CIT 382
Web Dev II

Week 4

Winter 2017

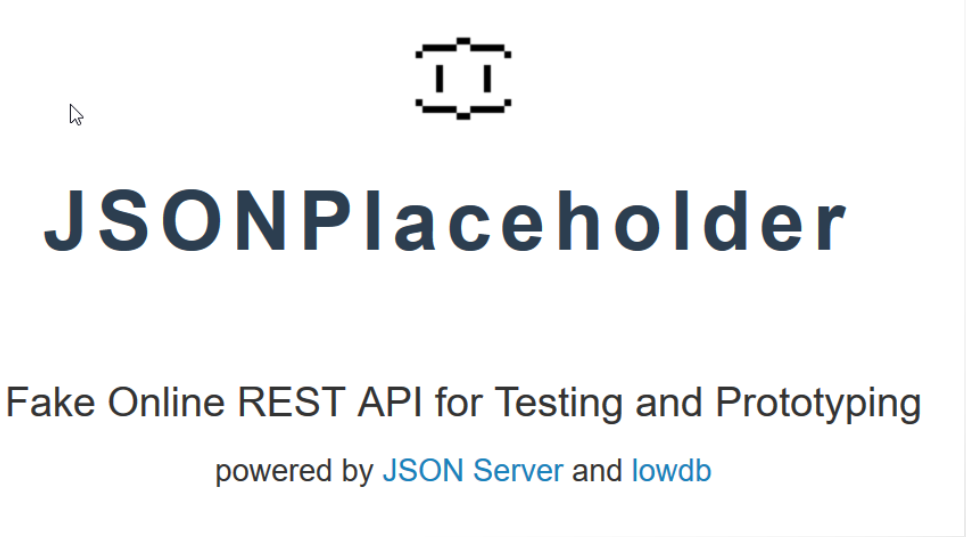
Phil Colbert

REST API Calls

- A very common architecture used by modern software is using HTTP calls to URLs that offer information
- These HTTP calls equate to an API, just as we are using for Project 2 and the OpenWeather website
- Using the HTTP protocol to interact with remote services is the foundation of REST (Representational State)
- REST is also referred to as RESTful web services, providing interoperability between computer systems over the Internet (or any network capable of supporting HTTP)

JSONPlaceholder

- To illustrate REST let's refer to the JSONPlaceholder website
- We can use this website to develop and test a working Meteor React application making REST calls



JSONPlaceholder

- The JSONPlaceholder website
  - Contains six different route-based resources
  - Supports a number of standard HTTP verbs
    - See the article [HTTP Verbs Demystified: PATCH, PUT and POST]
  - Supports individual records
- We can use our browser to "call" any of the routes and view JSON data directly in our browser
  - https://jsonplaceholder.typicode.com/users


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
```

HTTP Library

- To issue HTTP REST calls from our Meteor React web application, we need to
  - Add the http library using the C9 terminal

    ```
    meteor add http
    ```

  - Add a named import statement
  - Use the call() method
- Example

  ```
  import { HTTP } from 'meteor/http';
  HTTP.call(…)
  ```

HTTP call()

- The call() method of the http object supports two versions:
  - Asynchronous
  - Synchronous
- Typically, we prefer asynchronous versions, but for this particular functionality, we're going to use the synchronous version to simplify our programming
- Why do we care?
  - Synchronous calls "block," essentially waiting until a response is given before continuing to execute sequentially
  - Asynchronous calls require a callback function, making returning results from a method call more problematic

HTTP call

- When issuing an HTTP call(), we need to specify at a minimum two parameters, with a third parameter for options
  - Param1: HTTP verb as a string (e.g. 'GET')
  - Param2: URL as a string
  - Param3: Options (object of properties), see HTTP | Meteor Docs for a list of options
- Example:

```
import { HTTP } from 'meteor/http';
let url = 'https://jsonplaceholder.typicode.com/users';
let callResult = HTTP.call('GET', url, {});
```

HTTP Call Working Example

- A working Meteor React program, CIS 382 httpcall, is available for you to examine

REACT REST Calls

Powered by JSONPlaceholder

Return JSON data using REST call via HTTP package using Meteor and React.

Get User Data

- Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdottir V
- Glenna Reichert
- Clementina DuBuque

HTTP Call Working Example

- From the previous screen capture you can see the program provides a single button that when clicked retrieves the user data using REST, and iterates over the results, creating an unordered list of the users names
- In order to use HTTP call(), we need to distinguish between code that runs on the client (in our browser), and code that runs on the server

Client versus Server

- Meteor includes the following Boolean properties we use to enforce running code on either the client or server
  - React.isClient
  - React.isServer
- Why do we care?
  - For security, browsers prevent or limit the ability to reference different domains using AJAX (used for REST calls)
  - These "cross domain" requests must be run from the server
    - See Wikipedia Cross-origin resource sharing article

Meteor imports/api

- In the Meteor tutorials we've seen the creation of JavaScript API (code) files, and in our example program we've created imports/api/http.js
- Highlights of http.js
  - Tests Meteor.isServer
  - Adds a getRestData() method to Meteor.methods
  - Issues the http.call() request synchronously
  - Uses JavaScript try..catch..finally syntax to trap and handle errors
  - Returns JSON data for all return paths, including using JSON.stringify() to ensure we return a JSON string

Adding imports/api/http.js

- To make our API methods available throughout our Meteor code, we add an import to server/main.js
- Note that this import statement is neither a default, or named, import, as we have no export statements in http.js
- This syntax, according to the Meteor tutorials and Meteor Application Structure documentation supports importing JavaScript code directly (as well as support for HTML and CSS)

Anatomy of a Meteor REST Call

- Below are the steps to making a REST API HTTP call using Meteor and React
  - Isolate your REST call as server-side code only
  - Construct a React JSX component that accepts reactive data
  - Use the JSX component within a render() method
  - Use an event to call the isolated REST call code, and update the reactive data (using state, not props), thus causing Meteor and React to update the JSX component
  - The event code to call your REST code is also isolated, but isolated the only execute on the client

JSON

- REST services return information as a string, typically either in JSON format (e.g. JSON, JSONP, JSON-LD), or XML
- Within JavaScript, we will use the JSON.stringify as needed if we need to force information into the JSON format
- Additionally, we will need to use JSON.parse() to transform a JSON string into a JavaScript object

Handling Errors

- JavaScript, as with most programming languages, includes a syntax for trapping and handling errors
  - Errors may be logical errors that are trapped when data does not conform to the expected format
  - Errors may be as a result of incorrect data
  - Errors may be specifically caused by the code (thrown or raised) to enable handling of errors using the JavaScript error handling syntax
- JavaScript errors are trapped and handled using the try, catch, finally syntax

Try, Catch, Finally

- To establish an error trap, surround code with a try, catch block
- Any error that occurs within the trapped block will immediately proceed to the catch handler
- The catch handler includes an optional error parameter useful to diagnosing and reporting on the error
- The try, catch block also includes an optional finally block that is always executed, whether an error occurs, or not
  - If an error occurs, the finally block is executed at the end of the catch block
  - If no error occurs, the finally block is executed at the end of the try block

Error Testing

- Most REST API web services use a standard object and property format
  - statusCode: an HTTP numerical code that mimics the codes used by browsers (see Status Codes in HTTP)
  - content: Information as a string that embodies the core information requested and returned
- Other standard elements may be included, but even these two elements are not necessarily included from every RESTful web service
  - White House Web API Standards

Error Checking

- The working example includes a JSX component Typicode
- This component constructs the user list, so requires some amount of error handling to ensure the props data includes the necessary user properties
- One way to test if an object contains properties is to count the keys (property names) of an object
- In the working example, this test is used to determine if an empty list was passed to Typicode (the startup condition), or if JSON data from the REST call is available
  - If no data is available, the JSX component returns null to prevent rendering a result from the component

Browser Data versus API Data

- When using a browser to interact with a REST web service, the format of the JSON data displayed in the browser is not the same format received when issuing a REST API call via a programming language
- Typically the REST API call has additional objects and properties
- Let's compare the data returned from the sample website

Browser Data

```
▼ array {11}
  ▼ coord {2}
      lon : -123.13
      lat : 44.1
  ▼ weather [1]
    ▼ 0 {4}
        id : 803
        main : Clouds
        description : broken clouds
        icon : 04d
  base : stations
  ▼ main {7}
      temp : 281.061
      pressure : 1008.44
```

REST API Data


```
▼ root {1}
  ▼ array {4}
      statusCode : 200
      content : [\n  {\n    \"id\": 1,\n    '
                Graham\",\n    \"username\":
                 \"email\": \"Sincere@april.b
                {\n      \"street\": \"Kulas
                 \"suite\": \"Apt. 556\",\n
                \"Gwenborough\",\n      \"zip
                \"92998-3874\",\n      \"geo'
                \"-37.3159\",\n        \"lng'
                 }\n    },\n    \"phone\": \'
```

JavaScript and JSX

- Mixing JavaScript and JSX can often be problematic
- Traditional JavaScript coding techniques may be used, although the more modern ES6 techniques are encouraged, and often require less coding

JSX Iterations

- The working example includes code to iterate over the returned users using JavaScript and ES6 techniques
- Carefully examine each example, and note that each example utilizes the principle that JSX embedded variables need to be objects, so often a solution that creates an array of results works best

ES6 Iteration

- Of special note is the ES6 technique used to iterate over the users array returned from the RESTful call

```
results.map(user => listItems.push(<li key={user.id}>{user.name}</li>));
```

- The code uses the array map() method to push HTML list item elements onto an array
  - The user parameter represents each array element, and as an object, the object properties are available
- Why the li key attribute? Removing this attribute returns the following browser warning:
  - Warning: Each child in an array or iterator should have a unique "key" prop. Check the render method of `Typicode`. See https://fb.me/react-warning-keys for more information.
- The indicated website helps understand why key is needed:
  - Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity

React Proptypes

- Another feature of the working example is the use of React's Proptypes
- Proptypes are useful for type checking (compare to Meteor's check)
- Proptypes are also useful for indicating when a property is required, and this usage is demonstrated in the working example