U-IMPACTIFY

System Design

Fall 2020 CSCC01 Final Project

Ivin Able | Shree Shah | Simar Bassi | Harry Geng | Bryan Liu | Eric Li | Runjie Zhang

Table of Contents

Front-End CRC Cards

Landing Parent class: none Subclasses: none Responsibilities: - Directs the users to sign-up or login page Collaborators: - Topbar	**Signup** Parent class: none Subclasses: none Responsibilities: - Contains fields for signing up with personal information - Contains button to login if user has account - Send login details to backend (Firebase) Collaborators: - Topbar - Redux	**Login** Parent class: none Subclasses: none Responsibilities: - Contains fields for login credentials - Has a button to submit login form - Provides user with appropriate errors if any - Has links to signup page if user doesn't have an account Collaborators: - Topbar - Redux
Dashboard Parent class: none Subclasses: none Responsibilities: - Allows students to see courses - Allows instructors to see courses they've made - Allows organizations to see courses the entire organization are taking - Directs user to CoursesPage - Directs user to Giving Garden - Directs user to individual course pages Collaborators: - Navbar - Coursecard - Coursewrapper - Redux	**Topbar** Parent class: none Subclasses: none Responsibilities: - Directs the users to sign-up or login page Collaborator:	**Footer** Parent class: none Subclasses: none Responsibilities: - Provides the users with social media links to contact the developers Collaborator:
Navbar Parent class: none Subclasses: none Responsibilities: - Have a button to go back to dashboard - Have a button to go to Giving Garden - Have a button to go to Courses - Have a button to go to Profile	**Courseboard** Parent class: none Subclasses: none Responsibilities: - Shows all courses that a user can take - Directs user to each course Collaborators: - Navbar - Coursewrapper - Redux	**GivingGarden** Parent class: none Subclasses: none Responsibilities: - Provides a platform for users that cannot afford courses to ask for financial aid and organizations to provide free courses - Have a button to create a new post on GivingGarden Collaborators:

- Have a button to sign out Collaborators:		- Navbar - Redux - Postwrapper
Coursecard Parent class: none Subclasses: none Responsibilities: - Provide the user with brief information about the course - Have a button to enroll in the course Collaborators: - Courses - Redux - ProfilePopup	**Courses** Parent class: none Subclasses: none Responsibilities: - Provide the user with the course page - Have a button to enroll in the course - Have a button to access course content Collaborators: - Navbar - Redux	**Coursewrapper** Parent class: none Subclasses: none Responsibilities: - Show Coursecards with ability to scroll through - Dropdown button to show more info Collaborators: - Coursecard - Redux
Postwrapper Parent class: none Subclasses: none Responsibilities: - Show Postcards with ability to scroll through - Dropdown button to show more info Collaborators: - Redux - Post	**Post** Parent class: none Subclasses: none Responsibilities: - Provide the user with brief information about the course - Have a button to show interest in post Collaborators: - Redux - ProfilePopup	**Profile** Parent class: none Subclasses: none Responsibilities: - Provide user with the ability to modify profile description - Modify first name, last name, contact info. Collaborators: - Redux - NavBar
Redux Parent class: none Subclasses: none Responsibilities: - Stores data from the database in store for classes to use Collaborators: - Posts - Courses - Users	**CourseCreation** Parent class: none Subclasses: none Responsibilities: - Provide an instructor with the ability to create a course with all its content Collaborators: - Navbar - Redux	**CustomButton** Parent class: none Subclasses: none Responsibilities: - A wrapper component to easily create and customize a button Collaborators:

ScrollButton	**ProfilePopup**	**ProfileView**
Parent class: none Subclasses: none Responsibilities: - Renders a button that allows the user to scroll back to the top of the page Collaborators:	Parent class: none Subclasses: none Responsibilities: - Displays a popup of a user's details given a user's email Collaborators: - ProfileView	Parent class: none Subclasses: none Responsibilities: - Styles a user's details into a component for viewing Collaborators: - Users

Back-End CRC Cards

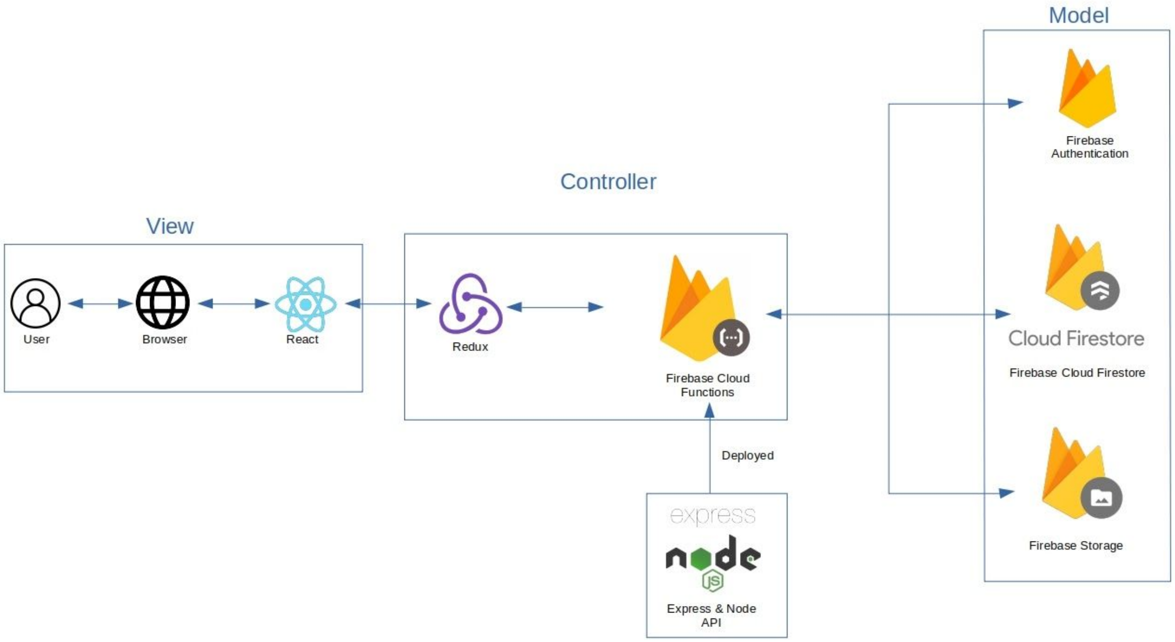
Users	**Courses**	**Posts**
Parent class: none Subclasses: none Responsibilities: - Signup - Login - Get user courses - Get course by id - Get user type - Get authenticated user details - Get user details by email - Update user details - Enroll in course Collaborators: - Courses - Posts	Parent class: none Subclasses: none Responsibilities: - Get course by id - Get all courses - Creates new courses - Uploads image to firebase storage - Generate upload URL to firebase storage Collaborators:	Parent class: none Subclasses: none Responsibilities: - Get individual posts by id - Get all posts - Make post Collaborators:

System Interaction with Environment

We assume that the user has proper internet access as well as a browser to access the internet with. Users may signup for the application with a valid email and password. They will then be able to access the application at any time, by logging in. Users, specifically instructors, can upload course content which include videos and PDFs. Any User will be able to upload an image as their User Avatar. Both of these features will require the user to allow the application to have access to their system file storage.

The web application may be started by first cloning the Git repository to your computer, and running both frontend and backend. To run frontend locally , go into the ./frontend folder and run the command `npm install` to download modules/libraries followed by `yarn start` to start the server on a port (default localhost:3000). To run the backend go into ./backend/functions and run the command `npm install` to download modules/libraries followed by `npm install -g firebase-tools’ to download the firebase command line tools and finally `firebase serve` to run the API service locally (default port 5000, the full url is given in console output).

Architecture of System



Component Breakdown

User	The user is someone who interacts with the UImpactify react application (the view) in order to perform actions to manipulate data provided by the model, which is delegated by the controller.
Browser	The browser allows the web application to be viewed so that the User can interact with it.
React	React is the frontend framework of the application which displays the state of the application to the user and allows the user to interact with the application.
Firebase Cloud Functions	These functions interact with the Model in order to retrieve and/or manipulate data stored in the database, or allow authentication.
Redux	Redux is the state management container which controls the current state of application. It also manages the communication between the View and Model; representing state modification on the view.
Express and Node API	API endpoints are accessed by Redux in order to perform application logic or data manipulation. These functions are then deployed to Firebase Cloud Functions so that it can be accessed by Redux.
Firebase authentication	Provides backend services and SDK's to authenticate users into the application, this includes authentication by identity providers such as Google, Facebook, and Twitter.
Firebase cloud firestore	A cloud database that can store long-term application data such as user information, courses, user posts.
Firebase cloud storage	A cloud database that is used to store media content such as images, pdfs, videos.

System Decomposition

Redux

Redux will be used as a state management container that allows us to change the UI according to what is required. When the User interacts with the UI, an action is dispatched to the redux store. This action does what is necessary such as interacting with the API endpoints in order to retrieve information from the Firebase Storage, which is then put into the Redux Store state. Redux reducers are what is used to change the state of the application based on the action that was dispatched. When the application state in the redux store is updated according to the action that was dispatched, the required objects, such as user courses, posts and more, can be retrieved by the UI from the store.

Firebase cloud firestore

Firebase cloud firestore will be used to store numerous things such as: user account information, courses, user posts. Relations between the User, the courses they take as a student or the courses they create as an instructor, and the posts they make for the Giving Garden will be handled by this database. This storage is for long-term application data and is a part of the Model in the system architecture; it will be accessed through the Firebase Cloud Functions to eventually return data that updates the state of the application in the Redux store.

Firebase cloud storage

Firebase cloud storage will be used to store media content for courses that are created and User information. This involves any PDFs, videos and images for the course or User avatar. Similar to firebase cloud firestore, this database will hold long-term application data and is also a part of the Model in the system architecture. This too will be accessed through the cloud functions and eventually update the application state for the UI to use.

React

React acts as a part of the View in the MVC architecture which creates the User Interface that allows the User to interact with the application. When the User interacts with the UI, Redux is used with React in order to change the state of the application and modify the UI accordingly. React displays errors that are returned by Redux when actions are being performed. In case of input validation, it is displayed on the UI (such as for login and signup) with specific error messages.

Firebase Cloud Functions

These are functions that are hosted on the cloud that are responsible for assessing the Firebase cloud firestore and Firebase cloud storage in order to retrieve and/or manipulate data. The functions are assessed by Redux actions when a User interacts with the UI. These functions are then responsible for sending application data back to Redux which uses reducers to store application state that can be assessed to modify the UI.

Firebase Authentication

Firebase authentication is a service that allows the controller to authenticate users into the application. Using Firebase, the controller can create a new user and save it to under authentication along with the password so that it is securely stored, then it can use the users credentials to authenticate the user into the application. Firebase authentication comes equipped with a FirebaseUI as a drop in authentication solution as well as SDKs that can be used to create custom sign-in methods. The API deployed on Firebase Cloud Functions contains endpoints that deal with sign-up/login, which use these SDKs to manually integrate Firebase Authentication into our application.

Express and Node API

Express and Node API is used to deploy the Controller for the backend which is responsible for creating the logic. It is deployed on the cloud so it uses a serverless framework which means that we as the developers are not responsible for scaling it. This is also what redux will send the http requests to.