

MVC-based Web App

Outline

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



Web Client

Request

Response



Web Server

Frontend development

HTML for page structure



CSS for styling



JavaScript for interaction



JavaScript

AJAX for partial page updates (without reload)



Backend development

Web API



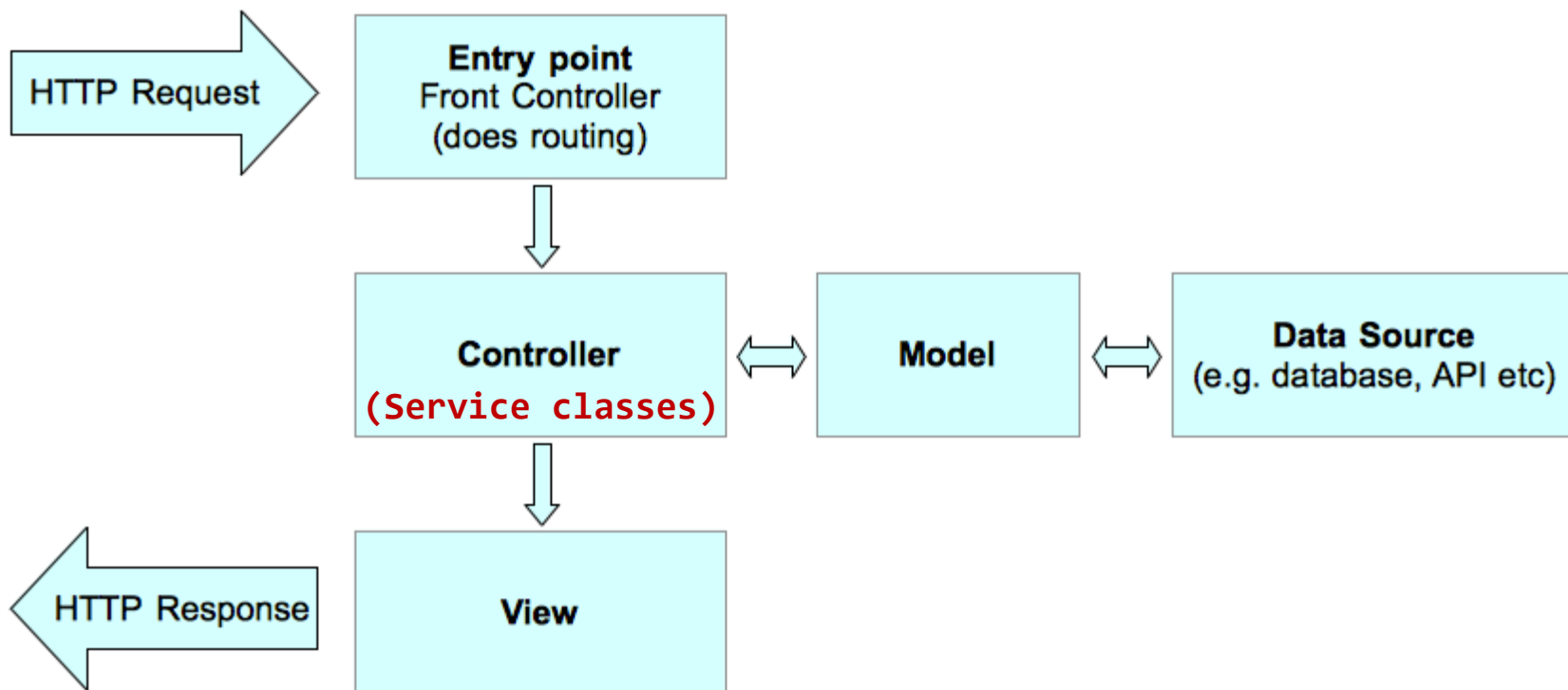
Views Template



Data storage



MVC-based Web App



Interaction between App Modules

Request comes from the browser

Routes decide which controller function to call

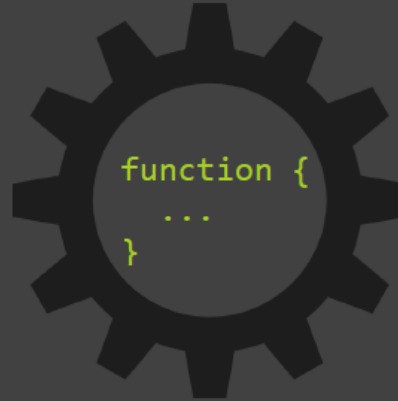
Controller calls the model to get user profile

Template inserts controller results into HTML file

`profileController.getProfile()`

`render('profile',
{ userProfile })`

`localhost:3000/hello/Tom`



Rendered HTML
`<p>Hello, Tom!</p>`

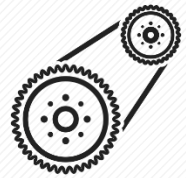
MVC-based Web application

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

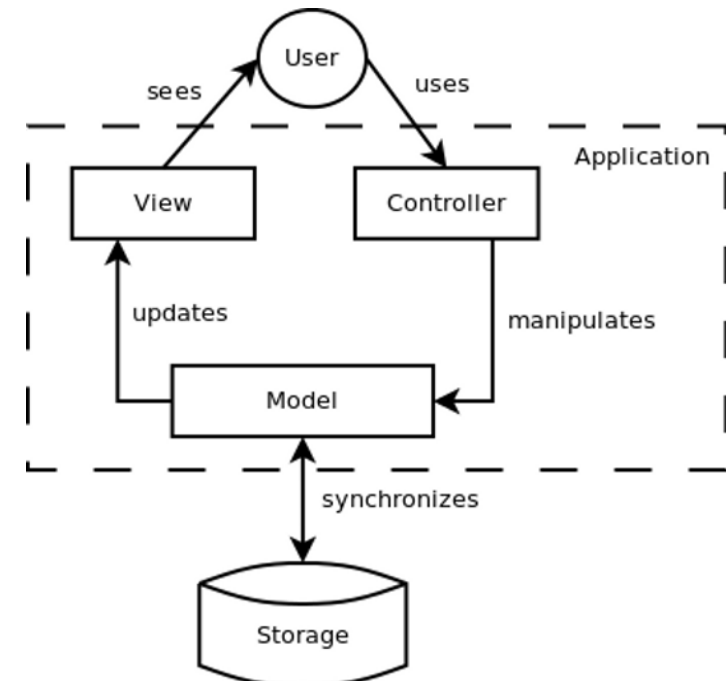


Model : implements business logic and **computation**, and manages application's data



View : responsible for

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

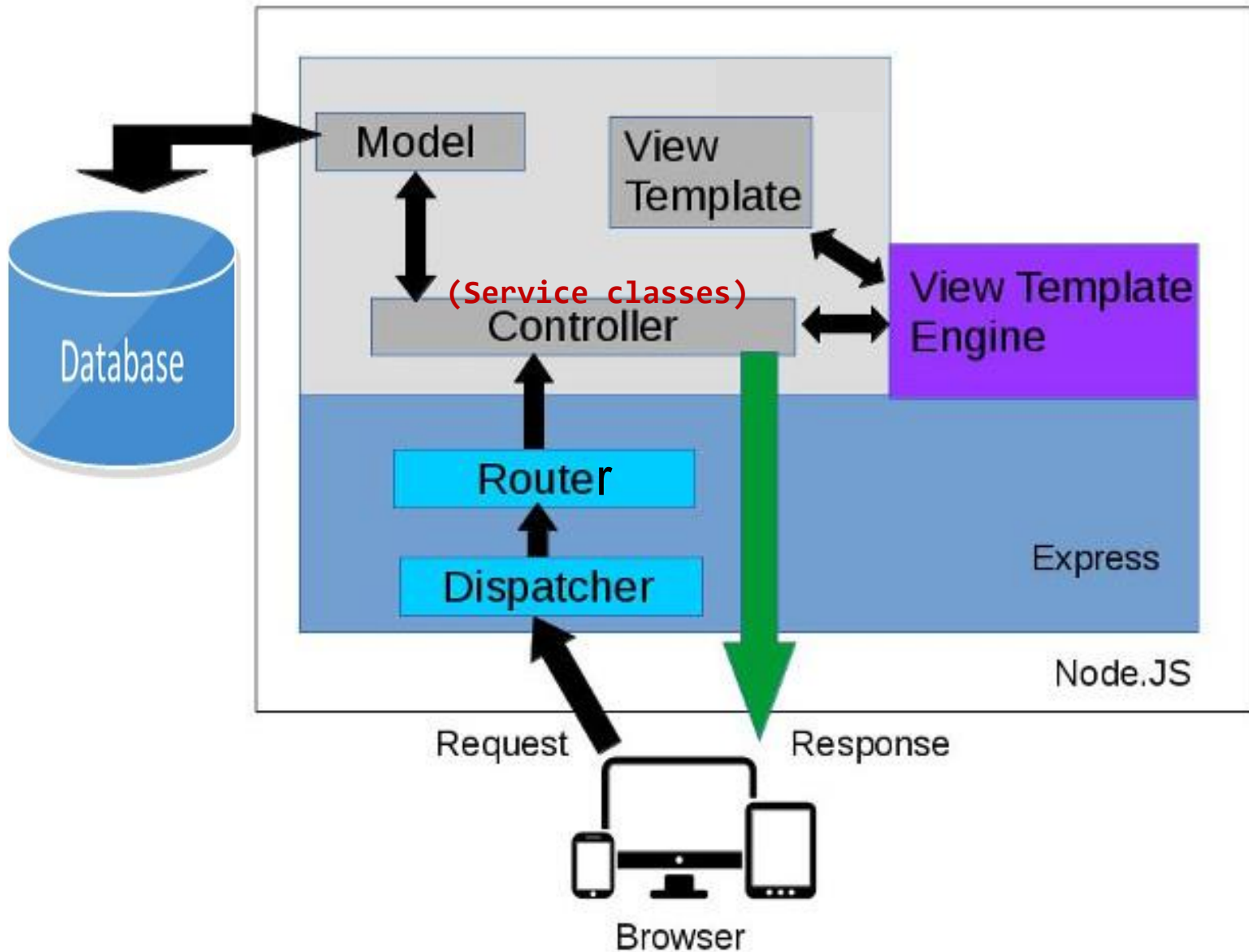


Advantages of MVC

- ***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 be 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

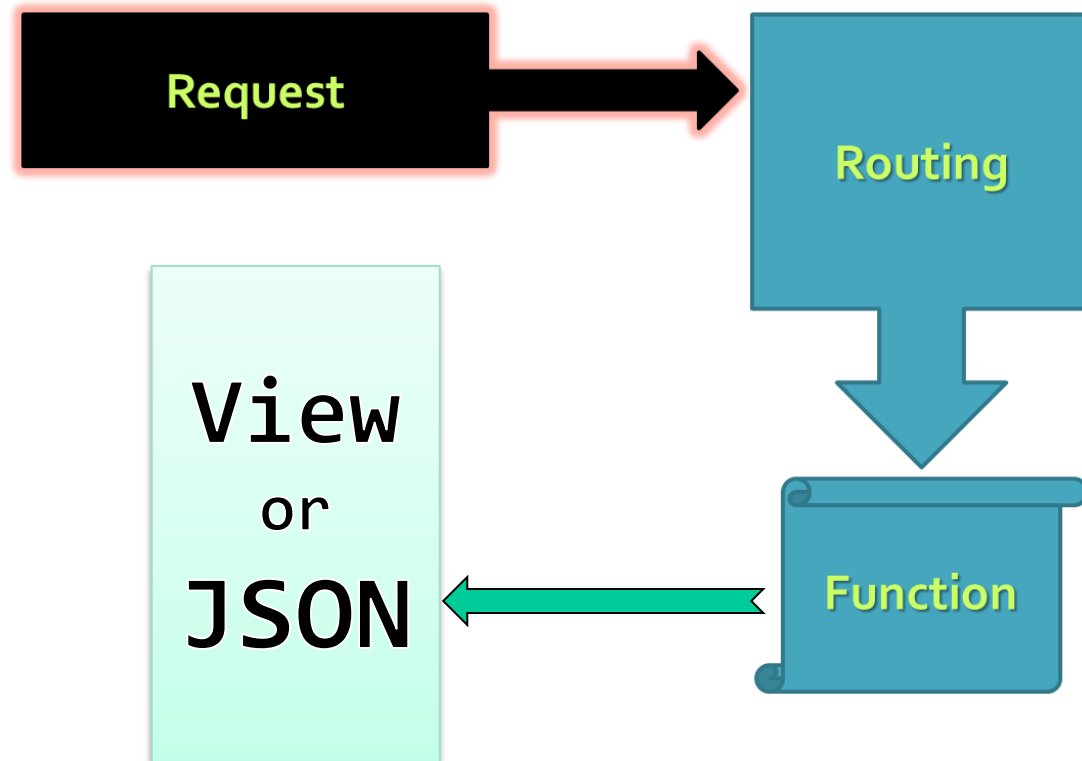
MVC is widely used and recommended particularly for interactive web-applications

MVC using Node.js Express



Views

Template using Handlebars



View Template

- **View template** used to dynamically generate HTML pages on-demand based on user input
- **View engine** (template engine) is a library that generates HTML page based on **a template** and a given **JavaScript object**
 - Provide cleaner solution by 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 UI templates
<http://handlebarsjs.com/>

Usage

- Add Handlebars script

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

- Create a template

```
const studentTemplate = `

|           |                            |
|-----------|----------------------------|
| StudentId | {{id}}                     |
| Name      | {{firstname}} {{lastname}} |

`
```

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

- Render the template

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

Creating HTML Templates

- HTML template has **placeholders** that will be replaced by **data** passed to the template
- Handlebars.js 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 elements

- `{{#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>
```

```
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"
  }
}
```

Server-side Rendering of Views



Client-side vs. Server-side Rendering of Views

- Client-side Views Rendering **frees the server** from this burden and enhances **scalability**
 - But one of the main disadvantages is **slower initial loading speed** as the client receive a lot of JavaScript files to handle views rendering
- Views could be generated on the server side to reduce the amount of client-side JavaScript and **speed-up initial page loads** particularly for slow clients, but this puts the rendering burden on the server
 - Web servers may render the page faster than a client side rendering. As a result, the initial loading is quicker.

Configure Handlebars 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 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`);
```

res.render

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

```
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

Resources

- Handlebars guide

<https://handlebarsjs.com/guide/>

- Learn Handlebars in 10 Minutes

<http://tutorialzine.com/2015/01/learn-handlebars-in-10-minutes/>