IGME 430: Rich Media Web App Dev II
Project 2 Overview

Table of Contents

Overview

Individually, you will need to create a web application using Express.js server architecture and a front-end developed with React.js. You may use your DomoMaker D/E as a base for this project, but you will need to significantly alter the project so that it neither looks nor functions like DomoMaker. Essentially you may use the DomoMaker project for its login system and basic code structure.

Failure to differentiate from DomoMaker, including not replacing the entire stylesheet, not removing all references to “Domo” in the code, not removing the images of Domos, etc. will result in a **50% grade deduction** on the final submission for the project.

You may also choose to start a codebase from scratch, but you will be required to reimplement things like a login system, sessions, etc. to meet the requirements of the project.

Application Description

The actual application is fairly open-ended. Your goal is to create a satisfying and engaging web application that users would actually use in the real world. An application like DomoMaker serves no actual purpose and does nothing for people. Your application will need to have a clearly defined use and will need to be implemented in such a way that it serves its purpose.

The application must have user accounts and a use for those user accounts. That means some sort of data privacy, where data is either private to a specific user or can only be seen by a subset of users. Public data can also exist. For example, if I were making a Twitter clone I could restrict my tweets so that only my followers could see them.

Your application needs to have a profit model implemented into it. You are **not required** to attempt to make profits, but the app should be designed in such a way that it would be possible. The profit model needs to be implemented. Examples of this include placeholder ads, a “premium” mode for user accounts, etc. You **should not** ask for, collect, or store financial information. For something like a premium model, it is sufficient to have an account page with a button that simply toggles premium mode to show the functionality working.

Your application should be a rich and engaging experience for the users. There should be a hook to get them interested. If this was a real application you were making to support yourself financially, you would want it to be of benefit to those using it. Part of the grade will be based on creativity, how engaging the experience is, the quality/polish of the experience, and going beyond the requirements.

Regardless of the application you choose to make, keep in mind that it should be scoped to be completed within a couple of weeks. **Please remember that a significant portion of the grade is based on polish.** Be sure to allocate time for polish, bug fixes, etc.

Project Ideas

Some projects that have been made in the past include project management tools, mood board creation applications, social media platforms, real-time and turn-based web games, etc.

When determining the concept for your project, there are a few questions to ask yourself.

1) Is this idea something that I can reasonably develop over the course of a few weeks? Unfortunately, most times that people set out to make very ambitious projects end in disaster. A contained and well-polished experience will be far better than a half-finished large-scale project. If you finish your minimum viable product with time to spare, add features to it until you are satisfied with it.

2) What kind of data will I be storing? What would the data model for that information look like? Is it feasible to implement a data model like that, as well as the controllers and views to handle it all?

3) How long will it take me to polish the front end? Remember that not only does your app need to work, but it should also feature a well-implemented user experience. A non-technical recruiter should open your app and be able to tell what it does. It should have a well-thought-out and implemented CSS stylesheet that presents the page well. It should have clear user interactions. All of this development will take time.

4) What would the profit model for a project like this be? How would I implement that?

5) Is this a project I am passionate about making? Oftentimes you will not be given the opportunity to work on something you want to make. If you don't have an idea you are passionate about you can certainly fall back on something, but having a project you *want* to make will make the whole process more enjoyable.

If you have a project idea but are unsure if it is feasible or acceptable for this project, feel free to speak with your professor to discuss the idea in depth.

Below are a few professional applications that could be accomplished (with a limited feature set) for this project.

Twitter - https://twitter.com/
Twitter is an excellent example of something doable for this project. Users can make accounts that allow them to 1) make tweets and 2) follow other accounts to see their tweets. Twitter also has some privacy settings to restrict information based on user preferences.

Imgur - https://imgur.com/
Imgur is a simple and effective image-sharing service. It is possible to upload images without an account and have them available for everyone on the internet to access. You can also create an account to upload unshared images that can only be accessed via a generated link or through your own account.

Agar.io (or similar games) - https://agar.io/
Agar.io, and many other ".io" games, store a reasonable amount of data such as leaderboards. They also keep track of private user data such as skins, etc. Real-time games like these make use of a technology called WebSockets, which we have not covered in class so be aware that this is at the "upper limit" of what is possible for this project. Also, keep in mind that developing the game will take considerable time without factoring in the development of the backend system and the menus that tie into it.

Project Requirements

The following are all the criteria your project must meet.

- The project must not contain any references to Domos and must be **significantly** different from DomoMaker. This includes an entirely new stylesheet (not just a recolor), no references to Domos anywhere in the code, no Domo images, etc. Failure to do this will result in a 50% deduction from the final submission grade.

- Client code must implement at least 5 dynamic React components besides login/signup/password change.
- At least some of the pages/views need to be dynamic, meaning they have variables and/or content that changes based on the user and/or context.

- You must have a login/signup system with session management.
    - Sessions must be persisted in Redis.
- You must have individual user accounts.
- You must have a password change system.
- Account passwords must be stored using a password-safe form of encryption (such as bcrypt).
- The server needs to both read from and write to MongoDB using Mongoose in addition to the login/signup/password change features.
    - Some amount of this data should only be accessible to a subset of users.

- Client code must make requests using fetch() or similar javascript API.
- Server code must implement GET and POST request handlers. Other HTTP methods are optional.
- The application must implement a 404 page or redirection for non-existent pages.
- Server code must return appropriate status codes where necessary.

- The Application must implement a proof of concept profit model.
    - You must **not** collect users' payment information.
    - A donation button is not acceptable as a profit model.
- The Application should be a satisfying and engaging user experience.
- The Application must have a definitive purpose and must be useful for something of value. That can be for entertainment or for practical use.
- The Application must be designed and implemented to professional standards.

- Client code must be written using React.
- Server code must be written in Node with Express.
- The project must use the Handlebars view engine.
    - If it makes sense to make an entire page with templates (ie. the page is static after it loads in the browser) you can use Handlebars templates. If the page is dynamic, use React. You still need to meet all React requirements above.

- You may not use a Content Management System (CMS) such as Ghost, KeystoneJS, etc.
- You may use external web APIs / services but it is not required. They may not be used in place of your own code.
    - Connections to external APIs and services must be done from your server code.
    - Connection information / API keys must be stored in config vars, not hardcoded into the project code.

Functional Requirements

- Client code must be transpiled and bundled (using webpack + babel, or other similar tools) before server runtime.
- Must use Git for version control in a repo that the professor can access.
- Must use ESLint with the Airbnb spec.
    - Airbnb spec cannot be edited/disabled through any means.
- Must use CircleCI for build testing.
- Server code should use libraries from npm listed in the package.json file.
- Must use a cloud service (such as Heroku) for deployment.
- Borrowed code and code fragments must be credited in code comments and in the documentation for the project.
- Separation of Concerns: your code should be appropriately broken up into files and functions based on the main functionality of those pieces of code.
- D.R.Y. - Don’t Repeat Yourself. If you have multiple nearly identical blocks of code, those should be factored out into separate functions.
- Code must be well commented. You don’t need to comment on every line. Have a comment for each function, and comments for confusing lines of code. Ideally, code should be “self-documenting”, meaning the variable and function names explicitly state what they are and what they do.
- Code must be free of runtime and ESLint errors.

Above and Beyond Work

As detailed in the grading rubric below, completing all the above requirements will yield a grade of 90% overall. The last 10% is reserved for above and beyond work. That is to say, to get above a 90% on this project you must go beyond the requirements.

Some examples of above and beyond work would include: utilizing a package from npm to aid in the development of your server, utilizing a CSS framework to increase the professional look and feel of your frontend application, utilizing features of libraries that we have not covered in class (such as React component composition, etc).

Points for above and beyond work will be distributed based on the size and quality of the implementations. Something that takes a line of code and a few minutes of work will net a few points. A larger scale integration that drastically improves the application will yield a maximum of 10 points.

Remember that a large goal of this project is to create something you can show off on your portfolio. The more you can differentiate yourself from your peers the better. Recruiters and employers like to see that you learned some tool on your own, as it’s something you’ll have to do frequently in the workplace.

Deliverables

There are two deliverables for this project. The first is a milestone (worth 30% of the final project grade), and the second is the final submission (worth 70% of the final project grade). The entire project accounts for 30% of the final course grade. Each part is due on the dates listed on their respective MyCourses dropboxes.

Milestone Submission

The milestone submission should be a “proof of concept” for your application. It should, to some extent, show your core features working on both the server and client.

It is okay if things are not polished at this stage, or if they still have references to DomoMaker. It should, however, be well beyond DomoMaker in terms of functionality.

Things can be unstyled, not work 100% of the time, features can be missing, etc. We simply need to see some significant progress underway by the time of the milestone submission.

You must submit the following things for the prototype submission.

- A live Heroku link to your application.
- A git repo containing the code for your project. If the repo is private, be sure to give access to your professor.
- A CircleCI link showing an (ideally) passing build of your most recent commit.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Documentation submitted to the MyCourses dropbox, **separate from the .zip file**. See below for details.

Documentation

You must submit a documentation document with your code and link submissions. This should be in a .doc, .docx, .pdf, or .txt outside of your .zip folder. In the documentation, answer the following questions and provide endpoint documentation.

Questions

Answer the following questions in full sentences. Please break out your answers into individual paragraphs so that they are easier to parse.

- What is the intended purpose of your application?
- What work has been completed for this milestone?
- What work is left, and how do you plan to complete it?
    - What does your timeline/roadmap look like to finish on time?
- How are you using React?
    - What components have you made?
    - What components do you still plan to add?
- What data are you storing in MongoDB?
    - What data do you still need to store?
- What is your profit model?
    - Have you implemented it yet?
        - If so, how?
        - If not, what is your plan to implement it?
- Do you have a plan for going above and beyond? If so, what is it?
- If you used any borrowed code or code fragments, where did you get them from? What do the code fragments do? Where are they in your code?

Endpoint Documentation

You must document each data endpoint (each one listed in your router.js) with the following information:

- The URL of the endpoint
- The supported methods of the endpoint (GET, POST, etc)
- What middleware the endpoint uses.
- Any query/body parameters that the endpoint accepts/requires.
- A brief description of what the endpoint does and returns.

Example:

URL: /getDogByName
Supported Methods: GET, HEAD
Middleware: Requires Secure, Requires Login
Query Params: dogName (the name of the dog being searched for)
Description: Retrieves information about a specific dog.
Return Type(s): JSON

Final Submission

This project is worth 30% of the final grade for the course. 30% of that grade comes from the prototype. The other 70% comes from this final submission. The final submission may not be submitted late.

You must submit the following things for the final submission.

- A live Heroku link to your application.
- A git repo containing the code for your project. If the repo is private, be sure to give access to your professor.
- A CircleCI link showing a passing build of your most recent commit.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Documentation submitted to the MyCourses dropbox, **separate from the .zip file**. See below for details.

Documentation

Answer the following questions in full sentences. Please break out your answers into individual paragraphs so that they are easier to parse.

- What is the purpose of your application? What does it do?
- How are you using React?
    - What components do you have?
- What data are you storing in MongoDB?
- What went right in the development of this project?
- What went wrong in the development of this project?
- What did you learn while developing this project?
- If you were to continue, what would you do to improve your application?
- If you went above and beyond, how did you do so?
- If you used any borrowed code or code fragments, where did you get them from? What do the code fragments do? Where are they in your code?

Endpoint Documentation

You must document each data endpoint (each one listed in your router.js) with the following information:

- The URL of the endpoint
- The supported methods of the endpoint (GET, POST, etc)
- What middleware the endpoint uses.

- Any query/body parameters that the endpoint accepts/requires.
- A brief description of what the endpoint does and returns.

Example:

URL: /getDogByName
Supported Methods: GET, HEAD
Middleware: Requires Secure, Requires Login
Query Params: dogName (the name of the dog being searched for)
Description: Retrieves information about a specific dog.
Return Type(s): JSON

Grading Rubric / Checklist

The grading process for projects is deductive. Each project starts with a base grade of 90%. Then, based on the table below, points are either reduced or increased.

Description	Value (%)
Above and beyond work.	+1 to +10
Code/project contains references to DomoMaker, including (but not limited to) a stylesheet reminiscent of DomoMaker, references to Domos in the code, Domo images, etc.	-50
Client code does not implement at least 5 dynamic React components besides login/signup/password change.	-5 per missing component
Client does not have any dynamic views or pages.	Up to -20
Application does not have a login/signup system with sessions.	Up to -20
Application does not have individual user accounts.	Up to -15
Application does not implement a password change system.	-5
Server does not store passwords using a proper encryption format.	-20
Server does not read and write to and from MongoDB in addition to login/signup/password change.	Up to -15
Application does not implement a 404 page or redirection for non-existent pages.	-5
Application does not implement a proof of concept profit model.	-10
Application does not provide a rich and engaging user experience.	-5 to -20
Application does not have a definitive and useful purpose.	-5 to -20
Application is not designed and implemented to professional standards.	-5 to -20
Application does not have stable performance / is throttled due to hardcoding, etc.	-5 to -20
Client does not properly utilize fetch or a similar API for making data requests to the server.	Up to -10

Project does not make proper use of required technologies (React, Node, Express, MongoDB, Redis, Handlebars, etc.)	-10 per technology
The project uses a CMS such as Ghost, KeystoneJS, etc.	-100
If the project uses external APIs: connections to external APIs / services are not done from the Node server.	-10
If the project uses external APIs: connection information for APIs are not stored in config vars and are instead hardcoded into the codebase.	-5
Server does not use context-appropriate status codes.	-3 per code
Server API does not support the following methods: GET, POST	-5 per method
Project does not use Git for version control or the professor cannot access the code repository on GitHub.	-20
Project does not use ESLint with the Airbnb spec.	-20
Airbnb spec has been modified, or ESLint errors have been manually silenced.	-5 per instance
Code is not free of ESLint errors.	-5 per error
Code is not free of runtime errors.	-5 to -50 based on severity
Project does not use CircleCI.	-10
Project does not use Heroku or another cloud platform.	-20
Borrowed code or code fragments are not commented in the code and the documentation.	-100
Code is not well organized and commented.	-5 to -20
Submission does not include documentation with the proper questions answered / with proper endpoint documentation.	-10

A Note on Git Usage

While it will not count directly against you, there should be a substantial number of commits in your project repository by the time of the final submission. Proper utilization of version control is essential in the industry.

If you only push your entire project to your repo after it is complete, that is a failing on your part to understand why git exists. Git is only useful if you regularly commit your code. Without doing so, rollback points are not created and you cannot undo mistakes you have made without manual code editing.

Be aware that recruiters and tech teams do look at things like git history.