System Design Document

GROUP MJAARNS

Mutasem, Julio, Andy, Anaqi, Rebecca, Nazmus, Sneha

CRC CARDS

Class name: Company profile	
Parent Class: None Subclasses: None	
Responsibilities: ● Display information about a company	Collaborators: ● Banner, Biography, Employees, ProfileInfo

Class name: Instructor profile	
Parent Class: None Subclasses: None	
Responsibilities: ● Display information about a instructor	Collaborators: ● Banner, Biography, ProfileInfo

Class name: Partner profile	
Parent Class: None Subclasses: None	
Responsibilities: ● Display information about a Partner	Collaborators: ● Banner, Biography, ProfileInfo

Class name: Entrepreneur profile	
Parent Class: None Subclasses: None	
Responsibilities: ● Display information about a Entrepreneur	Collaborators: ● Banner, Biography, ProfileInfo

Class name: Banner	
Parent class: None Subclasses: None	
Responsibilities: ● Display the banner image ● Display the profile picture	Collaborators: ● None

Class name: ProfileInfo	
Parent class: None Subclasses: None	
Responsibilities: ● Display the profile specific information	Collaborators: ● None

Class name: Biography	
Parent class: None Subclasses: None	
Responsibilities: ● Display the biography of the user	Collaborators: ● None

Class name: Employees	
Parent class: None Subclasses: None	
Responsibilities: ● Display the Employees of a Company	Collaborators: ● None

Class name: Documents	
Parent class: None Subclasses: None	
Responsibilities: ● Display the documents of a company	Collaborators: ● None

Class name: Settings	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays the name, username and email of the user. ● Allows the user to update their first name, last name, username, email and password.	Collaborators: ● None

Class name: Employee	
Parent Class: None Subclasses: Partner, Entrepreneur, Company, Instructor	
Responsibilities: ● Keep track of employees	

Class name: Routes	
Parent Class: None Subclasses: None	
Responsibilities: ● Links the frontend pages with their respective APIs	Collaborators: ● Register ● Login ● authSettings ● updateSettings

Class name: Header	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays and links to other frontend pages at the top of the page as a navbar	Collaborators: ● None

Class name: Register	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays the register form. ● Allows the user to register into the application. ● Send the information from the register form to userAction.	Collaborators: ● Selection ● Header ● userAction

Class name: Selection	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays the selection criteria for registering into the website	Collaborators:

Class name: Login	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays the login form ● Send the information from the login form to userAction	Collaborators: ● Header ● userAction

Class name: Types	
Parent Class: None Subclasses: None	
Responsibilities: ● Actions of the application	Collaborators:

Class name: userAction	
Parent Class: None Subclasses: None	
Responsibilities: ● Sends the information of the registration and login form from frontend to backend via API	Collaborators: ● Types

Class name: settingAction	
Parent Class: None Subclasses: None	
Responsibilities: ● Sends the information of the update settings form to the backend via the API.	Collaborators: ● Types

Class name: reducers/Index	
Parent Class: None Subclasses: None	
Responsibilities: ● Combine all reducers of the application	Collaborators: ● userReducer

Class name: userReducer	
Parent Class: None Subclasses: None	
Responsibilities: ● Tracks the changes of states in register and login	Collaborators:

Class name: settingReducer	
Parent Class: None Subclasses: None	
Responsibilities: ● Tracks the changes of states when user is updating	Collaborators: ● Types

Backend CRC

EndPoint: GET(/profile/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Fetches the user at the given id from the database ● Populate the information for the specific user and return it	Collaborators: ● Models ● controller

EndPoint: POST(/register)	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets the user information given from the frontend and saves it to the database	Collaborators: ● Models ● controller

EndPoint: PUT(/profile/edit/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Takes the info sent and updates it accordingly in the database.	Collaborators: ● Models ● controller

EndPoint: POST(/login)	
Parent Class: None Subclasses: None	
Responsibilities: ● Checks for the user in the database with the unique email to see if a user matches. Then, checks the password of that user to see if it matches ● Sends confirmation to the frontend	Collaborators: ● Models ● controller

EndPoint: POST(/profile/auth)	
Parent Class: None Subclasses: None	
Responsibilities: ● Checks for the user in the database with the unique email to see if a user matches. Then, checks the password of that user to see if it matches ● Authenticates user to be able to update the information ● Sends confirmation to the frontend	Collaborators: ● Models ● controller

EndPoint: PUT(/profile/update/settings)	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays current information of the user, such as name, username and email ● Allows user to update their information ● Sends confirmation to the frontend	Collaborators: ● Models ● controller

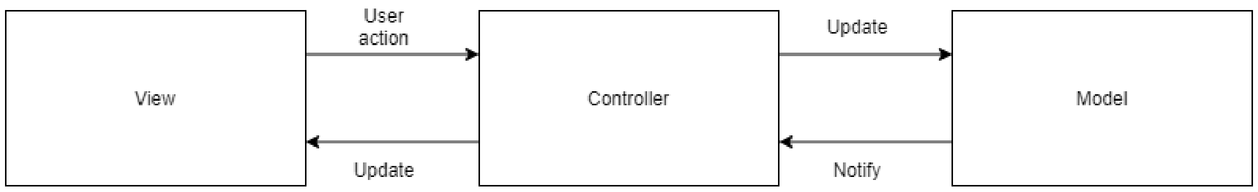
DESCRIPTION OF SYSTEM INTERACTION

Everyone is using macOS/Linux or a Linux virtual machine from windows, thus this is the recommended environment. The MERN framework is being used with MongoDB as the database which we are currently using locally. We are using Mongoose to speed up development. Express.js is used for the backend. React is used for the frontend with bootstrap, and Node.js is the runtime for the entire application. The assumption is that anyone who wants to develop or run the application should have all of these applications or frameworks installed.

SYSTEM ARCHITECTURE

Our group used a variation of the model-view-controller architecture discussed in class. In this design, we have a view, which represents the front end components of the project and what the user interacts with. Through this interaction, an event will be signaled to the controller. The controller will then figure out which is the correct response. The model is what talks to the controller and represents the database. It holds our schema as well as the information needed for the application. The controller may fetch or update information from the model as needed.

A link has been provided for a detailed explanation



https://www.intuz.com/blog/guide-on-mvc-vs-mvvm

SYSTEM DECOMPOSITION

Each page has its respective View, Controller, and Model components. The view component of a page interacts with the Controller to send user input and receive information to view to the page. Before sending, this component will do some basic input validation and ensure that the user does not enter bad input. The Controller interacts with the model to retrieve and add information to the database. The controller is also divided up into smaller components like the register controller which deals with all events related to registration. There will be validation in the controller to ensure that request failures are caught and reported appropriately. Furthermore in the model, the database schemas will have rules for each field which mongoDB will enforce so that bad input will never be posted into the database.