Roommate Finder – Project Documentation

Authors

1. Rana Mohsen – 231001373

2. Wesal Soliman – 231001686

3. Rama Mousa – 231001373

4. Raneem Khaled – 231001424

Course

Software Engineering – Nile University, Cairo, Egypt

Instructor

Dr. Mohamed Hassan Elgazzar

Content

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.

1. Introduction

Roommate Finder is a web application designed to help users find suitable roommates based on preferences such as location, budget, and gender. Current solutions, such as social media posts, are inadequate because posts can be lost or ignored, and there is no structured search functionality. Roommate Finder addresses these issues by providing:

- Secure authentication
- Searchable posts
- Optional filtering for amenities
- Structured and responsive user interface

This report documents the full project architecture, implementation, and UML diagrams.

2. Requirements and Design

2.1 Functional Requirements

- Register and login securely
- Create, edit, delete roommate posts

- Search and filter posts (required: city, budget, gender; optional: WiFi, AC, pets)
- Only authenticated users can manage posts

2.2 Non-Functional Requirements

- Password hashing (bcrypt)
- JWT authentication for security
- Responsive and user-friendly interface
- Scalable backend for multiple searches

2.3 Use Cases & User Stories

2.3.1 Use Case Example – Create Post

- **Actor:** Registered User
- **Goal:** Create a roommate post
- **Precondition:** User is logged in
- **Flow:** Fill form → Submit → Post stored → Confirmation shown
- **Postcondition:** Post appears in search results

2.3.2 User Story Example:

As a user, I want to filter roommate posts by city and budget so that I can find relevant matches quickly.

2.4 System Specifications

- **Frontend:** React.js, HTML, CSS, JavaScript
- **Backend:** Node.js, Express.js
- **Database:** MongoDB Atlas
- **Authentication:** JWT, bcrypt

2.5 Data Models

- **Logical Model:** User, Roommate Post, Authentication Token
- **Physical Model:** MongoDB collections for Users and Posts, with document references for relationships

3. System Architecture

3.1 Backend Structure

- **models/** – Mongoose schemas for Users and Posts
- **controllers/** – Logic for signup, login, search, posts
- **routes/** – Express routes for each module
- **server.js** – Entry point and environment setup

3.2 Frontend Structure

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

3.3 Workflow

1. Client interacts with the frontend UI
2. Frontend calls backend API endpoints
3. Backend processes requests and queries MongoDB
4. Backend responds with JSON or errors
5. Frontend renders data for the user

4. User Authentication

4.1 Signup

- Provide name, email, password
- Check for duplicate email
- Hash password (bcrypt)
- Store user in MongoDB
- Return JWT token

4.2 Login

- Provide email and password
- Verify against hashed password
- Return JWT token if successful
- Prevent multiple logins using valid token

5. Roommate Search Functionality

5.1 Required and Optional Filters

Required: City, Budget, Gender
Optional: WiFi, AC, Pets

6. Post Management

- Create, edit, delete posts
- Only post owner can modify
- Frontend dynamically updates posts

7. Error Handling

- Signup: 400 if user exists, 500 for server errors
- Login: 400 for invalid credentials, 500 for server errors
- Post/Search: 500 for server errors

8. Security Considerations

- Hash passwords before storing
- JWT authentication and verification
- Environment variables for sensitive data
- Prevent multiple logins with tokens

9. 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 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

10. Implementation and Testing

10.1 Coding Standards

- Modular file structure
- Clear naming conventions
- Comments for complex logic

10.2 Modular Code

- Backend functions reusable for search and validation
- Frontend separated into components and pages
- API service layer handles requests

10.3 Testing

- **Unit Tests:** Backend functions, password hashing, queries
- **Integration Tests:** API endpoints, database interactions

- **Frameworks:** Jest, React Testing Library
- **Error Tests:** Invalid inputs return proper errors

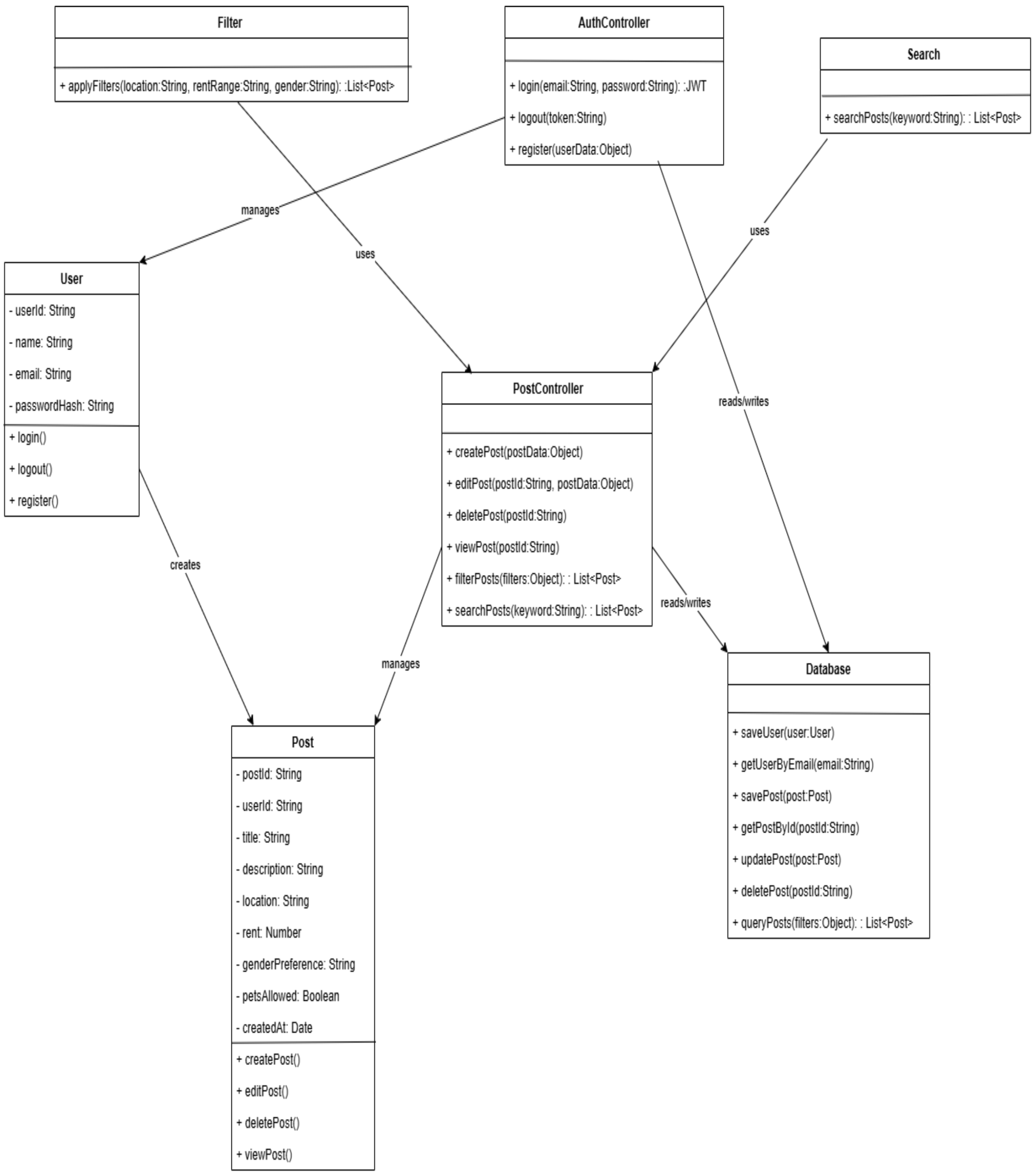
11. Future Improvements

- Email verification and password reset
- Frontend notifications
- Pagination and sorting of search results
- Real-time chat between users

12. UML Diagrams

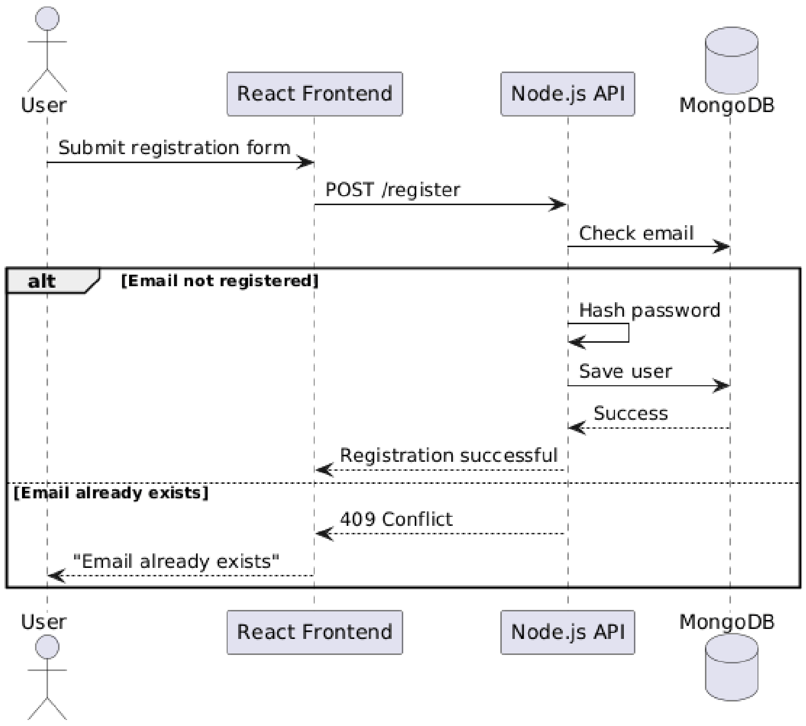
Use Case Diagram

Class Diagram

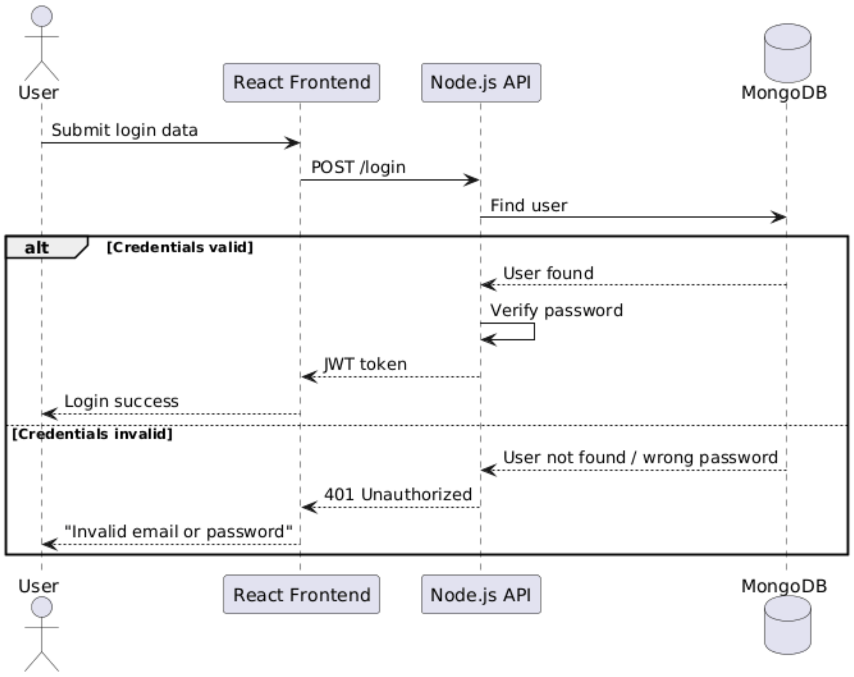

Filter
+ applyFilters(location:String, rentRange:String, gender:String): :List<Post>
AuthController
+ login(email:String, password:String): :JWT
+ logout(token:String)
+ register(userData:Object)
Search
+ searchPosts(keyword:String): : List<Post>
manages
uses
uses
reads/writes
User
- userId: String
-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

Sequence Diagrams

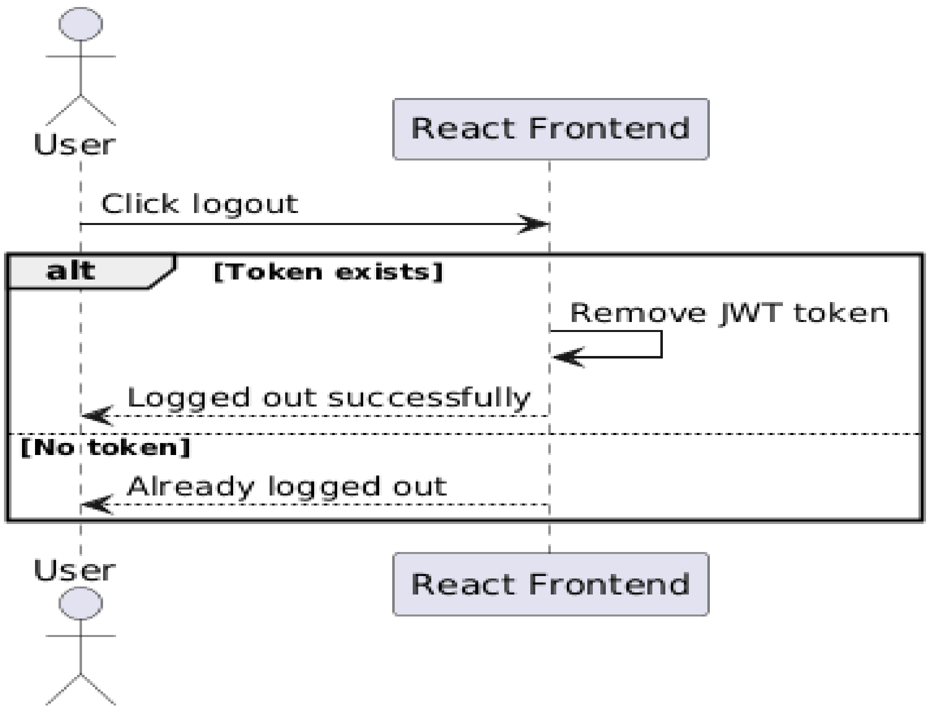
- **Register:**



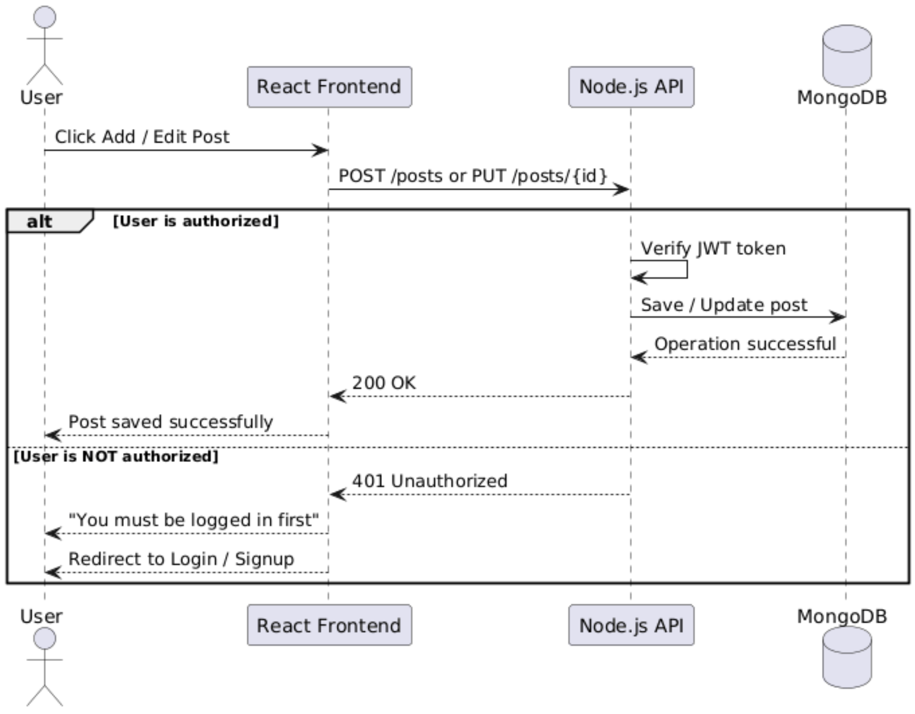
- **Login:**



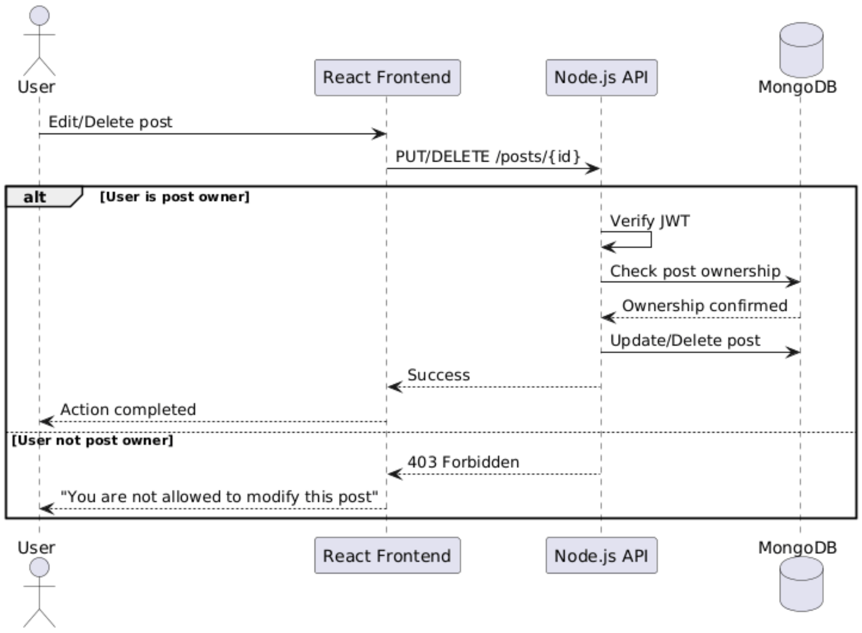
- **Logout:**



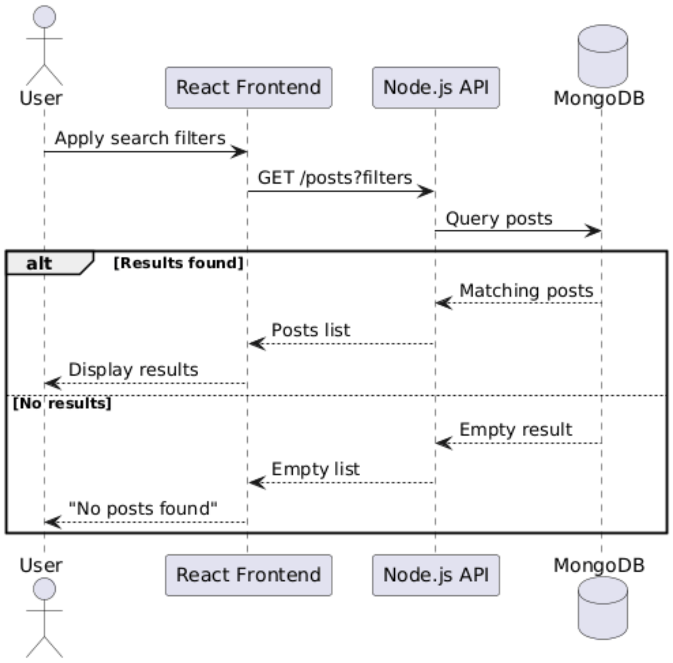
- **Create/Edit Post:**



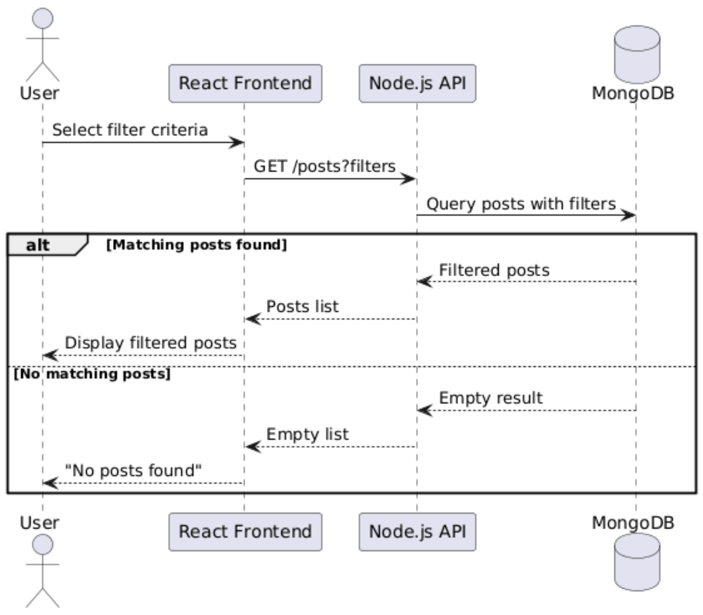
- **Edit/Delete Post:**



- **Search Posts:**



- **Filter Posts:**



13. Conclusion

Roommate Finder provides a secure, scalable, and user-friendly platform for finding roommates. By implementing full backend authentication, dynamic search filters, post management, and frontend-backend integration, the system is ready for deployment and future enhancements.

Limitations: Real-time chat is not yet supported.

14. References

1. Node.js Documentation: https://nodejs.org/en/docs/
2. Express.js Documentation: https://expressjs.com/
3. MongoDB Manual: https://www.mongodb.com/docs/

4. Mongoose Documentation: https://mongoosejs.com/docs/
5. JWT Introduction: https://jwt.io/introduction/
6. bcrypt Node.js: https://www.npmjs.com/package/bcrypt
7. UML 2.5 Specification: https://www.omg.org/spec/UML/2.5/