High-level Architecture

P-10:Odysseum

Student ID	Name
25100283	Muhammad Affan naved
25100225	Mohammad Haroon Khawaja
25100212	Shahrez Aezad
25100097	Pir M. Shahraiz Chishty
25100023	Luqman Aadil

Table of Contents

1. Introduction

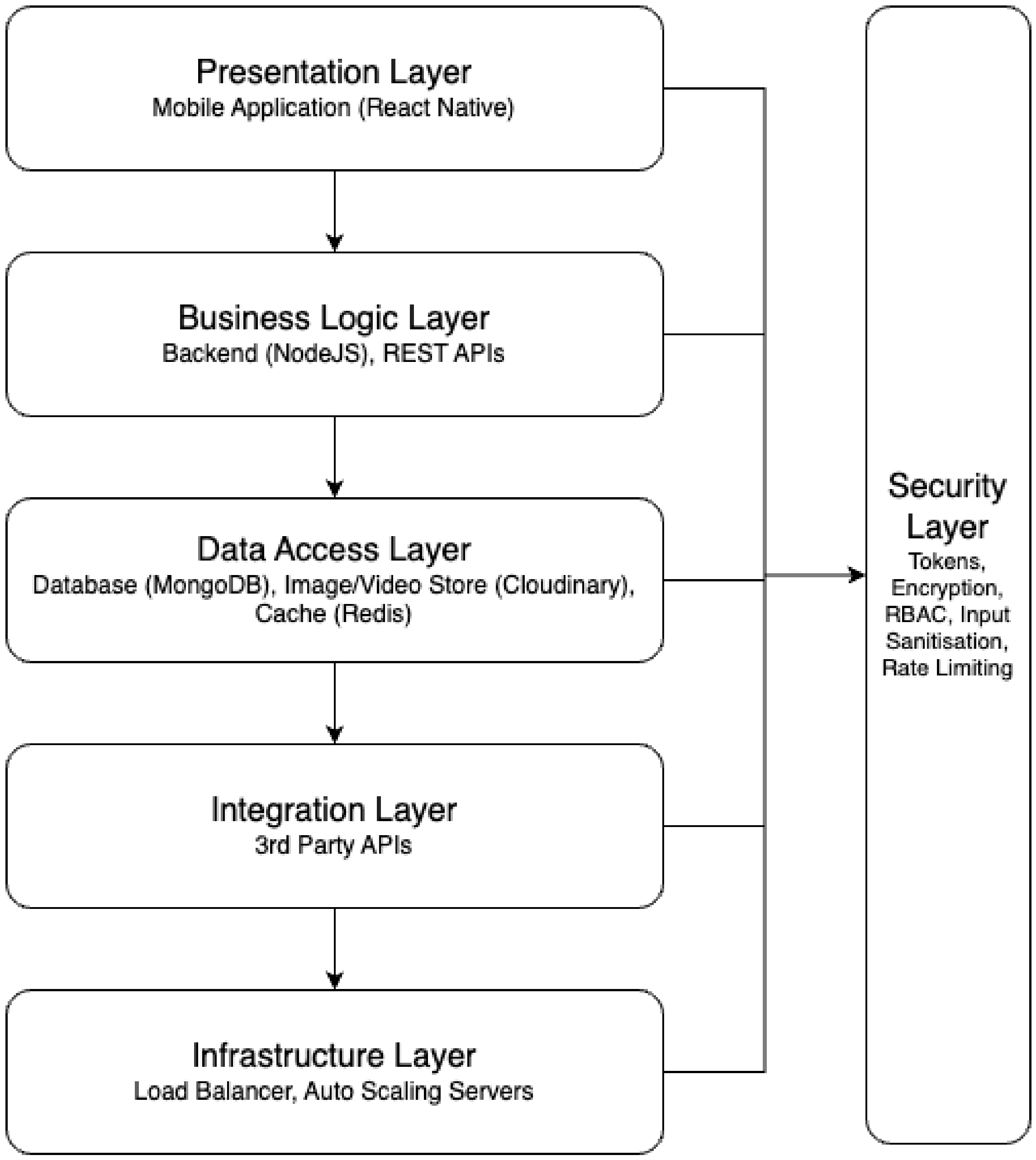
This project aims to develop a travel/social network application to help travelers plan their next trip using just one app. Rather than relying on blogs and pages from different online outlets, the app would be a one-stop solution for all travelers. The application will provide users a platform to search for various tourist destinations they may be interested in visiting and what these destinations offer, such as accommodation, sightseeing, dining, nightlife, historical sites, and tour guides. Combining all these services onto one platform would improve the travel experience and allow users to make well-informed decisions based on destination information.

Nowadays, travelers face a fundamental problem: finding accurate and relevant information. They have to rely on large commercial travel agencies that only have profit-driven goals or on personal connections that provide limited details and advice. This gives travelers an experience far from fulfilling, while local businesses gain limited benefits. The purpose of this app is to serve as a networking app to connect like-minded travelers and local service providers such that both parties benefit, with travelers having a fulfilling experience visiting their destinations and the local business being allowed to boost the economic growth in the region.

As stated above, potential users of this app include travelers themselves, administrators and local businesses which are but not limited to hotels, restaurants, and tour guides.

2. System Architecture

2.1 Architecture Diagram


Presentation Layer
Mobile Application (React Native)
Business Logic Layer
Backend (NodeJS), REST APIs
Data Access Layer
Database (MongoDB), Image/Video Store (Cloudinary), Cache (Redis)
Integration Layer
3rd Party APIs
Infrastructure Layer
Load Balancer, Auto Scaling Servers
Security Layer
Tokens, Encryption, RBAC, Input Sanitisation, Rate Limiting

2.2 Architecture Description

2.2.1 Presentation Layer

The presentation layer is responsible for the user interface and user experience of the social media app. It includes the components that users interact with directly, such as forms, buttons, navigation, and visual content.

- ➔ Components
  - ◆ Mobile App: Developed using React Native, this component allows users to access social media features on mobile devices, such as posting updates, liking posts, and interacting with friends.

- ➔ Interactions
  - ◆ Users interact with the presentation layer through the app or website, which sends requests to the business logic layer to perform server-side actions (e.g., fetching posts, searching locations, messaging users/groups).

2.2.2 Business Logic Layer

The business logic layer contains the core application logic that processes user requests, manages data flow, etc. This layer is supposed to handle all server-side operations and ensure that the application behaves as intended.

- ➔ Components
  - ◆ Backend: Contains the main server which is implemented using NodeJS and express routing to provide REST APIs for requests from the front end.
  - ◆ Authentication: This component is responsible for making sure all data transactions that happen between presentation and business logic layers are secure and authorized. This is implemented via JWT tokens and its relevant authentication middleware.
- ➔ Interactions
  - ◆ The presentation layer sends HTTPS requests to the backend services to perform various operations (e.g., user login, fetching posts, sending a message).
  - ◆ Upon receiving a request, the backend services process the request and may interact with the data access layer to retrieve or modify data.
  - ◆ Authentication services verify user credentials and generate JWT tokens, which are sent back to the presentation layer for subsequent requests.

2.2.3 Data Access Layer

The data access layer manages data storage and retrieval. It abstracts the underlying database systems, providing a clean interface for the business logic layer to interact with data.

- ➔ Components
  - ◆ Database: This will be a NoSQL database (MongoDB) that will store all user data, from login credentials to posts and recommendations.
  - ◆ Image/Video Storage: Since this is a social media app, we will use Cloudinary to store all images or videos users upload to make a post.
  - ◆ Cache System: Optionally, we may add another component for caching frequently requested data that does not change (e.g. fetch user profile, comments, message) in order to reduce unnecessary calls to the database. If we choose to implement it, we will use Redis.
- ➔ Interactions
  - ◆ The business logic layer communicates with the data access layer to store, retrieve, and update data. For instance, when a user creates a post, the backend service will send a request to the database to save the new post and will store any image or video on Cloudinary.
  - ◆ The caching component can be queried before hitting the database to reduce response times, improving performance for popular data requests.

2.2.4 Integration Layer

The integration layer facilitates interactions with external services and APIs. This layer manages data exchange between the social media app and third-party systems, such as authentication providers like Google.

- ➔ Components
  - ◆ The only components here would be the third-party providers whose APIs and other services may be used.
- ➔ Interactions
  - ◆ The business logic layer interacts with the integration layer when it requires external services. For example, during user authentication, the backend service may call the OAuth provider to verify user credentials.
  - ◆ Responses from external services are then passed back to the business logic layer, which processes the information and returns results to the presentation layer.

2.2.5 Security Layer

The security layer encompasses mechanisms that protect data and ensure secure access to application resources. This layer ensures that only authorized users can access certain functionalities and that data is protected during transmission.

- ➔ Components
    - ◆ JWT Tokens: Used for stateless authentication, allowing users to access protected routes after successfully logging in.
    - ◆ Encryption: Ensures sensitive data (e.g. passwords) is stored securely and is only accessible to authorized users.
    - ◆ Role-Based Access Control: Ensure that only certain users are allowed to access sensitive functionalities of the backend system. (e.g. admins)
    - ◆ Input Validation and Sanitization: Ensure all user inputs are validated to prevent injection attacks and strip out harmful code or scripts from inputs to avoid Cross-Site Scripting (XSS) attacks.
    - ◆ Rate Limiting: Prevent brute force attacks by limiting the number of API requests from a single IP or user
- ➔ Interaction
    - ◆ The security layer interacts with both the business logic and data access layers. For instance, when a request is made to the backend, the security layer checks the validity of the JWT token before allowing access to protected resources.
    - ◆ The security measures are applied whenever data is transmitted between layers, ensuring encryption for data in transit (e.g., HTTPS).

2.2.6 Infrastructure Layer

The infrastructure layer represents the environment where all backend components are deployed. It encompasses the physical and virtual resources required to run the application, including servers, load balancers, and databases.

- ➔ Components
    - ◆ Load Balancer: Distributes incoming traffic across multiple backend servers to ensure optimal resource utilization and availability.
    - ◆ Backend Servers: These servers host the backend services, which can be scaled horizontally based on traffic demand (auto-scaling).

- ➔ Interactions
    - ◆ The load balancer receives requests from users and distributes them to available backend servers in the infrastructure layer.

- As traffic increases, the auto-scaling feature automatically adds or removes backend servers based on the load, ensuring that the application remains responsive and available.

2.3 Justification of the Architecture

Pros:

1. **Separate Layers:**
   Each layer has a distinct, individual role, such as the presentation layer that handles user interaction and the business logic layer that handles managing core functionality such as user requests.
2. **Server Scalability:**
   The architecture will support horizontal scaling due to the infrastructure layer where all the backend components are deployed. This will ensure that the system can efficiently handle large amounts of traffic.
3. **Security:**
   The security layer, along with its components that include JWT tokens, encryption, role-based access, rate limiting, etc, ensures that the sensitive data within the databases are protected and access is controlled. Potential security breaches will also be minimized due to the security layer.
4. **Extensibility and Integration:**
   The integration layer enables easy interaction with third-party APIs and external services. This makes it easier to add new features, such as social media logins without affecting the rest of the system.
5. **Fault Tolerance and Availability:**
   The use of load balancers in our infrastructure ensures that the backend servers distribute services across multiple layers. If one server goes down, the others can contribute by handling its requests and improving server availability.

Cons:

1. **Complexity:**
   Having multiple layers makes our system more complex in terms of developing and managing. The interactions between layers can add a layer of overhead, especially when scaling the system.
2. **Server Latency:**
   Third-party app extensibility along with multiple layers of abstraction can introduce latency in data processing and handling, leading to slower response times.
3. **Security Overhead:**
   Implementing security components, such as JWT tokens, can add computational overhead, which might affect server performance, especially when dealing with high amounts of traffic.

Justification for architecture

This architecture is well suited for our app because:

1. **It is Modular**
   Each layer operates independently, allowing easy modifications, updates, and scaling of components.
2. **It is Scalable**
   The app can handle varying traffic through horizontal scaling of resources. This will dynamically adjust due to the server demand and amount of user traffic, making the server adaptable for a growing user base.
3. **it is Secure**
   The security layer ensures data protection and user privacy, complying with standard security requirements.

Implementation of Non-Functional Requirements by the Architecture

1. **Scalability**
   Our infrastructure layer supports horizontal scaling to accommodate an increase or decrease in user traffic. The load balancers ensure the even distribution of user requests.
2. **Security**
   The non-functional security requirements are addressed by the security layer components. Encryption, JWT tokens, and other security measures ensure that our non-functional security requirements are met.
3. **Reliability**
   The architecture provides fault tolerance with load balancing and auto-scaling features, ensuring the system remains available even during high traffic or server failures.
4. **Maintainability**
   The distinct layers separate the responsibilities of the system, making the system more maintainable. This would allow our team to work on different layers without affecting the other components.

3. Tools and Technologies

Frontend:

- React Native (Version: [0.75])

Backend:

- Node.js (Version: [v22.9.0])

Database:

- MongoDB (Version: [8.0])
- Cloudinary (Version: [2.5.1])

Deployment/Cloud Hosting:

- AWS or Azure (AWS EC2, S3, or Azure App Service)

APIs:

- Integration with various third-party APIs (e.g., Google Maps API, weather services, accommodation APIs)

Version Control:

- GitHub for code repository and collaboration

Code Editor/IDE:

- Visual Studio Code (Version: [1.94.2])
- Postman (API testing) (Version: [v11.16])

Package Management:

- NPM(node package manager) (Version: [10.9.0])

4. Hardware Requirements

Development Machines:

1. Processor
   Minimum: Intel Core i5 (9th gen or onwards)/ AMD Ryzen 5
   Recommended: Intel Core i7/ AMD Ryzen 7/ Silicon Macs
2. RAM
   Minimum: 8gb
   Recommended: 16gb
3. Storage
   Anything above 256gb would be enough
4. GPU
   No dedicated graphics required. Integrated gpu will suffice.
5. Operating System
   Windows 10/Linux Ubuntu 20.04/MacOS Catalina (Or Later)
6. General
   Backup Solutions(External Storage), Internet, Peripherals.

Deployment Servers:

We will be using a cloud based deployment server. Preferably Amazon Web Services or Microsoft Azure.

1. Processor
   Quad-core processor (Intel Xeon/ AMD EPYC or higher)
2. RAM
   8GB or Higher
3. Storage
   256GB or Higher
4. Network Bandwidth
   100mbps connection
5. Backup And Storage
   Cloud Storage (S3 or Blob)
6. Load Balancer
   A load balancer is required for better optimization and request distribution.
7. Operating System
   Should be linux based. Since they are scalable and optimized for web apps.
8. Database Server:
   Should be able to manage different types of databases including relational databases, NoSQL databases, and specialized data storage systems.

5. Who Did What?

Name of the Team Member	Tasks done
Muhammad Affan Naved	Section 2.2
Pir M. Shahraiz Chishty	Section 4
Luqman Aadil	Section 3
Shahrez Aezad	Section 2.1
Mohammad Haroon Khawaja	Section 2.3

6. 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)
Section 2.2	Pir M. Shahraiz Chishty, Mohammad Haroon Khawaja
Section 3,4	Muhammad Affan Naved
Section 2.1,3,2.3	Shahrez Aezad, Luqman Aadil