Handlebars

Objectives

- To understand semantic templating using Handlebars
- To understand the basic syntax for Handlebars-based web applications
- To create an Express connection to a front-end application which takes in MySQL data and prints it to the screen using Handlebars
- To use HTML/jQuery GET, POST, PUT, and DELETE commands
- To create full-stack web applications that will Create, Read, Update, and Delete data from a MySQL database
- To use Express.js, MySQL and Handlebars together to create a dynamic application

Demo: Handlebars Lunch

```
$ atom 04-HandlebarsLunch/server.js
```

```
$ node 04-HandlebarsLunch/server.js
```

http://localhost:3000/weekday

http://localhost:3000/weekend

http://localhost:3000/lunches

- Like past activities, we are using Express
- Unlike past activities, we are using Handlebars

Handlebars

http://handlebarsjs.com/

Handlebars is what is known as a "Semantic Templating" framework for JavaScript and HTML.

- Frameworks like these are used as a replacement for constructing long strings of HTML within your JavaScript code while also providing programmers with a simpler method through which to dynamically create or to fill HTML elements.

Somebody Google ‘semantic’

Semantic templating creates a ‘common language’ between our HTML && JavaScript so we can keep them separate

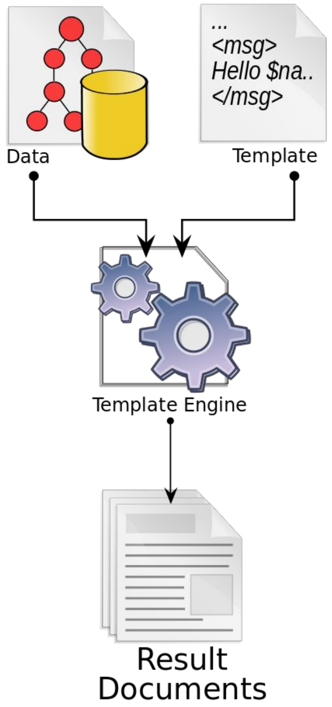
{{ }}

Handlebars makes it so you can pass back-end variables to your front-end HTML simply by including the name of the variable you are sending from the server inside of two sets of curly-brackets within your front-end code.

Example: {{VALUE}}.

Template Engine || Processor || Parser

- Software designed to combine templates with a data model to produce result documents
    - encourages organization of source code into operationally-distinct layers (see e.g., MVC)
    - enhances productivity by reducing unnecessary reproduction of effort
    - enhances teamwork by allowing separation of work based on skill-set (e.g., artistic vs. technical)



https://en.wikipedia.org/wiki/Template_processor

Review: Handlebars Lunch

Handlebars expects to find ‘views’ and ‘layouts’ directories in specific locations.

Take a look at the folder structure in 04-HandlebarsLunch

- A views directory at root level
  - A layouts directory within that

Note in server.js we declare: defaultLayout: "main"

```
app.engine("handlebars", exphbs({ defaultLayout: "main" }));
```

We use the Express method .engine() to say that for any files in our template engine ending with ‘handlebars’ call exphbs() and use “main” as the defaultLayout

RTFM https://expressjs.com/en/4x/api.html#app.engine

```
app.set("view engine", "handlebars");
```

We set the ‘view engine’ of our Express app to ‘handlebars’

RTFM https://expressjs.com/en/guide/using-template-engines.html

```
app.get("/weekday", function(req, res) {
  res.render("index", lunches[0]);
});
```

RTFM http://expressjs.com/en/api.html#app.render

.send() vs. .render()

- one sends data
- the other renders HTML

{{{ body }}}

We use triple curly-brackets to tell our program to read and render HTML elements

Double curly brackets are expressions and used to pass values

```
<h1>{{lunch}}</h1>
```

indindex.handlebars is taking in the variable {{lunch}} and is placing it within <h1> tags and then inserting itself into main.handlebars in {{{body}}}

Activity: Handlebars Lunches

- With a partner, explain server.js, main.handlebars & index.handlebars
- Research {{#each}} and be prepared to explain all-lunches.handlebars

{{#each}}

{{#each}} is a Handlebars helper.
It’s essentially a for-loop which iterates through an array and appends the values to your HTML

Activity: Ben & Jerry’s App (15 min)

Review: Ben & Jerry’s App

Demo: Handlebars Animals

```
$ node server.js
```

- http://localhost:3000/dog
- http://localhost:3000/all-pets
- http://localhost:3000/all-non-pets

Activity: Handlebars Animals (20 min)

See 07-HandlebarsAnimalsBase for instructions

Review: Handlebars Animals

#parkinglot

What is REST?

~~What we’re all going to need after this~~

Representational State Transfer

REST is a Set of Standards for the Web

RESTful applications:

- uniform interface (there are only a few basic requests you can make)
- stateless (no information is retained by either sender or receiver)
- keep the server separate from the client
- layered system in that there may be intermediary servers between the client and the database with which they are working
  - cacheable (the ability to store copies of frequently accessed data in several places along the request-response path)

http://www.restapitutorial.com/lessons/whatisrest.html

What is a state?

In information technology and computer science, a program is described as stateful if it is designed to remember preceding events or user interactions; the remembered information is called the state of the system.

stateless

vs.

stateful

What is a URI?

Uniform Resource Identifier

A string of characters used to identify a resource

RTFM https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

A URL is a URI, but not all URIs are URLs

RTFM https://en.wikipedia.org/wiki/Uniform_Resource_Name

What are the two HTTP requests we currently use?

GET && POST

Activity: PUT && DELETE

With your neighbor, research PUT && DELETE requests and be prepared to report back

Review: DELETE

DELETE requests are used to delete a row from a database

Review: PUT vs POST

PUT requests are used to update a row within a database

PUT requests are used when the client knows or specifies the URI

POST requests are used to create new data in a database

POST requests are used when the client is allowing the server to create a URI

idempotence

```
x = 1 * 1;
```

```
x += 1;
```

PUT is idempotent, so if you PUT an object twice, it has no effect.

RTFM https://en.wikipedia.org/wiki/Idempotence

SFW?

RESTful APIs

1. HTTP GET should be used for all retrieval. It should never be used to create, update, or do things.

2. HTTP POST should be used for creating. It shouldn’t be used to update or get a resource. If a URI had never existed before now and you’re going to create it and make it hold some data, use POST.

3. HTTP PUT should be used for updating — meaning replacing a collection with different data. The URI should have existed before.

4. HTTP DELETE should be used for deleting.

Lunch

Demo: Using HTTP Requests

```
$ cd 08-TaskSaver
```

```
$ node server.js
```

Look!

- Lines 1-33 are boilerplate
- We are collecting data from a MySQL database in our routes
- We use POST in the index.handlebars form
- In server.js, our .post route takes the HTML form POST and inserts it into our db with req.body.task

Activity: Using HTTP Requests (15 min)

See 08-TaskSaver

Set up the database and run the app

With your neighbor, go line-by-line through the code

Activity: Wishes (15 min)

See 09-Wishes for instructions

Review: Wishes

Demo: Putting & Deleting

Note

- We require the ‘method-override’ package. Why?
    - DELETE and PUT were removed from HTML5 forms. Why?
        - No payload. Human error.
    - We use ‘method-override’ to build queries that force our form to send DELETE and PUT requests
        - ?_method=DELETE
        - ?_method=PUT

RTFM https://www.npmjs.com/package/method-override

Activity: Day Planner (15 min)

See 10-DayPlanner

Setup the database, npm install, and run the server

With your neighbor, walk through the code line-by-line

Activity: Watch List

See Slack for instructions

Review: Watch List