HIGH LEVEL ARCHITECTURE

UNI-BUBBLE

STUDENT ID	NAME
22100339	TAYYAB ALI

TABLE OF CONTENTS

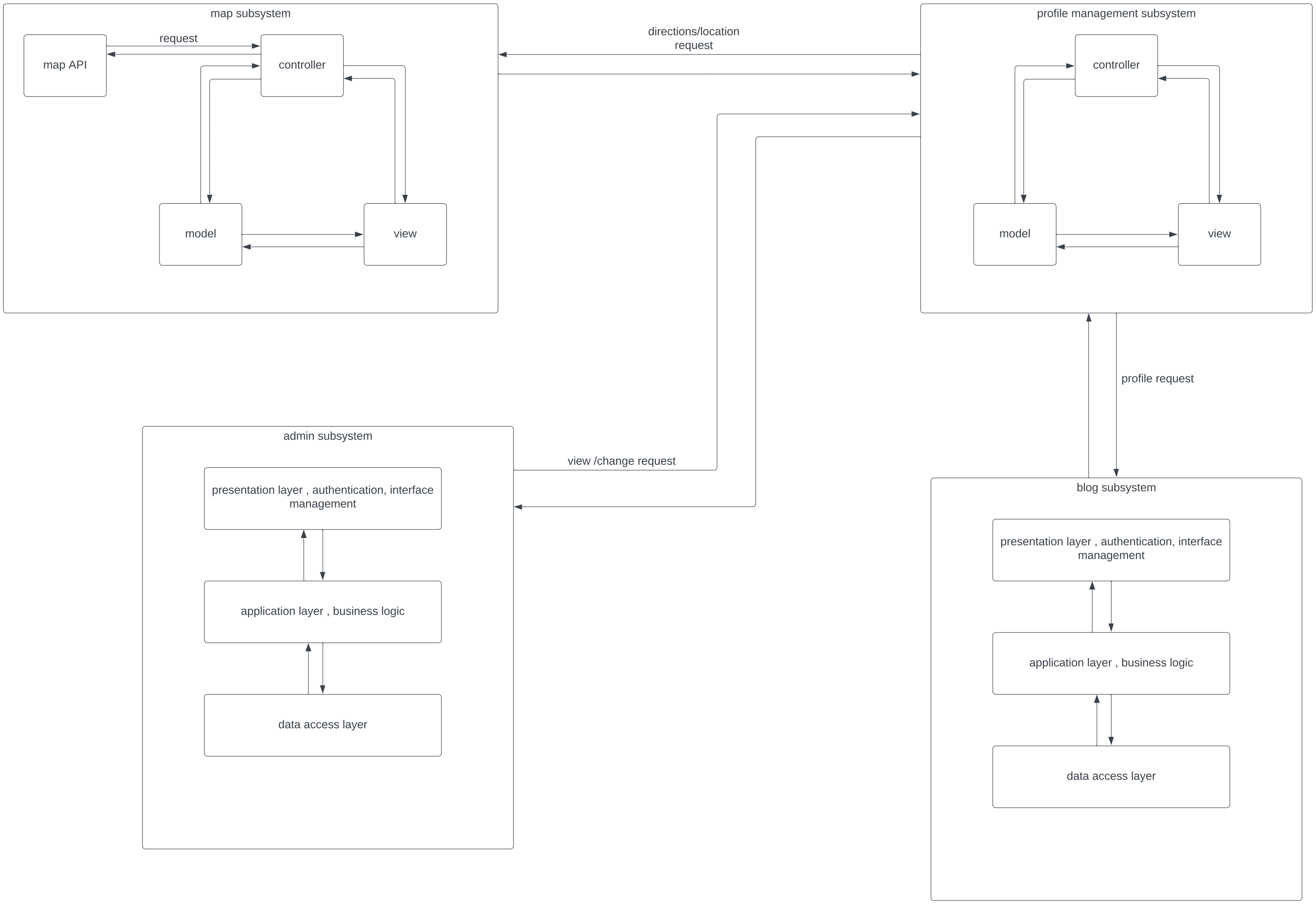
1. Introduction

The project is aimed at universities and colleges. Through this web application users will be able to search for locations inside of their college such as classrooms, offices and eateries among others. Users will be able to access their location within the college and get directions to their desired location.
Furthermore the application will provide an interface for students to connect with their instructors by viewing instructor profiles, and to check if a certain instructor is currently available. Instructors will in turn be able to share their current status on the app along with other details.
The application will allow users to rate eateries/ restaurants within the college and share their reviews on a university blog. This blog will be visible to all the student who have been registered with the university in the application.
It will allow visitors to the university campus to find their way around and search for locations as well.

The potential users of the application are enrolled university students, instructors, vendors, campus visitors such as families, friends, delegates from other universities etc.

2. System Architecture

2.1. Architecture Diagram


map subsystem
map API
request
controller
model
view
directions/location request
profile management subsystem
controller
model
view
profile request
admin subsystem
view /change request
presentation layer , authentication, interface management
application layer , business logic
data access layer
blog subsystem
presentation layer , authentication, interface management
application layer , business logic
data access layer

2.2. Architecture Description

There are four subsystems in the architecture as shown above
1. Map subsystem
2. Profile management subsystem
3. Admin management subsystem
4. Blog subsystem

1.Map subsystem :

this subsystem handles all use cases which deal with fetching locations and getting directions on a map . The subsystem will use a model view controller architecture architecture, the model will hold all database related information and operations and how they are performed. The controller will receive requests and pass them on to the model or the view as required by the specific request. The view will handle how the information is portrayed to the user. The subsystem will also use an external api to access map related information.

2.Profile management subsystem :

this subsystem handles all use cases that are related to user profiles. The different profiles and pages of students, instructors , restaurants, visitors are all handled by this subsystem. Operations such as creating, editing profiles or adding information such as reviews or timings are all handles by this subsystem.

the architecture to be used is an mvc architecture, the model component will handle how profile information is stored and accessed from the relevant database. The view component will handle how the profiles and profile data are displayed to the users. The controller will take in requests and handle them by passing them onto the the relevant component (i.e view and model) and return an approprite response

3.Admin management subsystem :

this subsystem handles all the use cases that are performed by admins and validators of the system. Operations such as deleting and validating users, locations, etc are all handled by this subsystem

the architecture used for this subsystem is a layered architecture.

The first layer is the presentation layer. This is where requests are received from the user interface. This layer handles authentication of the user. It processes whether the user has access to certain operations and displays the results.

The second layer is the application layer this deals with all the business logic related to the request and the operations being performed

the third layer is the data access layer, this handles how data is accessed from the database and performs operations on the database as requested from the above layer

4.Blog subsystem :

the blog subsystem handles the blog feature pf the application and all use cases related to the blog feature, I.e. posting and viewing blogposts. The architecture to be used is a layered architecture.

the first layer is the presentation layer which displays the blogposts to the user. The second layer is the application layer which handles the business logic and decides which blogposts from the database are to be displayed to user and in what order , the third layer is the data access layer which handles the storage and fetching of blogposts from the database

Interactions between the subsystems:

- The map subsystem receives requests from the user interface and from the profile management subsystem to get locations which can be accessed from user profiles or to set locations which may later be accessed from user profiles
- The profile management subsystem receives requests from the admin management subsystem and the blog system to display a profile.
- The profile management subsystem also receives requests from the admin subsystem about changes in the profiles such as validation or deletion of profiles or profile data

2.3. Justification of the Architecture

MVC architecture (map and profile management subsystems)

Pros :
MVC enforces a clear separation of concerns between the View (user interface), Controller (user interactions and logic), and Model (data and business logic).

1. individual components can be reused across different parts of the subsystem or even in other subsystems.
2. MVC makes it easier to test the map and profile subsystem components in isolation. By separating user interface logic (View) from the business logic (Model) and user interactions (Controller), unit testing becomes more straightforward and effective.
3. As the subsystems grows, MVC provides a scalable foundation.
4. MVC enhances code maintainability. As changes or updates are required, developers can focus on the specific component affected without affecting the entire subsystem.

Cons:

1. Complexity: Implementing MVC can introduce some initial complexity, especially if the subsystem is relatively small and doesn't have complex user interactions or data manipulations
2. Overhead: MVC can lead to increased code overhead due to the need to manage the communication and coordination between the components.

MVC is appropriate for the subsystems due to its potential complexity, the need for scalability, and the separation of concerns it offers. The subsystems involves various data handling, user interactions, and data presentation. By structuring the subsystem with MVC, it becomes easier to manage these complexities and allows the developer to work concurrently on different parts of the subsystem.

Non functional requirements such as security and maintainability are supported by the mvc architecture

Layered architecture (admin and blog subsystem)

Pros**:**

1. Layers allow for modularity and separation of concerns, so developers can work on and test the layers separately without affecting the other layers

2. The system will be easier to maintain as errors can be identified in one particular layer and fixed
3. Layers can be updated or changed independently of the other layers if there needs to be a change made in business logic or presentation

Cons:

1. Communication between layers can be costly and might lead to overheads if there is significant data passing between layers
2. Making layers might increase the complexity

The layered architecture allows for security and authentication in the layers and this is required for an admin system. It also allows for features and changes to be added to the system with ease in the layered system without affecting the other layers

3. Risk Management

3.1. Potential Risks and Mitigation Strategies

<List down top 10 potential risks and their mitigation strategies>

Sr.	Risk Description	Mitigation Strategy
	Change in requirements	Assess impact of requirements changes and adapt well
	Difficulty in integrating subsystems and API leading to delays	Plan integration early in development stages and conduct sufficient testing to identify and solve problems
	Failure to meet non functional requirements	Prioritise non functional requirements along with functional requirements, allocate sufficinet time so these requirements are met
	Size of project is underestimated	Thorough investigation of requirements and planning, after consulting with experts.
	Time to complete project is underestimated	Investigate the use of a program generator
	3rd parties dependencies risk	Have a contingency plan in place in case the Third party disruptions or changes
	Uncontrolled expansion of project scope leading to delays	Define scope clearly ahead of time,
	Data privacy, security and unauthorised access risk	Implement security measures such as encryption and role based access to data
	Developers are not trained in the technologies required leading in low quality product and delays	Provide training to deal with skill gaps, consult with seniors or mentors if required
	System cannot handle heavy loads of traffic and is not scalable	Conduct sufficient load testing and identify bottlenecks ahead of time so they may be optimised

4. Tools and Technologies

- React.js (v18.2.0)
- Mongodb (v6.0.6)
- Node.js (v18.2.1)
- express.js (v4.18.2)
- HTML5
- CSS 3
- GitHub
- Vscode (v1.68.1)

5. Hardware Requirements

Development machine :

- CPU : Multicore processor (intel core I5)
- RAM : 8GB
- Storage : SSD
- OS : macOS
- Connectable to internet

Deployment server :

- CPU : high performance Multicore processor
- RAM : 8GB
- Storage : high storage capabilities
- High bandwidth and fast internet connection required
- Available 24/7

6. Who Did What?

Name of the Team Member	Tasks done
Tayyab Ali	All tasks

7. Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Section Title	Reviewer Name(s)
All sections	Tayyab Ali