HOCHSCHULE FULDA
UNIVERSITY OF APPLIED SCIENCES

Global Distributed Software Development – AI5088 (WiSe24/25)

Master Team Project – Fall 2024

"Fulda Student Hub"

Milestone 4

January 23, 2025

Supervisor:

Prof. Dr. Todtenhöfer, Rainer

Team Members		
Member Name	**Email**	**Role**
Huzaifa Khatri	Huzaifa.khatri@informatik.hs-fulda.de	Team Lead & Backend Affinity
Shafi Shaik	Shafi.shaik@informatik.hs-fulda.de	Github Master
Muhammad Hassan	Muhammad.hassan@informatik.hs-fulda.de	Backend Lead
Divyansh Dahiya	Divyansh.dahiya@informatik.hs-fulda.de	Backend Affinity
Devansh Negi	Devansh.negi@informatik.hs-fulda.de	Frontend Affinity
Aniq Ahmed	Aniq.ahmed@informatik.hs-fulda.de	Frontend Lead

Revision History				
Member Name	**Date Submitted**	**Date Revised**	**Remarks**	**Version**
Huzaifa Khatri	23.01.2025	-	-	1.0
Huzaifa Khatri	06.02.2025	05.02.2025	Added more usability test results, peer reviews and QA tests	1.1

Contents

1. Product Summary

Fulda Student Hub is the ultimate solution for Fulda students in need of fast, secure, and reliable housing. As the student population grows, our platform focuses exclusively on student accommodation, streamlining the rental process for a user-friendly experience. With our **Bidding Functionality**, landlords can quickly select tenants with real-time bids, making urgent tenant searches simple and efficient. **Lease Agreements** are provided digitally, offering proof of contract for peace of mind. Our **Trust Score** feature uses a customized rating system to evaluate landlords, ensuring prospective tenants always make informed choices.

Stay connected with ease through the **Group Chat**, allowing multiple people to communicate at once. Organize your schedule effortlessly with the **Calendar View for Appointments**, providing a clear, intuitive interface to manage all your housing-related appointments. Plus, discover nearby **Nearest Utilities** like supermarkets, hospitals, and bus stops, making life in Fulda even more convenient. Verified by active moderators and designed to eliminate outdated listings, Fulda Student Hub ensures safe and efficient housing experience for students, especially those new to the local market. With a scalable, growing platform, we're building a stronger community for students, landlords, and everyone in between.

1.1 All Product Functions:

- **Homepage:** Displays key information and navigation links for quick access to features.
- **Search (Including Search Field Validation):** Enables users to search for properties with input validation.
- **Search Results:** Shows property listings based on search criteria with essential details.
- **Filtering:** Refines search results by price, location, property type, and amenities.
- **Search Details and Maps (If Applicable):** Displays detailed property info and a map (if applicable).
- **Messaging/Contact Agent/User (If Applicable):** Allows communication between users and agents/landlords for inquiries or scheduling.
- **Data Upload:** Enables landlords to upload property listings with necessary details, images, and documents.
- **Dashboards (User, Admin):** Displays saved properties, messages, and appointments. Allows management of listings, and platform activity.
- **Bidding Functionality:** Landlords can access a real-time bidding feature for accommodations, allowing quick and urgent tenant selection when needed.
- **Lease Agreement:** A digital copy of the lease agreement, serving as proof of contract between the parties.
- **Trust Score:** A customized rating system that evaluates landlords to provide better choices for prospective tenants
- **Group Chat:** Allows messaging between more than two people, enabling group conversations.

- **Calendar View for Appointments:** An intuitive user interface that provides a better view of all scheduled appointments for users.
- **Nearest Utilities**: Users can view nearby utilities, such as supermarkets, fuel stations, hospitals, schools, bus stops, etc.

Product URL: Fulda Student Hub

Target Audience
Our primary target audience consists of Hochschule Fulda students as well as landlords within the Fulda area. By focusing on this community, **Fulda Student Hub** aims to become the ultimate platform for students seeking reliable and secure housing. Our platform offers students a streamlined rental process, ensuring access to updated listings, secure contracts, and tools for better decision-making. For landlords, it offers a simplified way to find reliable tenants, access real-time bidding, and ensure transparency in the rental process. Together, we foster a secure and efficient housing experience for all.

1.2 Unique Selling Propositions (USPs)

1. **Exclusive Hochschule Fulda Integration**
   Fulda Student Hub is designed specifically for Hochschule Fulda students ensuring a secure and trusted environment within the university community. This exclusivity guarantees that the platform is used by the right audience, enabling focused collaboration and engagement between students and landlords.

2. **Bidding Functionality**
   Landlords can access a real-time bidding feature, allowing for quick tenant selection when urgent housing needs arise. This feature ensures a seamless and efficient rental process for both parties.

3. **Trust Score**
   Our customized rating system evaluates landlords, providing students with valuable insights to make informed housing decisions.

4. **Lease Agreement**
   A digital copy of the lease agreement is provided for each contract, ensuring both students and landlords have clear proof of their rental arrangements.

5. **Group Chat**
   Enabling seamless communication, the group chat feature allows multiple users to discuss, negotiate, and share information, fostering better collaboration and transparency.

6. **Calendar View for Appointments**
   An intuitive interface lets users manage and view all their housing appointments with ease, ensuring no meeting is missed.

7. **Nearest Utilities**
   Students can quickly identify nearby utilities such as supermarkets, fuel stations, hospitals, schools, bus stops, and more, making it easier to settle into their new homes.

1.3 Marketing Channels

1. **Campus Outreach and Awareness Campaigns**
   We will engage in on-campus marketing initiatives to raise awareness and generate interest among our target audience. These efforts include distributing flyers, hosting information sessions, and partnering with student organizations to promote **Fulda Student Hub** as the go-to platform for student housing.

2. **Social Media Campaigns**
   Utilizing platforms such as Instagram, Facebook, and Twitter, we will share updates, testimonials, and success stories to increase visibility. Social media will be used to showcase platform features, engage with users, and encourage them to explore the site.

With **Fulda Student Hub**, we're building a stronger, more connected community that simplifies the search for student housing, making it safer and more efficient for everyone involved.

2. Usability Test Plan

2.1 Search Functionality with advanced filters

2.1.1 Test Objectives

The primary objective of this usability test is to evaluate the "**Search**" functionality within the Fulda Student Hub web application. Specifically, the test aims to assess the ease of use, effectiveness, and overall user satisfaction when searching for rental accommodations. By identifying any usability issues or pain points in the search process, the goal is to enhance the user experience, improve search accuracy, and ensure that students can efficiently locate properties matching their preferences. This evaluation will help refine the interface and functionality to better serve the target user base—students seeking rental accommodations.

2.1.2 Test Background and Setup

System Setup: The usability test will be conducted on the Fulda Student Hub web application, accessible via the URL: https://fulda-student-hub.publicvm.com/app. Testing will be performed using the latest versions of **Google Chrome** and **Mozilla Firefox** browsers on desktop and laptop environments to ensure compatibility and performance across commonly used platforms.

Starting Point: Participants will begin the test on the **homepage** of the Fulda Student Hub, where the search functionality is prominently displayed.

Intended Users: The target participants for this usability test are university students who are actively seeking to rent apartments through the Fulda Student Hub. These users are expected to have varying levels of technical proficiency, ranging from novice to intermediate users who have basic familiarity with online search forms but no prior experience with this platform to ensure unbiased results.

URL of the System: The specific URL to be tested is: https://fulda-student-hub.publicvm.com/app/home

What is to be Measured: This test will focus on the usability of the search feature. User satisfaction will be evaluated using a **Likert-scale questionnaire** administered after the completion of the search task. **Effectiveness and efficiency** will be measured through task success rates, completion times, and user feedback.

2.1.3 Usability Task Description

Instructions to the Tester:

1. **Preparation:**
    - Ensure you have access to a stable internet connection and are using either Google Chrome or Mozilla Firefox on a desktop or laptop.
    - Log into your student account on the Fulda Student Hub using your credentials.

2. **Task Execution:**
    - Locate the search bar on the homepage.
    - Enter the following search criteria:
        - Location: "Fulda"
        - Room Type: "Single Room"
        - Price Range: €300 - €600
    - Use the "Advanced Filters" to select at least two amenities (e.g., Wi-Fi and Parking).
    - Click the "Let’s Go" button to perform the search.
    - Review the search results and identify one property that meets your criteria.
    - Complete a short Likert-scale questionnaire to share your experience.
3. **Post-Task Assessment:**
    - After completing the search task, please fill out the following Likert-scale questionnaire based on your experience.

Measuring Effectiveness: Effectiveness will be measured by the percentage of testers who successfully find a property matching their search criteria without requiring additional guidance.

Measuring Efficiency: Efficiency will be assessed by recording the time taken to complete the search process and the number of steps or clicks required to identify a suitable property.

Likert Scale Questions: Please indicate your level of agreement with the following statements regarding your experience with the search feature:

1. **Ease of Use**:
    - "The search bar and filters were intuitive and easy to use."
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree
2. **Effectiveness**:
    - "The search results matched my expectations based on the criteria I entered."
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree

3. **Overall Satisfaction**:
    - "I was satisfied with the responsiveness and clarity of the search process."
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree

Additional Notes:

- **Test Moderation**: The test will be moderated to observe participants in real-time, allowing for immediate feedback and clarification of any issues encountered during the search process.
- **Recording**: Sessions will be recorded with participants' consent to facilitate detailed analysis of user interactions and behaviors.
- **Participant Diversity**: Efforts will be made to include participants with diverse technical backgrounds to ensure comprehensive usability insights.

By conducting this usability test, we aim to identify and address any barriers that students may face while searching for accommodation, thereby enhancing the overall functionality and user experience of the Fulda Student Hub.

2.2 Upload Student Application

2.2.1 Test Objectives

The primary objective of this usability test is to evaluate the **"Upload Student Application"** feature within the Fulda Student Hub web application. Specifically, the test aims to assess the ease of use, clarity of instructions, and overall user satisfaction when students upload their application documents. By identifying any usability issues or pain points in the upload process, the goal is to enhance the user experience, streamline the application process, and ensure that students can efficiently submit all required documentation without frustration or error. This evaluation will help in refining the interface and functionality to better serve the target user base—university students seeking apartment rentals.

2.2.2 Test Background and Setup

System Setup: The usability test will be conducted on the Fulda Student Hub web application, accessible via the URL: https://fulda-student-hub.publicvm.com/app. Testing will be performed using the latest versions of **Google Chrome** and **Mozilla Firefox** browsers on desktop and laptop environments to ensure compatibility and performance across commonly used platforms.

Starting Point: Participants will begin the test by logging into their student accounts on the Fulda Student Hub. Upon successful login, they will select any available Listing and navigate to where they can choose to either **Contact the Landlord** or **Apply for a Listing**. For this test, participants will focus on the **Apply Now** option, which directs them to the **Application Form**.

Intended Users: The target participants for this usability test are university students who are actively seeking to rent apartments through the Fulda Student Hub. These users are expected to have varying levels of technical proficiency, ranging from novice to intermediate users who are comfortable navigating web applications but may have different experiences with document upload functionalities.

URL of the System: The specific URL to be tested is: https://fulda-student-hub.publicvm.com/app/home

What is to be Measured: This test will focus on **user satisfaction** related to the upload feature. User satisfaction will be evaluated using a **Likert-scale questionnaire** administered after the completion of the upload task. The questionnaire will measure perceptions of ease of use, clarity of instructions, and overall satisfaction with the upload process.

2.2.3 Usability Task Description

Instructions to the Tester:

1. **Preparation:**
    - Ensure you have access to a stable internet connection and are using either Google Chrome or Mozilla Firefox on a desktop or laptop.
    - Log into your student account on the Fulda Student Hub using your credentials.
2. **Task Execution:**
    - Select any available Listing after logging in.
    - Select the **Apply Now** option.
    - Fill out the required details, including your name, address, and other necessary information.
    - Upload the following documents:
        - **Government ID** (JPEG, JPG, PNG, GIF, AVIF, or PDF; required, 1 attachment)
        - **Financial Proof** (JPEG, JPG, PNG, GIF, AVIF, or PDF; required, 1 attachment)
        - **Enrollment Certificate** (JPEG, JPG, PNG, GIF, AVIF, or PDF; required, 1 attachment)
        - **Other Documents** (optional; multiple attachments allowed)
    - Agree to the terms and conditions check.
    - Submit the completed application form.
    - Upon successful submission, you should see an **Application Successful** confirmation message.
3. **Post-Task Assessment:**
    - After completing the upload process, please fill out the following Likert-scale questionnaire based on your experience.

Measuring Effectiveness: Effectiveness will be measured by the number of participants who successfully upload all required documents without encountering errors or requiring assistance during the process.

Measuring Efficiency: Efficiency will be assessed by recording the time taken to complete the upload process and the number of steps or clicks required to successfully submit the application.

Likert Scale Questions: Please indicate your level of agreement with the following statements regarding your experience with the application upload feature:

1. **Clarity of Instructions:**
    - "I was satisfied with the responsiveness and clarity of the search process."
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree
2. **Ease of Use:**
    - “I found the upload process efficient and straightforward.”
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree
3. **Overall Satisfaction:**
    - “Overall, I am satisfied with the application upload feature.”
        - ☐ 1 - Strongly Disagree
        - ☐ 2 - Disagree
        - ☐ 3 - Neutral
        - ☐ 4 - Agree
        - ☐ 5 - Strongly Agree

Additional Notes:

- **Test Moderation:** The test will be moderated to observe participants in real-time, allowing for immediate feedback and clarification of any issues encountered during the upload process.
- **Recording:** Sessions will be recorded with participants' consent to facilitate detailed analysis of user interactions and behaviors.
- **Participant Diversity:** Efforts will be made to include participants with diverse technical backgrounds to ensure comprehensive usability insights.

By conducting this usability test, we aim to identify and address any barriers that students may face while uploading their applications, thereby enhancing the overall functionality and user experience of the Fulda Student Hub.

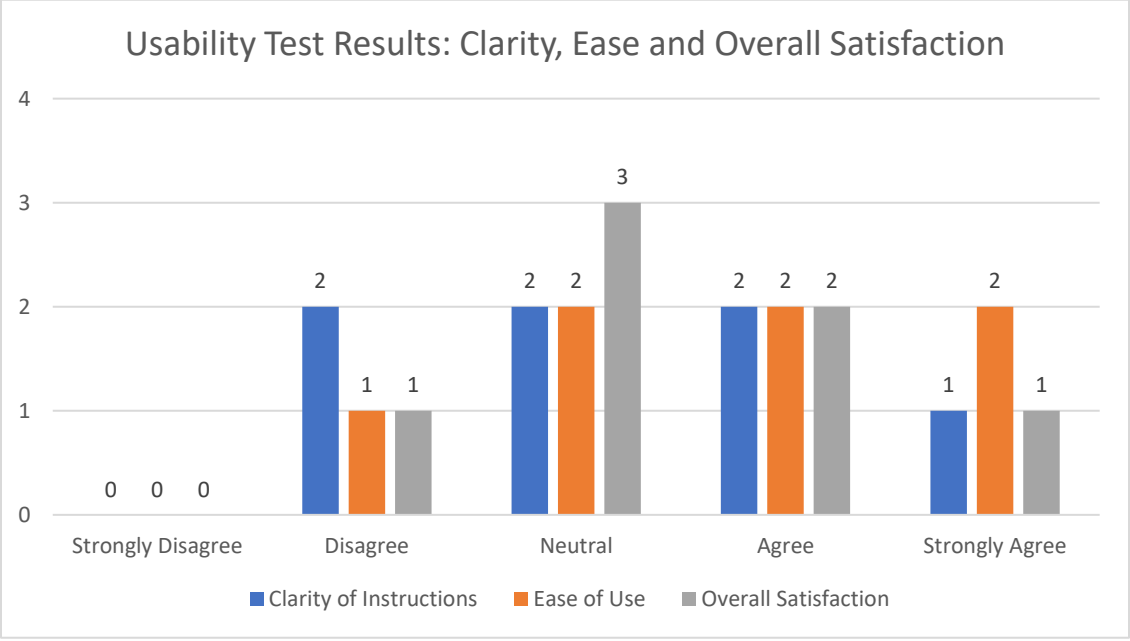
Post Usability Testing

Testing Users:

- Test User 1 (MSc in Global Software Development)
- Test User 2 (MSc in Global Software Development)
- Test User 3 (BSc in Mechatronik)
- Test User 4 (MSc in International Management)
- Test User 5 (MSc in International Management)
- Test User 6 (MSc in International Food Business and Consumer Studies)
- Test User 7 (BSc in International Business and Management)

Test Results:

Name	Clarity of Instructions	Ease of Use	Overall Satisfaction
Test User 1	4 – Agree	3 – Neutral	4 – Agree
Test User 2	3 – Neutral	5 – Strongly Agree	3 – Neutral
Test User 3	2 – Disagree	4 - Agree	3 – Neutral
Test User 4	5 – Strongly Agree	4 – Agree	5 – Strongly Agree
Test User 5	3 – Neutral	2 – Disagree	4 - Agree
Test User 6	4 - Agree	5 – Strongly Agree	3 – Neutral
Test User 7	2 – Disagree	3 – Neutral	2 – Disagree



3. QA Test Plan

Test Objective

The goal of this QA test is to test every main feature on the website **Fulda Student Hub**, which is a web project done by six of the master students for the course "Global Distributed Software Development". Fulda Student Hub is an online platform exclusive to students at Fulda University of Applied Sciences. Users can register and login, browse and search for available accommodations while other residents of Fulda city can play the role of being landlords if they would like to rent out their property(s). The feature which will be looked upon in this test is searching for suitable accommodation as a student and submitting rental application. The reason for this being that it is one of the main functions of the website which is very likely to be used by every user or at least the vast majority.

Hardware and software setup

Tested software:

- **Fulda Student Hub** website – URL: https://fulda-student-hub.publicvm.com/app - running on a remote EC2 instance on AWS.

Hardware used for testing:

- Mozilla Firefox browser 64 bit for Windows and Mac
- Google Chrome browser 64 bit for Windows and Mac
- Mozilla Firefox browser Android and iOS
- Google Chrome browser for Android and iOS

No.	Title	Description	Input	Expected Output	Result
1	Sign-up and Sign-in on platform	Any user should be easily able to register on platform and be able to sign in.	1. If new user, then click on register link and create a new account with required information in fields 2. If existing user, then input user email and password.	1. Successful account creation 2. Redirect to homepage on successful login	Pass
2.	Search and Advanced Search	Users should be able to view available listing on platform with normal search or advanced searching with multiple filters	Open homepage and view available listings. Optionally, perform search with one or multiple filters. For filter selected input keyword or select price range, etc.	List of available listings shown depending on any filter criteria if applied.	Pass
3.	Student application submission	Student users specifically should be able to submit rental application on a listing.	Input required details in fields and upload documents on rental application.	A submitted rental application with details input by user	Pass
4.	Send message	Users should be able to send messages to a listing owner.	Send message to the contact person on a listing.	The receiving user should receive a message in real time by the sender	Pass
5.	Create a new Property	Landlord users should be able to create new properties	Input property details and upload related images	Item of the new property visible to the creator	Pass
6.	Create a new listing	Landlord users should be able to create new listings from their properties	Select a property, input any additional details and upload any additional images	Item of the new listing visible to the creator and in the list of all available platform listings.	Pass
7.	Live Bidding on Rental Listings	Users should be able to participate in live bidding on rental listings.	1. Select a rental listing with live bidding enabled. 2. Place a bid with a valid amount. 3. Ensure bid is processed and reflected in the system.	1. Bid successfully placed. 2. Updated highest bid visible to all users.	Pass
8.	Document Upload Functionality	Users should be able to upload the documents in all over system	1. Navigate to the document upload section. 2. Select a valid document file (PDF, PNG, JPG). 3. Submit upload.	1. Document successfully uploaded and visible in user profile or application.	Pass
9.	No Loss of Data While Processing	Ensure no data is lost when users submit rental applications or listings.	1. Fill out rental application or create a listing. 2. Submit the form. 3. Simulate network failure and refresh the page.	1. Data should be saved as a draft or automatically recovered. 2. User should be able to continue without data loss.	Pass

10.	Manage Listings via Admin	Admins should be able to review and take action on listings violating platform policies.	1. Log in as admin. 2. Navigate to reported or flagged listings. 3. Approve, reject, or modify the listing.	1. Admin successfully reviews and takes action on reported listings. 2. Status of listing updated accordingly.	Pass
11.	Manage Properties via Admin	Admins should be able to review and take action on properties violating platform policies.	1. Log in as admin. 2. Navigate to reported properties. 3. Approve, reject, or modify the property.	1. Admin successfully reviews and takes action on reported properties. 2. Status of property updated accordingly.	Pass

4. Code Review

For the purpose of code review, we opted with a pair-peer review strategy. Where we formed groups of two each reviewing each other's code and later onwards as a team we shared our findings for any further betterment possibilities.

1. Divyansh Dahiya (Reviewer) → Muhammad Hassan (Reviewee)

Before:

```
uploadPromises.push(
      // Hassan, this is using await inside uploadPromises.push, which is unnecessary
and can lead to confusion
      // Code review by Divyansh
      await uploadToS3(file).then((uploadedFile) => {
        mediaEntries.push({
model_name: "application",
          model_id: application.application_id,
          media_url: uploadedFile,
media_type: "government_id",
          media_category: "image",
        });
      })
    );
```

After:

```
uploadPromises.push(
      uploadToS3(file).then((uploadedFile) => {
mediaEntries.push({
          model_name: "application",
          model_id: application.application_id,
media_url: uploadedFile,
          media_type: "government_id",
          media_category: "image",
});
      })
    );
  }
```

Before:

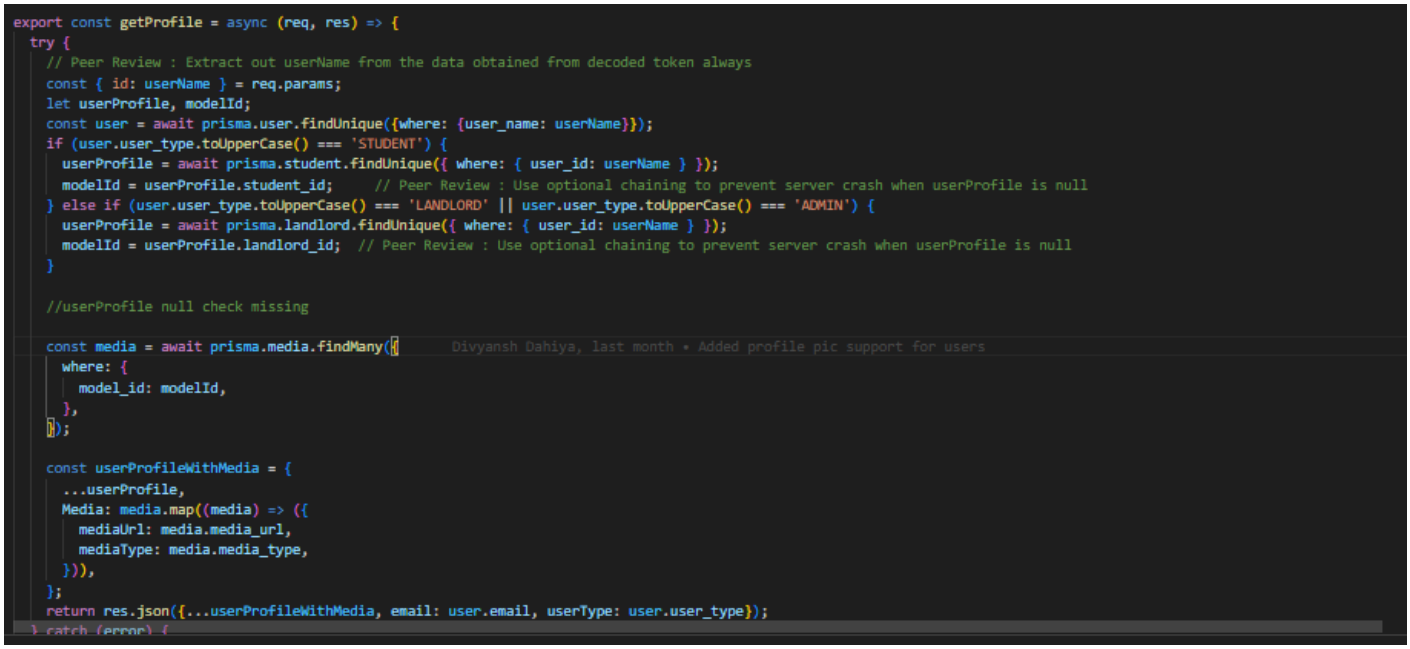
```
const application = await prisma.application.create({
     data: {
       listing_id: listing_id,
       student_id: userName,
student_card_id,
       full_name,
       contact_number,
current_address,
       application_status: "PENDING",
       // Hassan, we can omit the explicit null check on remarks field. Prisma will
handle it as null by default.
       // Code review by Divyansh
       remarks: remarks || null,
},
   });
```

After:

```
const application = await prisma.application.create({
     data: {
       listing_id: listing_id,
       student_id: userName,
student_card_id,
       full_name,
       contact_number,
current_address,
       application_status: "PENDING",
       remarks: remarks,
     },
   });
```

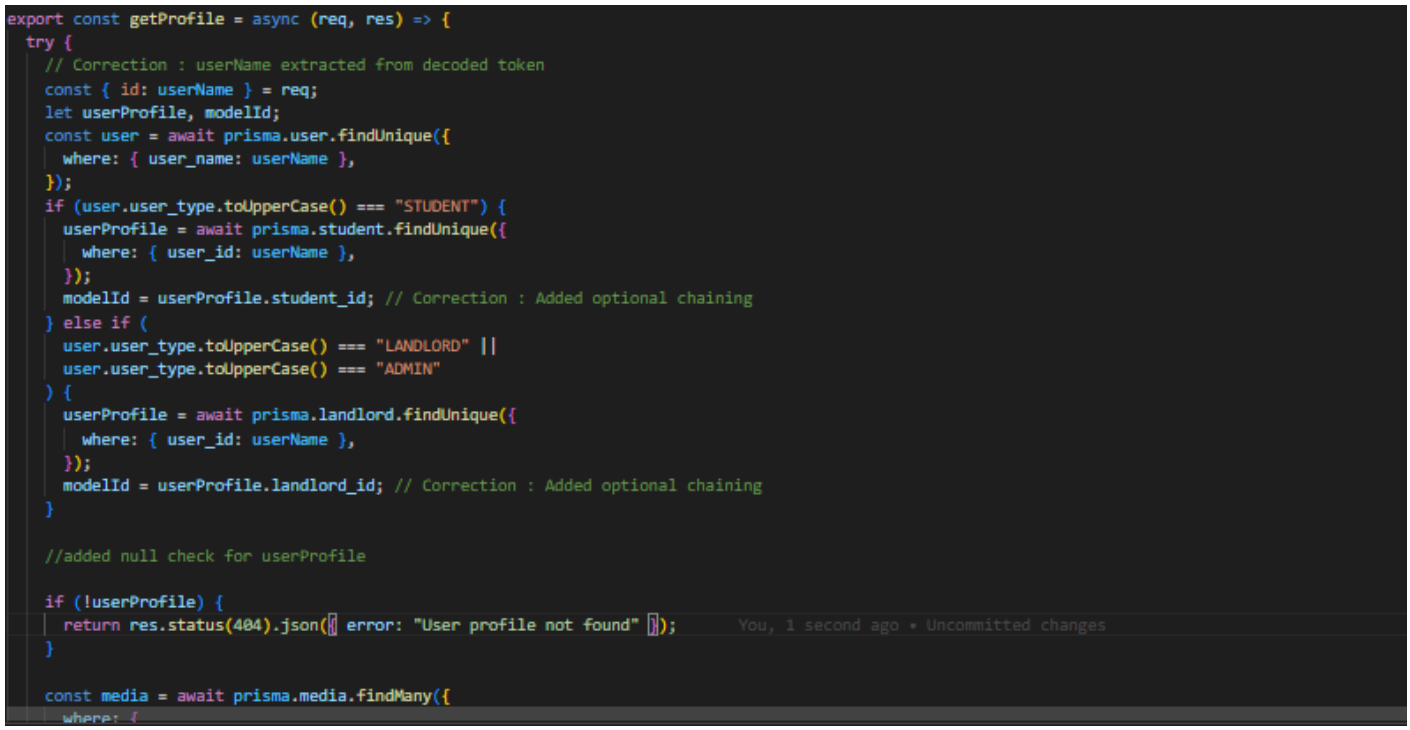
2. Muhammad Hassan (Reviewer) → Divyansh Dahiya (Reviewee)

Before:



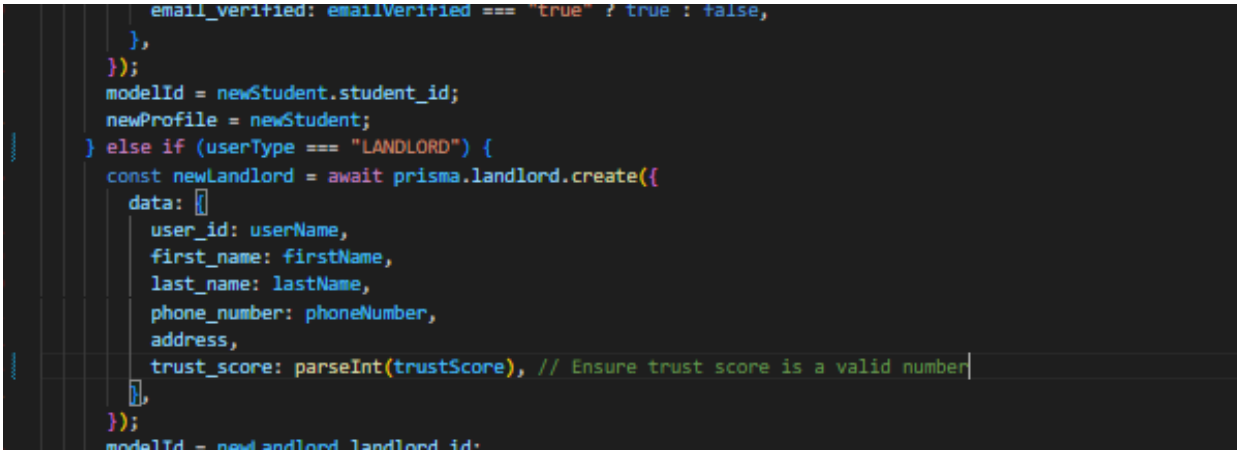
```
export const getProfile = async (req, res) => {
  try {
    // Peer Review : Extract out userName from the data obtained from decoded token always
    const { id: userName } = req.params;
    let userProfile, modelId;
    const user = await prisma.user.findUnique({where: {user_name: userName}});
    if (user.user_type.toUpperCase() === 'STUDENT') {
      userProfile = await prisma.student.findUnique({ where: { user_id: userName } });
      modelId = userProfile.student_id;     // Peer Review : Use optional chaining to prevent server crash when userProfile is null
    } else if (user.user_type.toUpperCase() === 'LANDLORD' || user.user_type.toUpperCase() === 'ADMIN') {
      userProfile = await prisma.landlord.findUnique({ where: { user_id: userName } });
      modelId = userProfile.landlord_id;  // Peer Review : Use optional chaining to prevent server crash when userProfile is null
    }

    //userProfile null check missing

    const media = await prisma.media.findMany({      Divyansh Dahiya, last month • Added profile pic support for users
      where: {
        model_id: modelId,
      },
    });

    const userProfileWithMedia = {
      ...userProfile,
      Media: media.map((media) => ({
        mediaUrl: media.media_url,
        mediaType: media.media_type,
      })),
    };
    return res.json({...userProfileWithMedia, email: user.email, userType: user.user_type});
  } catch (error) {
```

After:



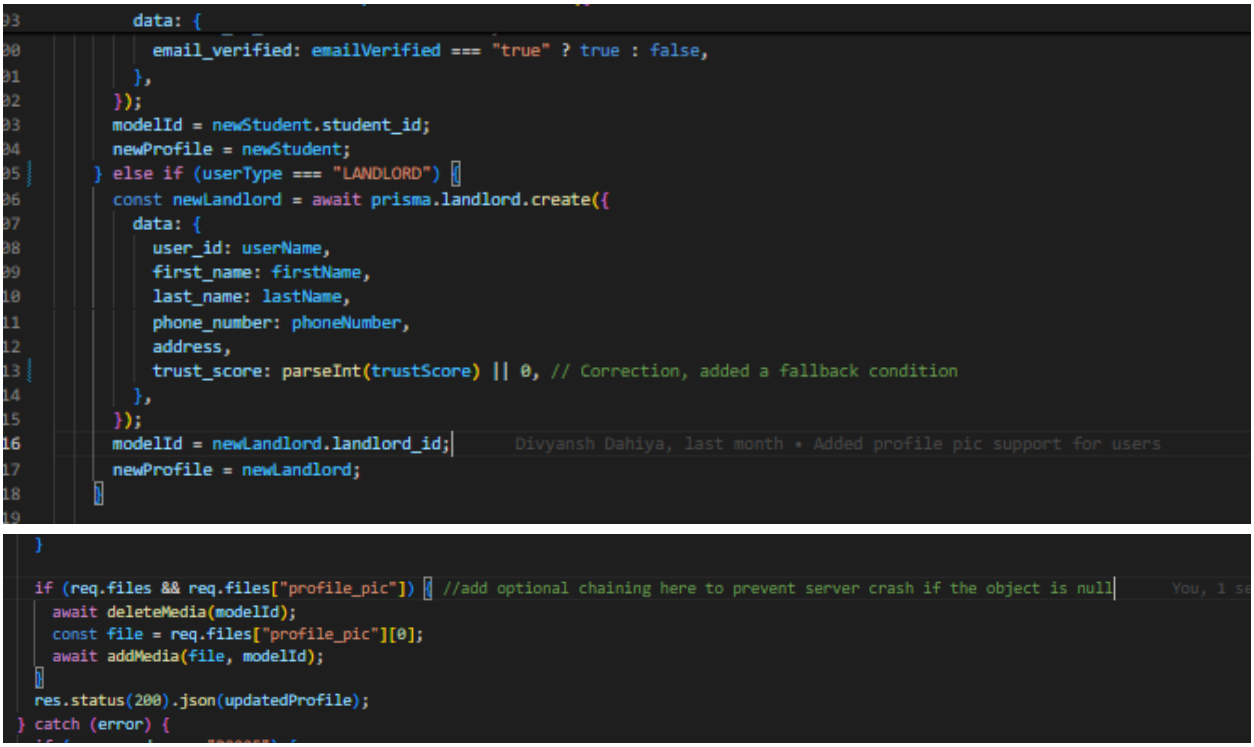
```
export const getProfile = async (req, res) => {
  try {
    // Correction : userName extracted from decoded token
    const { id: userName } = req;
    let userProfile, modelId;
    const user = await prisma.user.findUnique({
      where: { user_name: userName },
    });
    if (user.user_type.toUpperCase() === "STUDENT") {
      userProfile = await prisma.student.findUnique({
        where: { user_id: userName },
      });
      modelId = userProfile.student_id; // Correction : Added optional chaining
    } else if (
      user.user_type.toUpperCase() === "LANDLORD" ||
      user.user_type.toUpperCase() === "ADMIN"
    ) {
      userProfile = await prisma.landlord.findUnique({
        where: { user_id: userName },
      });
      modelId = userProfile.landlord_id; // Correction : Added optional chaining
    }

    //added null check for userProfile

    if (!userProfile) {
      return res.status(404).json({ error: "User profile not found" });      You, 1 second ago • Uncommitted changes
    }

    const media = await prisma.media.findMany({
      where: {
```

Before:



```
            email_verified: emailVerified === "true" ? true : false,
          },
        });
        modelId = newStudent.student_id;
        newProfile = newStudent;
      } else if (userType === "LANDLORD") {
        const newLandlord = await prisma.landlord.create({
          data: {
            user_id: userName,
            first_name: firstName,
            last_name: lastName,
            phone_number: phoneNumber,
            address,
            trust_score: parseInt(trustScore), // Ensure trust score is a valid number
          },
        });
        modelId = newLandlord.landlord_id;
```

After:



```
          data: {
            email_verified: emailVerified === "true" ? true : false,
          },
        });
        modelId = newStudent.student_id;
        newProfile = newStudent;
      } else if (userType === "LANDLORD") {
        const newLandlord = await prisma.landlord.create({
          data: {
            user_id: userName,
            first_name: firstName,
            last_name: lastName,
            phone_number: phoneNumber,
            address,
            trust_score: parseInt(trustScore) || 0, // Correction, added a fallback condition
          },
        });
        modelId = newLandlord.landlord_id;      Divyansh Dahiya, last month • Added profile pic support for users
        newProfile = newLandlord;
      }
```

```
    }

    if (req.files && req.files["profile_pic"]) { //add optional chaining here to prevent server crash if the object is null
      await deleteMedia(modelId);
      const file = req.files["profile_pic"][0];
      await addMedia(file, modelId);
    }
    res.status(200).json(updatedProfile);
  } catch (error) {
```

3. Huzaifa Khatri (Reviewer) → Shafi Shaik (Reviewee)

Before:

```
// Peer review:
// 1. Refactor following function to follow the format as other functions in the file
// 2. Adding a try/catch block in order to process better error handling
async function getLandlordId(user_name) {
  const dbUser = await prisma.user.findUnique({
    where: { user_name },
    include: { Landlord: true },
  });
  if (!dbUser || !dbUser.Landlord) {
    throw new Error("Associated landlord profile not found");
  }
  return dbUser.Landlord.landlord_id;
}
```

After:

```
async function getLandlordId(user_name) {
  try {
    const dbUser = await prisma.user.findUnique({
      where: { user_name },
      include: { Landlord: true },
    });
    if (!dbUser || !dbUser.Landlord) {
      throw new Error("Landlord profile not found");
    }
    return dbUser.Landlord.landlord_id;
  } catch (error) {
    handleListingError(res, error, "retrieving landlord ID");
  }
}
```

Before:

```
    });
  } catch (error) { // Peer review: A considerable option, since similar error handling and print output is kept here.
  // Moving the repetetive code to a separate function and reusing that function
    console.error("Error retrieving listings:", error);
    res.status(500).json({ success: false, error: "An unexpected error occurred while retrieving listings" });
  }
};
```

After:

```
    });
  } catch (error) {
    // console.error("Error retrieving listings:", error);
    handleListingError(res, error, "retrieving listings");
  }
};
```

```
// Error handler utility
const handleListingError = (res, error, context = "") => {
  console.error(`Error ${context}:`, error);
  res.status(500).json({
    success: false,
    error: `An unexpected error occurred while ${context}`
  });
};
```

Before:

```
export const createListing = async (req, res) => {
    try {
      if (!req.user || !req.user.userName) {
        return res.status(401).json({ success: false, error: "User not authenticated" });
      }

      // Peer review: Consider adding null validation for properties being fetch from req.body
      const { property_id, title, description, status, rent, room_type_id } = req.body;

      // Validate landlord ownership of the property
      const landlord = await prisma.user.findUnique({
        where: { user_name: req.user.userName },
        include: { Landlord: true },
      });
```

```
        return res.status(401).json({ success: false, error: "User not authenticated" });
      }

      // Peer review: Similar null validation check on following variable before processing database transactior
      const { id } = req.params;
      const landlord_id = await getLandlordId(req.user.userName);

      const listing = await prisma.listing.findUnique({
        where: {
          listing_id: id,
          property: {
            landlord_id,
          },
        },
        include: {
          property: true,
          room_type: true,
```

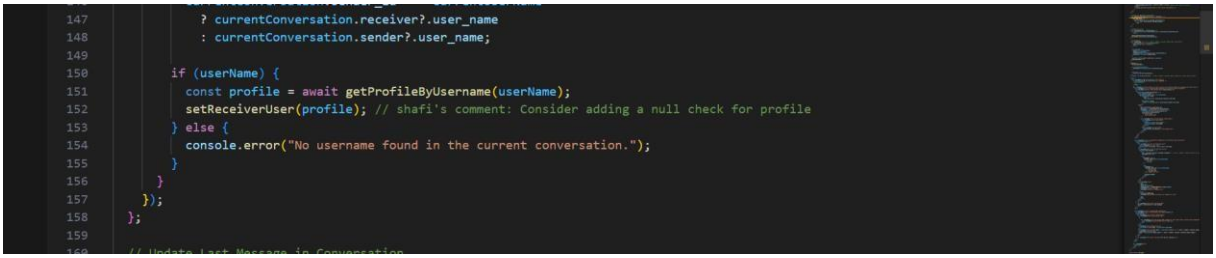
After:

```
24    }
25
26    export const createListing = async (req, res) => {
27        try {
28          if (!req.user || !req.user.userName) {
29            return res.status(401).json({ success: false, error: "User not authenticated" });
30          }
31
32          const { property_id, title, description, status, rent, room_type_id } = req.body;
33          if (!property_id || !title || !description || !status || !rent || !room_type_id) {
34            return res.status(400).json({
35                success: false,
36                error: "All fields are required"
37            });
38          }
39
```

```
206       }
207
208       const { id } = req.params;
209       if (!id) {
210         return res.status(400).json({
211             success: false,
212             error: "Missing ID"
213         });
214       const landlord_id = await getLandlordId(req.user.userName);
215
```

4. Shafi Shaik (Reviewer) → Huzaifa Khatri (Reviewee)

Before:

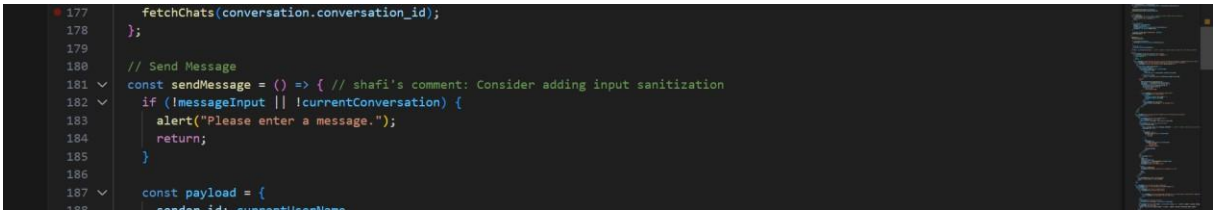


```
            ? currentConversation.receiver?.user_name
            : currentConversation.sender?.user_name;

          if (userName) {
            const profile = await getProfileByUsername(userName);
            setReceiverUser(profile); // shafi's comment: Consider adding a null check for profile
          } else {
            console.error("No username found in the current conversation.");
          }
        }
      });
    };

    // Update Last Message in Conversation
```

After:

```
    if (userName) {
      const profile = await getProfileByUsername(userName);
      if (profile != null) { // Update: Added a null check for profile and setting empty user to not occur any runtime errors
        setReceiverUser(profile);
      } else {
        setReceiverUser(null);
      }
    } else {
      console.error("No username found in the current conversation.");
    }
  }
});
```

Before:



After:

```
// Send Message
const sendMessage = () => {
  if (!messageInput || !currentConversation) {
    alert("Please enter a message.");
    return;
  }

  // Input sanitization
  let sanitizedMessage = messageInput.trim(); // Remove leading/trailing spaces

  // Prevent empty or excessively long messages
  if (sanitizedMessage.length === 0) {
    alert("Message cannot be empty.");
    return;
  }
  if (sanitizedMessage.length > 100) {
    alert("Message is too long. Limit is 100 characters.");
    return;
  }

  const payload = {
    sender_id: currentUserName,
    message: messageInput,
    conversation_id: currentConversation.conversation_id,
    created_at: new Date().toISOString(),
  };

  socketRef.current.emit("sendMessage", payload);
  setMessageInput("");
};
```

5. Aniq Ahmed (Reviewer) → Devansh Negi (Reviewee)

Before:

```
//PeerReview Issue 1:The dependency array only includes id, meaning if the token changes, the property won't be refetched.
 useEffect(() => {

  const fetchProperty = async () => {
    try {
      const data = userType === "ADMIN" ? await fetchPropertyByIdAdmin(id, token) : await fetchPropertyById(id, token);
      if ((userType === "ADMIN" && data) || data.success) {
        setProperty(data || data.data);
        setAddress(data.address || data.data.address);
        setDisplayedImages(data.Media || data.data.media);
      }
    } catch (error) {
      console.log("error", error)
      setError('Error fetching property details');
    }};
  fetchProperty();
 }, [id]);
```

After:

```
//Fix 1
 useEffect(() => {
   const fetchProperty = async () => {
     const token = localStorage.getItem('accessToken'); // Moved inside the function to avoid stale token

     try {
       const data = await fetchPropertyById(id, token);
       if (data.success) {
         setProperty(data.data);
         setAddress(data.data.address);
         setAmenities(data.data.PropertyAmenity.map(pa => ({
           amenity_name: pa.Amenity.amenity_name,
           amenity_value: pa.Amenity.amenity_value
         })));
         setDisplayedImages(data.data.media || []);
       }
     } catch (error) {
       setError('Error fetching property details');
     }
   };

   fetchProperty();
 }, [id]);  // Keeping the dependency minimal, since token is retrieved inside the function
```

Before:

```
//Peer review: 2.Instead of using index-based removal, consider using a unique key like amenity_name to
//avoid potential issues if two amenities have the same value.
const handleRemoveAmenity = (index) => {
  setAmenities(amenities.filter((_, i) => i !== index));
};
```

After:

```
//Fix 2
const handleRemoveAmenity = (name) => {
  setAmenities(amenities.filter((amenity) => amenity.amenity_name !== name));
};
```

```
    {/* Fix 2 */}
      <button
          type="button"
          onClick={() => handleRemoveAmenity(amenity.amenity_name)}
          className="text-red-500 hover:text-red-700"
        >
          Remove
      </button>
```

Before:

```
    {/*Peer Review 3: Use conditional rendering with animation for a better UX.*/}
    {error && <div className="text-red-500 text-center mb-4">{error}</div>}
```

After:

```
    {/*Fix 3 */}
    {error && (<div className="text-red-500 text-center mb-4 transition-opacity duration-300 ease-in-out">
    {error}
    </div>)}
```

6. Devansh Negi (Reviewer) → Aniq Ahmed (Reviewee)

Before:

```
//  Consider storing these in local storage or context for better user experience.
//    If the user refreshes, they won't lose their progress and would be usefull in website. (Devansh)
const [formData, setFormData] = useState({
  firstName: "",
  lastName: "",
  phoneNumber: "",
  ...(userType === "STUDENT"
    ? { university: "", studentIdNumber: "" }
    : { address: "" }),
  profilePic: null,
});

const [notification, setNotification] = useState({ ···
});

const [loading, setLoading] = useState(false);

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData({ ...formData, [name]: value });
};

const showNotification = (msg) => { ···
};
```

After:

```
//Store form data in local storage.
const storedFormData = JSON.parse(localStorage.getItem("profileFormData")) || {}; // Get stored data
const [formData, setFormData] = useState({
  firstName: storedFormData.firstName || "", // Use stored value or empty string
  lastName: storedFormData.lastName || "",
  phoneNumber: storedFormData.phoneNumber || "",
  ...(userType === "STUDENT"
    ? { university: storedFormData.university || "", studentIdNumber: storedFormData.studentIdNumber || ""
    : { address: storedFormData.address || "" }),
  profilePic: null,
});
//To save in localstorage whenever data changes
useEffect(() => {
  localStorage.setItem("profileFormData", JSON.stringify(formData));
}, [formData]);

const [notification, setNotification] = useState({ ···
});
```

Before:

```
    //Handle the response more comprehensively. Check status codes, etc. (Devansh)
    await createProfile(payload, accessToken);

    showNotification("Profile created successfully!");
    setTimeout(() => {
      if (userType === "STUDENT") {
        navigate("/Home", { replace: true });
      } else if (userType === "LANDLORD") {
        navigate("/landlord", { replace: true });
      }
    }, 2000); // Redirect after success
  } catch (error) {
    console.error("Error creating profile:", error);
    showNotification("Failed to create profile. Please try again.");
  } finally {
    setLoading(false);
  }
```

After:

```
const response = await createProfile(payload, accessToken);
if (response.status === 201) {
  showNotification("Profile created successfully!");
  setTimeout(() => {
    if (userType === "STUDENT") {
      navigate("/Home", { replace: true });
    } else if (userType === "LANDLORD") {
      navigate("/landlord", { replace: true });
    }
  }, 2000); // Redirect after success
} else {
  const errorMessage =
    response.data.message || "Failed to create profile."; // Extract error from the response
  showNotification(errorMessage);
}
```

5. Self-check on best practices for security

Major Protected Assets:

- The tables which provide persistence storage for data are media, users, chats, properties, listings.
- In addition to that, files that are being stored on the AWS S3.

List major threats for each asset above:

- Passwords should not be saved in plain text since can be misused if the database access is compromised and access to database is done through specific port only.
- Update and delete operations must be protected against unauthorized access.

For each asset say how you are protecting it:

- Passwords are hashed before being stored in the database.
- The update or delete operations related routes are protected using authentication middleware.
- All respective routes are also protected through authentication middleware developed for every different type of user meaning, Landlord routes can not be triggered via a student type user and vice versa.
- Frontend can only access the files or data stored from the database through APIs which are protected and always require a token for authentication. There is no direct access to the database from frontend.

Confirm that you encrypt password in the database:

We have used the BCrypt Hashing method which converts the password into hash while storing/retrieving from the database using salted hashing and custom number of iterations to build hash. This custom hash can only be validated through a set “salt” value which can be custom and embedded within code. This method is built intentionally slow to make brute-force impractical and the password execution time is constant to avoid timing attacks.

Confirmation of input data validation:

1. Login Page:
    a. Requirements:
        i. Hochschule Fulda Email is required.
        ii. Password of minimum length of 8 is required.

2. Sign Up Page:
    a. Requirements:
        i. First name a string is required.
        ii. Last name a string is required

iii. Email is an email validation (i.e. with @) with suffix allowed ['hs-fulda.de', 'ai.hs-fulda.de', 'informatik.hs-fulda.de']
iv. Password, at least 8 characters while containing at least one small letter, one capital letter, one special character and one number.
v. The repeat Password must match Password.

3. Search bars:
    a. Requirements:
        i. Search field has a maximum input of 40 characters.
        ii. Only amenities available in system should shop up for advanced filtering

4. Upload Media / Document:
    a. Requirements:
        i. Users should select appropriate file type (.PDF, .JPEG, .JPG, .PNG)
        ii. If there is a document upload limit, then user should not be able to upload more than allowed

5. Send Message:
    a. Requirements:
        i. Students should be able to contact landlords via messaging
        ii. If a chat between two users already exists, that chat should be opened instead of making a new one

6. Bidding:
    a. Requirements:
        i. Students should be able to place bidding on active live bidding sessions
        ii. The placed bid should be greater than the current highest bid
        iii. Only a landlord type user can start/end a bidding session

7. Lease Agreement:
    a. Requirements:
        i. Landlord must accept atleast one student application in order to generate a lease agreement soft copy
        ii. The lease agreement soft copy should be read-only and should show all necessary details for a rent contract

6. Self-check: Adherence to original Non-functional specs – performed by team leads

Non-Functional Requirement	Comments
Application shall be developed, tested and deployed using tools and servers approved by Class	Done
CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server	Done
Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers	Done
All or selected application functions must render well on mobile devices	Done
Data shall be stored in the database on the team's deployment cloud server.	Done
Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner	Done
No more than 50 concurrent users shall be accessing the application at any time	Done
Privacy of users shall be protected, and all privacy policies will be appropriately	Done
communicated to the users.	Done
The language used shall be English (no localization needed)	Done
Application shall be very easy to use and intuitive	Done
Application should follow established architecture patterns	Done
Application code and its repository shall be easy to inspect and maintain	Done
Google analytics shall be used (optional for Fulda teams)	Done
No e-mail clients shall be allowed.	Done
Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.	Done
Site security: basic best practices shall be applied (as covered in the class) for main data items	Done
Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today	Done
Modern SE processes and practices shall be used as specified in the class, including	Done
collaborative and continuous SW development	Done
For code development and management, as well as documentation like formal milestones	Done
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
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	Done
Demonstration Only" at the top of the WWW page. (Important to not confuse this with a real application).	Done