FEC Engineering Journal and Notes

- [ ] Project Repository: [project-catwalk](https://github.com/fec-bareminimum/project-catwalk)
- [ ] Notion Journal: https://www.notion.so/Front-End-Capstone-f9b9572fe78641eea624557e7dffc665

Contributors

- [Andy Chen](https://github.com/andy-ch3n)
- [Gabrielle Guo](https://github.com/ggbbi)
- [Lawrence Sun](https://github.com/lawsun03)
- [Spencer Lepine](https://github.com/spencerlepine)

Tech Stack

- [React](https://reactjs.org/)
- [Axios](https://www.npmjs.com/package/axios)
- [Express](https://expressjs.com/)
- [Node.js](https://nodejs.org/en/)
- [Jest](https://jestjs.io/)
- [React Test Library](https://testing-library.com/docs/react-testing-library/intro/)

API Docs

- [Atelier API](https://gist.github.com/trentgoing/d69849d6c16b82d279ffc4ecd127f49f)

Entries and notes taken during the Front-end Capstone (Project Catwalk). This was a project completed at Hack Reactor with a team of 4.

These are observations and lessons learned during this project. There were technologies covered that created new challenges throughout development.

Project Catwalk is a MVP client-facing retail web-portal. Browse catalog products with a modern customer experience.

Table of contents

- [9/14/2021](9-14-21.md)
- [9/15/2021](9-15-21.md)
- [9/16/2021](9-16-21.md)
- [9/20/2021](9-20-21.md)
- [9/21/2021](9-21-21.md)

- 9/08/2021

09/22/2021

Overview

With a MVP front-end client roughly feature complete, it was time to deploy the “finished” project to AWS.

Log

Deploying

Challenge/Motivation

Deploy a Node Express + React app to Amazon Web Services (AWS). Run an EC2 instance to serve static files to demo the project.

Actions Taken

Read through the documentation for starting an EC2 instance. We met up as a group to follow a walk-through and complete each initiation step on a team-mates computer. Created a Ubuntu 20 LTS instance through the AWS developer console. Connected via SSH through the terminal with the local public pem key file. We then installed software on the remote Ubuntu machine, such as Git, Nodejs, and Npm. We cloned the remote GitHub repository for the project onto the EC2 instance. We ran the build script with webpack, and installed all npm packages. With all of the static files ready to be served, we could run the Express server and visit the demo website.

Results Observed

This was a great team exercise and learning experience. We worked together to learn a new technology. We had to debug issues with the server not running and incorrect ports. It was a good exercise to debug an unfamiliar tool and go iterate over steps taken.

09/21/2021

Overview

With a minimum viable product complete, our group was ready to work on optimizing the React + Express server.

Log

Optimizations

Challenge/Motivation

Optimize the React app and Express Node.js server. Improve the performance and network request speed. Ensure the site renders images quickly and does not leave the user waiting. Aim for a contentful paint time under 2000ms.

Actions Taken

For the client, we worked on modifying the webpack configuration for bundling and minifying the javascript files. We also tried code splitting so the browser would not deal with large build files. We worked on refactoring how the images were loading in the browser. We explored more npm packages for uglifying the javascript, and caching methods. Those two did not result in marginal performance gains however.

Results Observed

The client improved 5-10 performance points (from Chrome's Lighthouse tool) when we implemented a library called "react-cool-img" ([Article](https://hackernoon.com/a-easy-way-to-handle-image-ux-and-performance-with-reactjs-introducing-react-cool-img-u2jt362o))
which handled image compression for the page. With many product images to load, it was important to render these efficiently. React-cool-img improved to an 85 performance and 3 second paint load time.

The biggest performance boost for the site was a [compression](https://www.npmjs.com/package/compress
) package on the Express server serving static files. This was a middleware that attempted to compress and network responses between the server and client. This dramatically improved the network speed transfers from over 2000ms to 600ms. The overall Lighthouse performance score went from low 50s to 70-80s. Since a client could fetch files/images faster, the page was

able to display a contentful paint in about 2500ms, which was much better than 6000ms plus previously.

09/20/2021

Overview

After making a lot of progress with the React app, it was time to move on from placeholder data. We needed to work as a team and come up with a system to fetch and render product information from an external API.

Log

Global App State

Challenge/Motivation

Merging and connecting each module off the project. Working together as a team to design the sharable/maintainable state management. Fixing errors and refactoring to follow similar patterns throughout the code base.

Actions Taken

Built the Context hooks to ensure React component state interoperability. Each module has slightly different approaches to handle an API request and store the data. We worked together to explain each of our approaches and the functionality needed in the app state. After we refactored each component and context, the app shared data and displayed what was expected again.

Results Observed

This step was critical to have a working MVP. Each module must work together , and the only way to make the codebase maintainable is to carefully design it this way. There is now way to avoid this team collaboration. It also helps improve the quality of the code because each member must write code that is explainable and readable.

09/16/2021

Overview

With most of the feature requirements listed out, it was time to build and practice TDD. I worked on breaking down features into modular components, and then determined how to isolate functionality and write a unit test for that. It helped to follow the practice of using feature-branches with Git to keep the code and updates organized. Keeping an organized system with a Trello Board ticket system also helped. This acted as a TODO list.

Log

React Testing Library

Challenge/Motivation

Write remaining tests for components dealing with state changes, and Context changes.

Actions Taken

Studied the React Testing Library further. The goal was to mock functions, mock API calls, and work with mock Context Providers to assert how components handle different data. First each component would need to properly render placeholder/hard-coded data. Once that was working, each component could be connected to the API with real data. The DOM would update for different features. When clicking a button, a Modal may be rendered, and I would need to test this.

Results Observed

Since it was still new material, writing tests with this library was a big investment. It took a lot of time to write these tests. There was still room for improvement since with the overall pattern of tests written. The unit tests were not robust, which rendered them less beneficial if they break in the long run.

09/15/2021

Overview

With the barebones Components for the Related Products module created, it was time to write more in-depth tests and add state functionality. The code base had everything properly configured for the group project now (webpack bundling, jest test configuration, development file serving scripts, continuous integration, and pre-commit hooks).

Log

Sprint Planning

Challenge/Motivation

Break down each component into sections and list out what tests need to be written for each. Without writing any code, I would create a list of functionality, visual/rendering, and state management tests for these components. Overall, specify/plan from the business requirements what would be tested.

Actions Taken

Create a TODO list for the next requirements:

- ProductCard: test for Price Sale renders when given (over default price)
- StarRating: test renders rating integer with 5 star rating
- ActionButton test (for Related Section)
    - Test value is a star icon
    - Test action opens comparison div
- ActionButton test (for Outfit Section)
    - Test value is a plus symbol
    - Create tests for ProductsContext
- Connect ProductCard to ProductsContext

        - Should fetch more details + styles object if props do not include them
    - Create tests for CartContext
    - Outfit Section
        - Should match list from CartContext
    - RelatedContainer
        - Test the related DIV renders first in DOM
        - Test the outfit DIV renders second in DOM

Results Observed

With a list of clear steps to take for these features, it would be easier to isolate each feature. I could work through writing failing tests knowing exactly what to expect. This would assist the TDD process overall and establish a smooth flow from test to test.

Theme Context

Challenge/Motivation

Create two light/dark themes to toggle between and update the page styles with new color schemes. Create a global Theme for the React App and connect styled-components to them. Needed to tie this in with the current styling library “style components”, and not use bare .css files.

Actions Taken

Read through this amazing [article](https://css-tricks.com/theming-and-theme-switching-with-react-and-styled-components/) about setting up a theme in React. Have a reusable theme schema and pass this to any component in the React app without prop drilling. Set up the useTheme hook, and added the ThemeProvider to give ALL components access to the theme prop. Could not use react-bootstrap themes, since it would not be as customizable.

Results Observed

There were many ways to set up a theme. It was important to find a maintainable and straightforward Theme system, so any team member could easily create Components that would match the Dark/light theme. Needed to have a solid foundation for the Theme state, to avoid any major refactors down the road.

Theme Toggle Button

Challenge/Motivation

Create a theme toggle button to switch between light and dark mode styles. Have a theme provider with styled components. Store the selected theme in localeStorage.

Actions Taken

Found this
[article](https://css-tricks.com/a-dark-mode-toggle-with-react-and-themeprovider/) for a React toggle hook tutorial. Created a Theme hook to save a schema will theme data (of colors and styles). Stored the schema in local storage. Created a ThemeProvider to wrap all React Components in, so they have access to the `theme` value.

Results Observed

This was a very difficult feature to set up. The state was not being updated and it was hard to debug. Had to create a system to update state and keep track of the theme. This way, the page would rerender when theme updates (changes to dark), and styles change.

09/14/2021

Overview

With the code base configured, it was time to begin writing code. After breaking down the app features into smaller tasks, we could begin organizing and building the components. Now that we were familiar with the Git Workflow, it was a smoother process to develop and make iterations across the board.

Log

Functionality Tests

Challenge/Motivation

Testing the html tags being rendered is not not enough. Implement User event tests on Card Components.

Actions Taken

Work with [fireEvent](https://testing-library.com/docs/dom-testing-library/api-events/) function to simulate clicks and hovers for the component. Write asynchronous tests to check if state updates after click. Test if corresponding Context function is triggered on button click. Test the DOM to make sure elements disappear after being closed.

Results Observed

Starting to get more familiar with tests. These tests are much more valuable to ensure functionality and help identify when something breaks.

CircleCI

Challenge/Motivation

Integrate [CircleCI](https://circleci.com/) (continuous integration) to run tests for each pull request. Connect CircleCI to the repository and set up the `.circleci/config.yml` configuration file. Also looked into [Coveralls](https://coveralls.io/) to report coverage, but that was not necessary for now.

Actions Taken

Read through CircleCI [docs](https://circleci.com/docs/enterprise/quick-start/) to create a project. Create config.yml file with node version and test script instructions. Connect GitHub repository to CircleCI.

Results Observed

This will assist the development flow of the team with details for each test. It is always helpful to automate tasks and increase productivity. No need to pull a branch from GitHub just to see if the tests run.

Updating Webpack

Challenge/Motivation

Webpack does not bundle the css files. Hitting errors with watch options (webpack noticing file changes and re-bundling). Need to update the webpack configuration to correctly bundle Javascript files together, AND CSS files. Also need to fix the “watch” option to re-bundle the React app on save (for easy development).

Actions Taken

Implement webpack [css-loader](https://webpack.js.org/loaders/css-loader/) in the webpack.config.js file. This will look for `.css` files and correctly parse them. Also update the [watch](https://webpack.js.org/configuration/watch/) configuration for webpack, which will bundle + refresh files DURING development.

Results Observed

The code base can now handle .css files with the React app. The webpack watch configuration will allow quick development, where a file change will re-bundle, and the browser will already display the updated Components. This is a crucial time-saver.

Jest Mock Files

Challenge/Motivation

Jest throws an error when files import with “.css” or foreign extensions. Jest cannot parse these when a Component tries to import them.

Actions Taken

Research how to allow other file extensions beyond “.js” and “.jsx” with Jest. Found a solution to [Mock Files](https://jestjs.io/docs/webpack). Whenever a Component tries to import (e.g. import Dog from ‘.dog.png’), Jest will REPLACE this with a mock file. Created a `__mocks__` directory to store `mockFile.js` and `mockStyle.js`.

Results Observed

The code base can now handle .css files with the React app. The webpack watch configuration will allow quick development, where a file change will re-bundle, and the browser will already display the updated Components. This is a crucial time-saver.

Jest Mock Function + Provider

Challenge/Motivation

Jest Components render and try to access the Context. To test the context, created a Mock Provider with entering manual values. [Mocking Provider](https://polvara.me/posts/mocking-context-with-react-testing-library)

Actions Taken

I created a MOCK context to wrap the component in. I also created a [MOCK function](https://jestjs.io/docs/mock-functions) with a hard-coded name (updateDisplayedProduct). After simulating a button click, the component should have called that function stored in the Context hook.

Results Observed

This would allow me to build out many useful tests with Mock Functions and custom values passed into the Context from mock Provider. This enables changing the state and adding user functionality.

Card Carousel

Challenge/Motivation

Render a Card Carousel Component from a list of products. See Requirements:
(Related product lists will be shown as a list of product cards displayed in a carousel fashion scrolling horizontally.

> The number of related products will be finite. All of the related products should be present in the list. Due to screen limitations, any product cards that do not fit on screen initially, should appear offscreen in the carousel. On initial load, the list should be centered such that the first related product is all the way on the left hand side of the screen.

In order to navigate through and view the rest of the list, arrows will appear on the right and left hand edges of the list. Clicking the left and right arrow will scroll through the list displaying previous and subsequent cards in the list, respectively. Clicking on the arrow should only scroll through the list one product at a time.

When the first card is all the way on the left of the screen, and no previous cards exist to display, the left arrow should be hidden. This will be the case on initial page load. Similarly, when the last card appears on the far right of the list, the right arrow will be hidden.)

Actions Taken

Research making a React Card Carousel from Scratch. Needed to use state and buttons to trigger a Carousel scroll. The problem with that is connecting it to the styling library we use. Ended up using [react-mutli-carousel](https://www.npmjs.com/package/react-multi-carousel).

Results Observed

Determining whether to build a somewhat complex component from scratch was a trade-off. I looked through numerous articles to understand how to create the Carousel myself, but why reinvent the wheel here? It is less ideal to install ANOTHER npm package, but it is useful for this case.

09/13/2021

Overview

With the code base configured, it was time to begin writing code. After breaking down the app features into smaller tasks, we could begin organizing and building the components. Now that we were familiar with the Git Workflow, it was a smoother process to develop and make iterations across the board.

Log

Test Planning

Challenge/Motivation

Learn React Testing Library and write basic tests for react components.

Actions Taken

Read through many articles, examples, and docs to understand the React Testing Library framework. Looked through the business requirements and translated that IP into code (tests).

Results Observed

This process was meticulous in the beginning as a real investment in the TDD process, but it will make the code base more robust overall. I struggled to figure out what to test, but it appeared to be a system of pattern matching. As long as there was a clear/defined feature I was trying to implement, I could find a way to test it.

Jest Test Utilities

Challenge/Motivation

Create a custom test utility for Jest to WRAP the components in providers and connect useContext for the test environment. Also had to work out kinks with the testing and package setup for the team.

Actions Taken

Read through RTL [DOCS](https://testing-library.com/docs/react-testing-library/setup/) to tie together Context Providers. Had to research issues with extending jest expect statements with the Jest-DOM npm package.

Results Observed

There are unexpected pieces to configuration after using different libraries. Since we took the time to fix these issues and have a common configuration for tests, it will be easier to avoid issues/irregularities later down the road.

09/09/2021

Overview

Today was the second day of the project. We started off reviewing the sprint plan (what to build for the first work period) and following up on the repository configuration and team development from yesterday. We had a meeting to describe our game plan (development) for the next few days. We needed to finalize our decision for technologies also.

Log

Sprint Planning

Challenge/Motivation

create a short-term game plan of how to develop a few features.

Actions Taken

Review business requirements and product wireframe. Create a breakdown/sketch of how to modularize functionality across React Components. Write out steps to follow to create tests before adding components, and what order to follow for features.

Results Observed

This provided clear direction for how to move forward. Having the tasks split up would make it easy to transition, and also separate work on different repository branches.

Research Testing

Challenge/Motivation

Find out how to test React Components BEFORE writing any code, and add configuration for monolith repo and every team member.

Actions Taken

Browsed the docs for Jest, Enzyme, Istanbul, and other common testing/assertion frameworks. Set up a test repo with Istanbul to check code converge. Configure Jest and enzyme and run sample tests.

Results Observed

There were many testing/assertion options. I was able to skim through the docs and see the pattern they follow. Once we have a system to follow, it should be easy to have everyone write tests for our code. (React, and Node/Express)

Researching useContext

Challenge/Motivation

Determine the way to store state in App. Use React Context or Redux? What would be the easiest to collaborate on and organize?

Actions Taken

Read through React useContext hooks DOCS. Read through Redux DOCS. Discuss familiarity and thoughts. Ended up going with useContext to save time.

Results Observed

This was a great exercise to share thoughts and reason what technology to use for the project. It was important to understand that the decision we made would affect us down the line, serving as a foundation for the project state management.

Setting up the Linter

Challenge/Motivation

Fix the linter set with the eslint package and .eslintrc file. Ensure each repo follows a style convention. Set up a pre-commit hook to lint the code base before pushing code.

Actions Taken

Researched how to set up eslint through docs. Had trouble fixing an existing eslint config file, giving very obscure errors. Re-configured eslint with airbnb style guide to follow. Followed docs for eslint setup. Also read through docs for husky pre-commit hooks with git. Configured npm test and npm run lint command to execute BEFORE anybody was able to commit code, which would simplify code-review syntax errors.

Results Observed

A common lint configuration saves a lot of time with syntax errors and enforce style conventions to follow. This is crucial for working with a team where everybody has different coding styles.

09/08/2021

Daily summary of what you worked on & what you learned

Today was about reading through the project requirements and establishing team roles. After reviewing the business requirements, we worked together to plan out the workflow and organization of our work. We created a Trello board for our ticket system to keep track of tasks at hand. We created a monolith GitHub repository to store code and collaborate. I learned the practice of using Trello for tickets and how we could organize in-progress or to-do features. I also learned how to keep separate branches with GitHub to keep a tidy repo. I also learned how to do a code review before merging any code to the actual main branch, which is crucial for iteration.

What went well? What could be improved?

Overall, our team did a great job of getting familiarized with the project and how to navigate the setup. We were able to work through each task step by step without rushing anyone or wasting any time. An improvement could be communicating the task

at hand better? A few times we needed time to read through something and the group needed to make sure everyone was ready before moving on.

What went badly? What have you tried? What is the next step?
One thing that did not go quite smoothly was

What observations do you have on the work done today? Yours? Others?
Today was a productive day with lots of reading to digest all the requirements for the project. Today's work was a wide variety of brief topics, and there was a lot to keep track of. Everyone did their part and kept up with the group.

What pointers or advice would you give someone who is facing the day you just had?
I wish I'd….
Maybe it would have helped to scim over more of the docs/requirements before touching/creating anything? At some points there would be another Learn page that was missed before, and it cleared up the question we were asking.

What bugs/problems did you encounter? How did you explore it? How did you get past it?
We had bugs setting up the linter and getting the webpack configuration. After looking up on the errors on Stack overflow we were able to determine the syntax/naming errors for files on configuration API.