Master Team Project Fall 2024

NeuAnfang

Team 3

Member Name	**Role**
Zubeena Shireen Sheikh zubeena-shireen.sheikh@informatik.hs-fulda.de	Team Lead and Document Master
Muhammad Zaid Akhter	Backend Lead
Masood Ahmed Mohiuddin	Frontend Lead
Humdaan Syed	GitHub Master
Andrii Kuripka	Team Member (Frontend)

Milestone 1

12/11/2024

Date Submitted	**12/11/2024**
Date Revised	

1. Executive Summary:

NeuAnfang was created to address a pressing challenge faced by Fulda University students: finding convenient, and affordable accommodation when moving to the city. For students relocating to pursue their education, navigating a new rental market can be daunting and stressful. With a focus on the unique needs of students, NeuAnfang facilitates a smoother transition, allowing them to spend less time worrying about finding a home and more time enjoying their new chapter in Fulda.

NeuAnfang is **an ad-free, one-stop housing solution** where landlords can post property ads, and students can easily search for a home that meets their needs. It offers several unique features not easily found on similar platforms. Such as a **rental price estimator** - which offers fair pricing insights for students and landlords, helping prevent overpricing and setting realistic expectations. With its **matching algorithm**, NeuAnfang connects landlords with suitable tenants and supports students in finding ideal homes faster. Plus, it also features **an appointment scheduler**, allowing both parties to set up viewings within the application—removing the hassle of managing appointments separately.

When students find a listing they like, they can connect directly with landlords through NeuAnfang's **real-time, in-app chat**. Security is a top priority: students can **rate interactions** with landlords and easily report any suspicious activity to moderators, making the platform safer for everyone. With both **list and map views**, students can explore housing options effortlessly. They can **save favorite listings**, **view trending ads** to discover popular areas, and use **Neighbourhood guide** to check commute times to key locations. NeuAnfang acts as a dedicated guide through the housing search.

For landlords, NeuAnfang provides a **dashboard** with valuable ad analytics to help optimize listings. This dual focus on students and landlords sets our platform apart, offering value for all users. Looking ahead, NeuAnfang plans to introduce a **chatbot** powered by a large language model (LLM) to answer common questions students have about housing, further enhancing support.

Our team consists of five enthusiastic students from Fulda University and each of us have experienced the challenges of finding housing here firsthand. We've drawn from these experiences to design a platform that truly addresses the needs of our peers. Together, we've crafted a solution that combines useful features with thoughtful design and functionality. With our strengths in backend development and a keen eye for aesthetics, we're confident NeuAnfang will be the go-to housing solution for students looking for a smooth and informed rental experience.

2. Personae and Main Use cases:

Following are the main personas for the application:

1. Student
Zaid is a full-time student at Fulda University of Applied Sciences. He recently moved from Pakistan to pursue his Master's in Global Software Development. He is also working a part-time job hence he has very limited time. Also, he is new in Germany and he has very little German skills. He is looking for an apartment near the university under a budget of EUR 450 per month.

Pain points:
- Finding affordable housing,
- Communicating with landlords,
- Understanding rental terms

2. Landlord
John is a property owner in Fulda. He recently renovated his apartment complex on Leipziger Strasse, right opposite to the university. He has 3 available apartments and wants to rent them out to students. He is looking for reliable tenants. He wants to be able to efficiently manage inquiries, screen tenants easily and co-ordinate viewings with potential tenants.

Pain points:
- Managing inquiries,
- Screening tenants,
- Coordinating viewings

3. Site Admin
Masood is a member of the startup team. He studies Master's in Global Software Development at Fulda University of Applied Sciences. He has 4 years of discord moderation experience. He wants to be able to easily manage all the site content, ensure site-reliability and ensure the content uploaded on the site is appropriate. He also needs to be able to block malicious users.

Pain points:
- Content review workload,
- Maintaining site quality,
- Handling disputes

4. Guest User

Shireen has been accepted to study at Fulda University of Applied Sciences. She is still in India, but she wants to start exploring apartments in Fulda. She wants to get a rough idea of how much it will cost and what types of apartments are available. She is not looking to rent an apartment immediately, so she wants to easily browse without having to create an account.

5. Subletter

Andrii is studying at Fulda University of Applied Sciences. He received an invitation for an internship in another city. During the internship, he has to pay rent so he doesn't lose his apartment. He decides to sublet his apartment while he is away. This allows him not to pay extra in rent when he is away.

6. Scammer

Humdaan is a malicious user. He wants to scam people on the internet and earn some quick money. He creates fake profiles on websites and attracts new students with low and affordable prices. He then asks for money before arranging a viewing and runs away. The likes of him must be stopped and banned.

Based on the above personas, following are the main use cases of the application:

1. UC-1: List Apartments

John has 3 available apartments and wants to rent them out to students. He visits NeuAnfang to create a new apartment listing. He uploads photos and provides details like rent, location, and amenities. He specifies requirements for potential tenants and waits for administrator approval before the listing goes live. Once the listing is approved John's apartment ad will be online for students to view.

2. UC-2: Search Apartments

Zaid is looking for affordable housing in Fulda. He visits NeuAnfang and sees a list of properties on the home page. He applies filters to find an apartment near the university, with a fitted kitchen and washing machine and under EUR 450. He then sorts his search by newest first. After clicking search he sees a list of apartments that match his needs. He can view quick info like postcode, monthly rent, title of the ad from the list view. He can also switch to the map view to see the apartments on a map.

3. UC-3: View Apartment Details

Once Zaid finds an ad he likes. He clicks on it to view the full details of the apartment. He can browse through photos, read the description uploaded by the landlord and see the available amenities. He can also see the exact location of the apartment on a map.

4. UC-4: Real-time Chat
Zaid likes the apartment he is viewing; he initiates contact with the landlord through the platform's messaging system. They can discuss viewing arrangements, ask questions about the property, and share basic contact information for further communication, all while maintaining privacy through the platform.

5. UC-5: Moderate Content
Masood reviews new listings in his queue, checking for compliance with platform guidelines, appropriate content, and accurate information. He can approve, reject, or request modifications to listings. He also monitors user interactions and responds to reported content or user complaints.

3. List of main data items and entities:

User is a generic term used for referring students and landlords together. The application has the following user types:

A **Guest** is an unauthenticated user. A guest can browse the platform and search for listings but they see a subset of the available information. Some information like the exact location of the apartment and landlords contact info is hidden from them.

A **Student** represents a verified user with an hs-fulda.de email domain. They form the primary user base of the platform, with capabilities to search listings, maintain a wishlist of preferred apartments, communicate with landlords, and save their search preferences. Students can report inappropriate content and must be able to demonstrate their affiliation with Fulda University.

A **Landlord** is a verified user who owns or manages rental properties. They can create and manage apartment listings, respond to student inquiries, and track the status of their properties. Landlords must provide valid contact information and maintain their property listings, including updating availability and responding to offers from potential tenants.

The **Site Admin** holds system-level privileges and is responsible for platform integrity. They review and approve new listings, moderate user content, manage user accounts, and handle disputes. Admins also monitor platform activity and can generate system reports for operational insights.

Some other main data items in the application are defined as follows:

An **Ad/Apartment** is the central entity of the platform, representing a rental property listing. It contains comprehensive property information including location, price, amenities, and media

content. Each listing has a unique identifier and maintains various states from creation through to rental or archival.

The platform offers two primary viewing modes: **List View** provides a structured, scannable format showing apartment summaries with key details and quick actions, while **Map View** offers a geographical representation of available properties, featuring interactive markers and distance-based searching from campus.

Search Criteria encompasses the filtering and sorting parameters users can apply to find suitable apartments. This includes price ranges, location preferences, property features, and distance from campus. Users can save their search profiles for future use.

The **Chat** system facilitates direct communication between students and landlords. Each conversation is tied to a specific property listing and maintains a complete message history. The system supports basic attachment sharing while adhering to platform security guidelines.

An **Offer** represents a formal expression of interest from a student to a landlord regarding a specific property. It includes proposed terms and maintains various states from initial submission through to acceptance or rejection.

The **Wishlist** feature allows students to save and organize their favorite listings. Users can add personal notes and set alert preferences for saved properties.

Notifications keep users informed of relevant platform activity. These can be system alerts, chat messages, or listing updates. Each notification has a priority level and expiration date where applicable.

A **Session** tracks user activity on the platform, maintaining security and authentication state. It includes login information, device details, and activity timestamps to ensure secure access to the platform.

The platform also maintains several status categories to track the state of various entities:

Listing Status:
Listings can be Pending (awaiting approval), Active (available for rent), Rented (transaction completed), Archived (no longer available), or Rejected (failed approval).

User Status:
User accounts may be Pending Verification, Active, Suspended, or Deactivated, reflecting the user's standing on the platform.

Chat Status:
Chat conversations can be Active, Archived, Blocked, or Reported, managing the flow of communication between users.

Offer Status:
Offers progress through states including Pending, Accepted, Rejected, Expired, or Withdrawn, tracking the rental negotiation process.

4. Initial list of functional requirements:

Following is an initial list of the functional requirements:

1. **Apartment Posting and Pricing:**
   Landlords can post apartment listings, providing essential details to attract potential student tenants. The system suggests a rental price based on the average cost per square meter, helping landlords set competitive rates.

2. **Share listings:**
   Landlords can share apartment listings on other social platforms, increasing visibility and reach for the property.

3. **Ad Status Management:**
   Landlords can indicate the availability of their property to students by mark listings as reserved, archived, or deleted.

4. **Student Profiles and Document Upload:**
   Students can create profiles and upload relevant documents, such as identification and proof of enrollment, to support their rental applications.

5. **List View of Apartments:**
   Students can browse all available apartments in a list format, making it easier to compare multiple properties at once.

6. **Map View of Apartments:**
   The exact location of the apartment can be viewed on an interactive map for a geographic overview and easy navigation, helping students understand its proximity to key areas.

7. **Save Search Criteria:**
   Students can save custom search criteria to streamline future searches with the same parameters.

8. **Detailed Apartment View:**
   Students can click on an apartment listing to see detailed information, such as amenities, photos, floor plans, and rental terms.

9. **Commute Information:**
   Students can view commute times to the university and nearby bus routes for better transportation planning.

10. **Wishlist Apartments:**
    Students can add apartments to a wishlist and access a dedicated section with all their wishlisted apartments for easy reference.

11. **Real-Time Chat:**
    The platform supports real-time chat functionality, allowing students and landlords to communicate instantly.

12. **Student Documents Sharing:**
    Students can message landlords to inquire about listings, with student profile documents shared automatically with the landlord after approval by the student.

13. **Chat File Sharing:**
    Users can upload images and documents within chat messages, streamlining information exchange.

14. **Block/Unblock Users in Chat:**
    Users can block or unblock others on chat if necessary, helping maintain privacy and security.

15. **Offer Apartment/Retract Offer in Chat:**
    Landlords can make a direct rental offer to students they are communicating with in the chat. Landlords can retract the offer before it is accepted or rejected by the student.

16. **Accept/Reject Offer and Rate Landlord:**
    Students can accept or decline a rental offer from a landlord.

17. **User ratings:**
    Students can rate landlords based on their experience. This would encourage landlords to post quality ads and be at their good behaviors and also increase reach for higher ranked landlords.

18. **View and Follow Landlord Profiles for Updates:**
    Students can view a landlord's profile, which includes all apartment listings they have posted on the platform. They can follow specific landlords to receive notifications on new postings and maintain easy access to their listings.

19. **Ad Approval by Admin:**
    The site administrator reviews and approves each apartment listing before it becomes visible to students, ensuring that inappropriate/fraudulent ads are not posted.

20. **Landlord Dashboard with Analytics:**
    Landlords can access a dashboard that provides insights into the performance of their listings (e.g., number of views, inquiries per listing) to help gauge interest and optimize listings.

21. **Customizable Notifications:**
    Both students and landlords can customize their notification settings (e.g., get notified about new listings in preferred areas, updates on inquiries).

22. **Advanced Filtering Options:**
    Allow students to filter listings by specific amenities (e.g., furnished, utilities included) for a more tailored search experience.

23. **In-App Apartment Viewing Scheduler:**
    Include a shared calendar feature where landlords and students can schedule viewings directly within the app, with reminders sent out to both parties.

24. **Message Templates for Landlords:**
    Landlords can create and reuse message templates for common inquiries, such as availability or terms, to save time in responding.

25. **Neighborhood Information Section:**
    Display information about the neighborhood, such as nearby amenities (e.g., grocery stores, parks), and popular spots for students.

26. **Matching Ads with Students:**
    Display potential ad matches for the student. A way of connecting similar ads and students. Helps students to find a high probability apartment quicker.

27. **Match Students for Landlord:**
    Enable landlords to view and connect with students who match their tenant criteria, based on factors like budget, move-in date, and preferred apartment features.

28. **Student Subletting:**
    Allow students to list and manage subletting options, enabling them to rent out their apartments to other students temporarily.

29. **Report ads:**
    Provide users with a reporting option to flag inappropriate or fraudulent ads for review by the site administrator.

30. **LLM bot for Accommodation Queries:**
    Integrating an AI chatbot to answer any and all accommodation-related questions (e.g., about the German rental system, common housing scams, finances and taxes, etc.)

31. **Show interested/views per ad:**
    Each ad will display real-time statistics showing the number of views. Aside from helping students gauge the popularity and competitiveness of a listing, this feature encourages landlords to optimize their ads based on engagement metrics.

32. **Trending/Popular ads:**
    Ads with the most views are tagged as "Trending." This feature allows students to see popular listings that may have higher demand, helping them make timely decisions on competitive apartments.

33. **Student Account Verification:**
    Implements a verification process to confirm that only students from Fulda University can create accounts and access listings.

5. List of non-functional requirements:

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server
2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
3. All or selected application functions must render well on mobile devices
4. Data shall be stored in the database on the team's deployment cloud server.
5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner
6. No more than 50 concurrent users shall be accessing the application at any time
7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.
8. The language used shall be English (no localization needed)
9. Application shall be very easy to use and intuitive
10. Application should follow established architecture patterns
11. Application code and its repository shall be easy to inspect and maintain
12. Google analytics shall be used (optional for Fulda teams)
13. No email clients shall be allowed.
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.

15. Site security: basic best practices shall be applied (as covered in the class) for main data items
16. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today
17. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
18. For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set-up by class instructors and started by each team during Milestone 0
19. The application UI (WWW and mobile) shall prominently display the following exact text on all pages "Fulda University of Applied Sciences Software Engineering Project, Fall 2024 For Demonstration Only" at the top of the WWW page. (Important to not confuse this with a real application).

6. Competitive analysis:

Our platform stands out among competitors like WG-Gesucht, ImmobilienScout24, Studierendenwerk, and Kleinanzeigen by focusing specifically on student rentals and offering unique features tailored to the needs of student tenants and landlords. Here are the key advantages of our platform:

1. **Student-Focused Rentals**: Unlike general rental or classified platforms, our platform is exclusively student-focused, ensuring listings and services cater directly to students' needs.
2. **Mobile Application:** WG-Gesucht, ImmobilienScout24, and Kleinanzeigen offer dedicated mobile apps, providing users with easy access to their services through standalone applications. Unfortunately, our platform doesn't have a mobile app at this time; however, users can still access all our features on mobile devices through our fully responsive website, which ensures an optimal browsing experience.
3. **Enhanced Verification and Safety:** Our platform includes student verification, which adds an extra layer of trust and security that general platforms lack, making it safer for both students and landlords.
4. **Commute Times and Map View:** Our map feature includes commute times to university, which is especially useful for students balancing housing location with academic schedules—an enhancement over standard map views on other platforms.
5. **Rental Price Recommendations:** We offer rental price suggestions, providing students and landlords with fair pricing insights, encouraging them to rely on this suggestion to avoid unreasonable expectations or overpricing. This feature is designed to promote fair pricing for students, a service not commonly available on other platforms.
6. **Comprehensive Landlord Dashboard:** Our platform supports landlords with a dedicated dashboard for managing properties, a feature that most competitors lack.

7. **User Ratings and Reviews:** By incorporating reviews for both students and landlords, we help foster transparency and trust, unlike most other platforms that do not offer user feedback systems.
8. **Profile Customization:** Both students and landlords can customize their profiles, allowing for a more personalized experience that sets our platform apart from others with limited or no customization options.

In summary, our platform combines student-centric features with advanced functionalities, providing a safer, more user-friendly experience tailored specifically for the student rental market. This specialization and range of features make it a superior choice for both students and landlords compared to broader rental or classified platforms.

FEATURES	OUR PLATFORM	WG-GESUCHT	IMMOBILIEN-SCOUT24	KLEIN ANZEIGEN	STUDIERENDEN WERK
EXCLUSIVELY FOR STUDENTS	✓	✗	✗	✗	✓
MOBILE APPLICATION	✗	✓	✓	✓	✗
USER RATING/REVIEWS	✓	✗	✗	✓	✗
RENTAL PRICE SUGGESTIONS	✓	✗	✗	✗	✗
TIME TO UNI ROUTES	✓	✗	✓	✗	✗
IN-APP CALENDAR	✓	✗	✗	✗	✗
ALL APPARTMENTS MAP VIEW	✓	✗	✓	✗	✗
PROFILE CUSTOMIZATION	✓	✓	✗	✗	✗

7. High-level system architecture and technologies used:

Following are the main technologies that will be used to develop the application:

Server Host: AWS EC2 t2.micro 1 vCPU 1 GiB RAM
Operating System: Ubuntu 22.04 Server
Database: MySQL 8.0 on AWS RDS
Web Server: NGINX 1.12.2
SSL Cert: Let's Encrypt (CertBot)
Server-Side Language: NodeJS
Additional Backend Technologies: ExpressJS, socket.io
Frontend Framework: ReactJS
Additional Front End Technologies: Material UI

Now moving on to system architecture. The application will use a three-tier web application design. Let's break down each tier:

1. Client Tier:

- Features three types of users: Students (who view ads), Landlords (who post ads), and Admins (who moderate content)
- Frontend is built using React.js
- Users interact with a web interface for various operations

2. Server Tier

- Running on Ubuntu VM in AWS EC2
- NGINX Web Server
    - Running on ports 443 (HTTPS) and 80 (HTTP)
    - Acts as a reverse proxy to Node.js server on localhost:3000
- Node.js Server
    - Running on port 3000
    - Handles API requests and business logic
    - Communicates with storage and database tiers

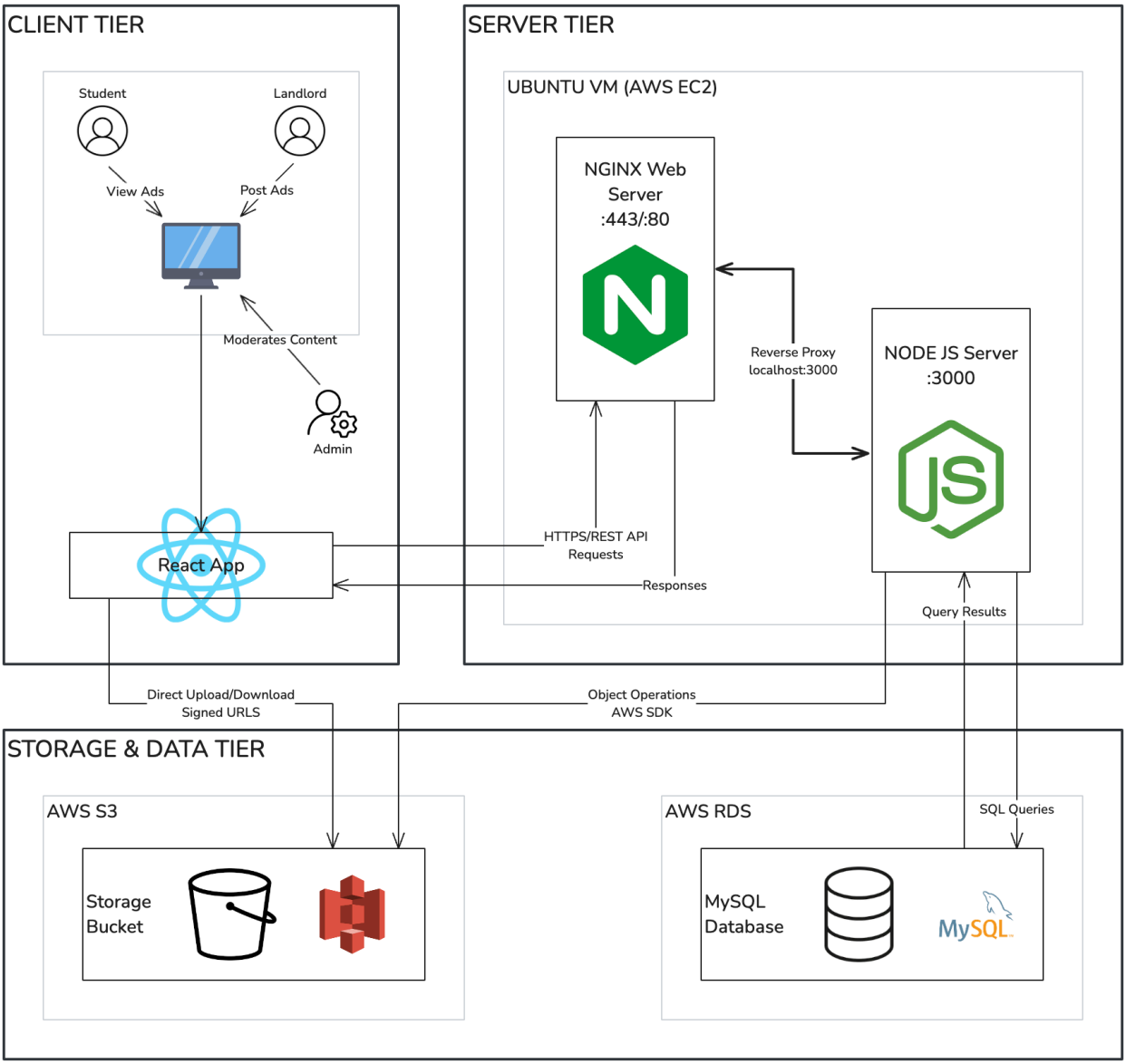
3. Storage & Data Tier:

- AWS S3
    - Used for storage bucket
    - Handles direct upload/download with signed URLs
- AWS RDS
    - Running MySQL database
    - Handles SQL queries and data persistence

Key Integration Points:

- HTTPS/REST API requests flow from React App through NGINX to Node.js
- AWS SDK is used for object operations with S3
- Direct file uploads/downloads use signed URLs for S3 access
- Database queries are executed from Node.js to MySQL

The architecture follows modern security practices with HTTPS and proper separation of concerns across tiers. It's scalable and uses industry-standard AWS services for storage and database operations. This architecture is better described by the diagram below:



8. Team and roles:

Member Name	**Role**
Zubeena Shireen Sheikh	Team Lead and Document Master
Muhammad Zaid Akhter	Backend Lead
Masood Ahmed Mohiuddin	Frontend Lead
Humdaan Syed	GitHub Master
Andrii Kuripka	Team Member (Frontend)

9. Checklist

Task	**Status**
Team found a time slot to meet (online) outside of the class	**DONE**
GitHub master chosen	**DONE**
Team decided and agreed together on using the listed SW tools and deployment server	**DONE**
Team ready and able to use the chosen back and front end frameworks and those who need to learn are working on learning and practicing	**ON TRACK**
Team lead ensured that all team members read the final M1 and agree/understand it before submission	**DONE**
GitHub organized as discussed in class (e.g. master branch, development branch, folder for milestone documents etc.)	**DONE**

Master Team Project Fall 2024
NeuAnfang
Team 3 (local)

Member Name	Role
Zubeena Shireen Sheikh zubeena-shireen.sheikh@informatik.hs-fulda.de	Team Lead and Document Master
Muhammad Zaid Akhter	Backend Lead
Masood Ahmed Mohiuddin	Frontend Lead
Humdaan Syed	GitHub Master
Andrii Kuripka	Team Member (Frontend)

Milestone 2
27/11/2024

Date Submitted	**27/11/2024**

Content and structure for Milestone 2 document for review by institutors:

1. Functional Requirements - Prioritised
2. List of main data items and entities
3. UI Mockups and Storyboards
4. High level Architecture, Database Organization
5. High Level UML Diagrams
6. Key risks for project at this time
7. Project management

1. Functional Requirements - Prioritised:

Following is a prioritized list of the functional requirements:

Priority 1 (Must have):

1. **Apartment Posting and Pricing:**

   1.1 Landlords can add detailed information about their properties, including rent, location, size, available amenities, and conditions.

2. **Share Properties:**

   2.1 Landlords can generate shareable links that can be shared on other social media platforms thereby increasing the reach of their apartment.

3. **Ad Status Management:**

   3.1 Landlords can mark ads as draft, pending, active, rented, archived, or deleted.

5. **List View of Apartments:**

   5.1 The list view is sortable by filters and includes thumbnail images for quick browsing.

   5.2 Each property in the list view displays a summary of key information such as proximity to Fulda University.

6. **View of Apartments:**

   6.1 An interactive map shows property markers, allowing students to zoom in on specific neighbourhoods.

8. **Detailed Apartment View:**

   8.1 The detailed view includes images and an expanded list of amenities.

11. **Real-Time Chat:**

    11.1 Chat features include read receipts and user online status.

18. **Ad Approval by Admin:**

    18.1 The site admin reviews and approves each property before it becomes visible to students.

18.2 Admins can communicate directly with landlords to resolve flagged issues.

19. Landlord Dashboard with Analytics:

19.1 The dashboard displays basic metrics such as the total number of views, inquiries, and wishlists for each property.

19.2 Landlords can mark properties as archived, deleted, or updated.

21. Advanced Filtering Options:

21.1 Filters include advanced criteria like apartment age, pet policies, etc.

21.2 Students can combine multiple filters for complex searches.

27. Student Subletting:

27.2 Students can specify furniture and utility usage conditions for sublets.

32. Student Account Verification:

32.1 Verification requires university email addresses.

32.2 Verified accounts are prominently tagged for enhanced trust.

33. Ad Search Feature

33.1 Students can search for properties using a search bar with keywords, such as type of apartment/room, and location (e.g., "studio apartments", “single room”).

33.3 Results are dynamically updated when students refine their search using filters and sort options.

Priority 2 (Desired):

1. **Apartment Posting and Pricing:**

1.2 The system suggests a price range based on the average cost per square meter and local rental trends, which landlords can adjust based on their preferences.

1.3 Landlords receive prompts to complete missing details to improve property visibility and accuracy.

2. **Share properties:**

   2.2 Sharing options include direct integration with popular platforms (e.g., Facebook, Instagram, WhatsApp).

3. **Ad Status Management:**

   3.2 Notifications are sent to students when the status of a property they interacted with changes.

4. **User Profiles and Document Upload:**

   4.1 Students must complete a profile, including name, university email, study program, and contact information.

   4.2 Landlords must create a profile that includes name, email, phone number, and address for identity verification.

   4.3 Students can opt to add supplementary documents, such as prior rental references, to strengthen their application.

   4.4 The document upload feature accepts multiple formats (e.g., PDFs, images) and verifies them for authenticity.

6. **Map View of Apartments:**

   6.2 Additional overlays show amenities such as grocery stores and public transport stops.

7. **Save Search Criteria:**

   7.1 Students can save multiple sets of search criteria with custom names for easier recall.

   7.2 Notifications are triggered when new properties match saved criteria.

9. **Commute Information:**

   9.1 Provides estimated commute times using public transport, biking, walking, or driving to key locations like university, groceries, drugstore and hospitals.

   9.2 Links to public transport schedules and route planners are included for convenience.

10. **Wishlist Apartments:**

    10.1 Students can organise apartment properties into wishlists.

10.2 Alerts notify students of changes in the status or availability of wishlisted apartment properties.

10.3 Students can add notes under their wishlisted apartments to record reminders or specific details about the property.

10.4 Wishlisted items can be selected, and their features can be compared side by side in a comparison view.

12. Student Documents Sharing:

12.1 Students can choose which documents to share during the inquiry process, maintaining control over their data.

12.2 The system securely encrypts shared documents to protect sensitive information.

13. Chat File Sharing:

Supports drag-and-drop file uploads for easy sharing of documents and images.

14. Block/Unblock Users in Chat:

14.1 Blocking a user disables further communication and hides chat history.

14.2 Users can manage their block list from their profile settings.

15. Offer Apartment/Retract Offer in Chat:

15.1 Landlords can attach terms and conditions to offers made through chat.

15.2 Notifications are sent to both parties when an offer is retracted.

16. Accept/Reject Offer and Rate Landlord:

16.1 Students can leave detailed feedback and ratings for landlords after accepting or rejecting an offer.

16.2 Ratings influence a landlord's visibility in searches.

17. View and Follow Landlord Profiles for Updates:

17.1 Landlord profiles display a cumulative rating and all active properties

17.2 Students are notified when landlords they follow post new properties.

19. Landlord Dashboard with Analytics:

19.3 Highlights the most popular properties and provides simple tips, such as "Add more photos" or "Adjust price for better visibility," to improve engagement.

20. Customizable Notifications:

Users can enable or disable specific types of notifications (e.g., new property alerts, status updates, or chat messages) individually.

22. In-App Apartment Viewing Scheduler:

22.1 Scheduling integrates with external calendars (e.g., Google Calendar).

22.2 Both parties can customise reminders for scheduled viewings.

23. Message Templates for Landlords:

Templates can include placeholders (e.g., #StudentName, #PropertyTitle) for personalization.

25. Matching Ads with Students:

25.1 Matches are prioritised based on how closely properties meet students' preferences.

25.2 Students receive a daily list of new matches.

27. Student Subletting:

27.1 Sublet properties are flagged as temporary and require landlord approval before posting.

28. Report ads:

28.1 Reporting options include categories such as "Fraudulent" or "Inappropriate Content".

29. LLM bot for Accommodation Queries:

29.1 The chatbot answers accommodation-related questions in English to assist students.

29.2 Provides general guidance on topics like rental processes, legal requirements, and housing tips.

30. Show interested/views per ad:

Metrics include unique views, number of inquiries, and shares.0

31. Trending/Popular ads:

31.1 Properties with high engagement are tagged as "Trending" for enhanced visibility.

31.2 Students receive alerts for trending ads matching their preferences.

33. Ad Search Feature

33.2 The search feature supports natural language queries (e.g., "apartments under €500 with a balcony").

33.4 Search results highlight the best matches based on students' preferences, saved search criteria, and popularity of properties.

Priority 3 (Opportunistic):

8. Detailed Apartment View:

8.2 The detailed view includes 360-degree virtual tours (if available).

11. Real-Time Chat:

11.2 A "quick reply" function allows landlords and students to use predefined responses.

24. Neighborhood Information Section:

24.1 Displays reviews from previous tenants about the neighbourhood.

24.2 Includes details about safety and affordability.

26. Match Students for Landlord:

26.1 Landlords can view a dashboard of matched students ranked by compatibility.

26.2 Direct communication options are enabled for matched students.

28. Report ads:

28.2 Users are notified of the outcome of their reports.

32. Student Account Verification:

32.3 Verification requires confirmation of enrollment.

2. List of main data items and entities:

User is a generic term used for referring students and landlords together. The application has the following user types:

A **Guest** is an unauthenticated user. A guest can browse the platform and search for properties but they see a subset of the available information. Some information like the exact location of the apartment and landlords contact info is hidden from them.

A **Student** represents a verified user with an hs-fulda.de email domain. They form the primary user base of the platform, with capabilities to search properties, maintain a wishlist of preferred properties, communicate with landlords, and save their search preferences. Students can report inappropriate content and must be able to demonstrate their affiliation with Fulda University.

A **Landlord** is a verified user who owns or manages rental properties. They can create and manage properties, respond to student inquiries, and track the status of their properties. Landlords must provide valid contact information and maintain their properties, including updating availability and responding to offers from potential tenants.

The **Admin** holds system-level privileges and is responsible for platform integrity. They review and approve new properties, moderate user content, manage user accounts, and handle disputes. Admins also monitor platform activity and can generate system reports for operational insights.

Some other main data items in the application are defined as follows:

An **Property** is the central entity of the platform, representing a rental property ad. It contains comprehensive property information including location, price, amenities, and media content. Each property has a unique identifier and maintains various states from creation through to rental or archival.

The platform offers two primary viewing modes: **List View** provides a structured, scannable format showing apartment summaries with key details and quick actions, while **Map View** offers a geographical representation of available properties, featuring interactive markers and distance-based searching from campus.

Search Criteria encompasses the filtering and sorting parameters users can apply to find suitable apartments. This includes price ranges, location preferences, property features, and distance from campus. Users can save their search profiles for future use.

The **Chat** system facilitates direct communication between students and landlords. Each conversation is tied to a specific property and maintains a complete message history. The system supports basic attachment sharing while adhering to platform security guidelines.

An **Offer** represents a formal expression of interest from a student to a landlord regarding a specific property. It includes proposed terms and maintains various states from initial submission through to acceptance or rejection.

The **Wishlist** feature allows students to save and organize their favourite properties. Users can add personal notes and set alert preferences for saved properties.

A **Review** is feedback from the student given to the landlord. It includes a rating and a comment about the student's experience with the landlord.

A **Landmark** is a place of interest like grocery stores, shopping centers, hospitals, gyms etc. They are used to prepare a neighbourhood guide showing routes and commute times from apartments to landmarks.

A **Report** is a flag by the user. A user can flag content on the site and report it to the admin, the admin can review this content for violation of site policies.

Notifications keep users informed of relevant platform activity. These can be system alerts, chat messages, or property updates. Each notification has a priority level and expiration date where applicable.

A **Session** tracks user activity on the platform, maintaining security and authentication state. It includes login information, device details, and activity timestamps to ensure secure access to the platform.

A **Template** is a quick way for landlords/students to chat. Users can predefine long messages and use it quickly in a chat scenario.

An **Appointment** refers to an apartment viewing appointment between a landlord and a student. **Reminders** can also be added to appointments.

The platform also maintains several status categories to track the state of various entities:

Property Status:
Properties can be Pending (awaiting approval), Active (available for rent), Rented (transaction completed), Archived (no longer available), or Rejected (failed approval).

User Status:
User accounts may be Pending Verification, Active, Suspended, or Deactivated, reflecting the user's standing on the platform.

Chat Status:
Chat conversations can be Active, Archived, Blocked, or Reported, managing the flow of communication between users.

Offer Status:
Offers progress through states including Pending, Accepted, Rejected, Expired, or Withdrawn, tracking the rental negotiation process.

3. UI Mockups and Storyboards:

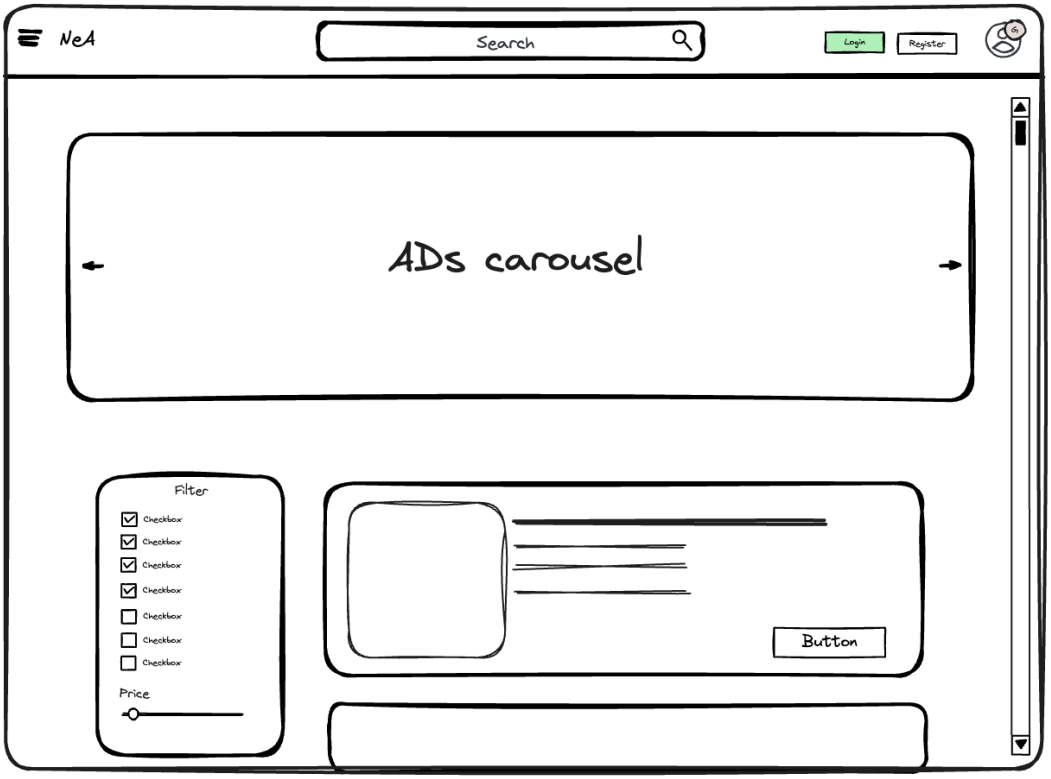


Fig 3.1: Home page (Guest)

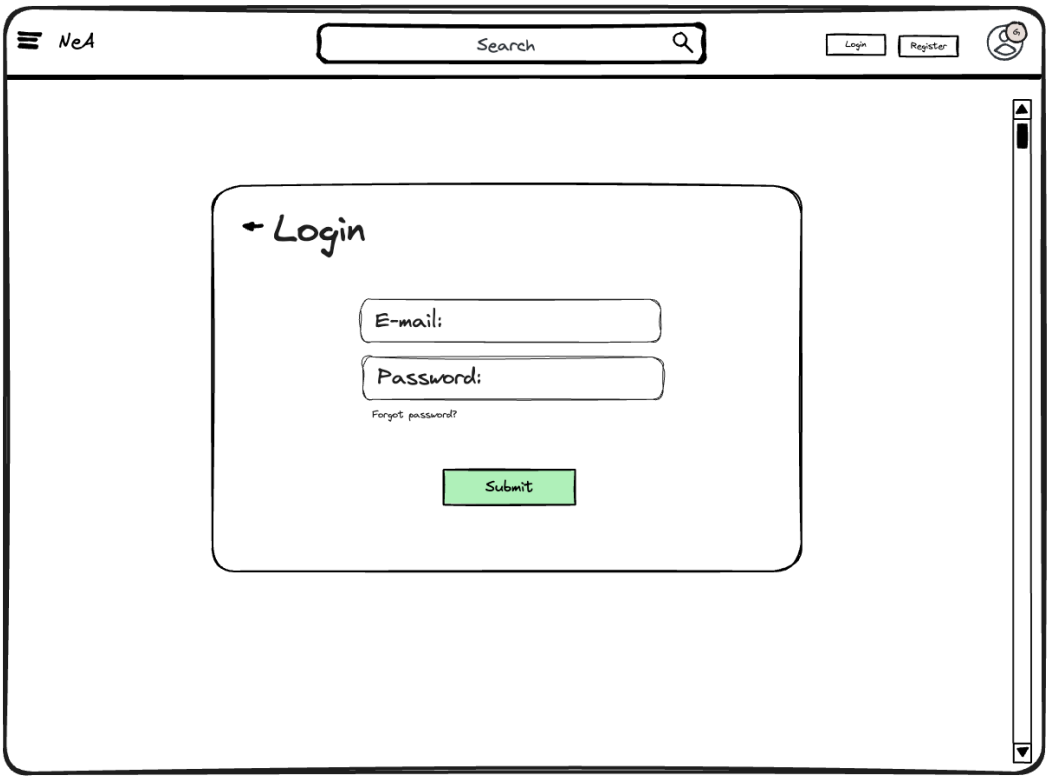


Fig 3.2: Login Page

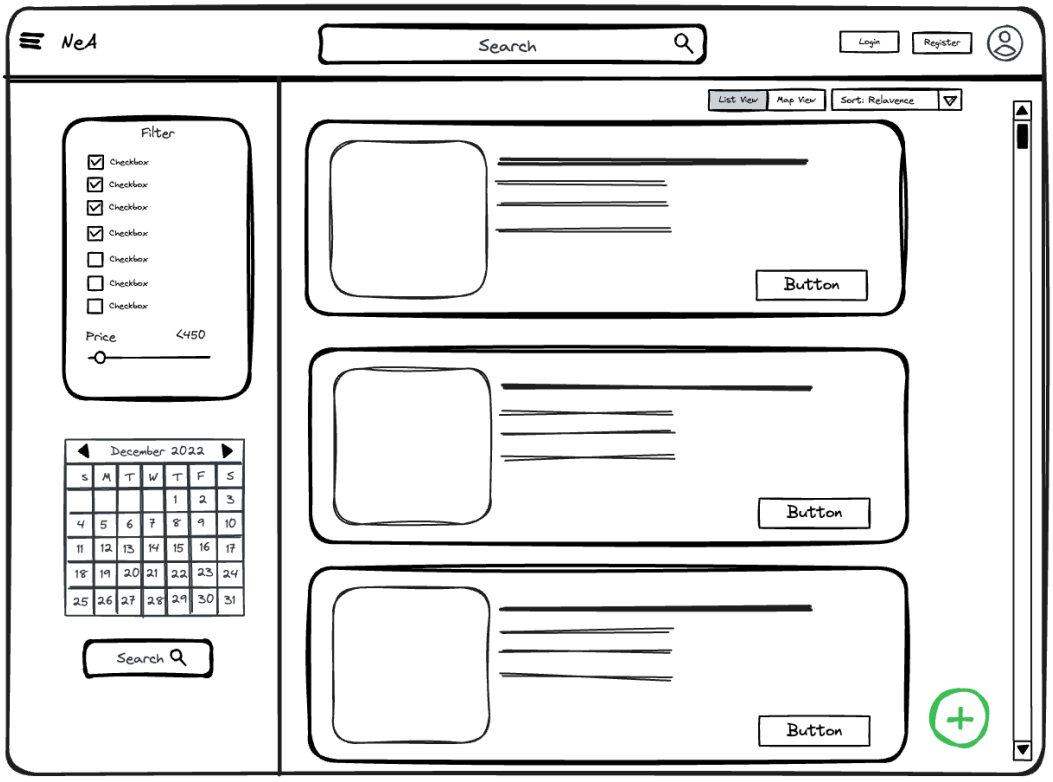


Fig 3.3: Search/List View (Landlord)

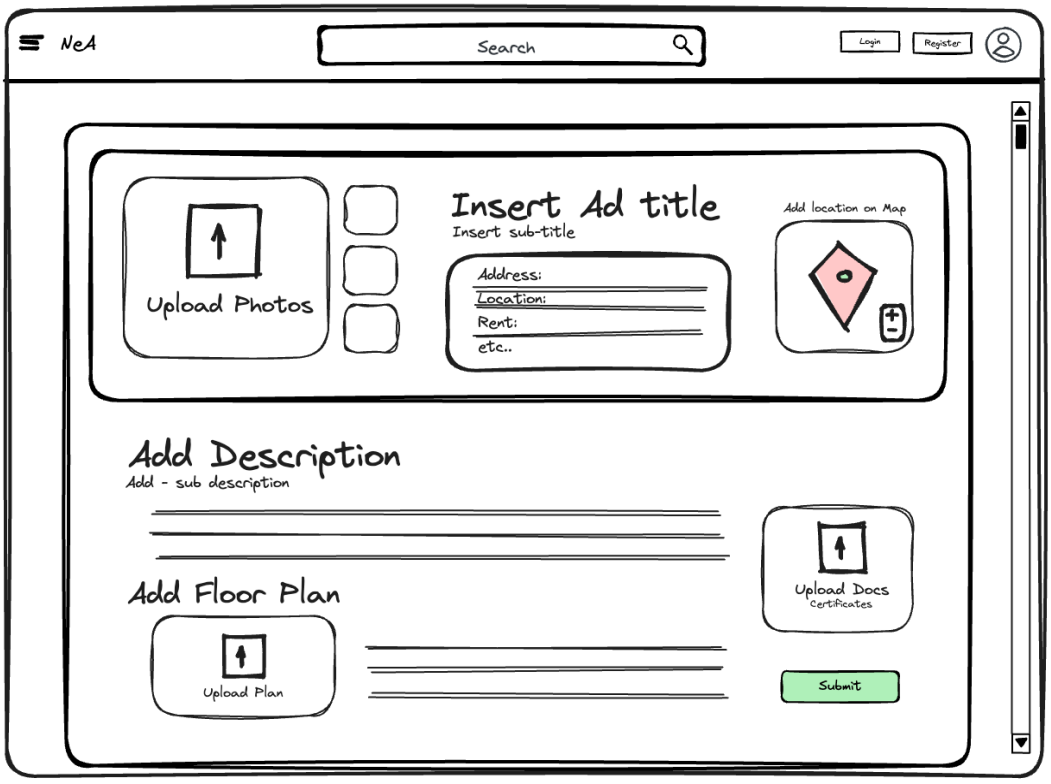


Fig 3.4: Create Property Page

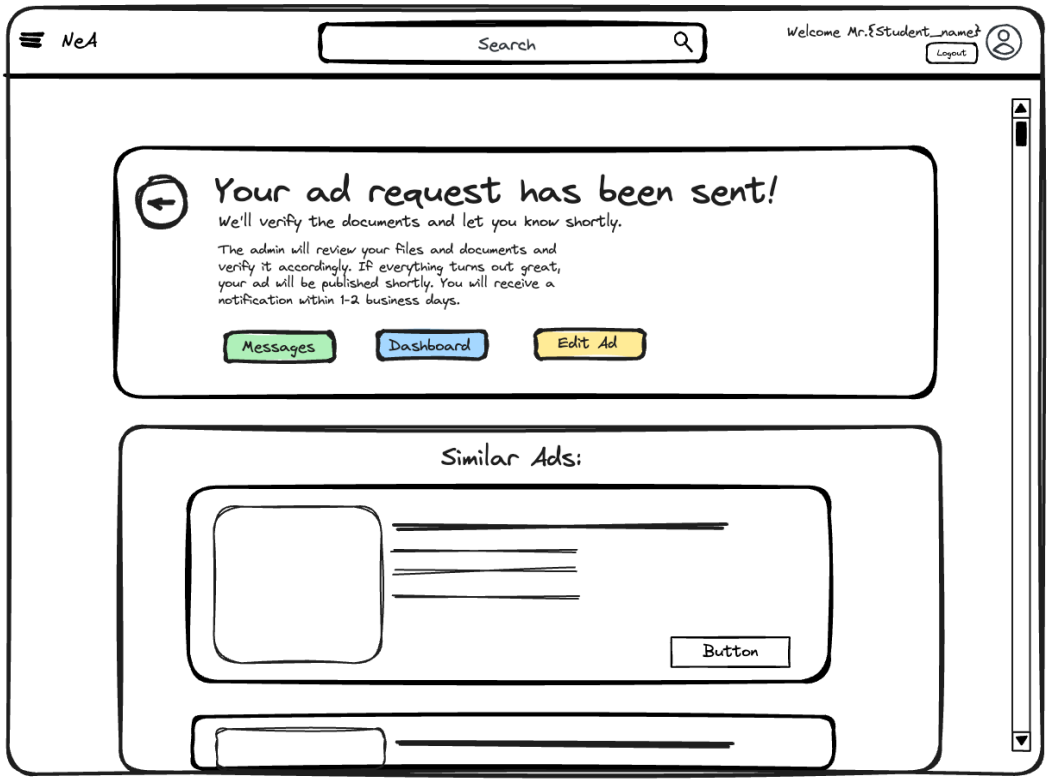


Fig 3.5: Property listing Submitted for Review

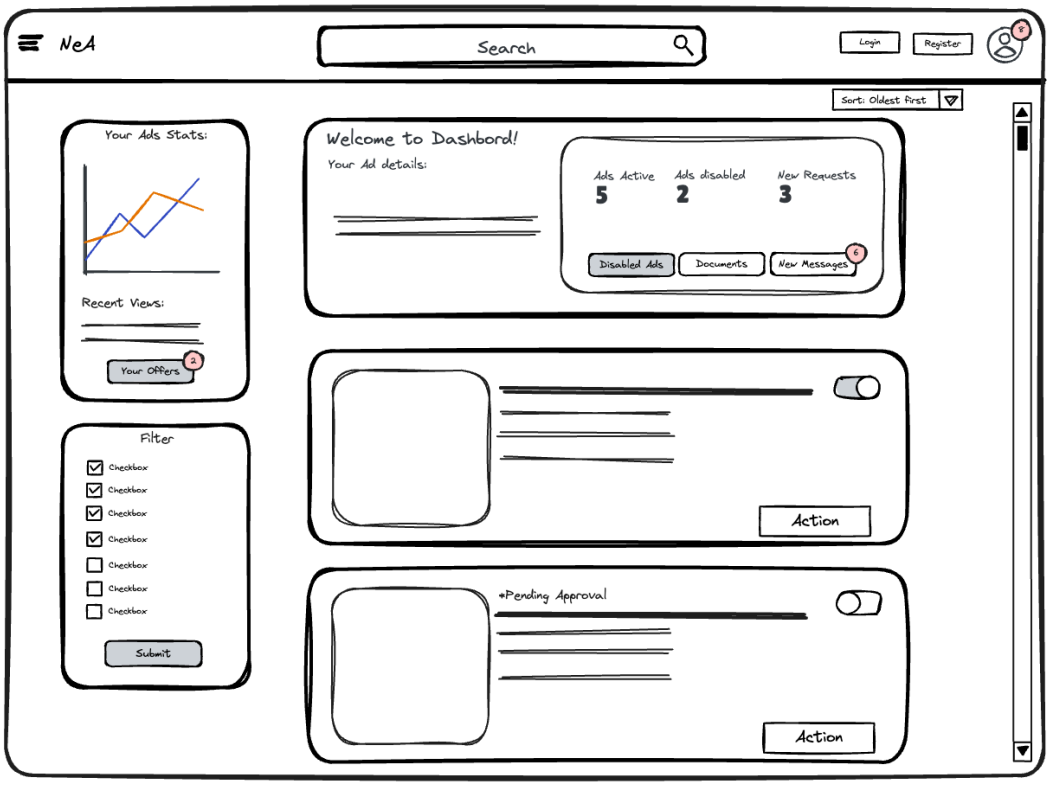


Fig 3.6: Landlord Dashboard

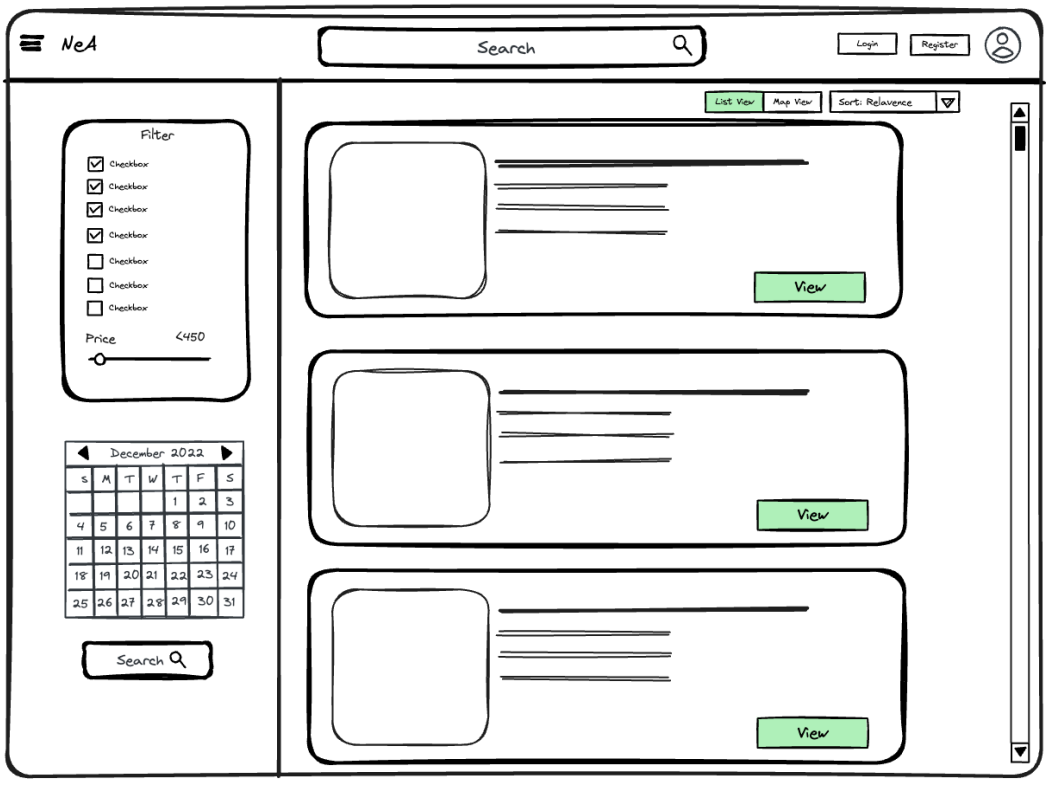


Fig 3.7: Search/List View (Student)

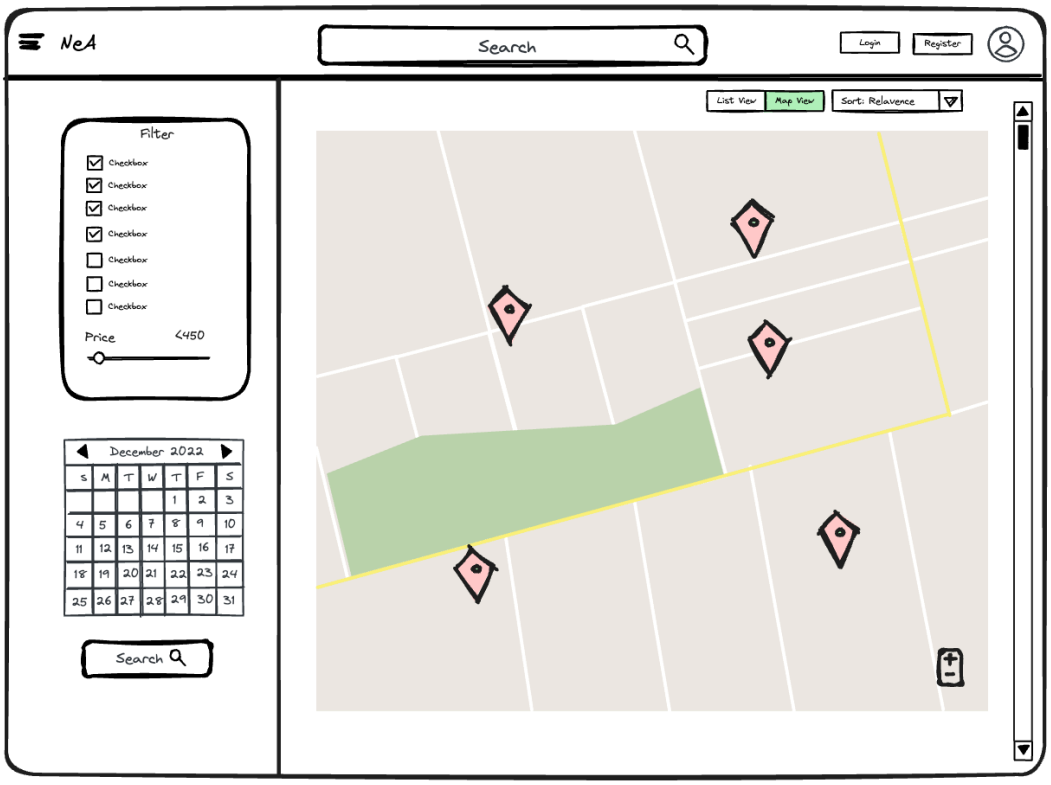


Fig 3.8: Map View

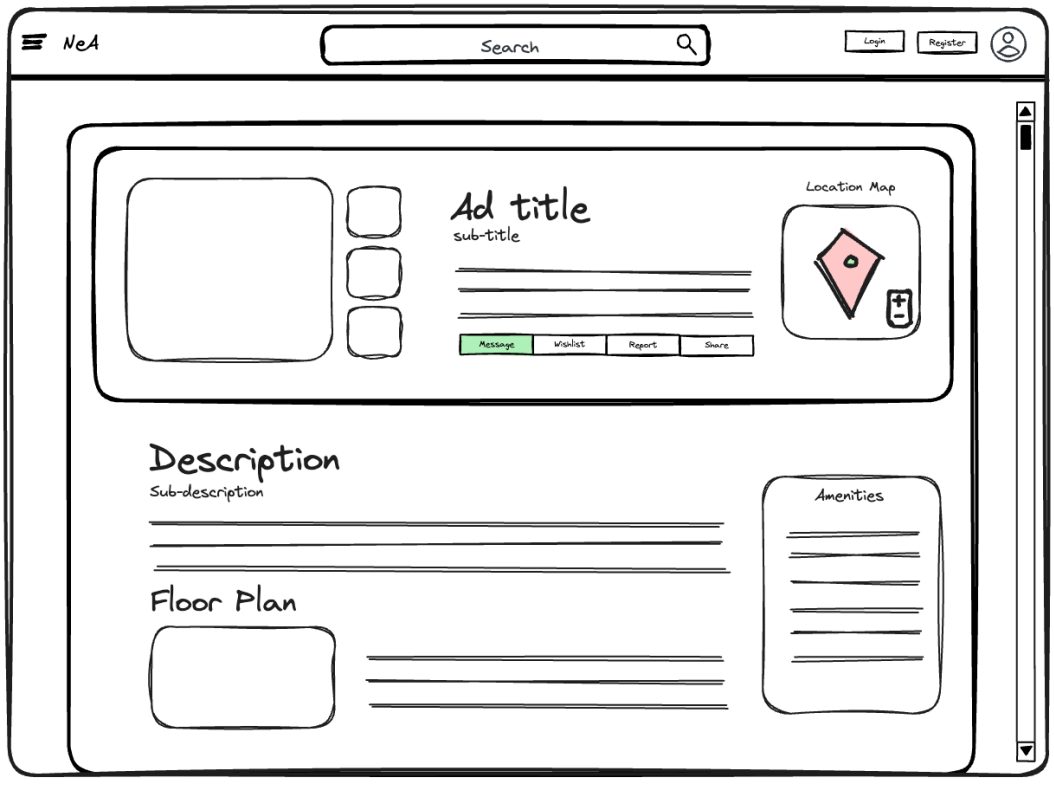


Fig 3.9: Property Details Page

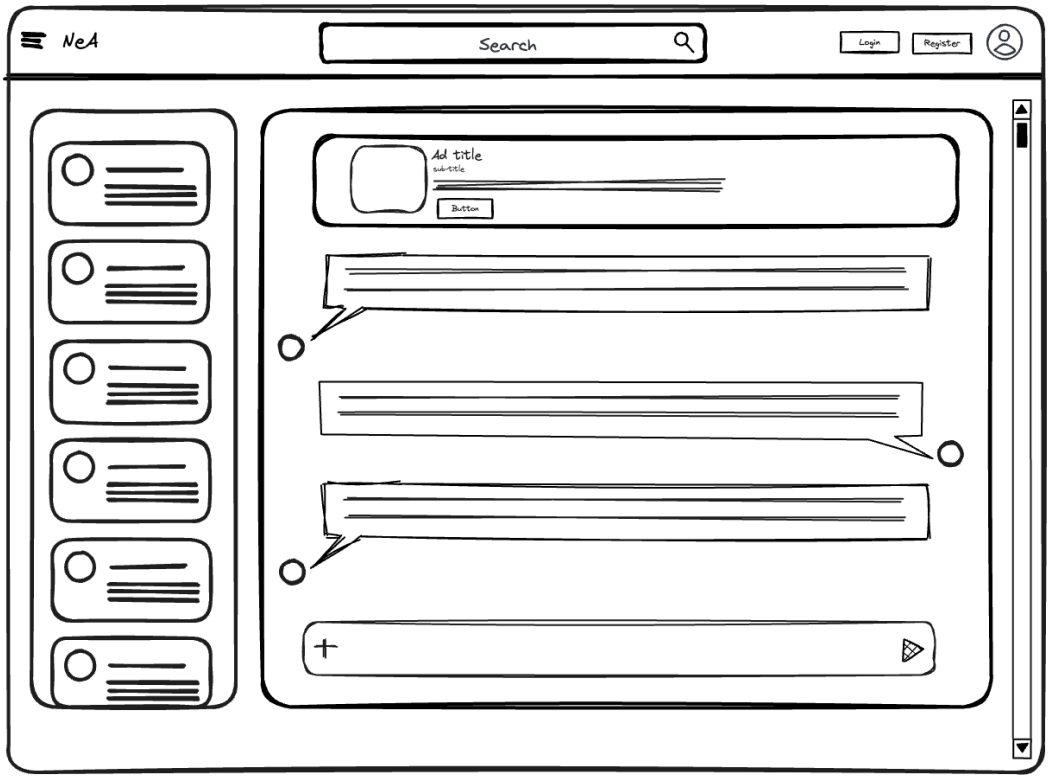


Fig 3.10: Real-time chat

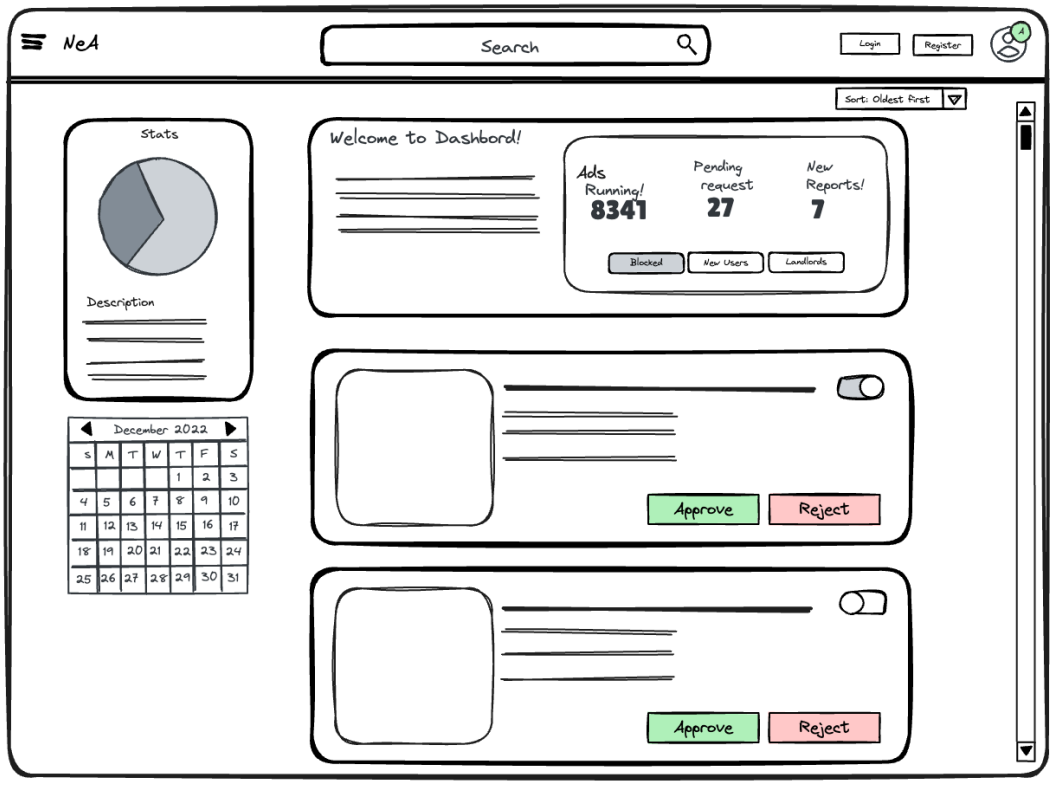
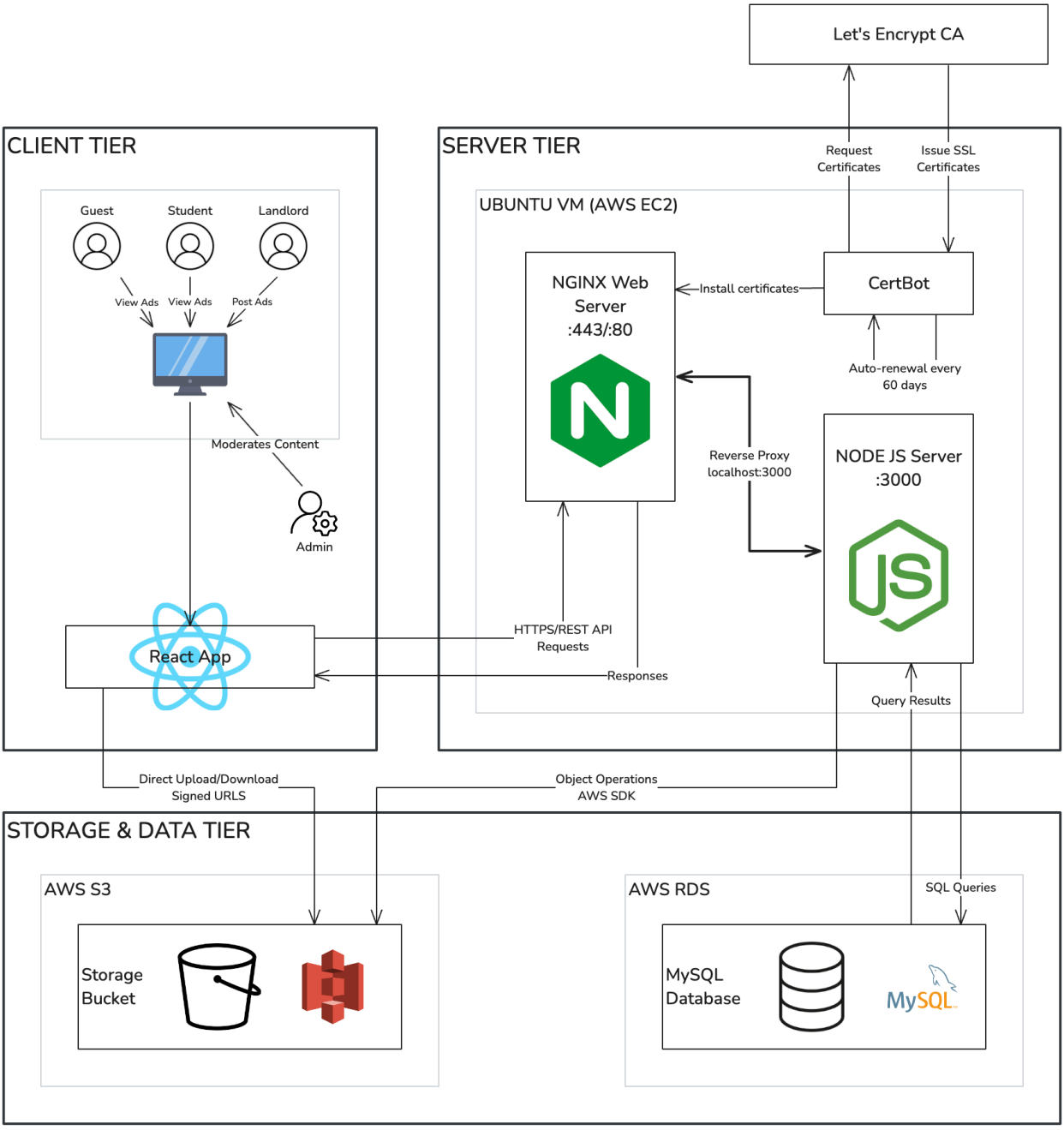


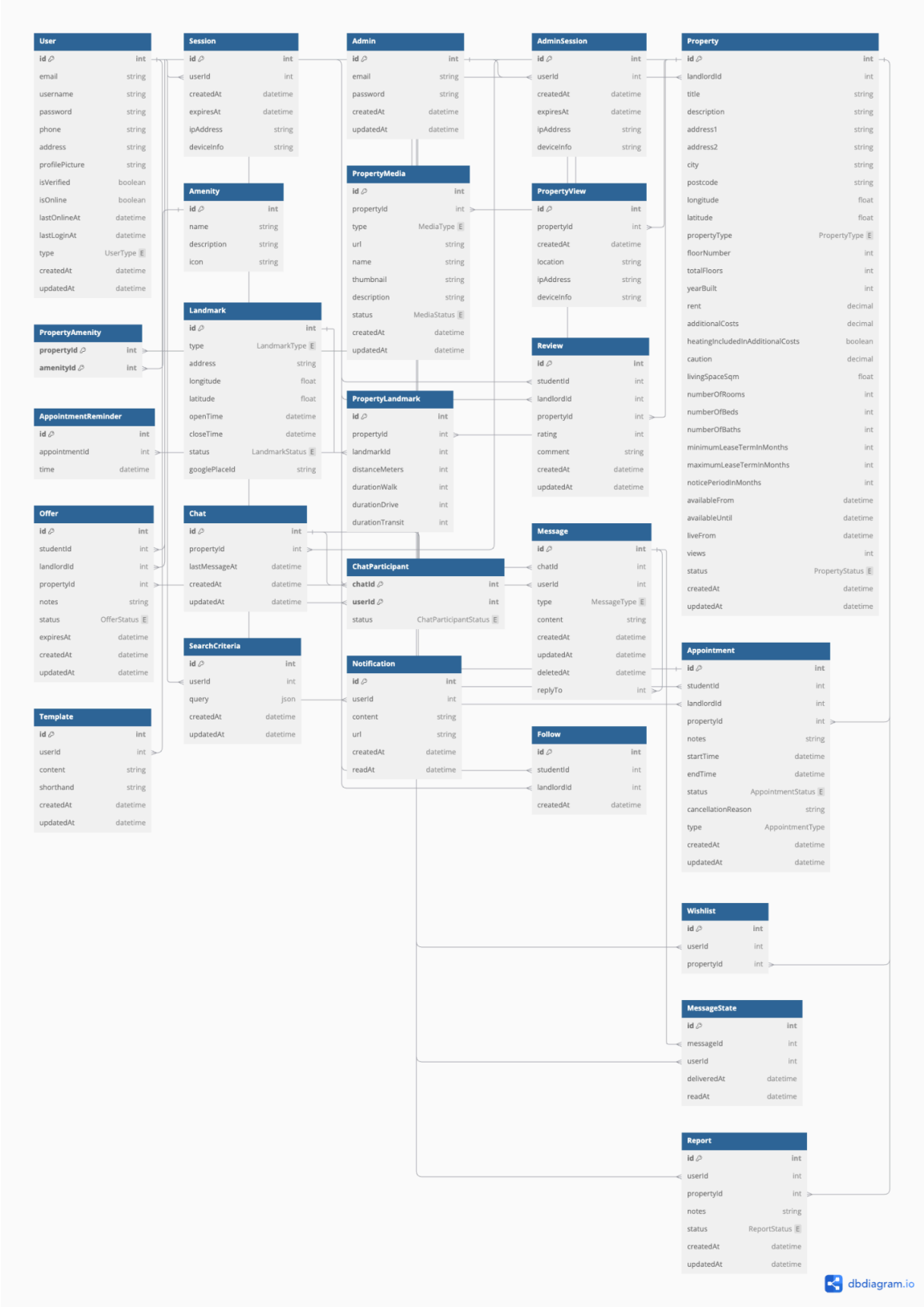
Fig 3.11: Admin Dashboard

4. High level Architecture, Database Organization:

High level Architecture:



Database Schema:



Following are the key entities defined in the above schema:

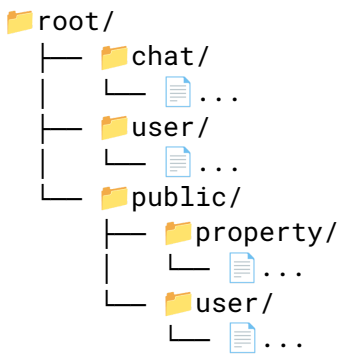
1. **Users & Authentication**
    a. User table is central with two types: STUDENT and LANDLORD
    b. Authentication handled through Session table
    c. Separate Admin and AdminSession tables for administrative access

2. **Property Management**
    a. Property is a core table linked to LANDLORD users
    b. Properties have detailed attributes (size, rent, location, availability)
    c. PropertyStatus tracks lifecycle (DRAFT → ACTIVE → RENTED etc.)
    d. Enhanced by:
        i. PropertyAmenity for features/facilities
        ii. PropertyMedia for photos/videos
        iii. PropertyView for tracking visits
        iv. PropertyLandmark for nearby points of interest and neighbourhood guide

3. **Interaction System**
    a. Appointment for apartment viewings
    b. Chat system with:
        i. ChatParticipant for user involvement
        ii. Message for communication
        iii. MessageState for read/delivery status
    c. Offer system for rental proposals
    d. Review for apartment/landlord ratings

4. **Supporting Features**
    a. Landmark for points of interest (schools, transport, etc.)
    b. Wishlist for saved properties
    c. SearchCriteria for saved searches
    d. Template for message templates
    e. Notification for user alerts
    f. Follow for student-landlord connections
    g. Report for issue reporting

Media Storage:
The application will use S3 to store all media such as images, files, and videos. The S3 Bucket is organized in the following structure:



The **root** folder of the S3 bucket contains private folders like **chat** and **user**. These folders store private files such as chat attachment or private user files. These files can only be accessed via a temporary signed URL that the system generates.

Within the root folder, there is a **public** folder. All content within this folder is public and accessible via long-lived URLs. Users can access content within this folder if they have a URL but not list all the contents of the folder. This folder stores public content like **property** images and **user** profile images.

The above structure is implemented by making the bucket private and adding a policy (like below) to make the public folder accessible:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::gdsd/public/*"
        }
    ]
}
```

Search Implementation:
For the vertical prototype search is implemented in a very simple manner by leveraging SQL queries only. We are using an ORM called prisma to manage all database queries so a typical query to search properties with rent between €400 - 800 looks like this:

```
const properties = await prisma.property.findMany({
  where: {
    totalRent :{
      gte: 400,
      lte: 800,
    }
  },
});
```

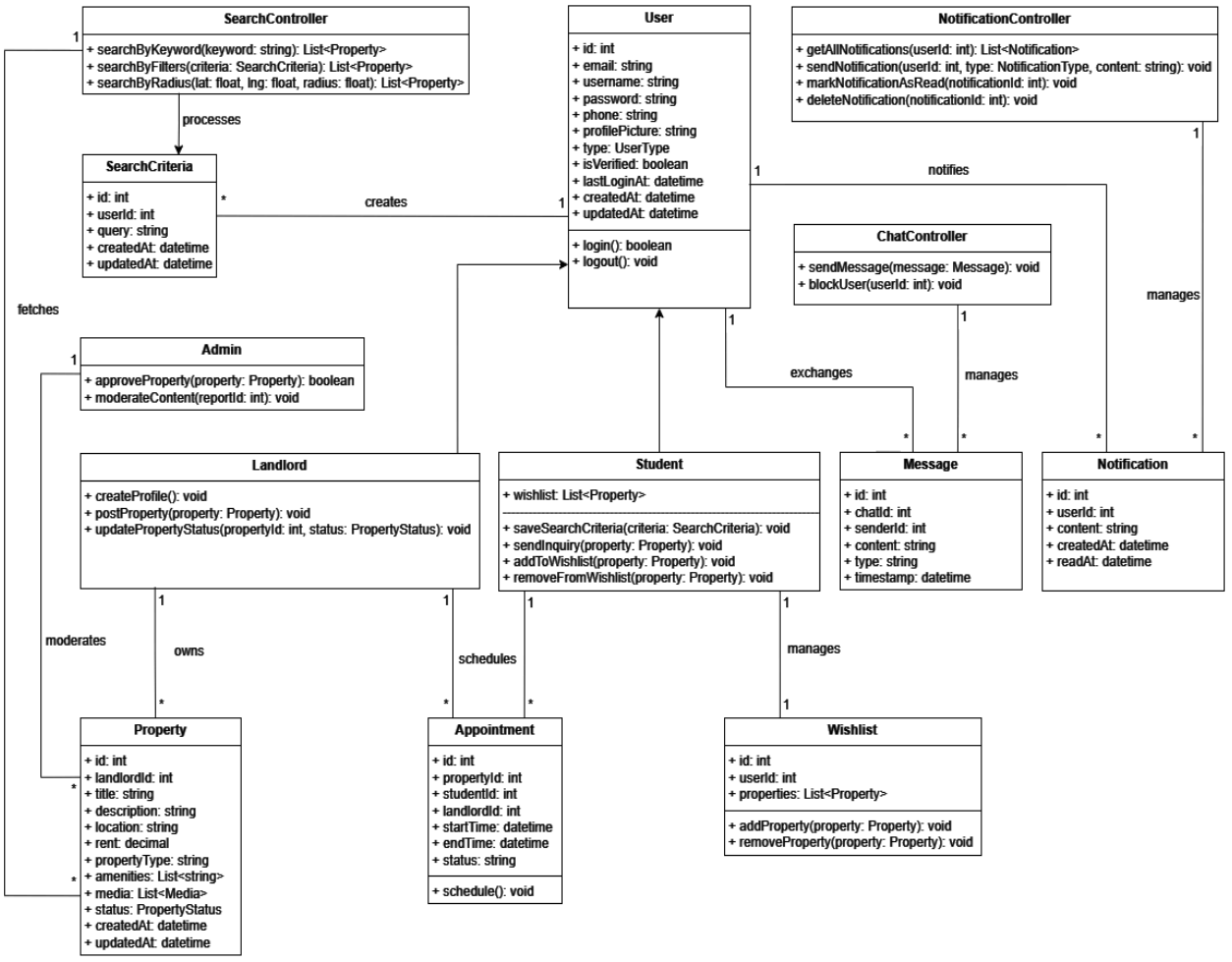
For the P2 implementation we want to develop a custom algorithm that is able to:

1. **Rank search results for users**:
   The algorithm will be able to display a feed that is tailored to each individual user. Each ad will receive a score based on the landlord rating, wishlist count, views, and user search criterion matching index (a score based on saved search filters). We will display the ads with the highest scores first in the results list.

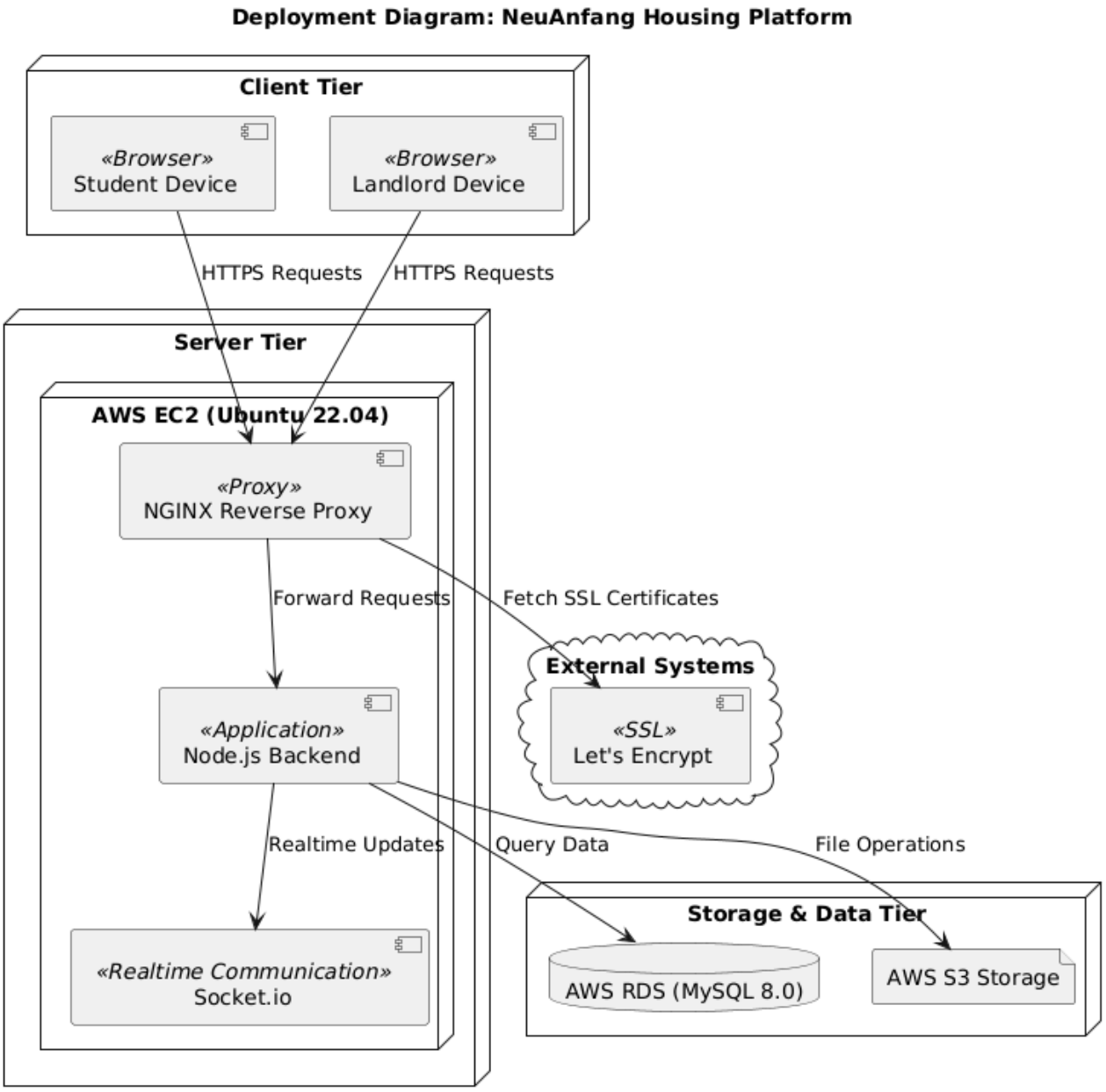
2. **Match students and landlords**:
   When a new ad is created, landlords receive a list of students based on the most recent saved search criteria. We can also obtain preferences through user interactions on the site; however, saved searches will serve as the primary source of data.

5. High Level UML Diagrams:

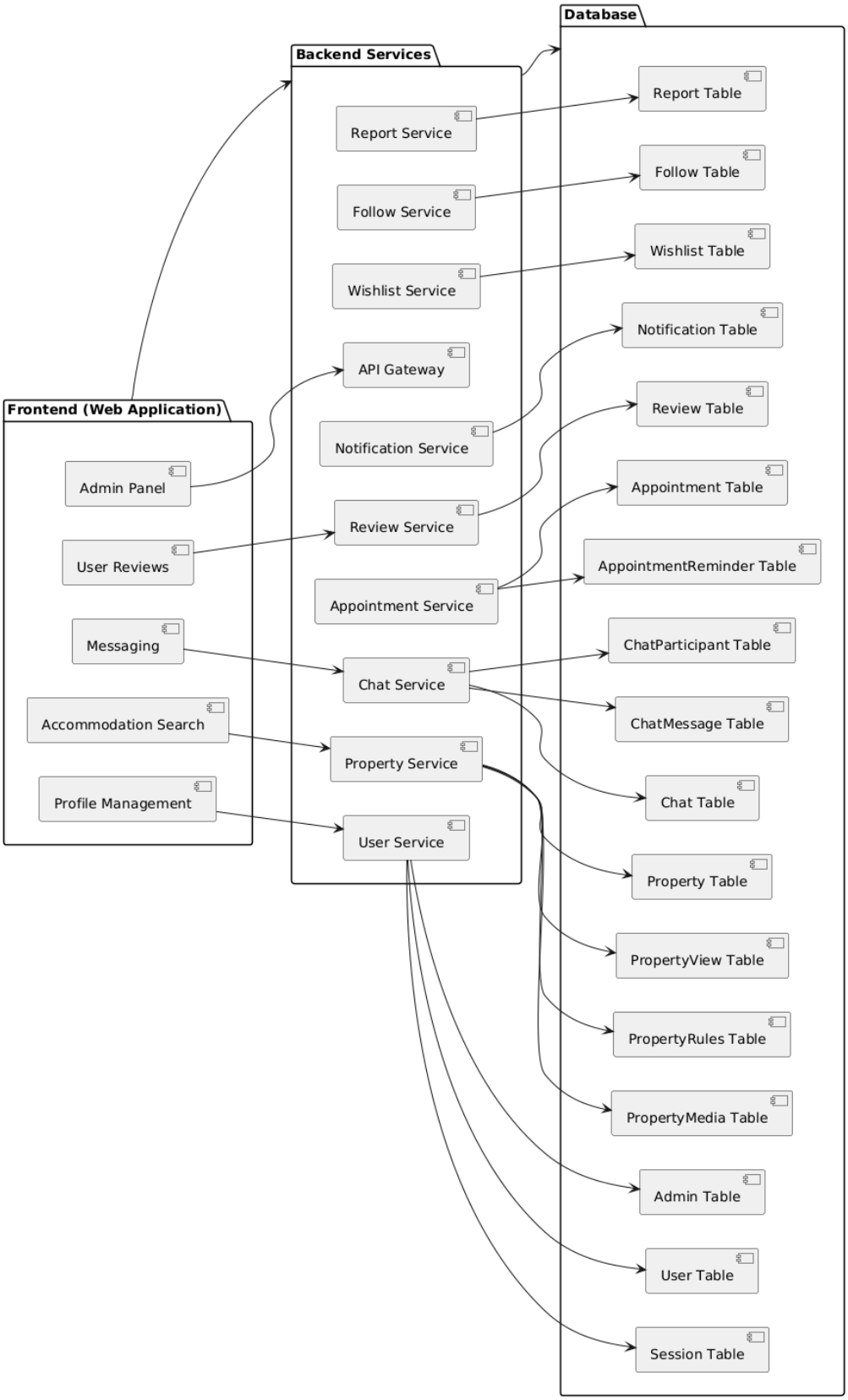
a) UML Class Diagram



b) UML Component and deployment diagram



Component Diagram: NeuAnfang Platform



6. Key risks for project at this time:

1. **Skills risks**
   Risk: Not all the team members are proficient with tools and tech stack required for the project.
   Plan for resolution:
   - Experienced members help others to catch up.
   - Team members dedicate time for self-learning during free hours.
   - Leads provide guidance and share expertise.

2. **Schedule Risks**
   Risk: There is a risk of missing deadlines due to resource constraints or unforeseen challenges.
   Plan for resolution:
   - Commit only to realistic goals.
   - Classify additional tasks as lower priority (P2) and handle them accordingly.
   - Allocate buffer time in the schedule to handle unforeseen challenges or changes.
   - Check status in the middle of the week to reassess timelines if needed.
   - Use project management tools for clear deadline visibility

3. **Availability Risks**
   Risk: A team member may become unavailable due to personal or health reasons.
   Plan for resolution:
   - Notify the team in advance to adjust plans.
   - Clear dependencies before the absence to minimise disruption.
   - Ensure key tasks are not dependent on a single individual
   - Maintain clear documentation for tasks to allow others to refer if needed.

4. **Technical Risks**
   Risk: Unforeseen technical issues, such as server failures or software dependencies, may arise.
   Plan for resolution:
   - Refer to documentation, Stackoverflow and other resources for troubleshooting.
   - Involving backend and frontend leads to resolving bottlenecks.
   - Communicate issues promptly and allocate buffer time for potential risks.

5. **Teamwork Risks**
   Risk: Communication issues, misunderstandings, or lack of coordination may disrupt teamwork.

Plan for resolution:

- Plan and discuss tasks thoroughly before starting work.
- Using tools like Trello to maintain transparency
- Ensure all members are on the same page and understand their roles.
- Foster respectful and open communication within the team.
- Encourage team members to take ownership of tasks and support one another when needed.

6. **Legal/Content Risks**

Risk: Use of images, data, or software in the project may violate copyright or licensing terms which can potentially lead to legal issues.

Plan for resolution:

- Verify that all images, data, and software used are either copyright-free or properly licensed.
- Use dummy data or publicly available datasets for testing and development.
- Regularly review copyright and licensing guidelines to ensure compliance.

7. Project Management:

How We Managed Tasks So Far:
For M2 and before, we utilized **Trello** for task management. Tasks were first listed in the backlog. Then we assigned them to team members based on their expertise and interests. Each task was categorized as *To Do*, *Doing*, or *Done* to track progress. We also created a *Resources* section in Trello, where team members could access key documentation and important information pertaining to the project.

For team communication:
We used **WhatsApp** for quick updates and **Google Meet** for online meetings.

Each week we had the following structure:

1. **Planning Meeting**: Held at the start of the week to discuss tasks and responsibilities. Preferred offline meetings but also online based on everyone's availability.
2. **Progress/Status Check**: Once a task was completed, team members would notify the group via WhatsApp. If required, we held quick midweek meetings (online/offline) to assess task progress and address any blockers.
3. **Readiness Check**: Held a day before the deadline to review if tasks were completed as required.

Moving forward, we will continue using the above discussed tools and methodologies we have practised so far for team communication and task management. Additionally, we will have clear documentation of individual programming tasks, handle critical tasks in the absence of team members, and define clear interfaces for frontend and backend communication.

Master Team Project Fall 2024
NeuAnfang
Team 3

Member Name	Role
Zubeena Shireen Sheikh zubeena-shireen.sheikh@informatik.hs-fulda.de	Team Lead and Document Master
Muhammad Zaid Akhter	Backend Lead
Masood Ahmed Mohiuddin	Frontend Lead
Humdaan Syed	GitHub Master
Andrii Kuripka	Team Member (Frontend)

Milestone 4
06/02/2025

Date Submitted	06/02/2025

Content and structure for Milestone4 document for review by institutors:

1. Product Summary
2. Usability test plan
3. QA test plan
4. Code review
5. Self-check on best practices for security
6. Self-check: Adherence to original Non-functional specs

1. Product Summary

Product Name: NeuAnfang

List of committed functions:

- NeuAnfang facilitates Fulda University students to find a place they can call home in their new beginnings in Fulda.
- **Apartment Posting and Pricing:** Landlords can add detailed information about their properties.
- **Share Properties:** Ads are shareable via links. This helps landlords to increase footprint by sharing on other social media platforms.
- **Ad Status Management:** Admins can mark ads as draft, pending, active, approved, or rejected.
- **List View of Apartments:** The list view is sortable by filters and displays a summary of key information.
- **Map View of Apartments:** An interactive map shows property markers, allowing students to zoom in on specific neighbourhoods.
- **Detailed Apartment View:** The detailed view includes images and an expanded list of amenities.
- **Real-Time Chat:** Real-time chat for students and landlords to communicate with each other. It features notifications for unread messages.
- **Ad Approval by Admin:** Ads go live only after site admin reviews and approves each ad.
- **Admin Comments:** Admins can add comments to communicate with landlords to resolve issues.
- **Analytics dashboard:** Ad creators can view and analyze their Ad reception with a dashboard with various KPIs like created, active, pending ads.
- **Advanced Filtering Options:** Filters include advanced criteria and users can combine multiple filters for complex searches.
- **Student Subletting:** Students can post Sublet ads which require approval by Admin.
- **Student Account Verification:** Fulda University students can register with verification of their university email address.
- **Ad Search Feature:** Users can search for properties using natural language queries.
- **Wishlist Apartments:** Students can add apartment properties to wishlists.
- URL to page: NeuAnfang

2. Usability test plan

Usability Task Description: Search
Participants will be given the following instructions for the usability test:

- **Task 1:** Imagine you are a student looking for a place to stay in Fulda. Use the search functionality on the platform to find properties that match your preferences (e.g., price range, number of rooms).
- **Task 2:** Apply relevant filters such as budget, and amenities to refine your search results.
- **Task 3:** Look at properties of interest and review their details, noting any difficulties or missing information.

Effectiveness Measurement:
Effectiveness will be measured by the user's ability to successfully search for a property, apply filters, view property details, and contact the property owner without encountering major issues.

Efficiency Measurement:
Efficiency will be measured by the time taken to complete each task and the steps required to perform each action within the property search functionality.

Likert Scale Questions for User Satisfaction (Property Search Functionality)
Participants will use a Likert scale to assess their satisfaction with the property search functionality after completing the usability tasks. They will rate their experience based on the following criteria:

Likert Scale rating description:

1	2	3	4	5
Very Poor	Poor	Neutral	Good	Very Good

Management Student - 1

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** The property search functionality was smooth and a seamless experience.

Q2: How easy was it to search for properties and apply filters? **A2:** Searching for properties and applying filters was extremely easy.

Q3: Did you encounter any issues during the property search process? - **A3:** No issues encountered; the platform worked flawlessly.

Q4: How relevant were the search results based on your input? - **A4:** The search results were highly relevant and accurately matched my criteria.

Q5: How was your overall experience using the platform for property search? - **A5:** Overall, the platform offers a user-friendly and efficient property search experience.

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	5/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	5/5
Inquiry Process	5/5

Management Student - 2

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Good

Q2: How easy was it to search for properties and apply filters? - **A2:** Satisfied

Q3: Did you encounter any issues during the property search process? - **A3:** No

Q4: How relevant were the search results based on your input? - **A4:** Perfect

Q5: How was your overall experience using the platform for property search? - **A5:** So good

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5
Ease of Use	5/5
Search Effectiveness	4/5
Information Clarity	5/5
Inquiry Process	3/5

News Reporter - 3

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Great Simple UI and I like it

Q2: How easy was it to search for properties and apply filters? - **A2:** Easy, Filter button and share is easily findable

Q3: Did you encounter any issues during the property search process? - **A3:** No, not an issue. Smooth process

Q4: How relevant were the search results based on your input? - **A4:** I found what I was looking for.

Q5: How was your overall experience using the platform for property search? - **A5:** Good overall.

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5

Ease of Use	5/5
Search Effectiveness	4/5
Information Clarity	5/5
Inquiry Process	4/5

German Landlord - 4

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Good

Q2: How easy was it to search for properties and apply filters? - **A2:** Fairly easy

Q3: Did you encounter any issues during the property search process? - **A3:** No

Q4: How relevant were the search results based on your input? - **A4:** Relevant

Q5: How was your overall experience using the platform for property search? - **A5:** Good

Likert Scale Questions Responses:

Criteria	**Response**
Overall Experience	4/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	3/5
Inquiry Process	3/5

Warehouse worker - 5

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** It was okay

Q2: How easy was it to search for properties and apply filters? - **A2:** Easy

Q3: Did you encounter any issues during the property search process? - **A3:** Not during search, but didn't know I needed to register to initiate contact with ad owners.

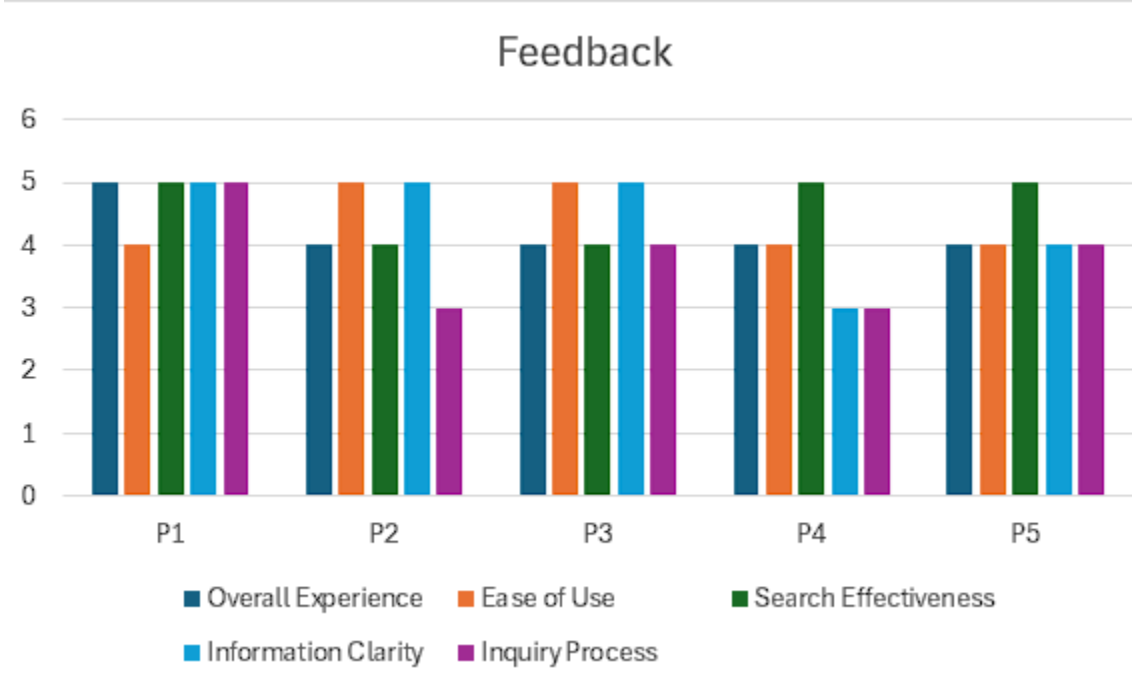
Q4: How relevant were the search results based on your input? - **A4:** Relevant, entered apartment type and matching results appeared

Q5: How was your overall experience using the platform for property search? - **A5:** Good

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	4/5
Inquiry Process	4/5

Feedback Chart:



3. QA test plan

Test objectives: The primary objective of this test is to ensure that the "Search" functionality on the homepage of the website operates as designed. The testing will be conducted across different hardware and browsers. Testers do not need to log in to perform this test.

HW and SW setup:

- **Hardware Setup:**
  - Windows laptops
  - Mac laptops
- **Software Setup:**
  - Latest version of Chrome browser for the Windows environment
  - Latest version of Safari browser for macOS
- **URL:** https://gdsdteam3.live/

Feature to be tested: Search functionality is tested for different sorts of user inputs which is a query text and applied filters. Initially the test is carried out for a user input text which does not match with any listings of the database. Then with a user input text which is present in the database. Next the validation of the 40 alphanumeric character limit of search text is tested. Finally, a combination of user input text and filters are verified.

Below table lists the details of tests to be carried out by the tester.

Test on Chrome

Test #	Test title	Test description	Test input	Expected output	Test results (Pass/ Fail)
1	Search with no matches	No match should result all the listings from the database	Enter a query that does not exist in the database. Eg. A string “xxxx”	Displays all the listings from the database	Pass

2	Search with matching results	Entering a query text should result in listings which include the query string in their titles	A string “2 room”	Displays listings relevant to the query	Pass
3	Search with filters	Search query by title along with filters should return results matching the filters	Enter a string “2 room”, Set the maximum price to 1600, Select filters - pets and smoking	Displays listings relevant to the query and filters	Pass
4	Price Filter	Search with both minimum and maximum price values	Enter “2200” in minimum price and “1200” in maximum price	Display error to not allow maximum price higher than minimum price	Fail
5	Character limit validation	Entering more than 40 characters should inform the user of character limit	Enter 41 “a” characters in the title field	Displays an input error	Pass

Test on Safari

Test #	**Test title**	**Test description**	**Test input**	**Expected output**	**Test results (Pass/ Fail)**
1	Search with no matches	No match should result all the listings from the database	Enter a query that does not exist in the database. Eg. A string “xxxx”	Displays all the listings from the database	Pass

2	Search with matching results	Entering a query text should result in listings which include the query string in their titles	A string “2 room”	Displays listings relevant to the query	Pass
3	Search with filters	Search query by title along with filters should return results matching the filters	Enter a string “2 room”, Set the maximum price to 1600, Select filters - pets and smoking	Displays listings relevant to the query and filters	Pass
4	Price Filter	Search with both minimum and maximum price values	Enter “2200” in minimum price and “1200” in maximum price	Display error to not allow maximum price higher than minimum price	Fail
5	Character limit validation	Entering more than 40 characters should inform the user of character limit	Enter 41 “a” characters in the title field	Displays an input error	Pass

4) Code Review:

Review 1

Code Writer: Muhammad Zaid Akhter
Reviewer: Humdaan Syed
Date: 05/02/2025
File reviewed: filters-section.jsx

Overview:
The filters-section.jsx file handles the property search filters for the home page. It allows users to filter properties based on title, price range, availability date, amenities), and search radius. It includes both desktop and mobile responsiveness using Mantine components. The form is managed using Mantine's useForm hook, and search parameters are updated via useSearchParams from React Router.

Strengths of the code reviewed:

1. Responsive Design: The component works well on both desktop and mobile devices. It uses Drawer for mobile filters and Paper for desktop filters, making it user-friendly on all screen sizes.

2. Good Use of Mantine Components: It uses Mantine UI elements like Stack, Title, TextInput, Select, and Checkbox, which help keep the design consistent and accessible without extra effort.

3. Dynamic URL Filtering: Filters update the URL using useSearchParams. This is helpful because users can bookmark, share filtered searches, and still see their filters when refreshing the page.

4. Separation of Concerns: The filter form's logic (Filters) is separated from the layout parts (DesktopFilters and MobileFilters). This makes the code easier to manage and update.

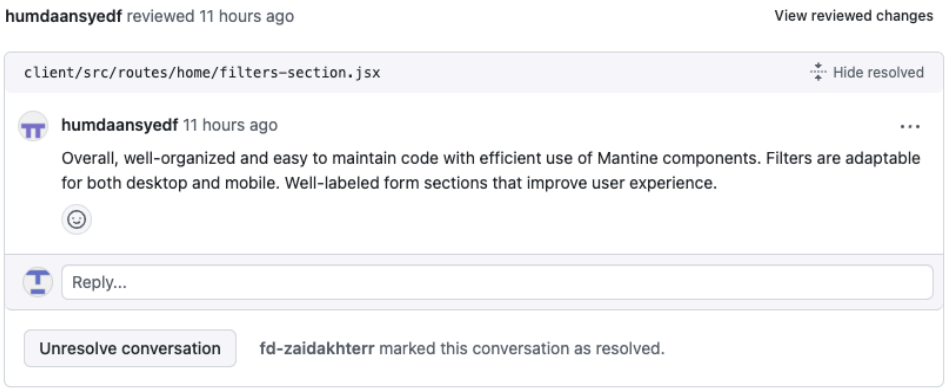
5. Reset Functionality: A simple Reset button allows users to clear all filters at once, improving the overall user experience.

Issues (if any):

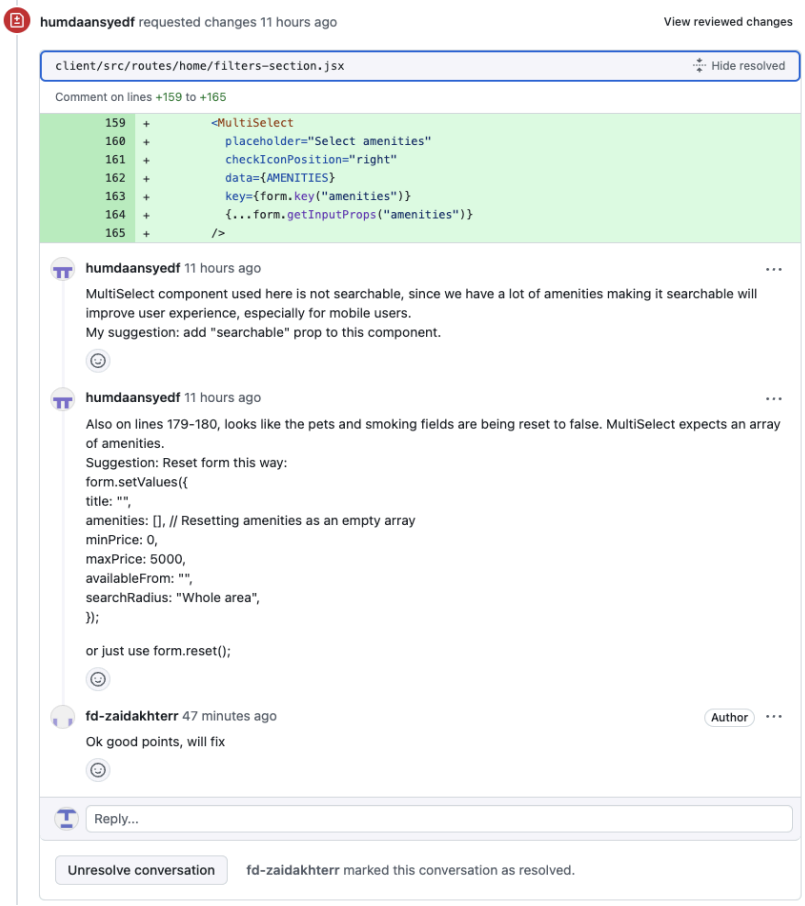
Issue	Code lines in file	Screenshot
Non-Searchable MultiSelect Component	The MultiSelect component isn't searchable, which can be inconvenient when there are many amenities.	Screenshot 1

Incorrect Reset for Amenities	The pets and smoking fields are being reset to false, but MultiSelect expects an array (since amenities are handled as an array)	Screenshot 2

Github Screenshots:



Screenshot 1



Screenshot 2

Review 2

Code Writer: Humdaan Syed
Reviewer: Andrii Kuripka
Date: 05/02/2025
File reviewed: Wishlist and Map view

Overview:
The pull request introduces changes in multiple components, including updates to the map-view.jsx, enhancements to the wishlist.js for authorization handling, and fallback image handling for properties. The changes aim to improve user interaction on the map, address potential authorization inefficiencies, and ensure better error handling.

Strengths of the code reviewed:

1. Clear User Interaction Design: The "View Details" button in map-view.jsx allows seamless user interaction with property details. The use of Button and Paper ensures a clean and consistent UI.

2. Authorization Logic: The authorization checks in wishlist.js demonstrate awareness of secure API call handling, ensuring that unauthorized users do not flood the API with unnecessary requests.

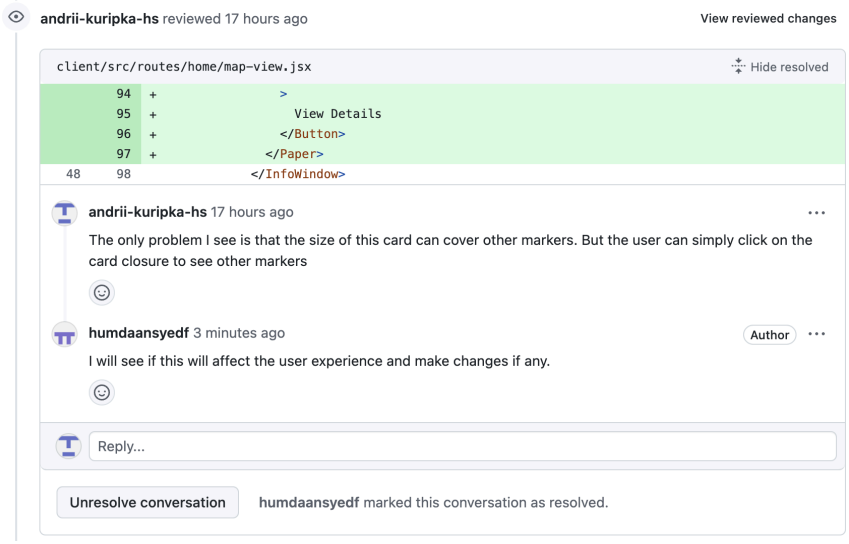
3. Fallback Image Handling: A fallback image has been added in map-view.jsx, preventing broken UI when property images are missing.

4. Responsive Improvements: The components reviewed, such as map-view.jsx, display responsiveness to varying scenarios.

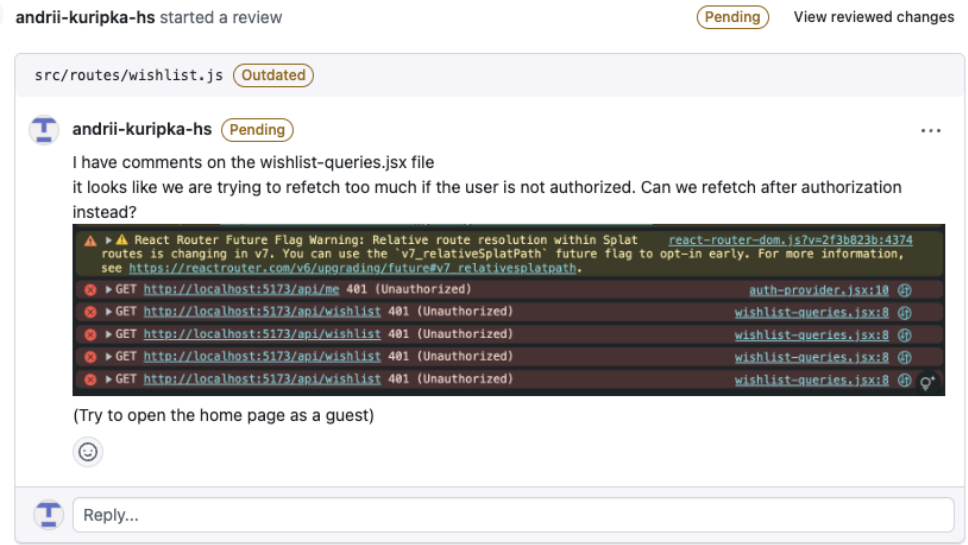
Issues (if any):

Issue	Feedback	Screenshot
Marker Obstruction by InfoWindows	The addition of property cards can obstruct other markers on the map, potentially frustrating users. While users can close the card, this could negatively impact user experience if multiple markers are clustered.	Screenshot 3
Refetching on Unauthorized Requests	Unauthorized users trigger excessive API refetches, which is inefficient and may lead to server strain.	Screenshot 4

Github Screenshots:



Screenshot 3



Screenshot 4

Review 3

Code Writer: Andrii Kuripka
Reviewer: Zubeena Shireen Sheikh
Date: 30/01/2025
File reviewed: Edit Property Page

Overview:
This pull request introduces multiple updates across the project, including:

- Preloading existing images in the ImageUploader component.
- Fixing image updates in the backend creator.js.
- Resolving a 500 error in edit-ad-page.jsx due to incorrect date formatting.

These changes aim to enhance the user experience by fixing bugs and improving functionality, particularly in property management and editing.

Strengths of the code reviewed:

1. Preloading Existing Images: In the ImageUploader component, the ability to load and display existing images from the database provides a smoother user experience for updating property listings.

2. Bug Fix for Date Format: The fix for the date formatting issue in edit-ad-page.jsx correctly uses ISO string formatting to prevent server-side 500 errors. The use of value.toISOString().substring(0, 10) ensures that only valid date strings are sent to the backend.

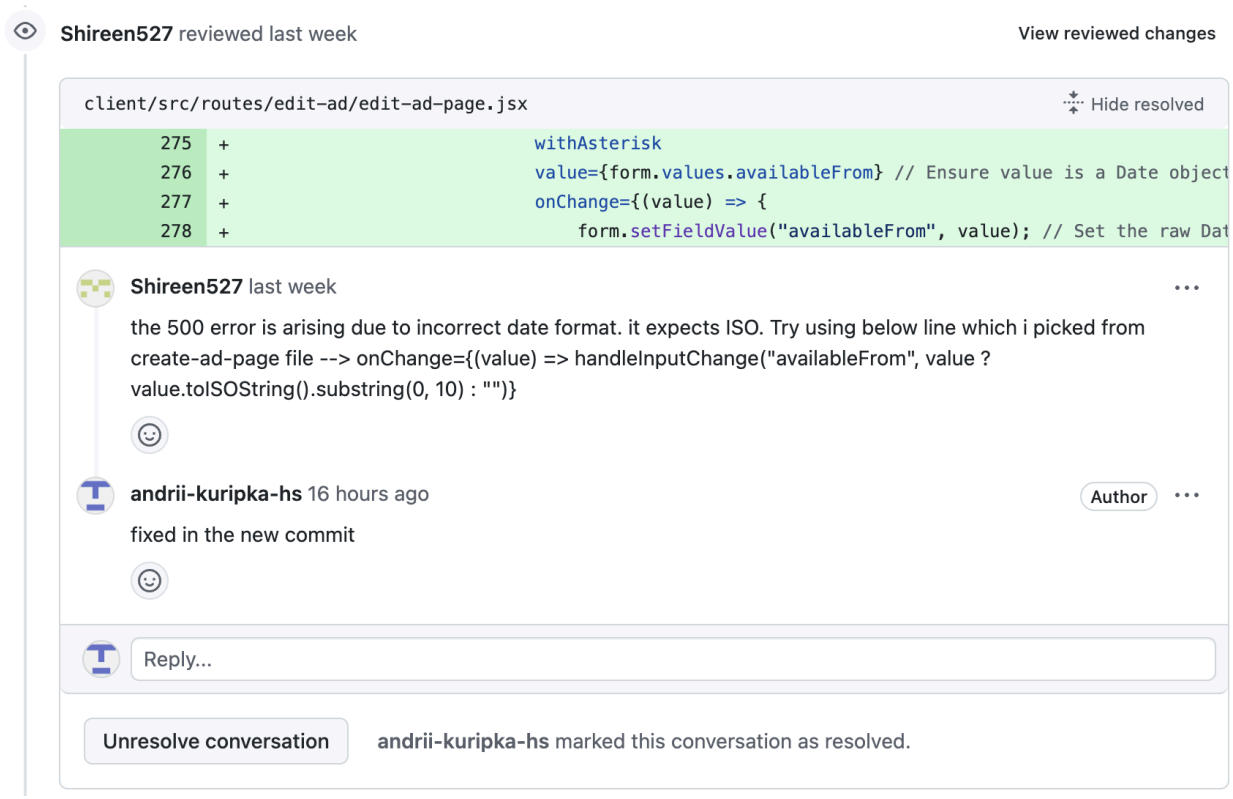
3. Effort to Update Images: Address image updates in the creator.js backend route. This shows a focus on ensuring backend and frontend synchronization for image uploads.

Issues (if any):

Issue	Feedback	Screenshot
Date Formatting in edit-ad-page.jsx	The fix addresses the immediate 500 error. However, the logic assumes value is always a valid Date object. Edge cases, such as null or invalid dates, might still cause issues.	Screenshot 5
Missing Logic for Image Updates	While the property update works, the lack of image update logic means users cannot modify the images associated	Feedback during 1-1 session

	with a property. This can lead to incomplete or outdated property listings.	

Github Screenshots:



Screenshot 5

Review 4

Code Writer: Zubeena Shireen Sheikh
Reviewer: Masood Ahmed Mohiuddin
Date: 05/02/2025
PR reviewed: Chat: refactoring, notifications

Overview:
The work on chat refactoring, and notifications handles socket events related to chat functionality, including joining rooms, sending messages, fetching chat history, and retrieving user interactions. It integrates Prisma for database operations and ensures real-time updates using Socket.IO. Additionally, add notification logic for chat interactions.

Strengths of the code reviewed:

1. Responsive Design: The component works well on both desktop and mobile devices. It uses Drawer for mobile filters and Paper for desktop filters, making it user-friendly on all screen sizes.

2. Good Use of Mantine Components: It uses Mantine UI elements like Stack, Title, TextInput, Select, and Checkbox, which help keep the design consistent and accessible without extra effort.

3. Database Interaction with Prisma: Queries are structured well with Prisma for fetching, creating, and updating chat and message records.

4. Good Real-time Messaging Implementation: The chat system efficiently uses socket.join(chatId) and socket.broadcast.to(chatId).emit() to handle user interactions in real-time. Notifications are triggered as expected.

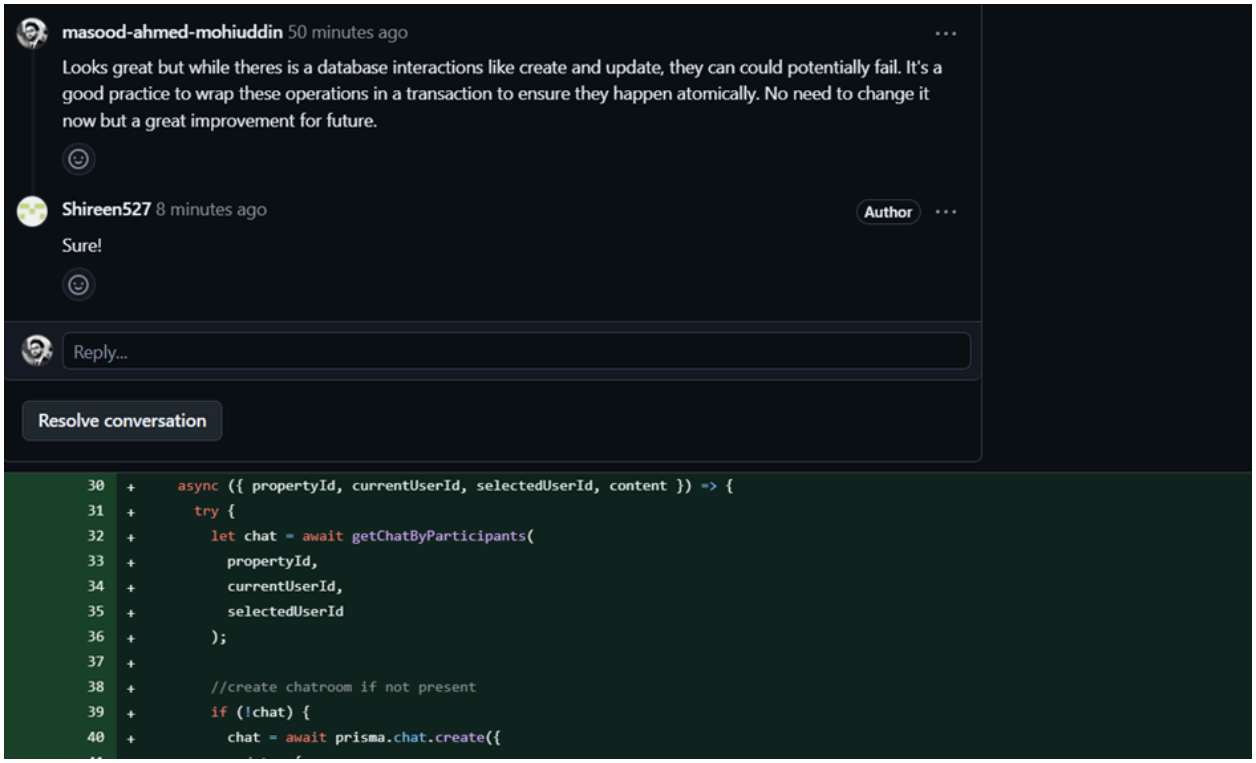
5. Good Use of React Query for Fetching Users: useQuery is implemented well to fetch chat users efficiently, improving API call structure.

6. Code Readability & Maintainability: Proper structuring and indentation.
   Clear separation of concerns for handling different socket events.

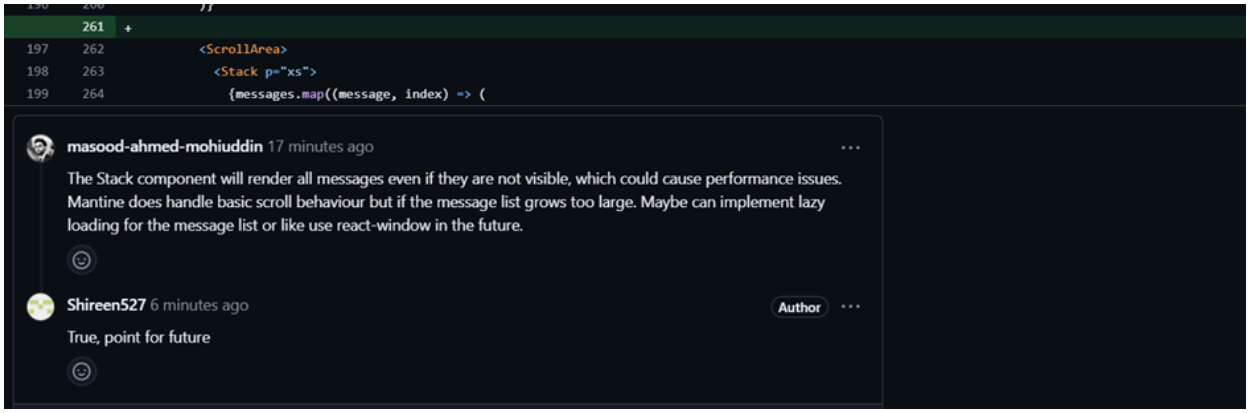
Issues (if any):

Issue	Feedback	Screenshot
Database Transactions. **Create** and **update** operations can fail, leading to partial state inconsistency.	Wrap database operations inside Prisma transactions	Screenshot 6
Performance Concern in Message Rendering. The Stack component renders all messages, even if not visible, causing potential UI lag.	Implement lazy loading or using libraries for better performance.	Screenshot 7
Fetching All Messages at Once. The query fetches all messages in a chat, which may slow down large conversations.	Implement pagination with skip and take parameters for fetching messages in chunks.	Screenshot 8
Queries like useChatUsers fetch all user data without caching.	Add staleTime or cacheTime options in useQuery to prevent unnecessary refetching.	Screenshot 9

Github Screenshots:



Screenshot 6



Screenshot 7

```
28 +       //get all messages in the chatroom
29 +       const messages = await prisma.message.findMany({
30 +         where: { chatid: chat.id },
31 +         orderBy: { createdAt: "asc" },
32 +       });
33 +
```

masood-ahmed-mohiuddin 12 minutes ago

Instead of fetching all messages at once, you can limit the number of messages and maybe add skip and take parameters for pagination throughtout for fetching, unread msgs and responses.

Shireen527 4 minutes ago Author

yes, this is good idea

Screenshot 8

```
1  + import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
2  +
3  + // Fetch users current user has chatted with
4  + export const useChatUsers = (currentUserId) => {
5  +   return useQuery({
6  +     queryKey: ["chatUsers", currentUserId],
7  +     queryFn: async () => {
8  +       if (!currentUserId) return [];
9  +       const response = await fetch(
10 +         `/api/chats/users?currentUserId=${currentUserId}`
11 +       );
12 +
13 +       if (!response.ok) {
14 +         throw new Error("Failed to fetch chat users");
15 +       }
16 +       return response.json();
```

masood-ahmed-mohiuddin 33 minutes ago

Clean code but If you have certain queries that fetch data that doesn't change frequently like here we have useChatUsers or useUnreadMessages you could add a staleTime or cacheTime option to control how long the data is cached before it is refetched. This can help reduce unnecessary network requests. Like cacheTime: 1000 * 60 * 10 which keeps cached data for 10 minutes. No necessay to do now but can do later.

Shireen527 6 minutes ago Author

yea, will note

Screenshot 9

Review 5

Code Writer: Masood Ahmed Mohiuddin
Reviewer: Muhammad Zaid Akhter
Date: 28/01/2025
File reviewed: Student document upload + UI changes

Overview:
This PR introduces several improvements and optimizations for handling document-related functionality, including:

- Fixes to prevent generating signed URLs for all documents on fetch.
- Implementation of a new "View" button in the UI that requests signed URLs dynamically for individual documents.
- Adjustments to the CORS setup to differentiate between development and production environments.
- Refactoring of backend logic to improve efficiency and reduce unnecessary S3 requests.

While the functionality is solid, there is room for further refinement, especially in aligning with existing project patterns (tanstack-query) and improving code reusability. Once these changes are addressed, the PR will significantly improve performance and maintainability.

Strengths of the code reviewed:

1. Signed URL Optimization: Eliminating the generation of signed URLs during the fetch of all documents reduces unnecessary overhead and S3 usage.

2. Separation of Concerns: The updated logic adheres to best practices by separating document metadata fetching and URL generation. This ensures the backend is more efficient and avoids overloading the frontend with redundant data.

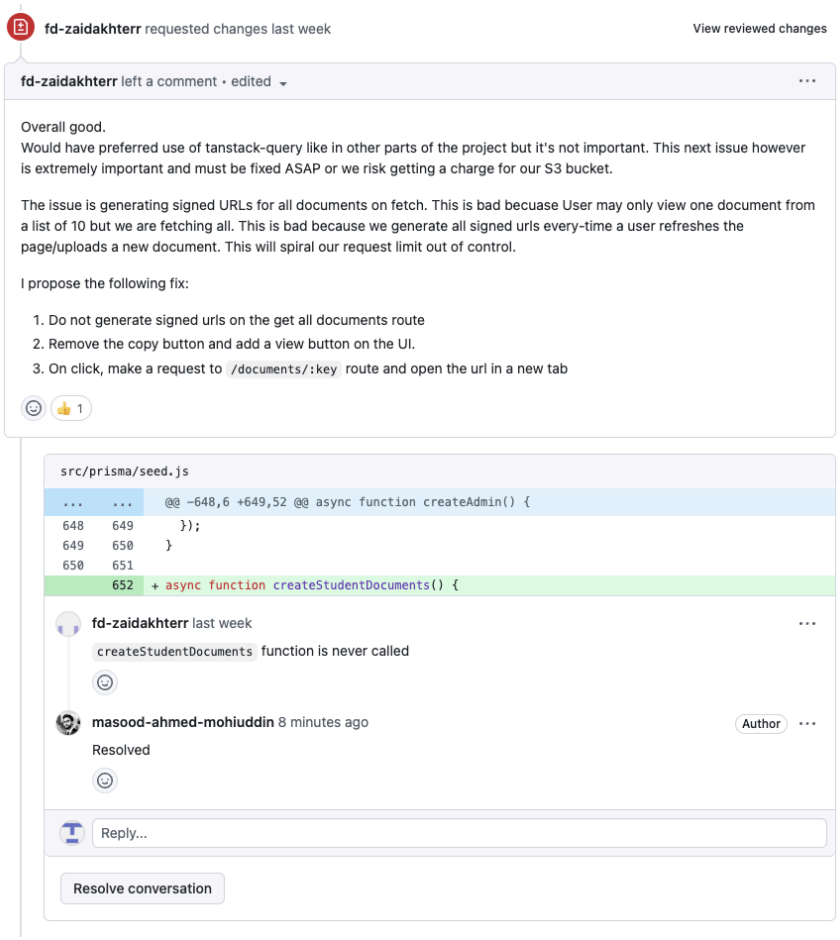
3. Improved CORS Setup: Adding an environment-based configuration for CORS prevents potential issues in production environments, ensuring secure and controlled API access.

4. Dynamic UI Enhancements: The "View" button now fetches signed URLs dynamically, addressing the limitation of short-lived signed URLs and enhancing user experience.

Issues (if any):

Issue	Feedback	Screenshot
Never called Function	The createStudentDocuments function was added but never called, making it redundant in the current implementation.	Screenshot 10
“Copy” Button Replacement	A better approach was suggested: replace the “Copy” button with a “View” button that makes a call to the /documents/:key route to fetch a signed URL dynamically and open it in a new tab.	Screenshot 11
Backend Signed URL Optimization	It was recommended to send document metadata only (without signed URLs) and generate the signed URL dynamically when requested via the /documents/:key route.	Screenshot 12

Github Screenshots:



Screenshot 10

client/src/routes/my-documents/my-documents.jsx

```
79 +                <td style={{ textAlign: "right", padding: "12px" }}>
80 +                  <Flex justify="flex-end" gap="lg">
81 +                    {/* Share Button */}
82 +                    <Tooltip label="Copy link">
```

fd-zaidakhterr last week

A copy button doesn't make sense for private urls or even short lived ones. We should remove it.

fd-zaidakhterr last week

A better approach would be to have a view button, this button would make a call to the backend `/documents/:key` route and get the signed url, then display it in a new tab

masood-ahmed-mohiuddin 7 minutes ago Author

Made changes accordingly

Reply...

Resolve conversation

Screenshot 11

src/routes/doc.js Outdated

Comment on lines 117 to 143

```
117 + documentRouter.get("/documents", async (req, res) => {
118 +   const userId = req.user.id;
119 +   if (req.user.type !== "STUDENT") {
120 +     return res.status(403).json({ message: "Access denied. Only students can access this." });
121 +   }
122 +
123 +   try {
124 +     const documents = await prisma.document.findMany({ where: { userId } });
125 +     const signedDocs = await Promise.all(
126 +       documents.map(async (doc) => {
127 +         const url = await getSignedUrl(
128 +           s3Client,
129 +           new GetObjectCommand({
130 +             Bucket: process.env.APP_AWS_BUCKET_NAME,
131 +             Key: doc.key,
132 +           }),
133 +           { expiresIn: 60 * 5 }
134 +         );
135 +         return { ...doc, url };
136 +       })
137 +     );
138 +     res.json({ documents: signedDocs });
139 +   } catch (error) {
140 +     console.error("Error fetching documents:", error);
141 +     res.status(500).json({ message: "Failed to retrieve documents" });
142 +   }
143 + });
```

fd-zaidakhterr last week

It would be better to not generate signed urls here and just send the docs with keys only. Then wen user click download/view on the UI use the `/documents/:key` route to get the docuement. Following reasons:

1. User may only view one document from a list of 10 but we are fetching all. This is wastefull specially consediring the already tight monthly limits on s3 requests
2. We generate all signed urls every-time a user refreshes the page/uploads a new document. Will spiral our request limit out of control. Fix at the soonest
3. The signed url is only valid for 5 minutes. User may stay on the screen for more than 5 minutes after which the url is basically useless

masood-ahmed-mohiuddin 7 minutes ago Author

Fixed

Reply...

Resolve conversation

Screenshot 12

5) Self-check on best practices for security

Major Assets and Their Protection:

1. Database (AWS RDS):
   - Threats:
     - Unauthorized access by external attackers
     - SQL injection attacks
   - Protection:
     - The secure database is hosted on RDS with public access blocked.
     - Access is strictly limited to the designated EC2 instance through a dedicated VPC.
     - Passwords are hashed using @node-rs/argon2
2. Object Storage (AWS S3):
   - Threats:
     - Exposing sensitive private student data due to misconfigured policies
     - Unauthorized data retrieval
   - Protection:
     - S3 is configured with a policy that allows public read-only access strictly for designated public content while keeping private user files inaccessible via public endpoints.
     - Access is managed and audited through AWS IAM roles and bucket policies.
3. Application Server and API Layer:
   - Threats:
     - Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF)
     - Injection attacks and unauthorized data exposure
   - Protection:
     - Robust input validation on the frontend using mantine forms
     - Strict CORs policies
     - Robust validation on the backend using zod schemas
     - Escaping user input via prisma
     - SSL Certificates configured on the server using Let's Encrypt

Password Encryption:

Passwords stored in the DB are encrypted using secure hashing algorithms (@node-rs/argon2) ensuring that even if the database is compromised, plaintext passwords are not retrievable.

Input Validation:

All inputs, including the search bar input (expected to be up to 40 alphanumeric characters), are validated rigorously. See example code for registration form validation using mantine forms below:

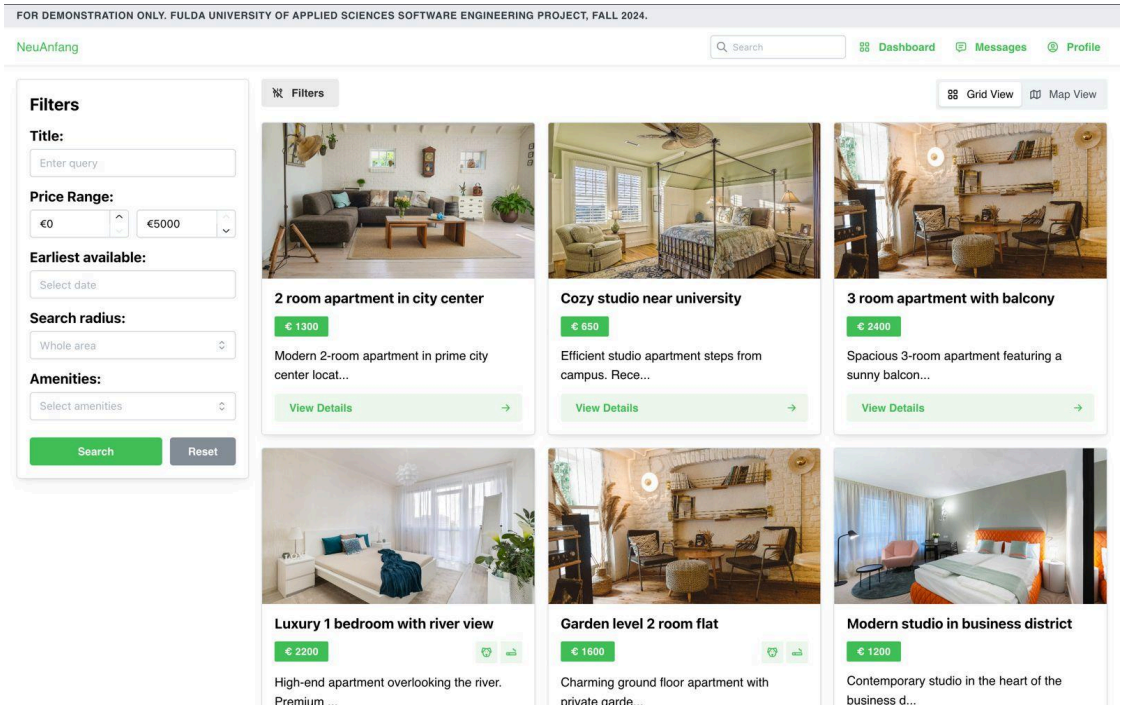
```
validate: {
  name: (value) => {
    return value.length < 2 ? "Name must have at least 3 letters" : null;
  },
  email: (value, values) => {
    const isEmail = /^\S+@\S+$/.test(value);
    if (!isEmail) {
      return "Invalid email";
    }
    if (values.type === "STUDENT" && !value.endsWith("hs-fulda.de")) {
      return "Please use your university email";
    }
    return null;
  },
  phone: (value) => {
    if (!value) {
      return null;
    }
    const isGermanMobile = /^(\+49|0049|0)(15|16|17)\d{8,9}$/.test(value);
    if (!isGermanMobile) {
      return "Invalid phone number";
    }
    return null;
  },
  password: (value) => {
    return value.length < 8 ? "Password must have at least 8 letters" : null;
  },
  confirmPassword: (value, values) => {
    return value !== values.password ? "Passwords do not match" : null;
  },
}
```

6. Self-check: Adherence to original Non-functional specs

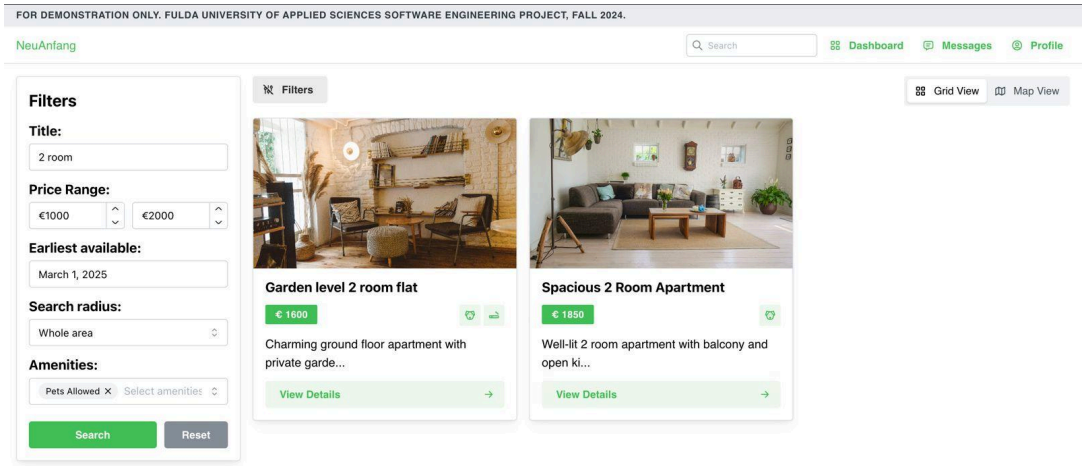
1. Application shall be developed, tested, and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from the chosen cloud
   Server **DONE**
2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers **DONE**
3. All or selected application functions must render well on mobile devices **DONE**
4. Data shall be stored in the database on the team's deployment cloud server. **DONE**
5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner. **DONE**
6. No more than 50 concurrent users shall be accessing the application at any time. **DONE**
7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. **DONE**
8. The language used shall be English (no localization needed) **DONE**
9. Application shall be very easy to use and intuitive **DONE**
10. Application should follow established architecture patterns **DONE**
11. Application code and its repository shall be easy to inspect and maintain **DONE**
12. Google Analytics shall be used (optional for Fulda teams) **NA**
13. No email clients shall be allowed. **DONE**
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. **DONE**
15. Site security: basic best practices shall be applied (as covered in the class) for main data items **DONE**
16. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today **DONE**
17. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development **DONE**
18. For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set up by class instructors and started by each team during Milestone 0 **DONE**
19. The application UI (WWW and mobile) shall prominently display the following exact text on all pages "Fulda University of Applied Sciences Software Engineering Project, Fall 2024 For Demonstration Only" at the top of the WWW page. (Important to not confuse this with a real application). **DONE**

Summary:

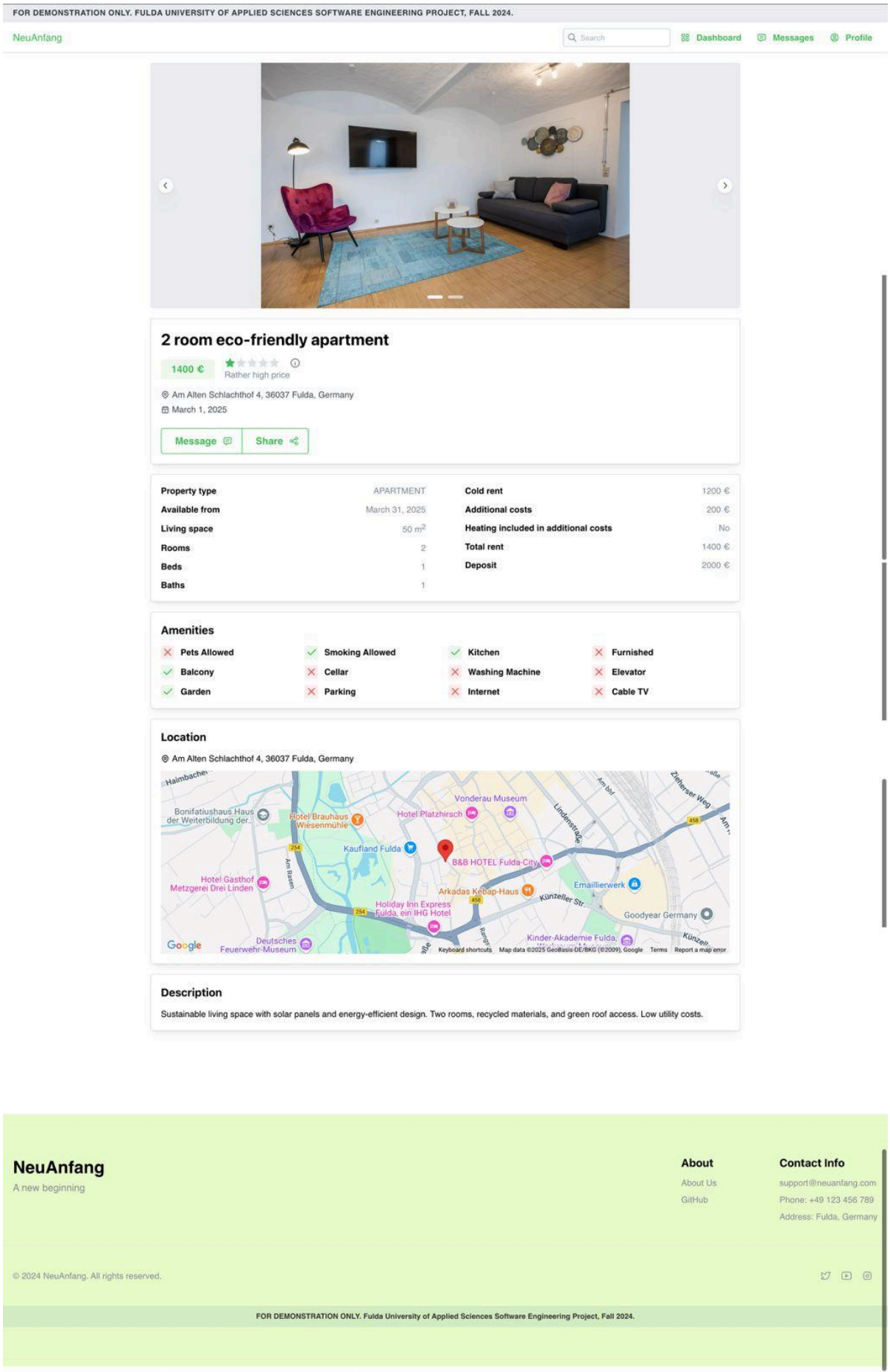
1.Home page



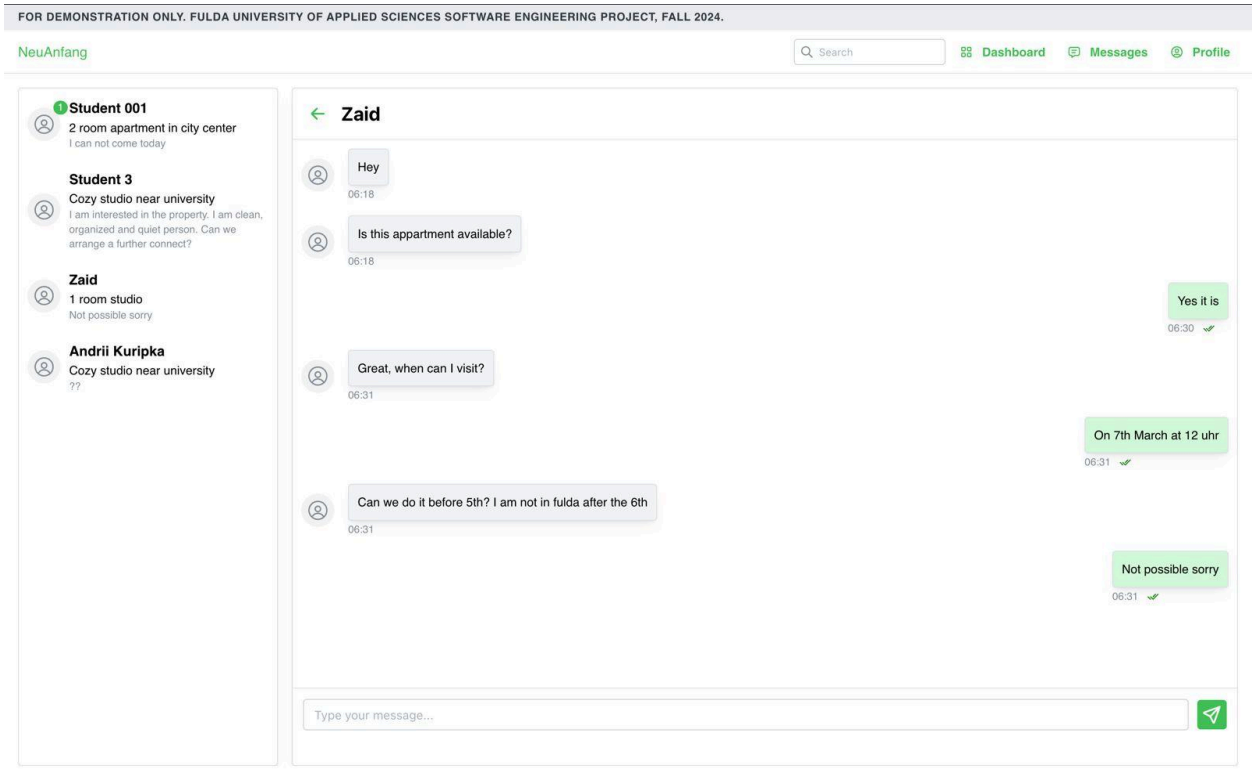
2.Search results page



3. Property detail page



4. Messages


FOR DEMONSTRATION ONLY. FULDA UNIVERSITY OF APPLIED SCIENCES SOFTWARE ENGINEERING PROJECT, FALL 2024.
NeuAnfang
Search
Dashboard
Messages
Profile
Student 001
2 room apartment in city center
I can not come today
Student 3
Cozy studio near university
I am interested in the property. I am clean, organized and quiet person. Can we arrange a further connect?
Zaid
1 room studio
Not possible sorry
Andrii Kuripka
Cozy studio near university
??
Zaid
Hey
06:18
Is this appartment available?
06:18
Yes it is
06:30
Great, when can I visit?
06:31
On 7th March at 12 uhr
06:31
Can we do it before 5th? I am not in fulda after the 6th
06:31
Not possible sorry
06:31
Type your message...