Easy App-ly
Systems Design Document
CSCC01H3 – Fall 2021

Instructor: Ilir Dema

Drop Table Team

Kourosh Jaberi | Jiale Shang | Christina Ma | Raymond Kiguru |

Jan Marchan | Mohammad Rahman | Anika Sultana

Table of Contents:

CRC Cards

Class: Profile	
Parent Class: none subclass: none	
Responsibilities - Stores data from form used to create profile for users - Returns a React Component for displaying a profile - Using axios makes API calls - Uploads a resume - Validates user inputs - Submits a profile form so a profile can be created for users	Collaborators - Login

Class: ViewProfile	
Parent Class: none subclass: none	
Responsibilities - Gets information about searched user - Returns a React Component for viewing a profile - Using axios makes API call - Displays a video player for users to view elevator pitches	Collaborators

Class: ActiveFeature	
Parent Class: none subclass: none	
Responsibilities - Displays the parent container for all user features	Collaborators - NavBar

Class: HostFair
Parent Class: none subclass: none

Responsibilities	Collaborators
- Renders the form used for organizing a job fair, allowing for date/time selection and information entry.	- None

Class: NavBar	
Parent Class: none subclass: none	
Responsibilities - Displays the dashboard drawer and dashboard components	Collaborators - Profile - Login - Logout

Class: Jobs	
Parent Class: none subclass: none	
Responsibilities - Displays the job page - Allows users to search for jobs based on company and job title	Collaborators - None

Class: ElevatorPitch	
Parent Class: none subclass: none	
Responsibilities - Displays the elevator pitch page - Allows elevator pitch upload	Collaborators - FileUpload

Class: Signupform	
Parent Class: none subclass: none	
Responsibilities - Stores data from signup form used to create a new user - Displays the sign-up page form	Collaborators - None

- Created custom styles for sign-up form - Validates user inputs - Submits user sign-up form by calling APIs using axios	

Class: Home	
Parent Class: none subclass: none	
Responsibilities - Displays the home page with notifications and calendars - Allows a user to search for other users through a textfield - Shows a notification button to user which can display all their notifications	Collaborators - HomeJobs - Logout - Notification - ViewProfile

Class: HomeJobs	
Parent Class: none subclass: none	
Responsibilities - Displays the components which shows users jobs best fitted for them in the home page	Collaborators - JobPosting

Class: JobPosting	
Parent Class: none subclass: none	
Responsibilities - Displays the components which shows a singular job posting	Collaborators - None

Class: Landing	
Parent Class: none subclass: none	
Responsibilities - Displays the landing page where users can choose to signup or login and get short introduction to Easy App-ly - Creates custom styles for buttons and text fields on the landing page - Redirects to sign in page when clicked signup - Redirects to login page when clicked login	Collaborators - Login - ModalDialogue

Class: ModalDialog	
Parent Class: none subclass: none	
Responsibilities - Gets props received from App.js	Collaborators Form

Class: App	
Parent Class: none subclass: none	
Responsibilities - Shows different components of our application like home, jobs, elevator pitch and profile	Collaborators - Home - Landing - CreatePosting - Jobs - Profile - ElevatorPitch

Class: Index	
Parent Class: none subclass: none	
Responsibilities	Collaborators

- Renders the app component	- App

Class: Login	
Parent Class: none subclass: none	
Responsibilities - Lets the user login and creates a user session	Collaborators - Profile

Class: Logout	
Parent Class: none subclass: none	
Responsibilities - Logs the user out and removes them from session	Collaborators - None

Class: CreatePosting	
Parent Class: none subclass: none	
Responsibilities - Lets users post a new job	Collaborators - FileUpload

Class: FileUpload	
Parent Class: none subclass: none	
Responsibilities - Component for uploading any files	Collaborators - None

Class: ScheduleMeeting	
Parent Class: none subclass: none	
Responsibilities - Lets users to input the required data to schedule a new meeting	Collaborators - None

Class: ExtensionLanding	
Parent Class: none subclass: none	
Responsibilities - Displays the landing page for chrome extension where users can login - Redirects to the homepage for chrome extension when user logged in	Collaborators - Login

Class: ExtensionHome	
Parent Class: none subclass: none	
Responsibilities - Display a button for auto-fill forms on current tabs with user's information - Display a button for logout from chrome extension	Collaborators - None

Class: NotificationDialog	
Parent Class: none subclass: none	
Responsibilities - Gives a dialog component to be used by Notification.	Collaborators - Notification

Class: Notification	
Parent Class: none subclass: none	
Responsibilities - Gives a list of notifications to users in the homepage	Collaborators - None

Class: CreateMeeting	
Parent Class: none subclass: none	
Responsibilities	Collaborators - None

- Communicates with Zoom API to schedule a new meeting	

Class: ViewJobFairList	
Parent Class: none subclass: none	
Responsibilities - Displays a list of upcoming job fairs to seekers, or displays a list of hosted job fairs to an employer	Collaborators - None

Class: ViewJobFair	
Parent Class: none subclass: none	
Responsibilities - Displays an employer's job fair posts to a job-seeker	Collaborators - None

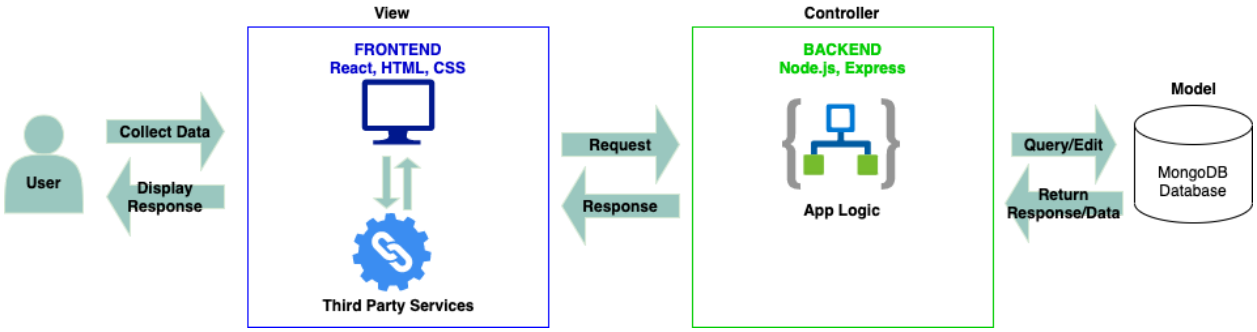
Class: JobFairPost	
Parent Class: none subclass: none	
Responsibilities - Allows an employer to create a job fair post	Collaborators - None

Description of System Interaction

Our program assumes that the user has node.js (14.18.0 or later) installed. The recommended operating systems to run this program are MacOS, Linus or Windows 7+. In order for users to run and view our web application on their local device, the steps have been documented in our README.md from Sprint 0, under the *Setup* section. The steps direct users on how to install all the required dependencies for the frontend and backend prior to running the web application. The instructions also advise users how to successfully run the backend and frontend of our web application.

Description of System Architecture

We are using the MERN (MongoDB, Express, React, Node.js) stack as our technologies of choice in developing this web application. The architecture follows how most web applications are structured. The frontend, which uses React, CSS, and HTML, is connected to the backend using HTTP. The frontend sends API requests to the backend. The backend receives the API request that will query/modify the database. The diagram below describes our high level system architecture.



System Decomposition

Users will access the web application by interacting with the web pages on our frontend. From here, users can make requests, which are packaged and sent as API requests to our backend. The backend will call on the required functions to query or modify the MongoDB database as required. Data is then sent back (if a query was requested) or confirmation to the user that the modification was successful. The API requests users can make are, but are not limited to:

- Creating an account
- Logging in
- Viewing available jobs
- Searching for a specific job

- View user profiles
- View interviews scheduled

All user inputs are validated before sending an API request to the backend to ensure the user input instructs a valid request. If an invalid user input passes the first line of defense in the frontend, the backend will also complete its own checks to validate if the request will cause damage to the database. If so, the request is stopped. These steps describe how our application identifies errors and how it is dealt with.

As an example, an error that can occur is when users are logging in. The frontend will ensure the user provides both a password and username (e.g. cannot be empty fields), and the password and username consist of legal characters. Once these requirements are met, an API request is made to the backend to query if the password and username match an existing pair in the database. The result of this query is returned to the backend and passed to the frontend. The user is then informed if the login was successful or not.