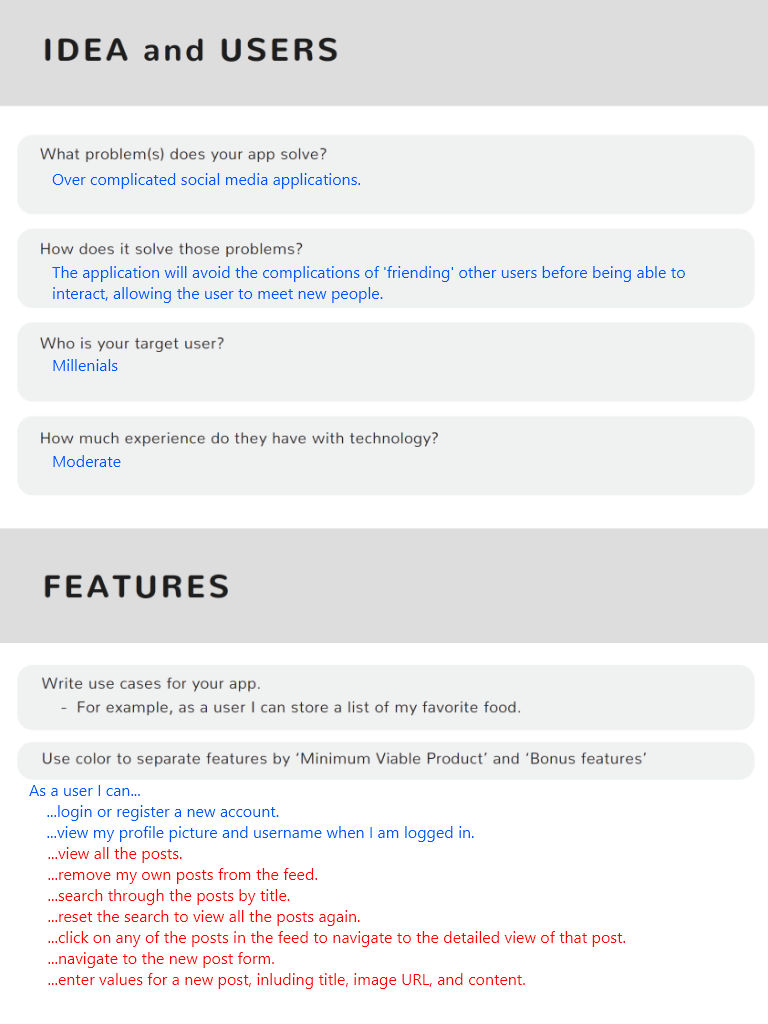
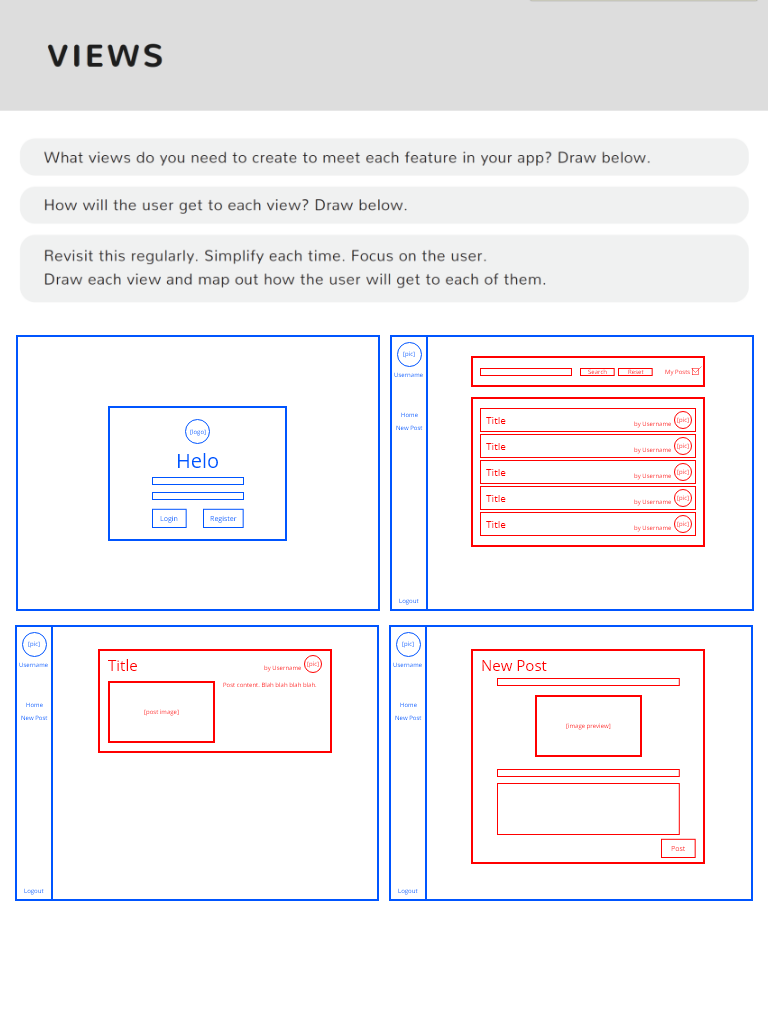
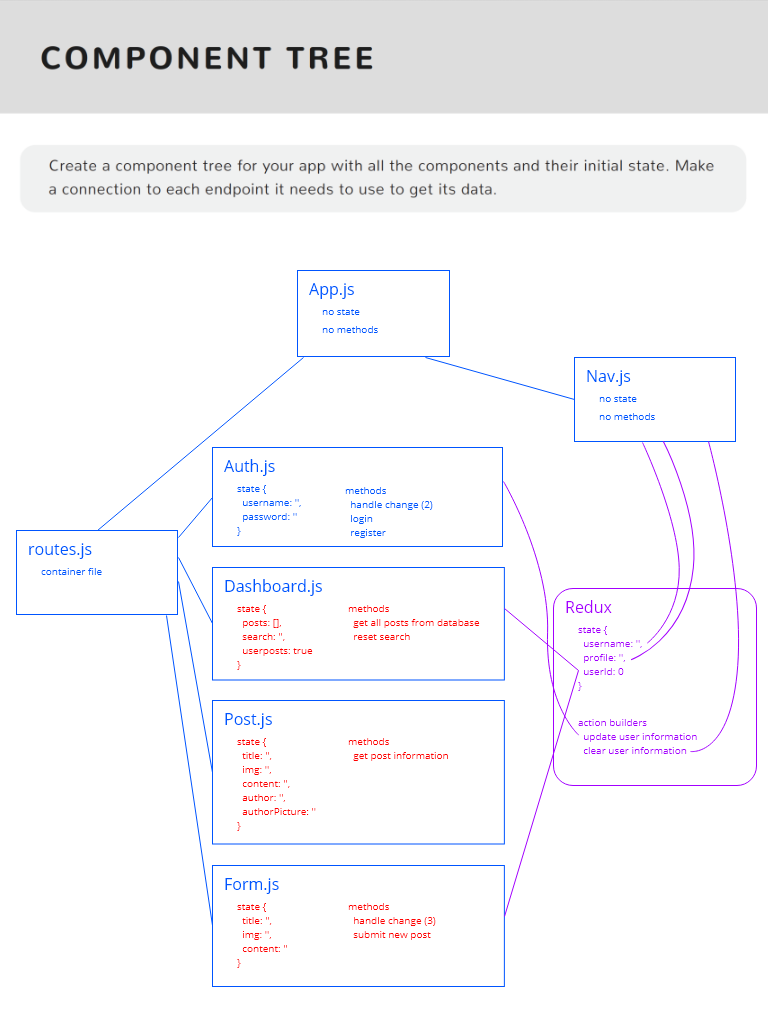
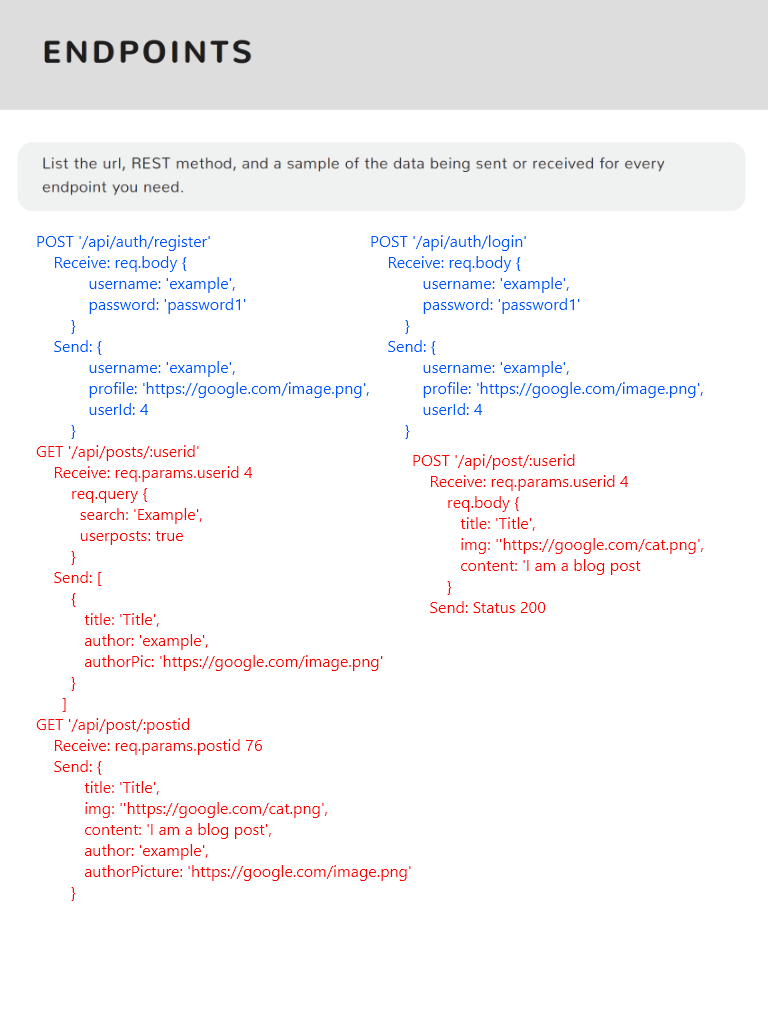
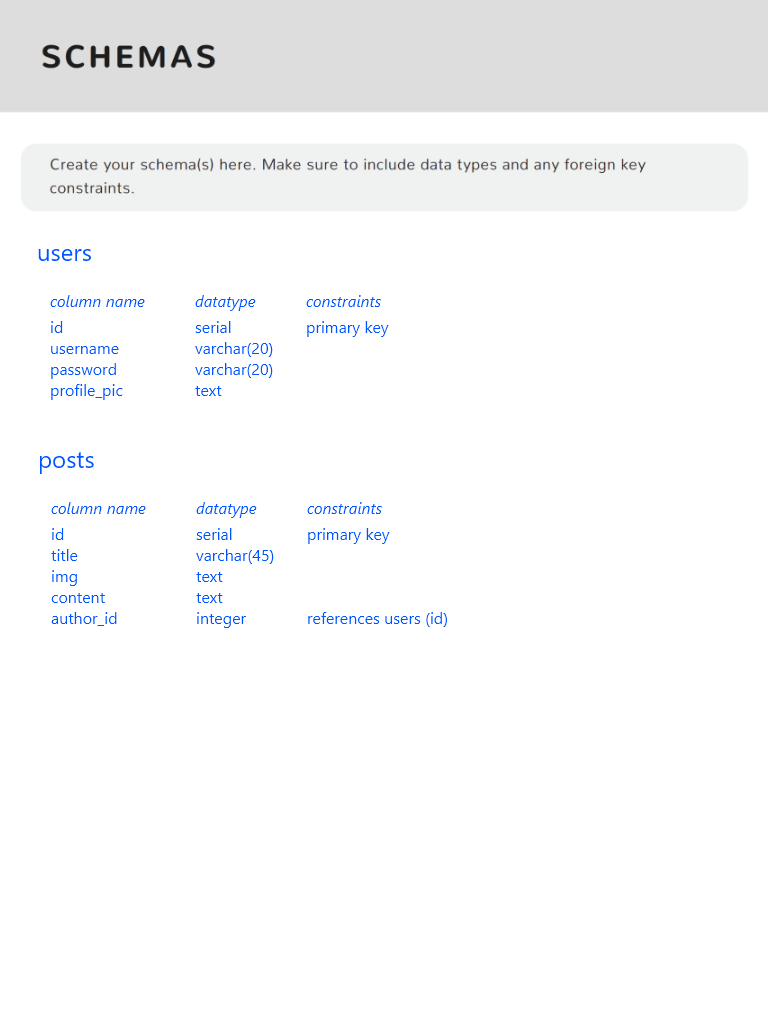


[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/idea_features.png)

[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/views.png)

[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/component_tree.png)

[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/endpoints.png)

[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/schema.png)

[](https://github.com/DevMountain/simulation-3/blob/master/assets/planning-sheet/tracker.png)

The intent of a personal project is to build something using technologies you have been taught, as well as to learn new ones on your own.

Include a link to your Github repository when you submit this assignment.

Requirements

Always required:

* Full stack CRUD site
  + Create/POST
  + Read/GET
  + Update (PUT or PATCH, used [correctly (Links to an external site.)Links to an external site.](https://stackoverflow.com/questions/28459418/rest-api-put-vs-patch-with-real-life-examples))
  + Delete/DELETE
  + Each of the above must manipulate the database (except GET).
* 1 foreign key and a JOIN statement – Joins make it so you can look at two different tables and the info that correlates between them
  + Yes, you need a JOIN. Usually this means you'd be displaying a list somewhere. Some sites, with the small scope of intended functionality, don't lend themselves to such a list with the database's data. A simple workaround for this is to create an admin page which displays the data. Such a page can even be public if it doesn't contain sensitive information; perhaps don't have a link to the admin page on the site itself but tell users in the README or a private email how to view it (e.g. by visiting #/admin). An example of such a list might be, for a store, all orders for all users. Or what about users without an order (use an outer join)?
* A fully planned project
  + Defined MVP features
  + Your expected path of getting points, something like this:

|  |  |
| --- | --- |
| Hosting | 10 |
| Redux | 20 |
| Responsive Design | 20 |
| Presentation Points | 10 |
| Authentication | 10 |
| TOTAL | 70 |

* + Defined API endpoints (Method, URL, short description)
    - Example: GET '/api/cars/:id' returns specific car based on passed in id
  + Defined database schema (Table names, column names and datatypes)
  + 20 tasks in Trello (or Trello equivalent)
  + **You need to pass off your plan with your mentor before you write any code**

You must also use additional technologies to gain points. The technologies and their associated points are listed in the rubric below. Make sure to read the description for any techs you're interested in for details on how to receive points for it.

Planning and Design

Project Planning

* Planning is extremely important. It's said, with tongue in cheek, "Weeks of programming can save you hours of planning". It can be tempting to jump right into code immediately at the start of a project, but you'll quickly start making mistakes and losing time. Taking a good amount of time to plan a project out, before you even write a line of code, will make the coding process significantly smoother and quicker.
* An important concept in planning is the "MVP". This stands for Mininum Viable Product. Simply, what is the smallest, simplest, least-amount-of-features version of your product that you could make that would still be considered completed and passable. For example: I am making a simple chat app. What are some "MVP Features"? The ability to type in a message, the ability to send a message, and the ability to recieve a message. If I want to add any other features, I need to carefully think if they are part of my MVP. A cool animation whenever I send a message? A notification sound when I recieve a message? Those aren't absolutely necessary for my app to function, so I need to put them aside until I finish all my MVP features. When planning, make a list of all the features your app **needs** to function and work on only them until they are done.
* Another important facet of planning is setting tasks for yourself. You can waste a lot of time trying to figure what exactly to do next. Switching constantly between thinking about what to do and actually doing can really slow you down. It's a good idea to take some time to split your project into tasks. The more the better! A good goal is to see if you can split your entire project into hour chunks. If you think you have made as many tasks as you can think of, break the tasks you have into smaller tasks. This will allow you to focus on actually doing the tasks, which will speed up your development times. Trello.com is a good site for task management.
* It's worth noting: plans don't need to be set in stone! You'll find the need for some extra endpoint, or you'll need to change your schemas up, or you'll realize a given feature really isn't MVP status; that's completely fine. It's important to be agile. It's important to plan, but it's also important not to waste time trying to come up with every single eventuality.
* To aid in planning, you might want to use the [project-plan.pdf (Links to an external site.)Links to an external site.](https://github.com/DevMountain/rubrics/blob/master/project-plan.pdf) document. It is optional, and you can use Trello instead or in addition.

Domain Registration

Domain Registration is the process for registering a domain name, which identifies one or more IP addresses with a name that is easier to remember and use in URLs to identify particular web pages. There are many domain registrars out there and not every choice will be the perfect fit in every way, so do your own research. That being said some of the more popular registrars are: namecheap, bluehost, HostGator, Google Domains and GoDaddy.

Points: A registered URL that simply redirects you to an IP will not count towards your domain registration points. It must be fully integrated to the point where your route urls will add on to the end of your registered domain. As in, [www.YourSite.com/#/Home (Links to an external site.)Links to an external site.](http://www.yoursite.com/#/Home) opposed to 138.68.255.191/#/Home and so on.

Gotchas:

* Be aware that registering for a domain name isn’t free. There are many sites that will offer a hosting package that will then give you a free domain name. Take “free” with a grain of salt. It is often cheaper to host elsewhere then wait for a promotion (they happen very often on GoDaddy) where you can register a domain name for $0.99/year.
* Speaking of hosting, that is something you must do before you start the process of hooking up to a domain name.
* It is highly recommended that you get the domain privacy package when registering for a domain name. That will protect against identity theft, prevents domain-related spam, and deters hijackers.
* If you are hosting with Zeit, you must upgrade your Zeit account to the paid version to use a custom domain name.

Responsive

Responsive design means that your application looks good regardless of the size of the screen. Whether the user is visiting your site from their laptop or phone it should keep an effective layout and proper functionality. This is done using responsive CSS, media queries, inline styling, and various other strategies. One main strategy is mobile-first design, meaning you design your site for a phone sized screen first and build it out for larger screen sizes.

Gotchas: Layout and design take alot of time. Making your site look good in multiple layouts will take even more time. BUT if there's a chance an employer may visit your site on their mobile device, this could be valuable time spent!

Custom

For any points you want that aren't mentioned here:

* you must get pre-approval
* there must be an objective way to define the requirement, as is done throughout this document

Techs

3rd party API

Approve by mentor first

Redux

You must read from and write to (modify) the Redux store. You do not technically need exported actions and action creators, but you must have a store, a reducer, and you must dispatch.

Stripe

Stripe is a set of APIs that allows you to take payments. There is a sandbox mode so that you don't have to do real payments. [https://stripe.com/docs (Links to an external site.)Links to an external site.](https://stripe.com/docs)

To get points: you'll need to collect payment information on the client. Eventually you'll need to charge the payment from your server code. If it works, you'll be able to prove it by seeing the new payment in your Stripe dashboard.

Amazon S3

Amazon's S3 service is a bucket to store files in the cloud.

Points:

* 10 points for creating your own S3 account and using files, such as storing images there and showing those images on your website.

Socket.io

Socket.io enables real-time bidirectional event-based communication. It is a great tool to show real-time analytics, instant messengers, push notifications, or do document collaboration. Socket.io works very well with and is simple to integrate into ExpressJS. The format of sockets in your server is similar to creating endpoints. The socket.io docs are not only great, they also have a quick tutorial on how to set up a chat application - [https://socket.io/get-started/chat/ (Links to an external site.)Links to an external site.](https://socket.io/get-started/chat/)

Points: To show your ability to use socket.io ,and get the full 20 points, you must create and use at minimum three unique listeners as well as three unique emitters on both the server and client individually (12 total). To get 10 points you must create one unique listener as well as one unique emitter on both the server and the client individually (4 total).

Gotchas:

* There are many ways to decide who is the recipient of a particular “emit”. Be advised that one way may work in a certain situation but may not in another. The docs have a great emit cheatsheet that will be valuable along the way.
* There are two important parts to Socket.io (server and client) and they both come with the ‘socket.io’ npm package.

ChartJS

In the world of data, visualization plays a key role in portraying the data. Being able to effectively display that data is made a little easier with ChartJS. ChartJS is an open source HTML library, [available as a React extension (Links to an external site.)Links to an external site.](https://github.com/jerairrest/react-chartjs-2) that makes displaying data in the form of a graph or chart fairly simple. The documentation is pretty straightforward, and integration with React is very doable.

Gotchas: ChartJS has certain limitations that other libraries, like D3, do not. While it is easier to use, it has fewer options and customizations. You have x amount of graph/chart options you can use and while interactivity is being added, it is not as updated as D3.

D3

D3 is more of a libary for creating and manipulating HTML than it is a charting libary like ChartJS. The possibilites with D3 are pretty endless. D3 can go way beyond graphs/charts and into other sort of graphics you may need. Like this: [https://bl.ocks.org/mbostock/1353700 (Links to an external site.)Links to an external site.](https://bl.ocks.org/mbostock/1353700). The many ways that you can visualize data with D3 make it pretty awesome.

Gotchas: D3's main drawback is it's difficult to incorporate, there is a steep learning curve.

The syntaxes are more complex than ChartJS because it involves creating each element of the chart. Sifting through the documentation can take awhile. Also, some high-level stuff, D3 directly manipulates the DOM while react is built around manipulating the virtual DOM. Getting React and D3 to play nicely together takes some fiddling. [Here's a good article for this (Links to an external site.)Links to an external site.](https://medium.com/@tibotiber/react-d3-js-balancing-performance-developer-experience-4da35f912484). Don't let these gotchas scare you, D3 is a valuable libary to have knowledge of to impress potential employers.

NodeMailer

Nodemailer is a module for Node.js applications to make sending emails very simple. There are only three simple steps from setting up to sending a Nodemailer. Step one is setting up a transport service that Nodemailer can use to send the emails (it can be any email that you have, or you can create a new one just for the project). Step two is setting up the message options. Message options is a fancy way to say “who sends what to whom”. The last step is just using the sendMail() method on your previously created transporter.

Be advised that there are many challenges with email. Worse still is that email behaves in a not fully predictable way, e.g. sometimes mail will go to spam, and sometimes that same email content, sent later, won't. This is not to discourage you from trying. The reality is that you'll almost certainly send automated email at some point in your career, so learning now still helps, but just to set your expecations.

Points: The ability to send an email with dynamic content with at least 5 in-line styles to a dynamic email address is the requirements for points.

Gotchas:

* Each email server is allowed to operate as it sees fit. The biggest issue is that your email might be flagged as spam by one provider and not another. The reasons messages are marked as spam are rarely given (as this knowledge would aid spammers).
* Email clients (the software that displays the email) can be picky about format. If you only need very simple email messages, you'll be fine. Be advised that all styling must be inline (no external stylesheets; you must use the style attribute), and some styles are ignored and thus don't work, such as flexbox. The rules change over time.
* Just to make sure that you are aware: all styling is in line styling. Styling will be a time sink if you choose to make a complicated email.

Twilio

Twilio has a wide range of services that can be used. They vary from sending SMS messages, phone calls, real-time video, as well as programmable chats all from your Node.js server. For the sake of this project, it would be recommended that you limit your Twilio experience to just the SMS service. Twilio’s SMS service is very easy to use after you go through the extensive set up it requires.

Points: Demonstrating the ablity to send specific information (that makes sense with your project's overall idea) though SMS will net you points here.

Gotchas:

* Twilio does not require a traditional “API key”. To set up your Twilio SMS service you must request an account SID and a Twilio Authentication token. You must then apply for a phone number that will be handling all of the message sending.
* There is a free account type (yay!) as well as paid accounts. With the free account, you are limited to sending messages to a single phone number that you choose during creation and only that number while using the free account. A free Twilio account can be great if you just want to be the sole receiver or the information. If you want to have your users receive messages, you’ll have to fork over some moneys per message sent. Consider a paid plan, so potential employers can try your site and type in their own phone number.

CSS-in-JS

CSS-in-JS is a way to keep styling scoped to a component, without letting styles affect the rest of the site. You can read more about it [here (Links to an external site.)Links to an external site.](http://blog.vjeux.com/2014/javascript/react-css-in-js-nationjs.html).

Examples of CSS-in-JS libraries:

* [Styled Components (Links to an external site.)Links to an external site.](https://github.com/styled-components/styled-components)
* [Emotion (Links to an external site.)Links to an external site.](https://github.com/emotion-js/emotion)
* [Glamor (Links to an external site.)Links to an external site.](https://github.com/threepointone/glamor)

**Points** At least 40 style properties must be from CSS-in-JS in the project. Said another way, 40 lines of CSS (like padding: 10px) must exist as CSS-in-JS styles. You may split those up across as many components as you want.

Build Tools

Less or Sass

Sass and Less are both CSS preprocessors, meaning they extend the CSS language, adding features that allow variables, mixins, functions, and many other techniques. They are a way to simplify your CSS workflow, making development and maintenance tasks easier. Though they are both essentially doing the same job, Sass is used far more in the industry and it is the recommended one if you choose to use a CSS preprocessor.

Points: Using at minimum one example each of variables, mixins, and inheritance as well as 50% of your project wide styling being done in Sass/Less format is the requirment for points.

Gotchas:

* You might hear the term Sass/SCSS, that’s because they are (for the most part) the same thing. SCSS is the preferred syntax when writing Sass. More on that [here (Links to an external site.)Links to an external site.](http://thesassway.com/editorial/sass-vs-scss-which-syntax-is-better)
* Sass/Less are tools that you can become very reliant on because of the simplicity that they bring to your development. Though it is something that you can simply implement in every project you create moving forward, it is important that you don’t forget/neglect the rules of basic CSS.
* Getting a preprocessor up and running might be a headache your first time around. Here is a great guide that will make set up a breeze. [https://github.com/missyjeanbeutler/sass-demo (Links to an external site.)Links to an external site.](https://github.com/missyjeanbeutler/sass-demo)

Webpack

Webpack is the build tool that powers create-react-app. Behind the scenes, create-react-app uses Webpack to perform build steps that eventually result in the bundle.js file your website uses. There is also the webpack-dev-server that is used to run the site in development mode, which allows for file watching and auto-refreshing. One of the neatest features is hot module replacement (HMR), where your new code replaces old code, and your site updates accordingly without doing a full refresh. You state is kept intact, which will make you super happy as a developer.

**Points** 20 points. To get points, you must create a webpack config file yourself. You may not use create-react-app's ejectfunctionality to build it for you.

Hosting

Note: hosting, in general, is tough. You should therefore not wait until the last moment to give it a try. It's tempting to do so, since it's sometimes thought of as the final step, but shouldn't be thought of in this manner because there are considerations to take into account. Depending on your technology choices, you might not be able to host with one solution vs others. See further consideration in the sections below.

Digital Ocean

A note on how Digital Ocean servers (and others like it) are structured. Your app will run on some port number above 1024, like 3000, 3030, or whatever you choose. However, the normal http port is 80 and https is 443, which require root/admin access. Therefore, the pattern is to run a "reverse proxy" tool, such as Nginx or Apache, which will accept web requests on those ports, and forward the traffic to your node server. When setting up your Digital Ocean droplet, ensure you choose to have Nginx, and follow steps to forward the web traffic between ports.

Gotchas:

While many students receive instant access to Digital Ocean, some students were told to wait while their account access request was reviewed. Ultimately some students are denied for reasons not explained by Digital Ocean's communications with those students.

Testing

Unit Tests

Unit tests check to make sure that the individual units (think functions) are working correctly. The idea is to create a suite of tests that, if passed, would give some surety that the functions in your app are set up correctly. Jest is a simple testing framework that you can use.

Points: 5 unit tests will get you full points

Gotchas:

Make sure you research how to set up and tear down your app in order to make sure that you can perform your tests on the right data.

Presentation

What to present

Here are general hints that aren't related to points:

You should practice presenting. Don't wing it.

You should present from the context of being a student at a bootcamp, as opposed to e.g. a solely business presentation or only a code presentation. Start off by mentioning the big picture of your idea. Keep this concise, although if there is background info, like a story to tell, that's fine. Then show the product you built (for most, that's a website), demonstrating its features. Transition to showing interesting/challenging code. You may take questions if you want, but it all counts toward your presentation time.

The idea is to get you in the practice of showing off your talents, and being able to communicate technical info, such as code.

Don't:

* show off registering for your site... assume everyone knows that concept
* mention your *experience* of the personal project, as there is already plenty of that discussion during day-to-day conversation
* mention what's broken or what you didn't get to

Here's a good example presentation: [https://youtu.be/ZG7\_U153Xqw (Links to an external site.)Links to an external site.](https://youtu.be/ZG7_U153Xqw)

Uses Time Effectively

Presentations should be 3 minutes long. To get points, you must be within 20 seconds of the 3 minute mark (2:40 - 3:20)

**Points** 10 points. 0 points if you go under or over.

Rubric

**Personal Rubric**

| Personal Rubric | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning OutcomeREQUIRED - Full CRUD | |  |  | | --- | --- | | 0.0 pts  Full Points | 0.0 pts  No Marks | | 0.0 pts |
| This criterion is linked to a Learning OutcomeREQUIRED - Foreign key and Join statement | |  |  | | --- | --- | | 0.0 pts  Full Points | 0.0 pts  No Marks | | 0.0 pts |
| This criterion is linked to a Learning OutcomeREQUIRED - MVP Plan  *You MUST pass this off with your mentor before you code* | |  |  | | --- | --- | | 0.0 pts  Full Points | 0.0 pts  No Marks | | 0.0 pts |
| This criterion is linked to a Learning OutcomeResponsive Design | |  |  | | --- | --- | | 20.0 pts  Full Points | 0.0 pts  No Points | | 20.0 pts |
| This criterion is linked to a Learning OutcomeRedux | |  |  | | --- | --- | | 20.0 pts  Full Points | 0.0 pts  No Points | | 20.0 pts |
| This criterion is linked to a Learning OutcomeHosted with Digital Ocean | |  |  | | --- | --- | | 10.0 pts  Full Points | 0.0 pts  No Points | | 10.0 pts |
| This criterion is linked to a Learning OutcomePresentation  *Use your time effectively when presenting* | |  |  | | --- | --- | | 10.0 pts  Full Points | 0.0 pts  No Points | | 10.0 pts |
| This criterion is linked to a Learning OutcomeAuthentication | |  |  | | --- | --- | | 10.0 pts  Full Marks | 0.0 pts  No Marks | | 10.0 pts |
| This criterion is linked to a Learning OutcomeOther Tech  *Pick different techs to add to your project to get the remaining points you need. ------------------------------------------------------------------- the following are worth 10 points each: - Amazon S3 - ChartJS - NodeMailer/SendGrid - Twilio - Sass/Less - Css in JS - Domain Registration - 3rd party API - other (must be approved) ------------------------------------------------------------------- the following are worth 20 points each: - Stripe - Sockets - D3 - Custom Webpack - Unit Tests - other (must be approved)* | |  |  |  |  | | --- | --- | --- | --- | | 30.0 pts  Full Marks | 20.0 pts | 10.0 pts | 0.0 pts  No Marks | | 30.0 pts |
| Total Points: 100.0 | | |