Software Design

Software Design Documentation

Table of Contents

1. CRC Cards
   a. Backend:
      - REST API Routes
      - Models
   b. Frontend:
      - Pages
2. Software Architecture Design Image
3. Meeting the Three Tier Architecture Specification
4. Environment dependencies.

CRC Cards

Backend

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

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

Class Name: Profile	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities • stores user's profile • uses' email, description, etc. • supplies CRUD operations on user's profile	**Collaborators:** 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

Class Name: Post	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• stores post information • supplies CRUD operations on posts	None

Class Name: Image	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• stores uploaded images • supplies CRUD operations on images	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 • Manages HTTPS request for profile CRUD • Gives error if user inputs are invalid or user does not have permission to edit the profile • Manages HTTPS request for timeline CRUD	**Collaborators:** Profile Timeline Like Image

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

Class Name: PostService	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities • Manages HTTPS request for post CRUD. (Deleting post also means deleting comments associated with it)	**Collaborators:** Post

Frontend

Class Name: LoginPage	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities • 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	**Collaborators:** • SignupPage • ProfilePage • AccountService

Class Name: SignupPage	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities • 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	**Collaborators:** • LoginPage • AccountService

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

Responsibilities	Collaborators:
• Displays and allow edits to user profile, user tags, timeline, and posts	• ProfileService • PostService • TagService • Rating • MomentieTag • MomentieTimeline • MomentiePost

Class Name: MomentieTag	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• Allows user to add and delete their tags • Displays list of 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 and allow user to edit the rating	None

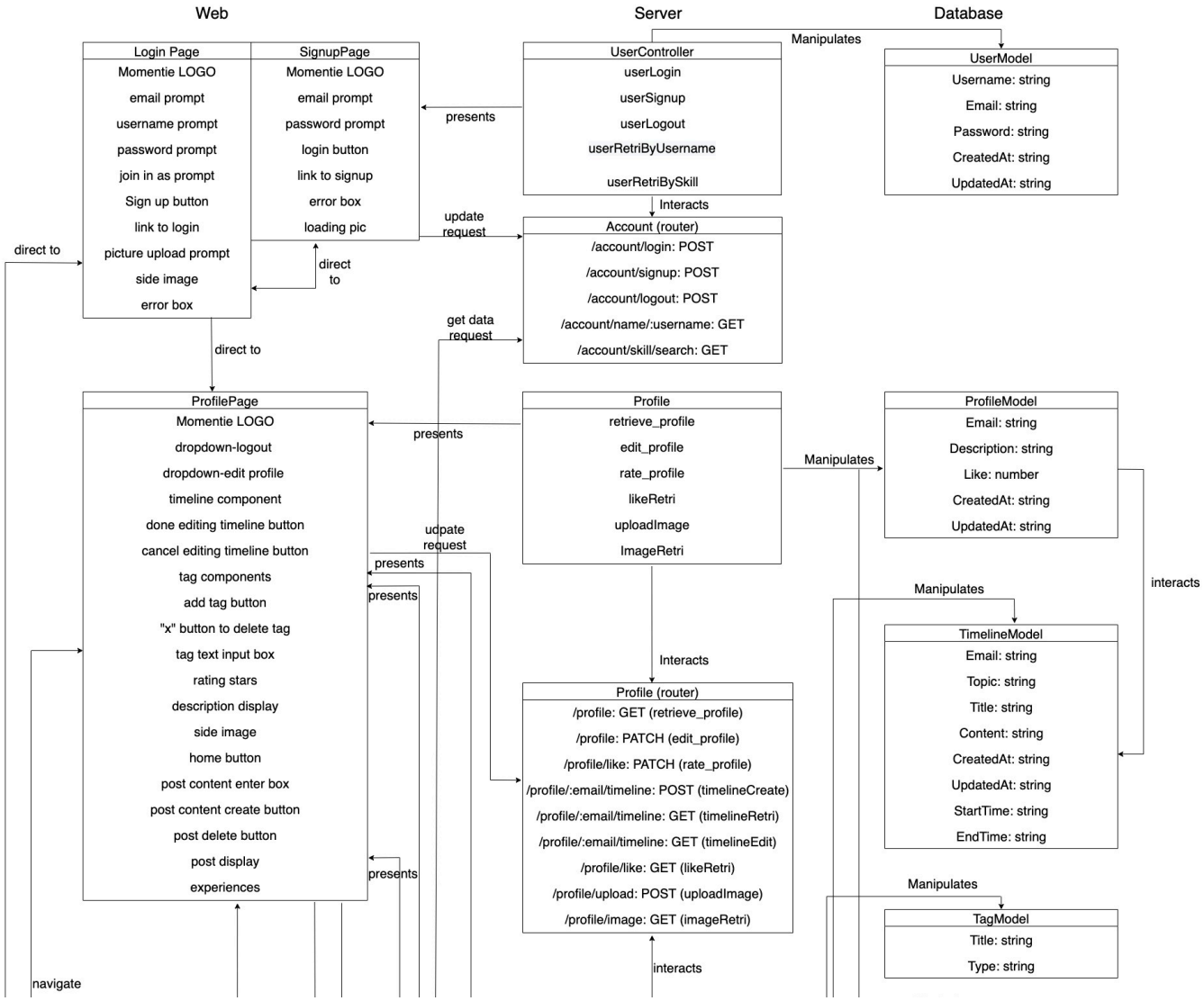
Class Name: MomentiePost	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• Displays a list of posts	None

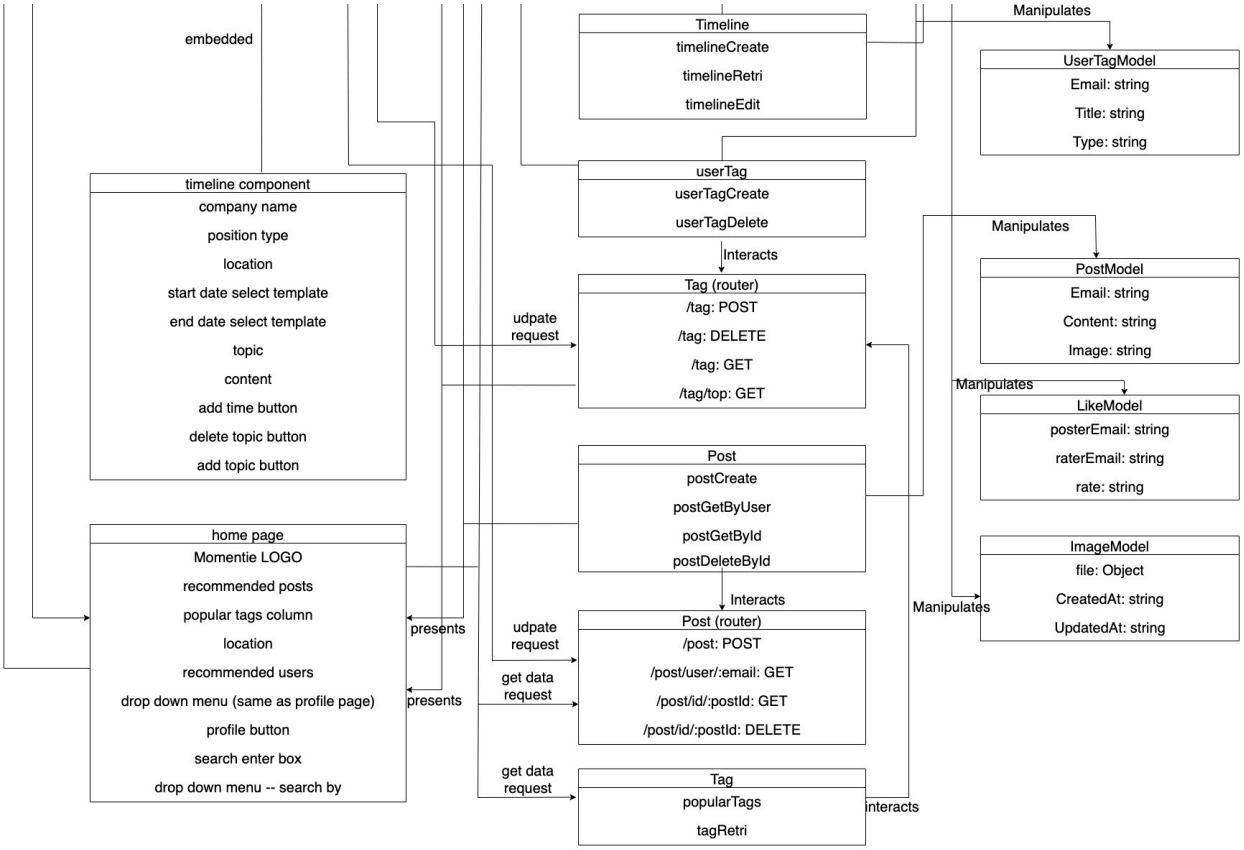
Class Name: HomePage
Parent Class: N/A **Subclasses:** N/A

Responsibilities	**Collaborators:**
• Allows user to search users with different filters such as by skill, email, username, etc. • Displays popular tags and posts by popular users	ProfileService TagService PostService MomentiePost MomentieTag MomentieUserList

Class Name: MomentieUserList	
Parent Class: N/A **Subclasses:** N/A	
Responsibilities	**Collaborators:**
• Displays a list of simplified user profiles • Allows navigation to the actual profiles	Rating

Software Architecture Diagram





Meeting the Three Tier Architecture (TTA) Spec.

> React is not MVC
> https://stackoverflow.com/questions/53729411/why-isnt-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 a Three-Tier Architecture (TTA) with the Client Tier being React, the Business Logic Tier being backend REST APIs, and the Data Tier being MongoDB.

Why exactly is this application Three-Tier Architecture?

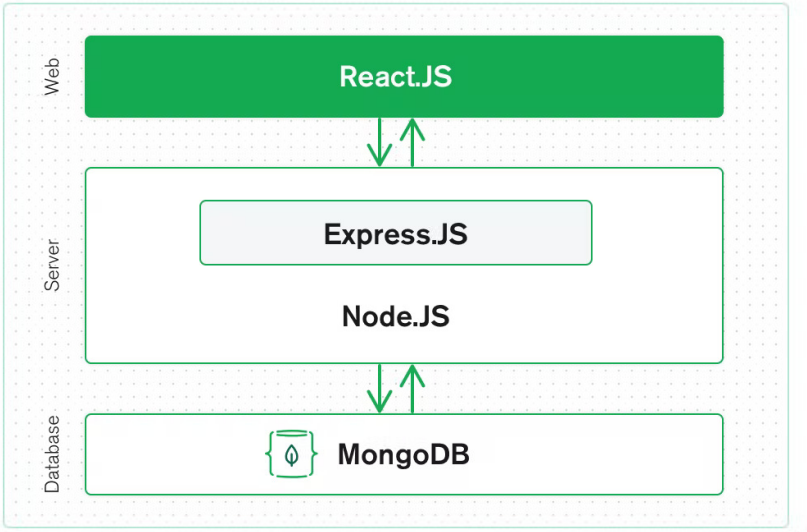
Following the well-known MERN stack, we show that our application follows the same methodologies that of a TTA.

React is our main client view. Using React Hooks, the user interface reacts to user changes and updates itself by communicating with the Business Logic. The Business Logic will not have a role in updating React nor will React directly change Data Tier. Thus, our user interface using React follows the standards of being a Client Tier.

The backend REST APIs setup using NodeJS and Express is our business logic. It is responsible to respond with data to requests from the Client Tier by communicating with the Data Tier. Upon reaching a conclusion, it simply sends the result/data back to the Client Tier without any attempt of updating the Client Tier using HTTP Protocols, etc. Any request from the Client Tier wanting to change the data in Data Tier also goes through our business logic server. Then, we have our REST APIs as a viable Business Logic Tier.

The database is hosted in a MongoDB Atlas Cloud server. The MongoDB API allows the Business Logic Tier to interact with the data storage, performing CRUD operations and complex queries. The database only responds to requests from Business Logic, no proactive requests are made. This makes our database a viable Data Tier.

All in all, our design followed a well-defined TTA which is the MERN stack.



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.