HTTP

The objective of this lessonis for you to get comfortable with the main concepts of HTTP. HTTP is the underlying protocol used by the World Wide Web. It's essential knowledge for developers who work with the web. At the end of it, you'll be able to identify common HTTP verbs and status codes, as well as demonstrating how HTTP is used by setting up a simple server.

When you finish, you should be able to

- match the header fields of HTTP with a bank of definitions.
- matching HTTP verbs (GET, PUT, PATCH, POST, DELETE) to their common uses.
- match common HTTP status codes (200, 302, 400, 401, 402, 403, 404, 500) to their meanings.
- send a simple HTTP request to `google.com`
- write a very simple HTTP server using ‘http’ in node with paths that will result in the common HTTP status codes.

Promises Lesson Learning Objectives I

Below is a complete list of the terminal learning objectives for this lesson. When you complete this lesson, you should be able to perform each of the following objectives. These objectives capture how you may be evaluated on the assessment for this lesson.

1. Instantiate a `Promise`object
2. Use `Promises`to write more maintainable asynchronous code
3. Use the `fetch`API to make `Promise`-based API calls

Promises Lesson Learning Objectives II

Below is a complete list of the terminal learning objectives for this lesson. When you complete this lesson, you should be able to perform each of the following objectives. These objectives capture how you may be evaluated on the assessment for this lesson.

1. Use `async`/`await`with promise-based functions to write asynchrnous code that behaves synchronously.

HTML Learning Objectives

The objective of this lessonis for you to know how to effectively use HTML5 to build semantically and structurally correct Web pages. HTML is the language that renders the cross-platform human-computer interfaces that made the World Wide Web accessible by the world! You'll be able to create structurally and semantically valid HTML5 pages using the following elements:

- html
- head
- title
- link
- script
- The six header tags
- p
- article
- section
- main
- nav
- header
- footer
- Itemized list tags
    - ul
    - ol
    - li
- a
- img
- Tabular-data tags
    - table
    - thead
    - tbody
    - tfoot
    - tr
    - th
    - td

Testing

The objective of this lessonis to ensure that you understand the fundamentals of testing and are capable of reading and solving specs. **This lesson is relevant**to you because good testing is one of the foundations of being a good developer.

When you finish, you should be able to:

- Explain the "red-green-refactor" loop of test-driven development.
- Identify the definitions of SyntaxError, ReferenceError, and TypeError
- Create, modify, and get to pass a suite of Mocha tests
- Use Chai to structure your tests using behavior-driven development principles.
- Use the pre- and post-test hooks provided by Mocha

Well-Tested Full-Stack To-Do Items

In this project, you will test a full-stack JavaScript and HTML application! You will write tests to make sure the code that was written for the project will meet the expectations of the requirements. Your tests will not have to be exhaustive. Instead, there are guidelines for your tests in each test file. Use those guidelines to implement the Web application.

The upcoming video provides you a full walk-through of the system as it is created. Then, once you understand how the application works from watching it be built, you will need to apply your knowledge of writing tests.

It may be hard. However, stick with it. You'll do great. Just take your time, write good tests, and you will be amazed at how much confidence that you will gain in writing code that comes together.

One of the ways that you can make this project more enjoyable is to vary the way that you pair on it. For each step,

- Discuss what the next feature is that you want to write
- One person writes a unit test to test the code
- Both examine the code and determine if there is any duplication to refactor into common functions or classes
- Loop, but swap who writes the unit test and who writes the code

At the end, you will leverage your test by swapping out the mechanism used to generate the HTML. This is the other part of writing good tests: tests give you the confidence to change code. If you do something that is "wrong" in that it breaks current expectations, the tests will tell you!

Tests are such an important part of a developers life. While some developers will complain about having to write them, when you start working on an existing "legacy" code base, making changes can cause a lot of stress unless you have the work of other developers' tests to make sure you don't unintentionally change a method in such a way to break code that you're not working on.

To get started,

- clone the project fromhttps://github.com/appacademy-starters/testing-an-existing-app-project
- change directory into the project
- run `npm install`to install the modules

If you want to run the server, type `node server.js`and go tohttp://localhost:3000/itemsto see what it does. You can add categories, add items, search for items, and complete items.

The code that you will test are the functions used to create the functionality of the data and the creation of the views, not the actual HTTP server. The video after shows a person writing the entire application. You will see what each piece does. Then, you will understand the *intent* of the code that you have to test.