System Design Document

GROUP MJAARNS

Mutasem, Julio, Andy, 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	Collaborators: ● None

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: ● None

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: ● None

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: ● None

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

Class name: Home	
Parent Class: None Subclasses: None	
Responsibilities: ● Home page for logged in user. ● Serves as a creation hub to create both posts and modules. ● Provides access to other major functionalities of the application.	Collaborators: ● HeaderAuth ● Post ● ModuleCard ● postAction ● moduleAction

Class name: Post	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays created posts ● Allows for editing, deletion, and commenting of posts.	Collaborators: ● postAction

Class name: ModuleCard	
Parent Class: None Subclasses: None	
Responsibilities: ● Shows the name of the module and the instructor’s name. ● Serves as a link between the home page and the actual module page.	Collaborators: ● None

Class name: AuthHeader	
Parent Class: None Subclasses: None	
Responsibilities: ● Authenticated navbar of currently logged in user ● Allows to navigate through all the major functionalities of the app.	Collaborators: ● userAction

Class name: module	
Parent Class: None Subclasses: None	
Responsibilities: ● Display information about the module created by the instructor ● Allows instructor to upload assignment and videos	Collaborators: ● AuthHeader

Class name: SearchHeader	
Parent Class: None Subclasses: None	
Responsibilities: ● Filtering types of Users ● Filtering Companies that seek funding	Collaborators: ● Profiles

Class name: Profiles	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays a list of Profiles based on filters	Collaborators: ● SearchHeader ● Profile

Class name: Profile	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays a basic information for a profile user ● Connect to the profile page of the displaying user	Collaborators:

Class name: ProfileSearchPage	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays a list of profile searches ● Display filter options for searches	Collaborators: ● AuthHeader ● SearchHeader ● Profiles

Class name: ProfileEditPage	
Parent Class: None Subclasses: None	
Responsibilities: ● Display the current user information	Collaborators: ● AuthHeader

● Allows user to upload profile picture and add changes to their profile	● EditGeneral ● EditCompany

Class name: EditGeneral	
Parent Class: None Subclasses: None	
Responsibilities: ● Display the current user general information ● Allows the user to edit the general information defined for all users	Collaborators: ● None

Class name: EditCompany	
Parent Class: None Subclasses: None	
Responsibilities: ● Display company profile information ● Allows company to upload documents and make changes to their company profile	Collaborators: ● None

Class name: moduleEdit	
Parent Class: None Subclasses: None	
Responsibilities: ● Allows the instructor to upload assignments, content and lectures to the module	Collaborators: ● Assignment ● Content ● Video ● AuthHeader ● moduleAction

Class name: moduleAction	
Parent Class: None Subclasses: None	
Responsibilities: ● Sends the information of the modules to the backend via the APIs	Collaborators: ● Types

<table>
<tr><td colspan="2">Class name: assignmentAction</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Sends the information regarding assignments, such as creation and fetching assignments via the APIs</td><td>Collaborators:
● Types</td></tr>
</table>

<table>
<tr><td colspan="2">Class name: moduleReducer</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Tracks the changes of states regarding modules</td><td>Collaborators:
● Types</td></tr>
</table>

<table>
<tr><td colspan="2">Class name: assignmentReducer</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Tracks the changes of states regarding assignments</td><td>Collaborators:
● Types</td></tr>
</table>

<table>
<tr><td colspan="2">Class name: AssignmentView/Assignments</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Displays card with all the single assignments</td><td>Collaborators:
● AssignmentView/Assignment</td></tr>
</table>

<table>
<tr><td colspan="2">Class name: AssignmentView/SingleAssignment</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Displays each assignment individually and a link that opens the assignment</td><td>Collaborators:</td></tr>
</table>

Class name: AssignmentUploadCard/Assignment	
Parent Class: None Subclasses: None	
Responsibilities: ● Adds assignments to the assignment card	Collaborators: ● AssignmentUploadCard/SingleAssignment

Class name: AssignmentUploadCard/SingleAssignment	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays assignments to be added and allows removal of assignments	Collaborators:

Class name: ContentView/Content	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays card with all the content	Collaborators: ● ContentView/SingleContent

Class name: ContentView/SingleContent	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays each content individually and a link that opens the content	Collaborators:

Class name: ContentUploadCard/Content	
Parent Class: None Subclasses: None	
Responsibilities: ● Adds content to the content card	Collaborators: ● ContentUploadCard/SingleContent

Class name: ContentUploadCard/SingleContent	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays content to be added and allows removal of content	Collaborators:

Class name: VideoView/Videos	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays card with all the videos	Collaborators: ● VideoView/SingleVideo

Class name: VideoView/SingleVideo	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays each lecture individually and a link that opens the lecture	Collaborators:

Class name: VideoUploadCard/Videos	
Parent Class: None Subclasses: None	
Responsibilities: ● Adds lectures to the lecture card	Collaborators: ● VideoUploadCard/SingleVideo

Class name: VideoUploadCard/Videos	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays lectures to be added and allows removal of lectures	Collaborators:

Class name: AssignmentView	
Parent Class: None Subclasses: None	
Responsibilities: ● Display student view of assignments added by instructor	Collaborators: ● assignmentAction

Class name: ContentView	
Parent Class: None Subclasses: None	
Responsibilities: ● Display student view of content added by instructor	Collaborators: ● moduleAction

Class name: LectureView	
Parent Class: None Subclasses: None	
Responsibilities: ● Display student view of lectures added by instructor	Collaborators: ● moduleAction

Class name: submission	
Parent Class: None Subclasses: None	
Responsibilities: ● Displays submissions of students and the assignment for which it was submitted	Collaborators: ● moduleAction ● assignmentAction

Class name: SingleAssignmentInstructor	
Parent Class: None Subclasses: None	
Responsibilities: ● Allows instructor to submit feedback file, comment and grade the submissions	Collaborators: ● assignmentAction

Class name: Zoom	
Parent Class: None Subclasses: None	
Responsibilities: ● Generates an encrypted signature for users to enter a meeting. ● Allows users to join a meeting as host or as a participant.	Collaborators:

Class name: InstructorView	
Parent Class: None Subclasses: None	
Responsibilities: ● Allows instructors to host/create meetings.	Collaborators: ● Zoom

Class name: StudentView	
Parent Class: None Subclasses: None	
Responsibilities: ● Allows entrepreneurs/students to join meetings.	Collaborators: ● Zoom

Backend CRC

<table>
<tr><td colspan="2">EndPoint: GET(/profile/{id})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Fetches the user at the given id from the database
● Populate the information for the specific user and return it</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: GET(/profile/getUsers)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Gets all the current users with populated information</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: GET(/profile/getImage/{id})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Gets the images for the user referenced by id</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: GET(/profile/getDocument/{name})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Gets the document by its name</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: GET(/profile/editImage/{id})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Saves the image to the user referenced by id</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: GET(/profile/addDocuments/{id})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Add documents to the user referenced by id</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: POST(/register)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Gets the user information given from the frontend and saves it to the database</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: PUT(/profile/edit/{id})</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Takes the info sent and updates it accordingly in the database.</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: POST(/login)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>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</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: POST(/profile/auth)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>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</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: PUT(/profile/update/settings)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Displays current information of the user, such as name, username and email
● Allows user to update their information
● Sends confirmation to the frontend</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

<table>
<tr><td colspan="2">EndPoint: POST(/createModule/:id)</td></tr>
<tr><td colspan="2">Parent Class: None
Subclasses: None</td></tr>
<tr><td>Responsibilities:
● Creates a new module given the id of the instructor who is creating it, and the name of the module.</td><td>Collaborators:
● Models
● controller</td></tr>
</table>

EndPoint: GET(/getrecmodules)	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets the last 10 modules that have been created and send them to the frontend ● Populates user who created the module before sending it to the backend	Collaborators: ● Models ● controller

EndPoint: PUT(/deletemodule)	
Parent Class: None Subclasses: None	
Responsibilities: ● Deletes module with the id that have been sent from the frontend in the request body	Collaborators: ● Models ● controller

EndPoint: PUT(/post)	
Parent Class: None Subclasses: None	
Responsibilities: ● Create a new post with the title, text, image and creates a post associated with the user id who created it, which is in the request body	Collaborators: ● Models ● controller

EndPoint: PUT(/comment)	
Parent Class: None Subclasses: None	
Responsibilities: ● Create a new comment with the commenter and the comment sent from the backend. ● Checks if there is a post associated with the comment so it can reference it inside the comment.	Collaborators: ● Models ● controller

EndPoint: GET(/getrec)	
Parent Class: None Subclasses: None	
Responsibilities: ● Fetches the most recent posts from the database. ● Populates the poster information and the comments related to that post.	Collaborators: ● Models ● controller

EndPoint: PUT(/editpost)	
Parent Class: None Subclasses: None	
Responsibilities: ● Edits the body of a post with the post id sent from the backend in the request body. ● Checks if there is a post with the post id sent in the request body. Send a 404 status code if the latter is the case.	Collaborators: ● Models ● controller

EndPoint: PUT(/deletepost)	
Parent Class: None Subclasses: None	
Responsibilities: ● Deletes the post associated with the post id that is sent from the frontend in the request body	Collaborators: ● Models ● controller

EndPoint: PUT(/post)	
Parent Class: None Subclasses: None	
Responsibilities: ● Create a new post with the title, text, image and creates a post associated with the user id who created it, which is in the request body	Collaborators: ● Models ● controller

EndPoint: GET(/getLecture/{name})	
Parent Class: None Subclasses: None	
Responsibilities: ● Fetches the mp4 video which has the file name {name}.	Collaborators: ● Models ● controller

EndPoint: POST(/assignment/create)	
Parent Class: None Subclasses: None	
Responsibilities: ● Create a new assignment with the user and file name associated with it.	Collaborators: ● Models ● controller

EndPoint: GET(/assignment/:{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets the assignment with the associated id.	Collaborators: ● Models ● controller

EndPoint: PUT(/assignment/edit/:{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Edits the information of the assignment associated with id.	Collaborators: ● Models ● controller

EndPoint: GET(/assignments/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets all of the assignments that the instructor with id {id} has posted.	Collaborators: ● Models ● controller

EndPoint: GET(/assignments/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets all of the assignments that the instructor with id {id} has posted.	Collaborators: ● Models ● controller

EndPoint: GET(/assignment/assignments/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets all of the assignments that the entrepreneur with id {id} has submitted.	Collaborators: ● Models ● controller

EndPoint: GET(/assignment/{id}/:name)	
Parent Class: None Subclasses: None	
Responsibilities: ● Gets all the assignments for the user with id {id} which has name {name}.	Collaborators: ● Models ● controller

EndPoint: POST(/assignment/marked/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Saves the document with the instructors markings and comments for the assignment with id {id}.	Collaborators: ● Models ● controller

EndPoint: POST(/assignment/marked/{id})	
Parent Class: None Subclasses: None	
Responsibilities: ● Saves the document that the entrepreneur wants to submit for assignment with id {id}.	Collaborators: ● Models ● controller

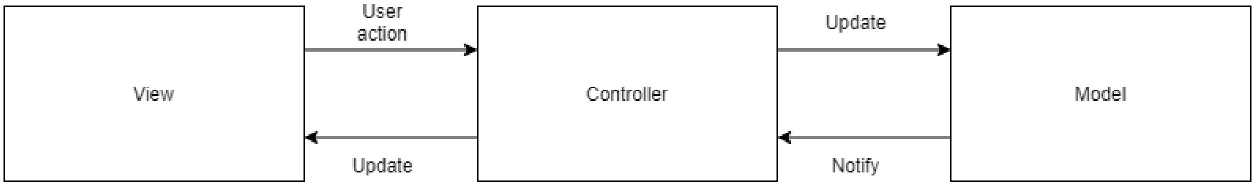
DESCRIPTION OF SYSTEM INTERACTION

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.