Software Design

Software Design Documentation 🔗

Table of Contents 🔗

CRC Cards 🔗

Backend 🔗

Class Name: User	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• stores user information • supplies CRUD operations on users	None

Class Name: Timeline	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• stores user's skills and experiences with corresponding date • supplies CRUD operations on timeline	None

Class Name: Profile	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• stores user's profile - uses' email, description, etc. • supplies CRUD operations on user's profile	None

Class Name: Tag	
Parent Class: N/A **Subclasses:** N/A	

Responsibilities	Collaborators:
• stores tags of users • supplies CRUD operations on tags	None

Class Name: UserTag	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	**Collaborators:**
• stores users and tags pairs • supplies CRUD operations on userTag	None

Class Name: Like	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	**Collaborators:**
• stores rate infromation • supplies CRUD operations on like	None

Services 🔗

Class Name: AccountService	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	**Collaborators:**
• Manages HTTPS request for user login, logout, sign up/register. • Creates a profile for each user created. • Gives error if user inputs are invalid	User Profile

Class Name: ProfileService	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	**Collaborators:**
• Manages HTTPS request for profile CRUD • Gives error if user inputs are invalid or user does not have permission to edit the profile	Profile

Class Name: TagService	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	**Collaborators:**
• Manages HTTPS request for tag CRUD	Tag UserTag

Frontend 🔗

Class Name: LoginPage	
Parent Class: N/A	
Subclasses: N/A	

Responsibilities	Collaborators:
• Allows user to enter the user name and password to login in • Can link to sign up page and profile page • Gives alerts for failed login and invalid inputs	• SignupPage • ProfilePage • AccountService

Class Name: SignupPage	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Allows user to enter the email, password, user name and chose the identity to sign up • Can link to login page • Gives alerts for failed signup and invalid inputs	• LoginPage • AccountService

Class Name: ProfilePage	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Displays user profile	• ProfileService

Class Name: TagProfileForm	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Allows user to add and delete their tags	None

Class Name: MomentieTimeline	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Displays and allows user to add, edit or delete on list of timelines	• MomentieTimelineItem

Class Name: MomentieTimelineItem	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Displays and allows user to add, edit or delete an item of a timeline	None

Class Name: Rating	
Parent Class: N/A	
Subclasses: N/A	
Responsibilities	Collaborators:
• Displays a rating	None

Software Architecture Diagram 🔗
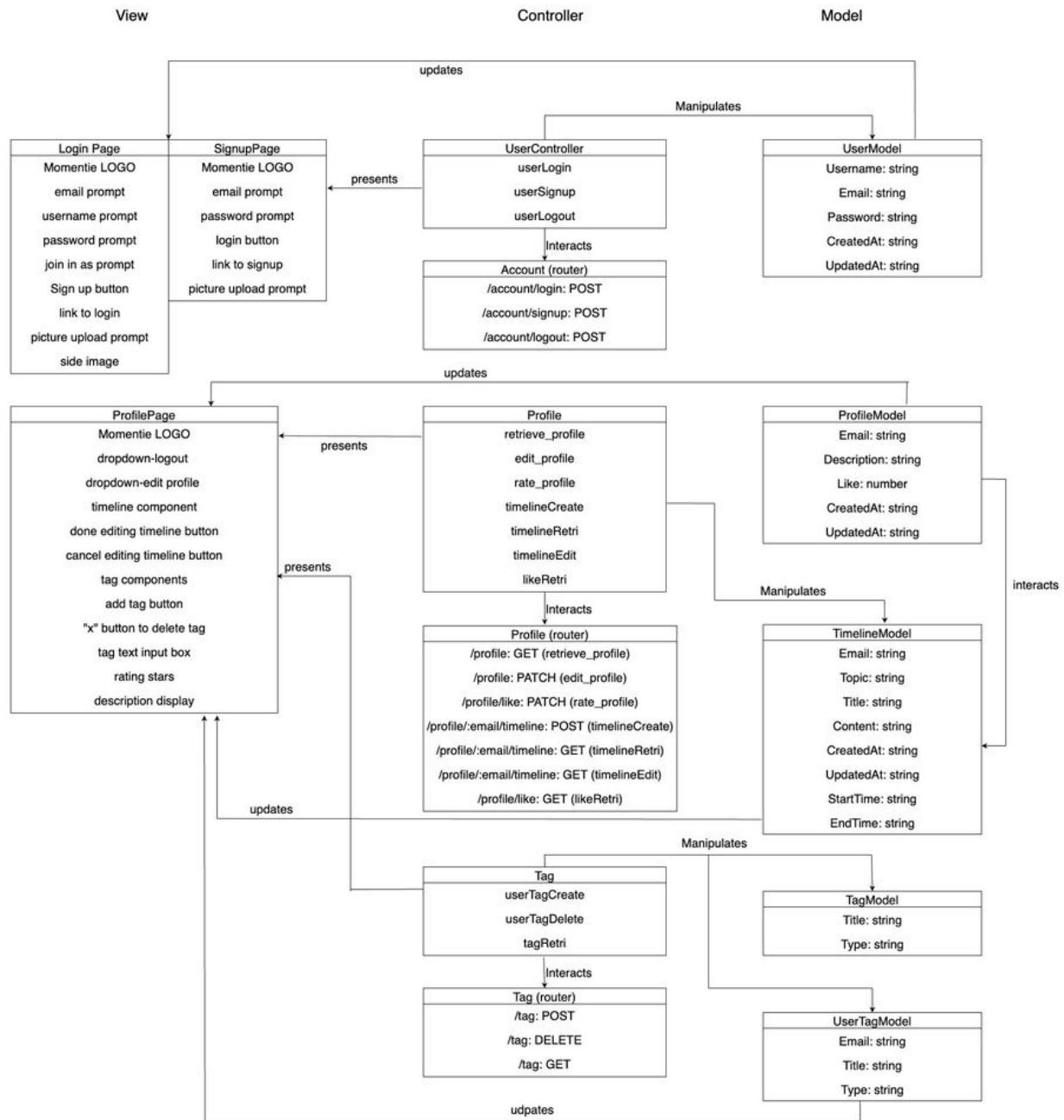
Meeting the MVC Spec. 🔗

> ℹ️ React is not really MVC, and we will take that into account
>
> 📄 Why isn't React considered MVC?

Introduction 🔗

For this project, this team is intending to use React as the front-end framework. React is only a front-end rendering library so that will be our view. Therefore, we classify our application as MVC as the View as React, Model as Mongodb, and Controller as the backend REST APIs.

Why exactly is this application MVC? 🔗

We consider the program as a whole to be MVC, rather than both Frontend and Backend both follow MVC.

We treat React as a pure user interface View. React will communicate with the Controller using HTTP requests. React best practices do not involve letting the controller tell it how to change the View. React Hooks are controllers and Components are Views. This means the Controller is separated into Frontend Controller and Backend Controller.

The Backend Controller then is our true controller. We use Express and its Route functionality to create RESTAPI. This allows the act of getting information from Model directly in View to be transferred to this Controller, thus fulfilling the backend part of the MVC.

The Model is done via Mongoose/MongoDB. We create schemas using Mongoose that provides query options for MongoDB collections. The Model is only used by the Controller and cannot change View directly, thus we follow MVC.

System Environment Expectation 🔗

We expect the operating system should be able to install the required Node environment mentioned in README. We communicate currently through the local host network. The system should also be able to interact with the internet and access MongoDB Atlas.