Roommate Finder
Project Documentation

1st Wesal Soliman
Software Engineering
Nile University
Cairo, Egypt
2.magdy2386@nu.edu.eg

2nd Rana Mohsen
Software Engineering
Nile University
Cairo, Egypt
r.mohsen2397@nu.edu.eg

3rd Rama Mousa
Software Engineering
Nile University
Cairo, Egypt
r.mohamed2373@nu.edu.eg

4th Raneem Khaled
Software Engineering
Nile University
Cairo, Egypt
r.khaled2324@nu.edu.eg

***Abstract*—This report presents the design, implementation, and documentation of the Roommate Finder website. The project includes both backend and frontend components, providing secure user authentication, flexible roommate search functionality, and a user-friendly interface. Key features include JWT-based login, password hashing, optional filters for searching roommates, and interactive UI components for posting and managing roommate listings.**

***Index Terms*—Roommate Finder, Backend, Frontend, Node.js, Express, JWT, MongoDB, Optional Filters, UML Diagrams**

I. Introduction

Roommate Finder is a web application designed to help users find suitable roommates based on their preferences such as location, budget, and gender. Current solutions are inadequate: people often post roommate requests on social media platforms, where posts get easily lost, there is no proper search functionality, and the posts are often not taken seriously by the audience—they can turn into jokes or memes, making it difficult to find genuine matches. Roommate Finder addresses these issues by providing a structured platform with secure authentication, searchable posts, and filtering options to ensure meaningful matches. The backend manages secure data handling, while the frontend provides a responsive and interactive interface for user interaction. This documentation covers the full project architecture, implementation, and UML diagrams.

II. Technologies Used

- **Backend:** Node.js, Express.js
- **Database:** MongoDB Atlas
- **Authentication:** JWT (JSON Web Tokens), bcrypt
- **Frontend:** HTML, CSS, JavaScript, React.js (optional)
- **Development Tools:** Nodemon, VS Code
- **Environment Management:** dotenv

III. System Architecture

A. Backend Structure

- **models/** - Mongoose schemas (User, Roommate)
- **controllers/** - Business logic (signup, login, search, posts)
- **routes/** - Express routes for each module
- **server.js** - Entry point, environment setup

B. Frontend Structure

- **components/** - React components for forms, posts, search
- **pages/** - Landing page, About Us, Login, Signup, Dashboard
- **services/** - API calls to backend endpoints

C. Workflow

1) Client interacts with frontend UI.
2) Frontend calls backend API endpoints.
3) Backend controllers process requests and interact with MongoDB.
4) Backend responds with JSON data, status messages, or errors.
5) Frontend renders data for the user.

IV. User Authentication

A. Signup

- User provides *name*, *email*, *password*.
- Email is checked for duplication.
- Password is hashed using bcrypt.
- User stored in MongoDB.
- JWT token returned for authentication.

B. Login

- User provides *email* and *password*.
- Checks for existing valid JWT token to prevent multiple logins.
- Password verified against hashed password in database.
- New JWT token returned upon successful login.

V. Roommate Search Functionality

A. Required and Optional Filters

Users can search for available roommate posts using a combination of required and optional filters:

Required filters:

- **City** – The city where the roommate is needed.
- **Budget** – Maximum rent the user is willing to pay.
- **Gender** – Preferred gender of the roommate.

Optional filters:

- **WiFi** – Availability of WiFi (true/false)

- **Air Conditioning (AC)** – Whether AC is included (true/false)
- **Pets** – Whether pets are allowed (true/false)

B. Backend Implementation

The backend dynamically builds a query based on the provided filters. Required filters must be present, while optional filters are included if specified by the user:

```
// Extract filters from query
const{city,budget,gender,wifi,ac,pets}
 =req.query;

// Validate required filters
if (!city || !budget || !gender) {
  return res.status(400).json({
  message:’City,budget,gender are required’
  });
}

// Build query object
const filter = {
  city: city,
  price: { $lte: Number(budget) },
  gender: gender
};

// Add optional filters if provided
if (wifi !== undefined)
filter[’amenities.wifi’] = wifi === ’true’;
if (ac !== undefined)
filter[’amenities.ac’] = ac === ’true’;
if (pets !== undefined)
filter[’amenities.pets’] = pets === ’true’;

// Fetch matching posts from database
const posts = await Post.find(filter);
```

C. Response

The backend returns a JSON object containing all posts that match the filter criteria:

```
res.status(200).json({ results: posts });
```

This implementation ensures both flexibility and accuracy, allowing users to search using the most relevant criteria while supporting additional preferences for amenities.

VI. Post Management

- Create, edit, and delete roommate posts.
- Backend ensures only the post owner can edit or delete.
- Frontend dynamically updates the post list.

VII. Error Handling

- Signup: 400 if user exists, 500 for server errors.
- Login: 400 for invalid credentials or already logged in, 500 for server errors.
- Search/Post actions: 500 for server errors.

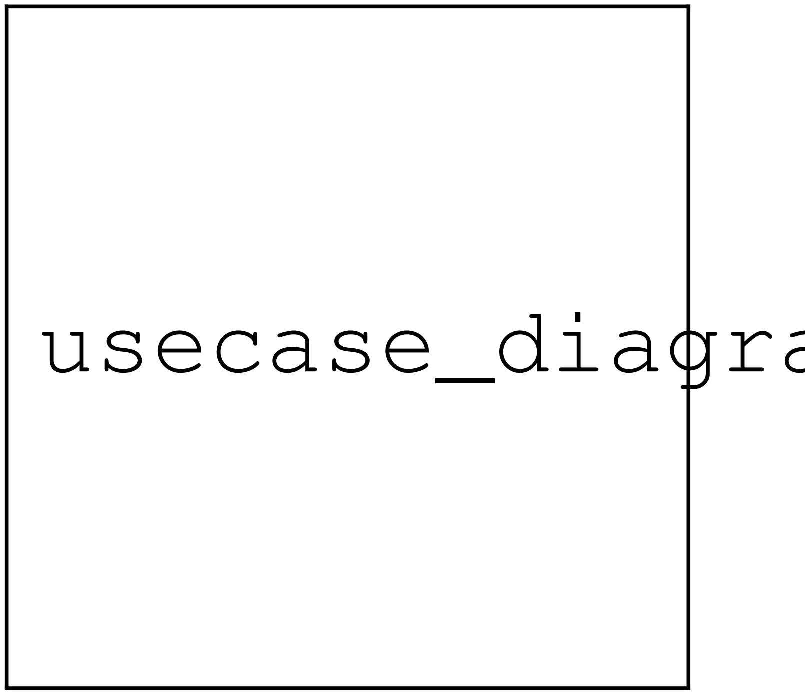

Fig. 1. Use Case Diagram of Roommate Finder

VIII. Security Considerations

- Passwords hashed before storage.
- JWT verification for secure authentication.
- Environment variables used for sensitive data.
- Prevent multiple logins using valid tokens.

IX. API Endpoints Summary

Endpoint	**Method**	**Description**	**Authentication**
/signup	POST	Register new user	No
/login	POST	User login	No
/logout	POST	User logout	Yes
/posts	GET	Retrieve all posts	No
/posts	POST	Create a new post	Yes
/posts/:id	PUT	Edit a post by ID	Yes
/posts/:id	DELETE	Delete a post by ID	Yes
/search	GET	Search posts with filters	No

TABLE I
Summary of Roommate Finder API Endpoints

X. Future Improvements

- Add email verification and password reset.
- Implement frontend notifications.
- Add pagination and sorting for search results.

XI. UML Diagrams

A. Use Case Diagram

The use case diagram illustrates the main interactions between users and the Roommate Finder system. It highlights core functionalities such as user authentication, post management, and roommate searching, providing a high-level view of system behavior from the user’s perspective.

B. Class Diagram

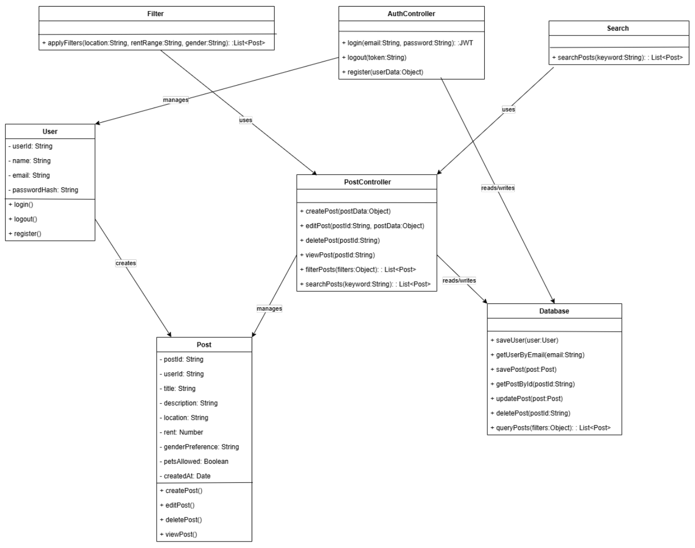
Filter
+ applyFilters(location:String, rentRange:String, gender:String): :List<Post>
AuthController
+ login(email:String, password:String): :JWT
+ logout(token:String)
+ register(userData:Object)
Search
+ searchPosts(keyword:String): : List<Post>
manages
uses
uses
User
- userId: String
- name: String
- email: String
- passwordHash: String
+ login()
+ logout()
+ register()
PostController
+ createPost(postData:Object)
+ editPost(postId:String, postData:Object)
+ deletePost(postId:String)
+ viewPost(postId:String)
+ filterPosts(filters:Object): : List<Post>
+ searchPosts(keyword:String): : List<Post>
reads/writes
creates
manages
reads/writes
Post
- postId: String
- userId: String
- title: String
- description: String
- location: String
- rent: Number
- genderPreference: String
- petsAllowed: Boolean
- createdAt: Date
+ createPost()
+ editPost()
+ deletePost()
+ viewPost()
Database
+ saveUser(user:User)
+ getUserByEmail(email:String)
+ savePost(post:Post)
+ getPostById(postId:String)
+ updatePost(post:Post)
+ deletePost(postId:String)
+ queryPosts(filters:Object): : List<Post>

Fig. 2. Class Diagram of Roommate Finder

The class diagram represents the static structure of the system, showing the main entities, their attributes, and relationships. It reflects how users, posts, and authentication data are modeled and how backend components interact at the data level.

C. Sequence Diagrams

These sequence diagrams represent the step-by-step interactions between the client, backend controllers, and database for the most important operations in the system.

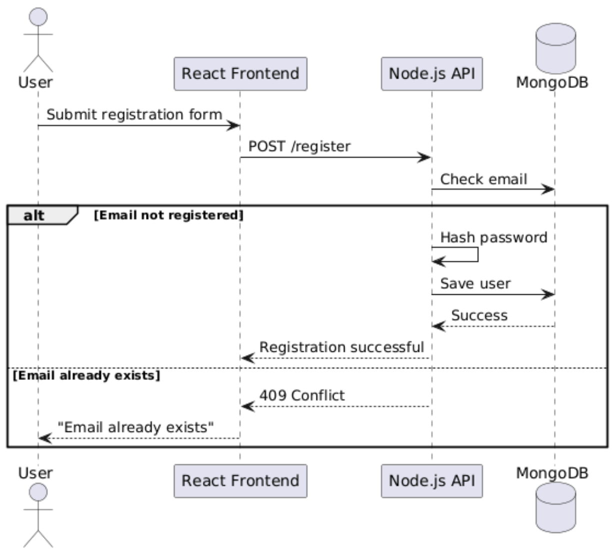

Fig. 3. Sequence Diagram 1: User Registration

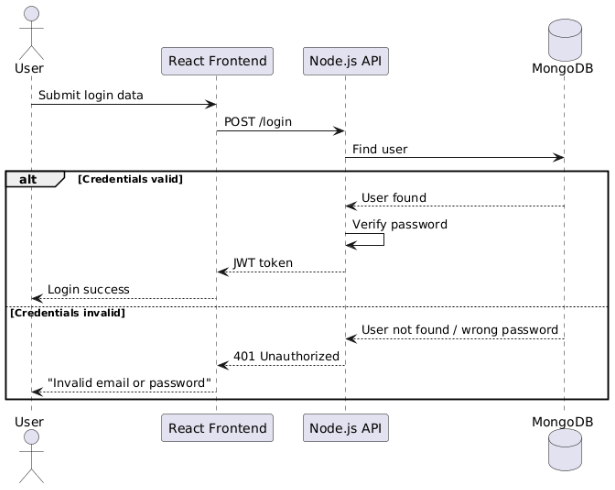

Fig. 4. Sequence Diagram 2: User Login

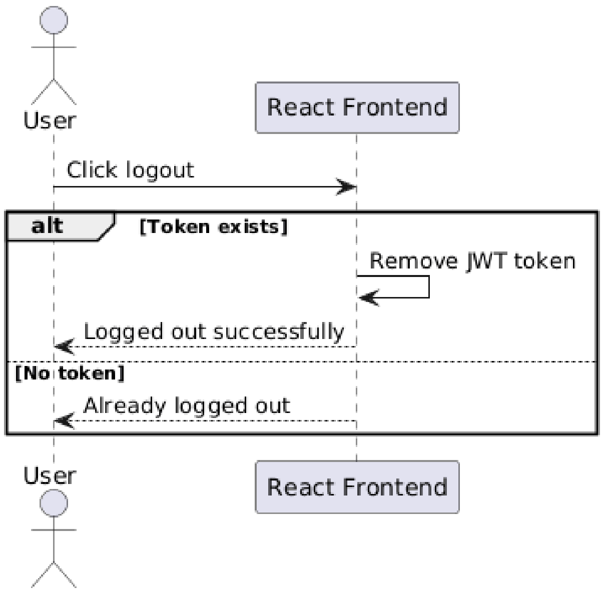

Fig. 5. Sequence Diagram 3: User Logout

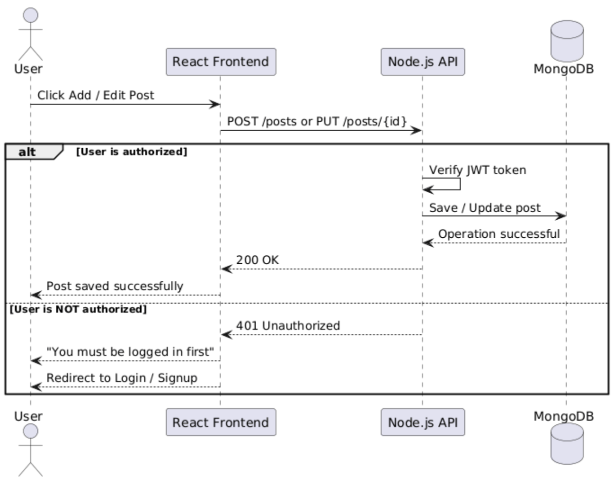

Fig. 6. Sequence Diagram 4: Create or Edit Post

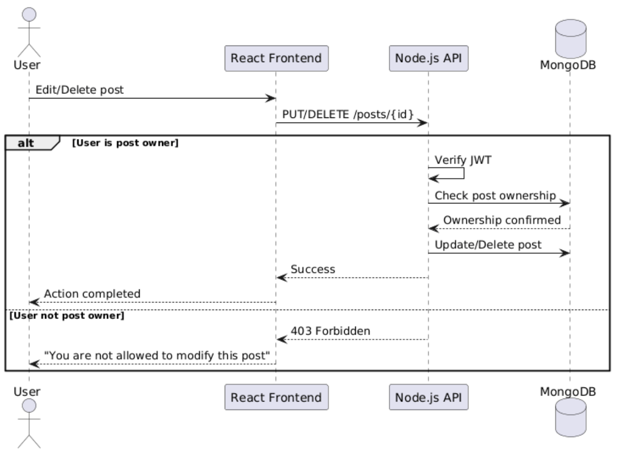

Fig. 7. Sequence Diagram 5: Edit or Delete Post

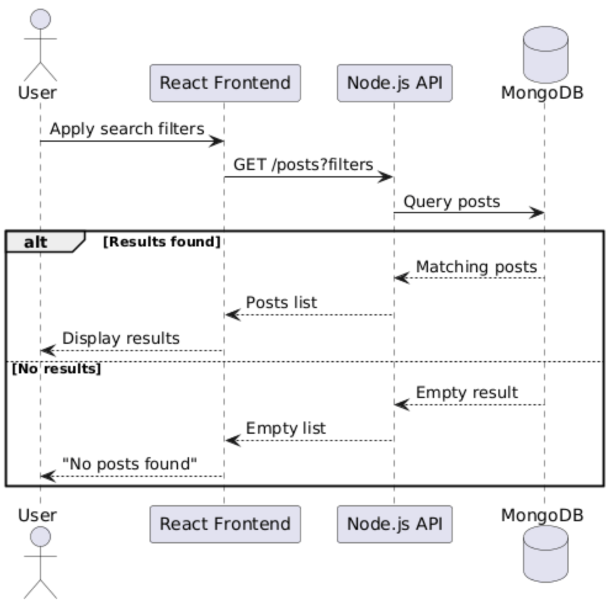

Fig. 8. Sequence Diagram 6: Search Posts

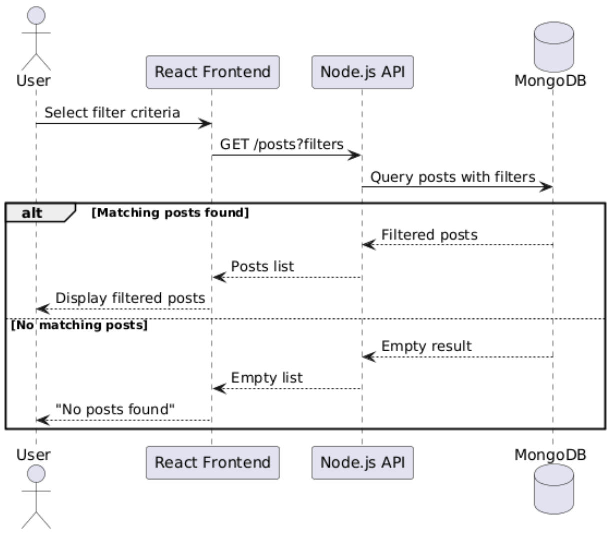

Fig. 9. Sequence Diagram 7: Filter Posts

XII. Conclusion

The Roommate Finder project provides a secure, scalable, and user-friendly platform for finding roommates. By implementing full backend authentication, dynamic search filters, post management, and integrating both frontend and backend components, the system offers a complete solution suitable for deployment and future enhancements. **Limitations:** The platform currently does not support real-time chat between users.

References

[1] Node.js Foundation, ”Node.js Documentation.” [Online]. Available: https://nodejs.org/en/docs/
[2] Express.js Team, ”Express.js Documentation.” [Online]. Available: https://expressjs.com/
[3] MongoDB, Inc., ”MongoDB Manual.” [Online]. Available: https://www.mongodb.com/docs/
[4] Automattic, ”Mongoose Documentation.” [Online]. Available: https://mongoosejs.com/docs/
[5] Auth0, ”JSON Web Tokens Introduction.” [Online]. Available: https://jwt.io/introduction/
[6] Open Source Community, ”bcrypt Node.js.” [Online]. Available: https://www.npmjs.com/package/bcrypt
[7] OMG, ”UML 2.5 Specification.” [Online]. Available: https://www.omg.org/spec/UML/2.5/
[8] S. Ambler, *The Elements of UML 2.0 Style*. Cambridge University Press, 2005.
[9] I. Sommerville, *Software Engineering*, 10th Edition. Pearson, 2015.
[10] C. Larman, *Applying UML and Patterns*, 3rd Edition. Prentice Hall, 2005.
[11] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd Edition. Addison-Wesley, 2018.
[12] J. Doe, A. Smith, ”Roommate Matching Applications: A Survey of Methods and Effectiveness,” *Journal of Information Technology Research*, vol. 13, no. 2, 2020.
[13] R. Lee, K. Chen, ”A Study on Web-based Roommate Finder Systems,” *International Journal of Computer Applications*, vol. 178, no. 7, 2019.