How to Run Our Code

Group Number:101

Team: Jacob Feldman and Andrew Mumm

To run our code, go to our project's website: MatthewEversDesign.meteorapp.com. From there, click on the diamonds to look at information of each project Matt created. The navbar allows you to change between the bio, home, and content sections of the website. Also, you can try resizing the window a bit to see the diamond grid change sizes.

The project backend is a NodeJS server (created by Meteor). We built the front end using a combination of AngularJS, Bootstrap, and Meteor. We also have access to a MongoDB database through Meteor, but we don't use it. We also used Galaxy, a deployment utility created by the same company that made Meteor, to deploy the website onto the web.

Checklist

The first place to look in our code is inside the imports/components folder. There are three folders inside of it (navbar, diamond, and carousel), each of which contains two files: an html template for the component, and a JavaScript file containing each component's respective code, including code for controllers and the actual export component statements. The carousel component we don't actually use in our website, but it was a feature that we originally thought we were going to use, so we made a working one. The diamond component contains functionality to pass in an image it displays, the template for the modal associated with it, the name of the project it's associated with, and the width and height of the image that is to be displayed. The modal templates for each project are stored in the imports/modals folder. Each is specific to the project that it is associated with, and each is imported dynamically by the diamond components based on the modalurl variable passed into the component via element properties in the DOM. The navbar component contains the code for the navbar at the top of the page

Static content, mostly pictures, we saved in the public folder, to be used by other parts of the project. The last part of the code we spent time working with was the client folder. Inside is all the code for the page that is actually served to the client, separated into three files: main.html, main.js, and main.css. Main.js is mostly import statements (every template must be imported into it, which took a while to figure out) with one angular module created at the bottom: 'matt-app.' This module statement contains all the dependencies required by the app to function. Main.html contains all the html that the end user can see displayed, along with some more javascript and css embedded into it. Lastly, main.css contains the majority of our CSS stylings; as you can see from all the commented-out lines, we spent a lot of time writing CSS while creating this file before we managed to produce the desired result.

Title Page

COMS 319

Group 101

Andrew Mumm

Jacob Feldman

We used the Meteor framework with AngularJS and Bootstrap as the front end in order to create a portfolio website for a friend.

Table of Contents

Overview

For this portfolio, we decided to help one of our friends, who is a Design student here at Iowa State, and build his website for him! He gave us a general design to work with at the beginning, and we checked back with him periodically to make sure the functionality we were creating was what he wanted and the way he wanted it. Usually, it wasn't exactly what he wanted, so we had to readjust the functionality or create new functionality for him. We decided to use the Meteor framework to develop this website. Meteor is a full-stack Javascript platform, which means we used NodeJS for the backend, and we had the ability to use a MongoDB database, but we didn't end up needing to. We used AngularJS and Bootstrap for the front-end part of the website. Lastly, we used Galaxy, a deployment tool created by the Meteor Development Team, to deploy the website to the web.

We had to use a back-end to host our web application. We learned about front-end frameworks like Angular and Bootstrap-UI. We decided to use these frameworks because we wanted to learn more about them and they worked well with meteor. We wanted to make a single page web app because we learned that this is the current trend of websites. We applied what we have learned about these frameworks into making a single page application that acts as design portfolio for a friend. Once we got passed the learning curve for these tools, development became easy. In conclusion, these tools fit the purpose of our project very well and provided us with a great new skillset to use in the future.

Creation: We created a single page web application with modern front-end frameworks that we have never used before.

Evaluation: There was a pretty steep learning curve with the frameworks that we used, but after learning how to use them we believe they are very quick and effective ways of building a modern website.

Analysis: The current trend of web apps is to make them as dynamic as possible and to try to include all of the content on a single. To do so a developer may have to learn to use new tools to make sure that all of the pieces of the project, front-end and back-end, work together properly.

New and Complex Section

We used Meteor, a full-stack Javascript framework, to build a website for my friend who is a Design student at Iowa State. Meteor has its own custom way of doing things, but it uses Node.js for the server side, MongoDB for database manipulation, and framework (s) of your choice for the front end. We chose to use AngularJS for a frontend framework. Since our group didn't do the Node.js project or AngularJS project, so this was the first time we actually used either of them. We also used Bootstrap (well, we used angular-ui-bootstrap, but that's just Bootstrap made with Angular) to help with front end code. This was also the first time we had an end user who intends to use the project we create. Because of this, we did more work than just write the website you'll see. Multiple times, we made functionality, showed it to my friend, then went back and changed or got rid of the functionality we had just created because he didn't like the way we'd implemented it. As a result, we have a website with a good amount of polish(for a project we created in a couple of weeks). We also learned a lot about CSS and how difficult it can be.

The area where we spent most of our time working on was the front-end functionality of this website since Meteor is designed to build single page applications. This is also a result of the single-page nature of the website and the fact that Meteor doesn't actually serve up html pages to the client (it uses "Data-On-the-Wire" instead). We used AngularJS to create components that could then be used repeatedly in our actual main page that we send to the client. We spent a lot of time figuring out how exactly we needed to go about creating these components since there weren't any examples or specific documentation for our exact combination of frameworks. As a result, we had to piece together what we needed to do from several different sources of documentation, and that took some time. As is often the case with JavaScript projects, we also had scoping issues when passing data around through components, controllers, and DOM elements.

The other major area of complexity was the actual deployment of the website to the internet. We originally tried to use Amazon Web Services' EC2 (Elastic Cloud Compute) to deploy the website, but we were having problems until we stumbled upon Galaxy, Meteor's own way to deploy the apps you create with their framework. Galaxy is a service that costs a small amount of money, and allows for a fairly streamlined way to deploy and modify your meteor app. It uses Docker and Meteor Up together to deploy the project to the web. As such, anyone can see the website we created from any device that can access the web.

Another thing we struggled with, as is normally the case when learning any new coding framework, was setup of the project. Setting up Meteor itself wasn't terribly hard, but making it play nice with our other frameworks that we used took a good amount of work. However, since this is no longer our first time learning frameworks like these, we managed to set up the frameworks themselves and our environment significantly faster than we did in the other two portfolios.

Bloom's Taxonomy

Analyzing:

We created a Meteor project for our third portfolio. This was the first time that we have ever used meteor. Meteor is a nodejs framework that is ideal for creating single page webapps. The front end frameworks that are supported by meteor are Blaze (built specifically for meteor), Angular, and React. We chose to use Angular for our frontend for this portfolio because we didn't do the Angular lab and we had already used React in the previous portfolio. We didn't use Blaze because it is still fairly new and has some bugs to it yet. Meteor is a very light weight back end framework. It is easy to install and once it has been installed it is already ready to run with a hello world setup. If the user navigates to the project directory on the command line and runs the command "meteor", the project will begin to deploy for testing on localhost port 3000. If you got to the browser and navigate to localhost 3000 the user will see a basic program that was created in blaze using meteor. We used this basic program to understand the file structure of meteor and how the layout of the program works. The setup for meteor is quite simple. There is a package.json file and a node modules folder that contains all of npm modules that you have installed to your project. There is a server folder that has a javascript file that runs the server-side code. There is a client folder that contains main.js, main.html, and main.css. This is the code that will actually get run on the client side. You can also create components and other dependencies for your project that will be referenced to in the imports folder. This is a quite simple layout without any unnecessary code and repetition. This allows meteor to stay light weight and easy to use.

Angular is a great framework for creating single page webapps which is what we were looking to do with this project. With Angular you are able to create components which are essentially like new html elements. These are useful for keeping your code clean and creating new functionality in html. Angular also allows for binding of elements which is great for keeping data consistent between the all aspects of the client. Angular integrates with meteor very well. There is actually a special Angular package that can be installed into meteor. Angular is also pretty straight forward and development with angular is pretty quick once you get the hang of things.

For deployment of our project we originally chose to use EC2 which is an amazon web services deployment option. When creating an instance of EC2 there are many different options and platforms to choose from depending on the purpose of the project. We chose to use a basic Linux tier because the project we are doing is fairly simple and that is what guides we have found recommended. The method for deployment to this container involved using a third party tool for meteor called Meteor Up or mup. Mup is used to streamline the deployment process of a meteor app. If you run mup init you will create a mup.json file that can be customized to contain the location for your project and the location of the container that mup needs to connect to along with any keys or credentials that are need to deploy to this container. Then you run mup setup to connect, upload, and build the project. Then you run mup deploy to actually launch the project. We kept encountering errors in the mup setup phase. There

was an issue with the connection that we couldn't figure out so we ended up trying to use and alternative method of deployment.

The next option that we chose to deploy to is a service that is provided for by the makers of meteor. It is a deployment service specifically for meteor projects called Galaxy. Galaxy makes deployment extremely straightforward. It packages and deploys meteor to a container on the web and allows you to give it your own url. This is what we ended up doing, and now our meteor project is successfully deployed to the internet through this process.

Evaluating:

Angular is a great framework for single page web projects like the one that we made for this portfolio. In our portfolio, there is only 3 different displays. We were able to have a smooth transition from display to display without the need for loading new pages. Angular allows for the binding of elements as well as being able to create new html elements called components. Being able to make new html elements provided us with a way to write really clean code in main.html which was the file that actually gets loaded and displayed to the user. We made three separate components for this project, navbar carousel, and diamond. The carousel ended up not being needed in the final draft of the site. The navbar takes up a large amount of space in an html file, but when it is in the form of a component it only takes up one line of code. This cleans up the code very fast. Meteor was also a great choice for the project that we choice to work on. Meteor is very lightweight and has a simple file structure that is easy to navigate around. With a simple short lived project like this, it is not a problem to use such a light weight framework. Meteor allows for easy deployment and testing on a local server. Our project is a portfolio website for a design student. With that said, the student wants to be able to have the website live on the internet so anyone can view it. Originally, we planned on using Amazon Web Services to host this project. We had read online that EC2 was a great option for hosting a meteor project. We attempted to deploy meteor in this method, but ran into some problems with Meteor Up, a 3rd party service for deploying meteor apps. The next option was to use Galaxy, a hosting service provided by the makers of meteor, specifically for hosting meteor projects. This was not the first option as I would be paying out of pocket for a short while to host this. Galaxy is very well documented, and the process of getting our meteor app up and running on Galaxy was quite simple given the guide provided. Once you have an app running on Galaxy it is very easy to update the project and to manage various aspects of your instance.

Overall, I am happy with the tools we chose to use for this project. Angular is great for single page web apps like our project. Meteor is lightweight and easy to deploy, something I have struggled with in other frameworks. Galaxy is a great service for hosting a meteor deployment, and it is great for updating and managing the project.

Creating:

The first part of the project that we worked on was setting up the meteor framework. We had to download an installer from the meteor website in order to install meteor on our machines. Then you have to navigate to the directory the you want to create a meteor project and run the command “meteor create <project name>”. This command will create a basic meteor project in the directory you are currently located. You can then navigate into this new directory and run the command “meteor” to launch the project on localhost:3000. You can then navigate to this location in the browser to see a hello world meteor project. The next thing that we did was uninstall blaze from meteor. Blaze is the frontend framework that comes with meteor. We are using angular for this project so we then ran the command “meteor npm angular” to install meteor to the project. We also wanted to install bootstrap for use with the project so we ran “meteor npm bootstrap-ui.” After this we had the basic setup for our project complete. The backend was ready to go and we could start writing code for the project. We then used angular to create a navbar component, a diamond component, and a carousel component. The navbar is used for navigating between the various view of the page. This website acts as a portfolio for a design student. The diamond components each house different project that the student has worked on. The diamond displays a picture of the project and when it is clicked on it will pull up a modal that we made for that project. We created 1 modal for each of the diamonds. The modal displays more information about the design project that the student worked on. In addition to the page with the diamonds we also made 2 other pages for our website. We made a page that gives a bio of the student and we made displays the contact information of the student. One of the major things that we wanted to do with our final portfolio is to create a polished and well put together product. We wanted to have a product with a greater production value than our previous projects. We used bootstrap to help us accomplish this task. Bootstrap allowed us to make each page format properly and give the overall website a complete look. The last part that we worked on was the final deployment of the project. Like I said, we wanted the website to be polished so we wanted to host is live on the internet. We used a hosting service called Galaxy, which is provided by the makers of meteor. Understanding how Galaxy worked was easy because the guides and documentation were easy to understand and are made specifically for a meteor project. After launching to the Galaxy hosting service the website is now available at mattheweversdesign.meteorapp.com.

From start to finish we created the back-end of our website, then we installed all of the packages that we would be using, next we created the front-end with angular and bootstrap, and lastly, we deployed the project to Galaxy for hosting on the internet. We were in close collaboration with the design student that the site was for, and this made it feel like a real-world project. We were essentially collaborating with a client from start to finish to give them a complete and polished product.