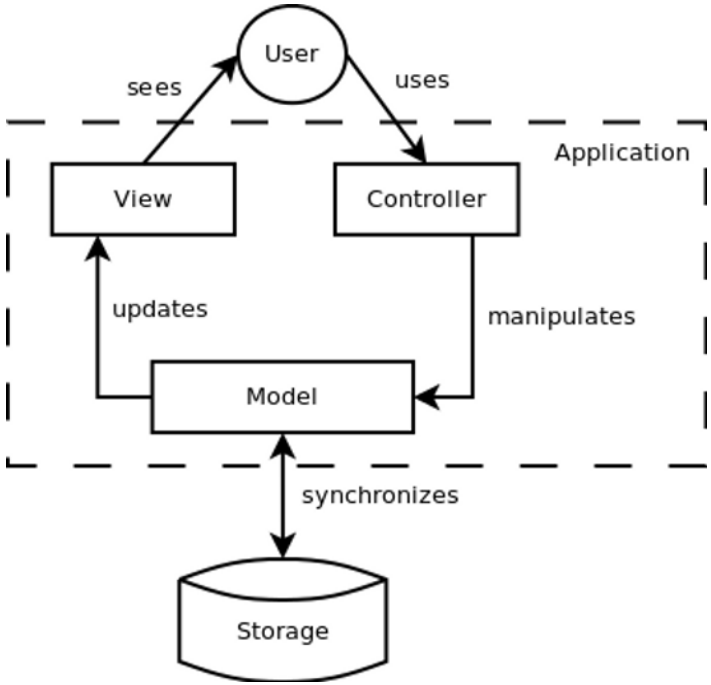
- accepts incoming requests and user input and **coordinates** request handling
- instructs the model to perform actions based on that input
  - e.g. add an item to the user's shopping cart
- decides what view to display for output



Model : implements business logic and computation, and manages application's data

View : responsible for

- collecting input from the user
- displaying output to the user

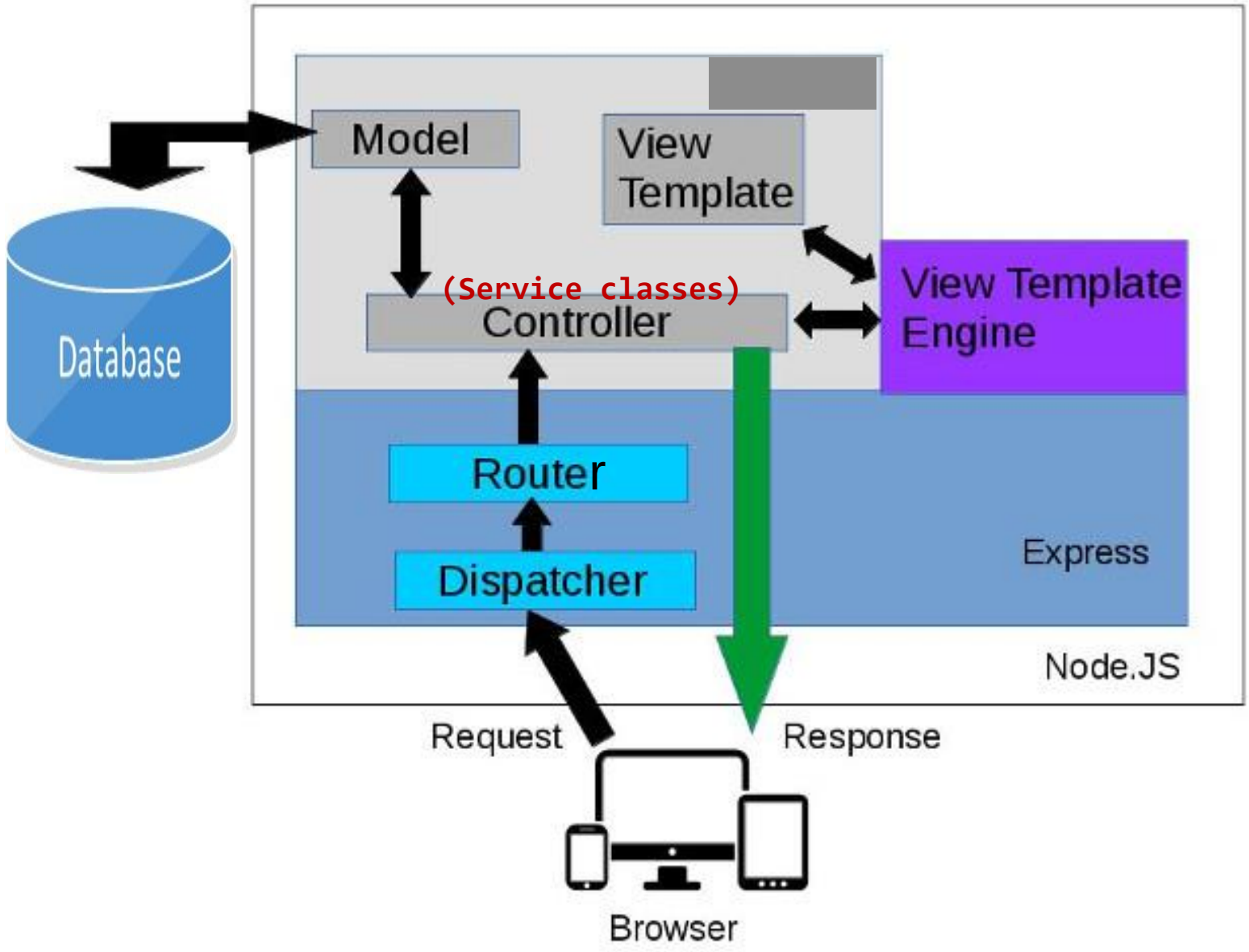


Advantages of MVC

- ***Separation of concerns***
  - Views, controller, and model are separate components. This allows modification and change in each component without significantly disturbing the others.
    - Computation is not intermixed with Presentation. Consequently, code is cleaner and easier to understand and change.
- **Flexibility**
  - The view component, which often needs changes and updates to keep the users continued interests, is separate
    - The UI can be completely changed without touching the model in any way
- **Reusability**
  - The same model can used by different views (e.g., Web view and Mobile view)
- **Allows for parallel teamwork**, e.g., a UI designer can work on the View while a software engineer works on the Controller and Model

MVC is widely used and recommended particularly for interactive web-applications

MVC using Node.js Express



View Template using Handlebars

View Template

- **View template** used to dynamically generate HTML pages on-demand based on user input
  - Template can use simple JavaScript Template literals
  - Template can use other syntax such as Handlebars
- **View engine** (aka template engine) is a library that generates HTML page based on a template and a given JavaScript object
  - Provide cleaner solution for separating the view
  - There are lots of JavaScript view engines such as Handlebars.js, KendoUI, Jade, Angular, etc.
  - **Handlebars.js** is simple library for creating client-side or server-side View templates http://handlebarsjs.com/

Handlebars View Templates

- View template has **placeholders** that will be replaced by **data** passed to the template
- Handlebars marks placeholders with double curly brackets `{{variable}}`
  - When rendered, the placeholders between the curly brackets are replaced with the corresponding value

Iterating over a list of objects

- **{{#each list}} {{/each}}** block expression is used to iterate over a list of objects

- Everything in between will be evaluated for each object in the list

```
<select id="studentsDD">
    <option value=""></option>
    {{#each students}}
        <option value="{{studentId}}">
            {{studentId}} - {{firstname}} {{lastname}}
        </option>
    {{/each}}
</select>
```

Template variable to be replaced with a value that is passed to the template

```
const students = [{
    "studentId": 2015001,
    "firstname": "Fn1",
    "lastname": "Ln1"
  },
  {
    "studentId": 2015002,
    "firstname": "Fn2",
    "lastname": "Ln2"
  }]
```

Conditional Expressions

- Render fragment only if a property is true
  - Using `{{#if property}} {{/if}}`
    or `{{unless property}} {{/unless}}`

```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{else}}
    <h1>Unknown Author</h1>
  {{/if}}
</div>
```

```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```

The with Block Helper

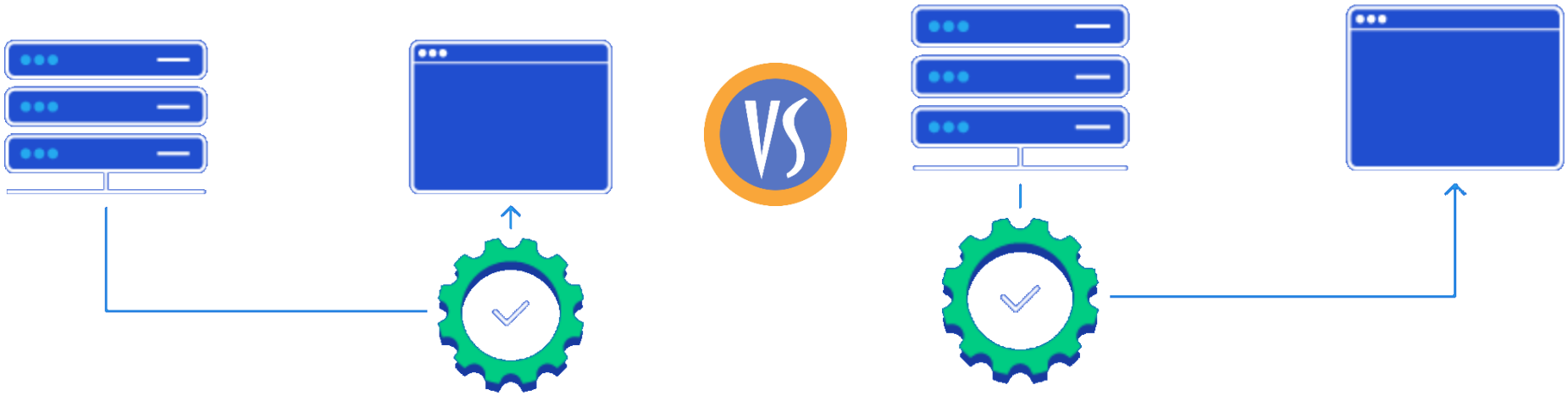
- `{{#with obj}} {{/with}}`
  - Used to minify the path
  - Write `{{prop}}` Instead of `{{obj.prop}}`

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

  {{#with author}}
  <h2>By {{firstName}} {{lastName}}</h2>
  {{/with}}
</div>
```

```
{
  title: "My first post!",
  author: {
    firstName: "Abbas",
    lastName: "Ibn Farnas"
  }
}
```

Client-side vs. Server-side Rendering of Views



Back

Client-side rendering (CSR)?

1. The user sends a request to a website (usually via a browser)
2. The browser downloads the HTML (containing HTML templates and static content), CSS and JS
3. Client-side JS makes Web API requests via AJAX to fetch dynamic data from the server
4. After the server responds, Client-side JS renders the Html template using the received data (The data from the API fill the template placeholders) then updates the page using DOM processing on the client browser

CSR using Handlebars

- Add Handlebars script

```
<script src="path/to/handlebars.js"></script>
```

- Create a template

```
const studentTemplate = '<p>{{firstname}} {{lastname}}</p>'
```

- Render the template

```
const student = {id: '…', firstname: '…', lastname: '…'};
const htmlTemplate = Handlebars.compile(studentTemplate);
studentDetails.innerHTML = htmlTemplate(student);
```

Server-side rendering (SSR)?

1. The user sends a request to a website (usually via a browser)
2. The server (more precisely a **model** object) performs the necessary computation to get/compute the results data
3. The server renders the Html template using the produced data
4. The produced Html content is sent to the client's browser

Server-Side Rendering (1 of 2)

1. Configure the View Engine

```
import handlebars   from 'express-handlebars';
const app           = express();
/* Configure handlebars:
 set extension to .hbs so handlebars knows what to look for
 set the defaultLayout to 'main'
 the main.hbs defines common page elements such as the menu
 and imports all the common css and javascript files
*/
app.engine('hbs', handlebars({ defaultLayout: 'main',
  extname: '.hbs'}));

// Register handlebars as our view engine as the view engine
app.set('view engine', 'hbs');

//Set the location of the view templates
app.set('views', `${currentPath}/views`);
```

Server-Side Rendering (2 of 2)

2. Call **res.render** method to perform server-side rendering and return the generated html to the client

```
app.get('/cart', (req, res) => {
    const shoppingCart = shoppingRepository.getShoppingCart();

    res.render('shopCart', { shoppingCart });
});
```

The above example passes the shopping cart to the **'shopCart'** template to generate the html to be returned to the browser

Client-side vs. Server-side Rendering of Views

- CSR **frees the server** from the rendering burden and enhances the app **scalability** (increased server ability to handle more concurrent requests)
  - But one of the main disadvantages is slower initial loading speed **of the first page** as the client receive a lot of JavaScript files to handle views rendering
  - Fast rendering after initial loading: second and further page load time is lesser, since all the supporting scripts are loaded in advance for CSR
- SSR **reduces** the amount of client-side JavaScript and speed-up the initial page loads particularly for slow clients
  - Web servers (having higher compute power) may render the page faster than a client-side rendering. As a result, the initial loading is quicker.
  - But this puts the rending burden on the server
  - Does **full page reload** to update the page

Client-side Rendering slow initial page load

Loading screen

Skeleton

Fully-rendered page

Resources

- Handlebars guide

https://handlebarsjs.com/guide/

- Learn Handlebars in 10 Minutes
  http://tutorialzine.com/2015/01/learn-handlebars-in-10-minutes/