

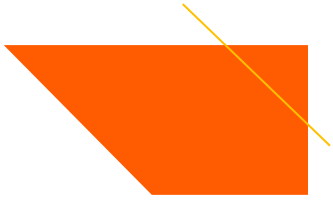
Web Application Development

COSI 152A

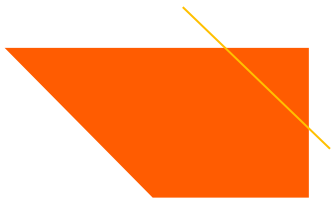


Main Point Preview

- In this lesson, you learn about:
 - templating engines and how to connect your routes to views.
 - embedded JavaScript (EJS)
 - passing data into views from controllers.



Templating Engine

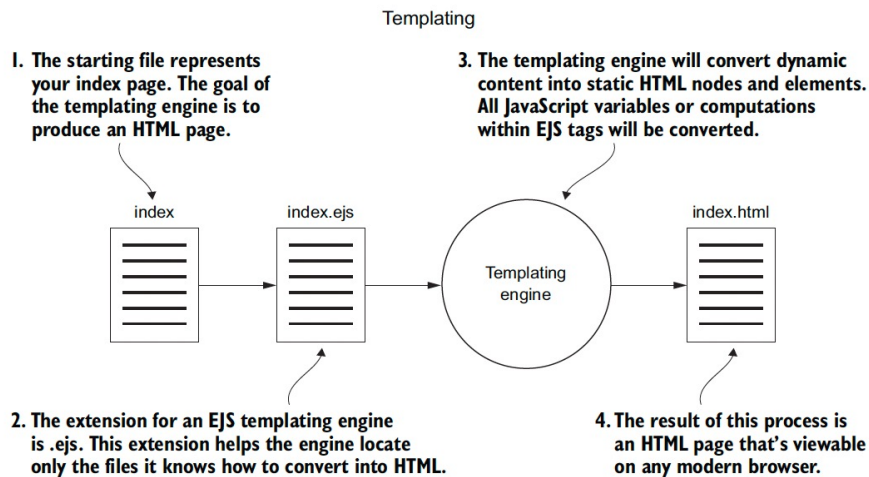


Templating

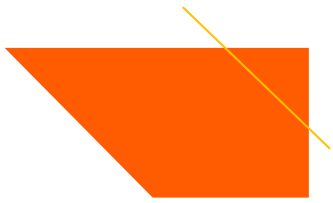
- Templating allows you to code your views with the ability to insert dynamic data.
- Express.js is so popular due to its ability to work with other packages and tools.
 - One such tool is the templating engine.
- There are many templating languages like EJS
 - With moderate HTML experience, EJS is the easiest and most effective

Templating Engine

- A templating engine is what Express.js uses to process your views and convert them to browser-readable HTML pages.
- Any non-HTML lines are converted to HTML, with values rendered where embedded variables once were.



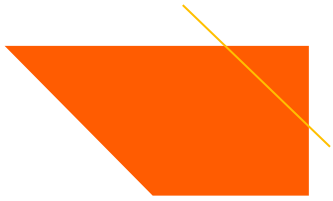
Converting EJS to HTML



Connecting a Templating Engine

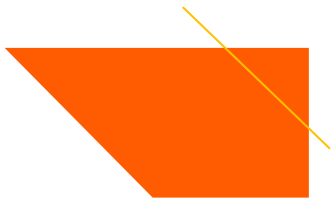
- Create and initialize a project called `express_templates`.
- Install `express` and create “controllers” folder with `homeController.js` file inside.
- In `main.js/index.js` file, require `express` module and `homeController.js`.
- Install the “`ejs`” package with the following terminal command:

```
npm install ejb --save
```



Connecting a Templating Engine

- Let the Express.js application know that you plan to use ejs for templating.
 - add `app.set("view engine", "ejs")` below your require lines in main.js.
 - this line tells your application to set its view engine as ejs.
 - application now expect EJS in your views folder in your main project directory.



Connecting a Templating Engine

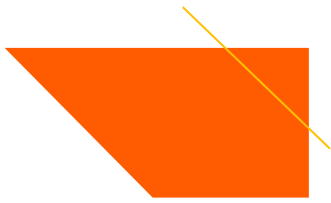
- Now that your application is ready to interpret EJS, create an index.ejs file in your views folder with following code:

```
<% let name = "Jon"; %>  
<h1> Hello, <%= name %> </h1>
```

Define and assign
a variable in EJS.

Embed a variable
within HTML.

- Above code uses the EJS syntax `<% %>` to define and assign a variable within your view.
 - everything within these characters runs as valid JavaScript.
 - each line of HTML can contain an embedded variable.
 - using `<%= %>` enables you to print that variable's value within the HTML tags.



Connecting a Templating Engine

- Last, create a route in main.js for the /name path:

```
app.get("/name", homeController.respondWithName);
```

- this route runs when a request is made to the /name path.
- then it calls the respondWithName function in the home controller.
- In homeController.js, add the respondWithName function as:

```
exports.respondWithName = (req, res) => {  
  res.render("index");  
};
```

Respond with a
custom EJS view.

- You used the render method on the response object to respond with a view from your views folder.



Connecting a Templating Engine

- Restart your application and visit `http://localhost:3000/name` in your browser.



Passing Data From Controllers

- The best way to use templates is to pass data from controllers to views
 - You don't want to define those variables directly in the view.
- To do so, remove the line in index.ejs that defines and assigns the name variable, but keep the h1 tag and its EJS contents.
- Change your route to take a parameter in its path and then send that parameter to the view.


```
app.get("/name/:myName",homeController.respondWithName);
```



Passing Data From Controllers

- Now the route takes a parameter at the end of the /name path.
- To use this parameter, you need to access it from your request params:
 - modify the `homeController.respondWithName` function:

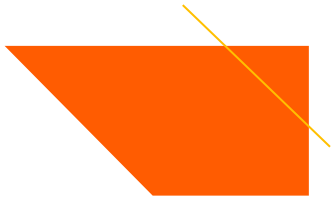
```
exports.respondWithName = (req, res) => {  
  let paramsName = req.params.myName;  
  res.render("index", { name: paramsName });  
};
```



Assign a local variable to a request parameter.

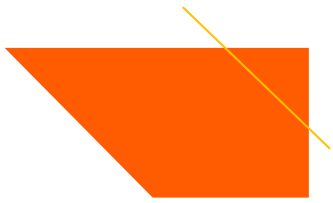
Pass a local variable to a rendered view.

- Now you are passing the name variable to your view in a JavaScript object



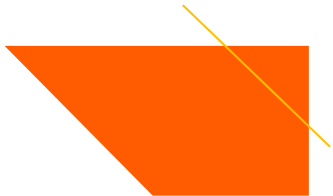
Layouts and Partials

- You can share view content across multiple pages
- Layout and Partials are the concepts used to create reusable and structured views or templates for rendering dynamic web pages.



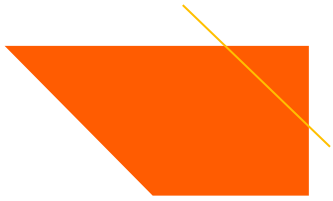
Layouts

- Layouts are templates that define the structure of your web pages
 - common structural elements like HTML `<head>`, navigation menus, and footers.
- Layouts often include placeholders where the unique content of each page can be inserted.



Setting Up Layouts

- Think of layouts as being the content that doesn't change from page to page when you browse a website.
 - The footer of the page or navigation bar might stay the same, for example.
- Instead of re-creating the HTML for these components, add them to a `layout.ejs` file that other views can share.



Setting Up Layouts

- To create layouts:
 - install the express-ejs-layouts package and require it in your main.js file

```
const layouts = require("express-ejs-layouts").
```

- let Express.js know to use this package in your main.js file.

```
app.use(layouts)
```



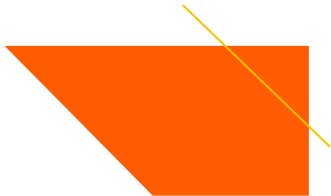

Setting Up Layouts

- Next, create a layout.ejs file in your views folder.
- You can start with some simple HTML in the layout file, as shown below:

```
<body>
  <div id="nav">NAVIGATION</div>
  <%- body %>
  <div id="footer">FOOTER</div>
</body>
```

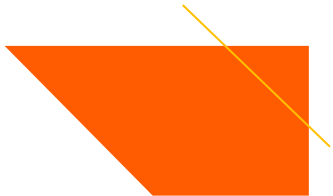
Wrap body with boilerplate HTML.

- The **body** keyword is a placeholder
 - It is used by Express.js and the layout express-ejs-layouts to fill other views' contents in its place.



Setting Up Layouts

- When you visit a route that renders a view, you'll notice the navigation and footer text with your rendered view in between.
- This layout will continue to render along with your view on every page load.
- To see, restart your application, and visit the `/name/:myName` path in your browser.



Setting Up Partials

- Partials work similarly to layouts.
- Partials are snippets of view content that can be included in other views.
 - For example, you may want to add a notification box on a few of the pages.



Setting Up Partials

- To create a partial for the navigation element:
 - move your code for that div to a new file called navigation.ejs.
 - place that file in a new folder called partials within your views folder.
 - then include that file within your layout.ejs file:

```
<%- include('partials/navigation') %>
```

- because the layout is already in the views folder, it needs to look in the partials folder on the same directory level to find the navigation partial.

Restart your application and visit the `/name/:myName` path again.



Thank You!