Towson University Student Housing (TUSH) Platform (Web)

Software Engineering (COSC612.101)

FALL 2024

Team 1

Group Members:

KWESI ASIEDU

DEAIRA CARRINGTON

CALVIN VENABLE

DANIEL SAMURA

JAYA PEDA VIGNESH REDDY DUGGEMPUDI

ANDREW TAOMINO

Planning and Scheduling

Assignee	Mail ID	Task	Hours	Dependency	Due date	Evaluation
KWESI ASIEDU	Kasied1@s tu dents.towso n. edu	System Modelling	8	None	10/27	100% Completed by due date
DEAIRA CARRINGTO N	dcarri6@st ud ents.towson .e du	Database, ML Components, db hosting	10	UI structure	10/27	100% Completed by due date
CALVIN VENABLE	cvenabl1@ st udents.tow so n.edu	UI, Use Case 5 Implementation	11	None	10/28	100% Completed by due date
DANIEL SAMURA	dsamura1@s tudents.tows on.edu	UI, Use Case 1 Implementation	12	None	10/28	100% Completed by due date
JAYA PEDA VIGNESH REDDY DUGGEMPUDI	jduggem1 @s tudents.to wso n.edu	System Revisions, Functional Testing	3	Completed Implementati on	10/28	80% Completed tests by due date, unable to finish system revisions
ANDREW TAOMINO (Coordinator)	ataorm1@s tu dents.tows on. edu	Middleware /API integration UI Integration	10	UI Completion, Database Completion	10/28	100% UI was not in a good shape until day of, so ran into delays for API implementat ion

Problem Statement

Towson University students face significant challenges when trying to find suitable housing through online platforms. Although various housing websites and apps exist, students often struggle to find listings that meet their specific needs, such as proximity to campus, affordability, and compatibility with student life. The current TU Off-campus housing and other listing websites lack a centralized and AI/ML user-friendly platform tailored to Towson

University students, leading to the following issues: fragmented listings, scams and unreliable listings, limited roommate matching and poor integration with campus resources.

Requirements

User Requirements

TUSH – The Towson University Student Housing (TUSH) is a web-based platform that streamlines the housing search process for on-campus students.

Payment System – A payment system that's integrated into the site to make payments for local transportation.

Roommate Match System – A personalized roommate matching system where students can connect based on shared, interest, schedules, and compatibility.

Search System – A advanced search system that filters things such as distance from campus, rental affordability, amenities, lease terms, and proximity to public transportation.

AI/ML System – A system that's used to drive together key components of the website such as the roommate search system.

Local Transport Integration System – A system that shows nearby transportation from the user's current location.

Interactive Map System – A Map system that shows current location and also the location of nearby bus stops.

Use Cases

Use Case No:	1	Use Case Name:	Housing Listing Posting
Actors:		Landlord, Student Housing System	
Description:		A landlord will use the platform to post a listing and add all features, amenities, pricing, location and pictures. The platform will allow the posting to occur online and notify students of new listings.	
Basic Course of Events		1. User will have completed the login process 2. User will select "add new listing"	

	3. User will enter the following details into a form in relation to the listing a. Name b. Location c. Pricing d. Location e. Pictures f. Amenities 4. The user will confirm that all listed information is accurate 5. The user will select upload. 6. Complete Housing Listing Posting
Alternate Path:	• If the listing details are an exact match from the same landlord a duplicate listing notification will appear, and the listing will not be posted • If the landlord is not verified an account verification request screen will appear, and the landlord will be redirected to the account verification process. • If the landlord no longer wants to submit a listing, they will be able to return back to the landlord dashboard
Pre-Condition:	• The landlord must be logged into the system • The landlord must have completed the background screening

Use Case No:	2	Use Case Name:	Landlord Background Verification
Actors:	Landlord, Student Housing System		
Description:	A landlord will use the platform to become a verified landlord for the housing system. The platform will allow a secure submission of official documents proving ownership or permission to lease a specific location as well as proof of identity documents.		

Basic Course of Events	1. User will have logged in 2. The user will select “Complete landlord verification process” 3. The user will answer the required verification question. 4. The user will confirm their responses. 5. The user will move to the next screen 6. The user will upload required documents 7. The user will confirm the upload of documents 8. The user will select submit
Alternate Path:	• If the landlord is already verified, then a notification will appear saying that the landlord has completed the verification process and will have the option to navigate back to the landlord dashboard or post a new listing • If the landlord has decided to not complete the process, they will be able to navigate back to the landlord dashboard
Pre-Condition:	• The landlord must be logged into the system

Use Case No:	3	Use Case Name:	Landlord Leasing Request Review
Actors:	Landlord, Student, Student Housing System		
Description:	A landlord will use the platform to review requests and responses to lease listings. The landlord should be able to see an interest list and profiles of students that have responded to the listing. The profile information should allow the landlord to view the student’s name, contact information (phone number and email address), and application status.		
Basic Course of Events	1. The user will have logged in 2. The user will select “Review Requests” 3. The user will select a link to review an individual request. 4. The user will respond or update the request. 5. The user will submit 6. Landlord Leasing Request Review completed		

Alternate Path:	• If there are no interest or application forms completed there will be a notification that states such and the landlord will be prompted to return to the dashboard. • If the landlord has not posted a listing to be able to have an application review a notification will appear that states such. The landlord will then be prompted to return to the dashboard or to post a new listing.
Pre-Condition:	• The landlord must be logged into the system • The landlord must be verified • The landlord must have a posting • The landlord must have responses to a listing or applications for a listing posted.

Use Case No:	4	Use Case Name:	Landlord Application Review and Approval
Actors:	Landlord, Student, Student Housing System		
Description:	A landlord will use the platform to review and analyze applications submitted for listings by each individual student. The landlord should be able to see all information submitted in the student's application. After a review the landlord should be able to approve or deny an application and state reasons for denial.		
Basic Cours of Events	1. The user will have logged in 2. The user will select "Review Application" 3. The user will select a link to review an individual application. 4. The user will approve or deny an application		

	5. The user will confirm their selection 6. The user will submit 7. Landlord Application Review completed
Alternate Path:	• If the landlord would like to return to the list of interest forms and applications, they will be able to access a return button
Pre-Condition:	• The landlord must be logged into the system • The landlord must be verified • The landlord must have a posting • The landlord must have applications for a listing

Use Case No:	5	Use Case Name:	Student Housing Search
Actors:	Student, Landlord, Student Housing System		
Description:	A student will use the platform to search for housing listings and view all the listing information while having the ability to filter and sort based on listing features, amenities, location, and pricing.		
Basic Course of Events	1. The user would have logged in 2. The user will select the search button 3. The user will enter their search criteria via text box 4. The user will select search 5. A list of search results will appear 6. The Student Housing Search has been completed		
Alternate Path:	• If there are no listings to match the search criteria the student will be prompted to adjust their search settings or return to the student dashboard.		

	• After viewing a listing if the student does not want to apply for the listing, they will be able to return to the search results • If the student, no longer wants to continue their search the student will be able to return to their dashboard
Pre-Condition:	• The student must be logged into the system • The student must be verified as a Towson University Student • The landlords must have posted listings • The listings must have all information necessary for a proper search to be performed

Use Case No:	6	Use Case Name:	Roommate Matching System
Actors:	Student, Roommate Matching System		
Description:	A student will use the system to enter personal lifestyle information about themselves to be matched with other students that are searching for roommates. Students will be able to view basic information (first name, last name, age, and lifestyle preferences)		
Basic Course of Events	1. The user would have been logged in 2. The user will select the find a roommate button 3. The user will be prompted to enter roommate profile information pertaining to lifestyle preferences 4. The user will confirm their entries 5. The user will select submit. 6. A list of matched potential roommates will appear. 7. The user will select and review each individual roommate 8. The user will send a message to a potential roommate. 9. The user will select finish when they have completed the process 10. Roommate matching will be completed		

Alternate Path:	• If the student swipes left, they will move on to the next profile • If the student, no longer wants to participate in the match search they will be able to return to their student dashboard • If there are no more possible profiles the student will be notified and advised to return at a later time.
Pre-Condition:	• The student must be logged into the system • The student must be verified as a Towson University Student • There must be other students seeking roommates • Student profile must be completed

Use Case No:	7	Use Case Name:	Students accessing Towson University resources
Actors:	Student, Towson University Resources integration		
Description:	A student will be able to search and review campus resource information that is available to the students including but not limited to dining, fitness, health, and shuttle resources.		
Basic Course of Events	1. The user would have been logged in 2. The user will select the Towson University Resources button 3. The user will review the list of Towson University Resources and select each resource to review 4. The user will click return to dashboard to complete the process		
Alternate Path:	• If the student, no longer wants to review Towson resources they will be able to return to their Student Dashboard		

Pre-Condition:	• The student must be logged into the system • The student must be verified as a Towson University Student • The Towson resources must be connected to the system.

Use Case No:	8	Use Case Name:	Students accessing local public transport information
Actors:	Student, Towson University Resources integration		
Description:	A student will be able to search and review local public transport information and gather access to the routes, location stops, and time of pick up.		
Basic Course of Events	1. The user would have been logged in 2. The user will select the View Local Public Transport button 3. The user will review the transit information provided by the local transit lines 4. The user will select return to dashboard to complete the process		
Alternate Path:	• If the student, no longer wants to review the public transport information they will be able to return to their Student Dashboard		
Pre-Condition:	• The student must be logged into the system • The student must be verified as a Towson University Student • Public Transportation resources must be connected to the system.		

Use Case No:	9	Use Case Name:	Students Carpool listing
Actors:	Student, Towson University Carpool platform		

Description:	A student will be able to search and review carpool listings by using destination, departure time, pick up location and arrival time. A student will also be able to post carpool listings on the platform with the above information.
Basic Course of Events	1. The user would have been logged in 2. The user will select the find a roommate button 3. The user will be prompted to a. View Postings i. The user will select view postings ii. The user will review and select individual postings iii. If the user would like to be a part of the listed car-pool they will select join iv. The user will be able to return to their dashboard to complete the carpool listings review b. List Postings i. The user will select list postings ii. The user will add the posting information iii. The user will confirm their entries iv. The user will select upload v. The user will be able to return to their dashboard to complete the carpool listings review
Alternate Path:	• If the student, no longer wants to review the carpool listing information they will be able to return to their Student Dashboard • If there are no carpool listings that meet the search criteria a notification will return the notification, and the student will be prompted to return to the dashboard or post a listing.
Pre-Condition:	• The student must be logged into the system • The student must be verified as a Towson University Student • There must be carpool listings

System Requirements

Requirement No:	**1**	**Requirement Name:**	Housing Listing Posting
Introduction	The Housing Listing Posting allows landlords to post available rental properties on the TUSH platform. The TUSH system must support this functionality.		
Inputs	Landlord Account information (Name, contact details, and verification status**),** Property location (city, street, zip code,), Property Type (Apartment, house, shared accommodation), Property Size (Square footage and the number of rooms, (bedrooms, bathrooms), Amount (rent, utilities, maintenance fees), Lease Terms (month-to-month, short-term), Availability, Amenities(parking, laundry, internet) and whether utilities are included), Images and Videos.		
Description	Landlord shall be able to post houses for listing to TUSH. A house listing posting shall be allowed if and only if: -The system must prompt landlords to complete verification -The system must provide a user-friendly form where landlords can input property information -The system must allow landlords to upload multiple images -The system must allow the integration of a map -The system must allow landlords to edit and update listing -The system must validate all inputs to ensure they are correctly formatted -The system must allow landlords to deactivate or repost listings.		
Outputs	Public Property Listing, Preview Function, Listing dashboard and Notifications.		

Requirement No:	2	**Requirement Name:**	Landlord Background Verification
Introduction		The Landlord Verification feature is designed to authenticate and verify landlords who wish to register and list properties on the TUSH system.	
Inputs		Personal details (Full name, contact information, and date of birth), Government Issued ID (scan copy of driver's license, passport), Proof of ownership (property deeds, utility bills), Bank details (optional for verification), Address and Housing registration.	
Description		Landlord Background must be verified for making listing to TUSH. A Landlord verification shall be allowed if and only if: -The system must allow that landlords personal details can be authenticated -The system must allow that landlords issued government ID can be verified -The system must validate all proof of ownership, bank details and housing registration provided by landlords. -The system must allow that all copies of landlords required official documents can be uploaded.	
Outputs		Verification Status update, Verified Landlord Badge and Notification.	

Requirement No:	3	**Requirement Name:**	Landlord Leasing Request and Application View
Introduction	The Leasing Request and Application View enables landlords to view student applications and manage leasing requests for their listed properties on TUSH.		
Inputs	Student name, contact info, student ID, move-in date, rental duration, proof of income, guarantor information, Rental history (optional). Property Listing information (rent amount, availability date, utilities).		
Description	Landlord Leasing Request and Application Review must be completed for making listing application to TUSH. A Landlord Leasing Request and Application Review shall be allowed if and only if: -The system must allow landlords to authenticated student details -The system must ensure the property listing information is always available -The system must allow students to submit application - The system must allow landlords to view all applications submitted -The system must allow landlords to view a detailed profile of each applicant. -The system must allow landlords to filter and search student applications.		
Outputs	Application List, Application Status Updates, Applicant profiles, Lease agreement, Feedback to applicants.		

Requirement No:	4	**Requirement Name:**	Landlord Individual Application Review, Approval and Rejection

Introduction	The Individual Application Review, Approval and Rejection allows landlords to review and approve or reject individual student applications for their properties on TUSH
Inputs	Student application details (full name, contact info, Towson University student ID, proof of income, bank statements, guarantor details, references, rental history), Landlord evaluation criteria (Student ID), and Application status inputs (approval, rejection, or requests for additional information).
Description	Landlord Individual Application Review, Approval and Rejection must be completed for making listing application to TUSH. A Landlord Individual Application Review, Approval and Rejection shall be allowed if and only if: -The system must provide landlords with the ability to view detailed application for each applicant -The system must provide a review dashboard allowing landlords to review applications for each applicant -The system must allow landlord to compare the student's application against criteria. - The system must allow landlords to update the status of each application -The system must allow landlords to request for additional information from each applicant. -The system must allow landlords to filter each student applications. -The system must allow landlords to Accept or Reject application.
Outputs	Application Review Dashboard, Application Status Updates, Request for Additional Information and Lease Agreement Generation and Notification.

Requirem ent No:	5	**Requirem ent Name:**	Student Housing Search

Introduction	The Student Housing Search allows students to search for available rental properties listed by landlords on the TUSH platform.
Inputs	Location, Property Type, Amenities, Price Range, Number of Bedrooms/Bathrooms, Move-in-Date, Lease Duration and Distance from School.
Description	Student Housing Search must be completed for making listing application to TUSH. A Student Housing search shall be allowed if and only if: -The system must allow students to perform basic search for available properties by entering keywords or selecting a location -The system must allow students to perform an advanced search by applying multiple filters, such as price range, number of rooms etc. -The system must allow students to sort search results by criteria such as price - The system must allow students to view available properties on a map, with pins indicating the locations of properties -The system should allow students to save searches based on their filter preferences -The system should allow students to mark properties as favorites for easy access later -The system should ensure AI/ML is used to enhance a good search experience
Outputs	Search Results, Detailed Property View, Favorites List, Map View, Notification and Contact Form.

Requirement No:	6	**Requirement Name:**	Roommate Matching System

Introduction	The Roommate Matching System will allow for students to match with other students based upon lifestyle information provided to TUSH.
Inputs	Student Account Information, like name, age, lifestyle preferences i.e. optionally, eating/sleeping/working/social habits, if the student already has property leased
Description	The Roommate Matching System will allow students to be able to match up with one another in order to become roommates in a property that they will be leasing. A Roommate Matching Search and Connection should be allowed if and only if: -The system must allow for students to be able to apply multiple filters to search for specific roommates based upon filters like age, name, certain lifestyle habits -The system should allow for saving filter preferences for ease of search -The system should give recommendations based upon saved filters -The system should provide a search history -The system must allow for students to view each other's profiles -The system should allow for students to contact one another, i.e. by allowing for access to their contact information -The system should allow students to add each other and approve the add to become roommates - The system should ensure AI/ML is used to support roommate matching based on students' preference
Outputs	Added Roommates, Favorited Filters, Search Results, Possible Roommate Information, Contact Information

Requirem ent No:	**7**	**Requirem ent Name:**	Students accessing Towson University resources
Introduction	A verified user should be able to access Towson University Resources through the system, being able to check things like dining information, fitness information, shuttle information, parking information.		
Inputs	System Account Information, Verification via Towson Login, searching for Towson Resources		
Description	A user verified Towson University student can search for basic Towson resources that would be useful for planning living conditions. Towson University Resources can be accessed by the application if and only if: -The user is logged in and verified as a Towson University Student -The system must provide basic information based on the Towson University resources the user is accessing -The system must provide information only relevant to possible decisions for living conditions -The system should allow for ease of returning to their Dashboard after accessing resources		
Outputs	Dining locations and costs, Shuttle times and pickup locations, parking pass times and costs, fitness information, library information, redirects to Towson University's official site		

Requirem ent No:	**8**	**Requirem ent Name:**	Students accessing local public transport information

Introduction	A verified student should be able to check local transport information, for example, like the MTA. The student can see pickup times, routes, pickup and drop off locations, delays.
Inputs	Location, Public Transport Name, Type of Public Transport, Account Information, Towson University Verification
Description	A verified Towson University Student can search for local public transport information. The Local Public Transport Information can be accessed if and only if: -The student is logged in and verified to be a Towson University Student - The system must allow for the user to apply filters based upon the name of the public transport, the type of public transport, the times they pick up and drop off, where each public transport picks up and drops off -The system must allow for redirects to available websites for the public transport -The system should make recommendations based upon affiliations with Towson University
Outputs	Delay Information, Pickup Times and locations, routes, public transport names and types

Requirement No:	9	**Requirement Name:**	Students Carpool listing

Introduction	Students should be able to create a carpool only accessible by those who are logged in to the system, providing information such as departure times, arrival times, destination, and meetup location. Students can then search for these and possibly flag that they will be joining the carpool.
Inputs	Student Account Information, Towson University Verification, Departure/Arrival Times, Destination, Meetup Location, Flagging to Join, amount of people allowed in the carpool
Description	A verified student can create a carpool, where only other verified students can access its information and join it. The Student Carpool Listing can be created or searched for if and only if: -The student creating the carpool is a verified Towson University Student -The students viewing the carpool are verified Towson University Students -The system must allow for searches based upon, departure/ arrival times, destination/meetup locations, how many people can join the carpool -The system must allow for students to join the carpool -The system must allow for the student who created the carpool to deny/remove students from the carpool -The system should make a recommendation when a roommate is attending a carpool -The system should allow for a student to delete a carpool they created
Outputs	Accepted and Denials for Carpool, Added/Removal from Carpool, Generation of Carpool, Deletion/creation of a carpool, recommendations for carpool, carpool searches

Use Case Diagrams

Class Diagrams

System Modelling

Major Use Cases for the TUSH System

- Use Case 1: Landlord Posts a Property Listing
- Use Case 2: Student Applies for a Property

Use Case 1: Landlord Posts a Property Listing

- **Analysis**

  Lifelines:

  1. **Landlord**
  2. **PropertyListingSystem**
  3. **VerificationSystem**
  4. **PropertyDatabase**

  Diagram Steps:

  1. **Landlord** activates the **PropertyListingSystem** to start the posting process **(postListing ()).**
  2. **PropertyListingSystem** requests **VerificationSystem** to verify the landlord's identity **(verifyLandlord ()).**
     - o **alt fragment**: If verification is successful, the system allows the process to proceed; if not, the process is terminated.
  3. **Landlord** inputs property details **(inputPropertyDetails ()).**
  4. **PropertyListingSystem** validates property inputs **(validateProperty ()).**
     - o **loop fragment**: The system checks the details until all inputs are valid.
  5. **Landlord** uploads images and videos **(uploadMedia ()).**
  6. **PropertyListingSystem** sends property details and media to the **PropertyDatabase** for storage **(storeProperty ()).**
  7. **PropertyDatabase** confirms the successful storage (confirmListing ()).
  8. **PropertyListingSystem** sends a confirmation message to the **Landlord** **(sendConfirmation()).**

- **Use Case 2: Student Applies for a Property**

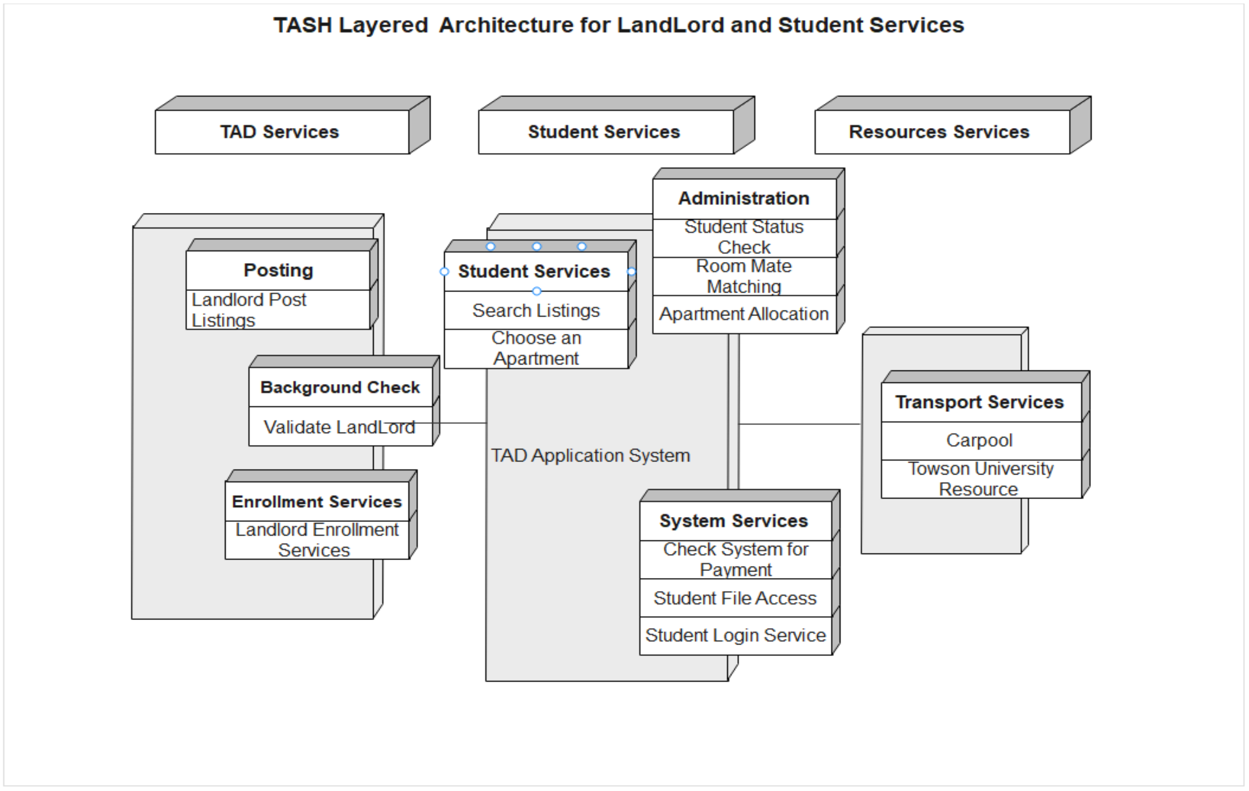
  Lifelines:

  1. Student
  2. SearchSystem
  3. PropertyListingSystem
  4. ApplicationManagementSystem
  5. Landlord

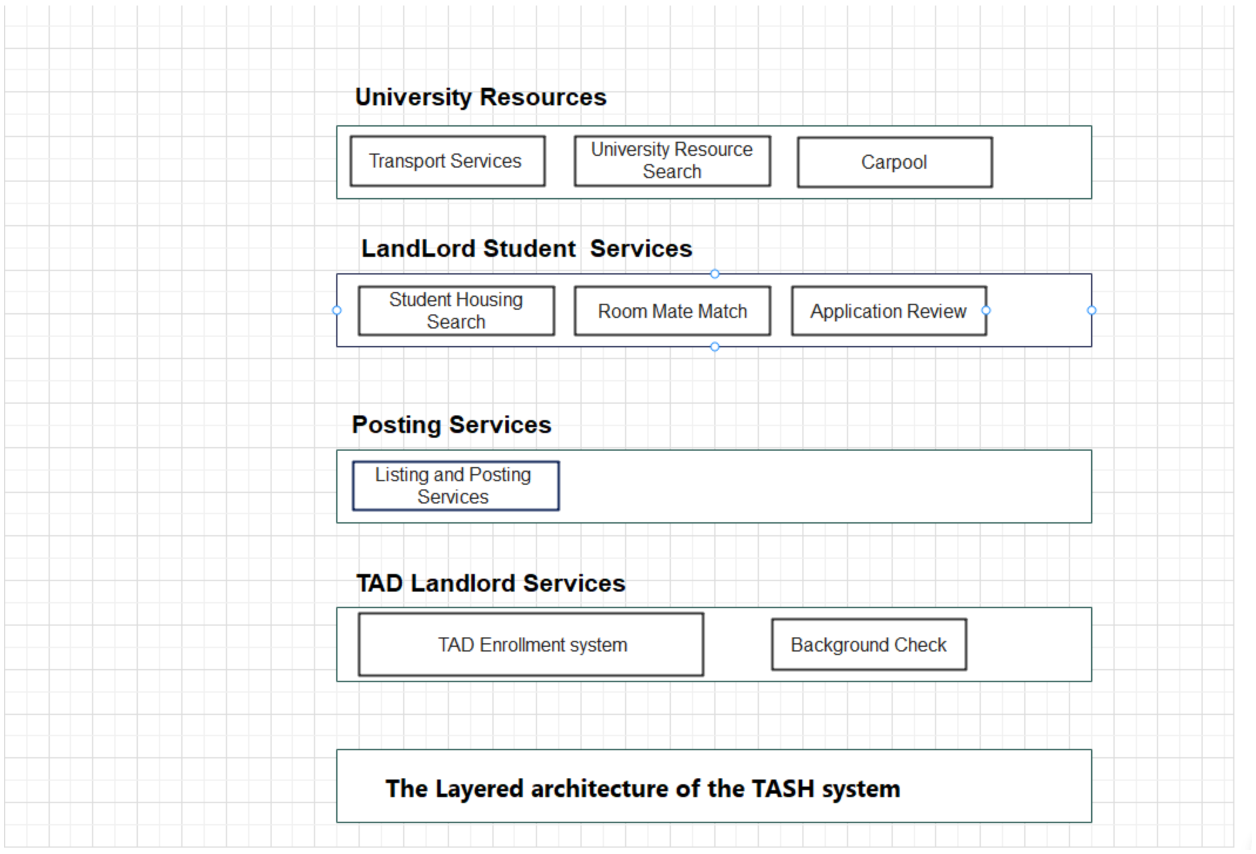
  Diagram Steps:

  1. **Student** searches for properties via the **SearchSystem (searchProperty ()).**
  2. **SearchSystem** retrieves property listings from the **PropertyListingSystem (retrieveListings ()).**
  3. **Student** selects a property and applies **(applyForProperty ()).**
  4. **ApplicationManagementSystem** collects the student application details **(collectApplicationDetails ()).**
  5. **ApplicationManagementSystem** verifies student details and application completeness **(verifyApplication ()).**
     - **alt fragment**: If the application is incomplete, the system requests additional information from the **Student**.
  6. **ApplicationManagementSystem** submits the application to the **Landlord (submitApplicationToLandlord ()).**
  7. **Landlord** reviews the application **(reviewApplication ()).**
  8. **Landlord** sends feedback, either **approval** or **rejection** via the **ApplicationManagementSystem (updateApplicationStatus ()).**
     - **alt fragment**: If the application is accepted, a lease agreement is generated.

C:\Users\asied\Documents\Software Engineering\Applicaton Program Architecture---.eddx

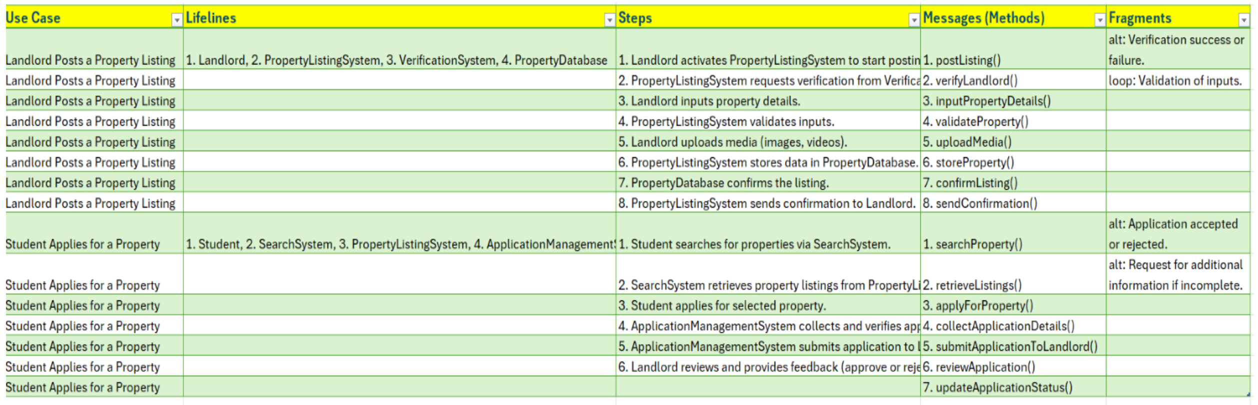


The Layered architecture of the TASH system



<u>C:\Users\asied\Documents\Software Engineering\TUSH_Sequence_Diagrams_Details.xlsx</u>

TUSH: Sequence Diagrams Details Base on the Use Cases



Use Case	Lifelines	Steps	Messages (Methods)	Fragments
Landlord Posts a Property Listing	1. Landlord, 2. PropertyListingSystem, 3. VerificationSystem, 4. PropertyDatabase	1. Landlord activates PropertyListingSystem to start postin	1. postListing()	alt: Verification success or failure.
Landlord Posts a Property Listing		2. PropertyListingSystem requests verification from Verifica	2. verifyLandlord()	loop: Validation of inputs.
Landlord Posts a Property Listing		3. Landlord inputs property details.	3. inputPropertyDetails()	
Landlord Posts a Property Listing		4. PropertyListingSystem validates inputs.	4. validateProperty()	
Landlord Posts a Property Listing		5. Landlord uploads media (images, videos).	5. uploadMedia()	
Landlord Posts a Property Listing		6. PropertyListingSystem stores data in PropertyDatabase.	6. storeProperty()	
Landlord Posts a Property Listing		7. PropertyDatabase confirms the listing.	7. confirmListing()	
Landlord Posts a Property Listing		8. PropertyListingSystem sends confirmation to Landlord.	8. sendConfirmation()	
Student Applies for a Property	1. Student, 2. SearchSystem, 3. PropertyListingSystem, 4. ApplicationManagement	1. Student searches for properties via SearchSystem.	1. searchProperty()	alt: Application accepted or rejected.
Student Applies for a Property		2. SearchSystem retrieves property listings from PropertyLi	2. retrieveListings()	alt: Request for additional information if incomplete.
Student Applies for a Property		3. Student applies for selected property.	3. applyForProperty()	
Student Applies for a Property		4. ApplicationManagementSystem collects and verifies app	4. collectApplicationDetails()	
Student Applies for a Property		5. ApplicationManagementSystem submits application to l	5. submitApplicationToLandlord()	
Student Applies for a Property		6. Landlord reviews and provides feedback (approve or reje	6. reviewApplication()	
Student Applies for a Property			7. updateApplicationStatus()	

Architectural Patterns Related to the TUSH System

Layered Architecture

- This pattern applies to the system's overall structure, as it separates functionalities into distinct layers:
    - **Presentation Layer**: The interface with the landlord, including **postListing ()** and **inputPropertyDetails ().**
    - **Business Logic Layer**: Logic within **Property Listing System**, handling the core process flow and validation **(validateProperty ()).**
    - **Service Layer**: This encompasses interactions with the **Verification System** and PropertyDatabase for verification and data storage.

Client-**Server Architecture**

- This pattern is relevant in the Landlord and Property Listing System interaction. Landlord acts as a client initiating requests, while **Property Listing System** acts as a server, processing these requests and interacting with other subsystems on behalf of the client.

Repository Architecture

- The PropertyDatabase utilizes a repository pattern. It serves as a central database where all property details and media are stored and retrieved. The **Property Listing System** interacts with the PropertyDatabase to persist data using a clear separation of data storage and business logic.

Pipe-and-Filter Architecture

- This pattern can be seen in the validation steps. The **Property Listing System** acts as a filter, validating each part of the landlord's input **(validateProperty ())** in a loop until all required data is correctly provided. Similarly, media files are uploaded and processed in steps, which could be managed using individual filters for various media types or quality checks before storage.

Diagram Steps and Patterns in Action

Step 1 (Landlord activates PropertyListingSystem): **Client-Server** pattern, where Landlord (client) initiates a request to start the posting process on PropertyListingSystem (server).
Step 2 (PropertyListingSystem requests VerificationSystem to verify Landlord): **Layered** and **Client-Server** patterns, with PropertyListingSystem acting as a client to the VerificationSystem (server) to perform an external verification step.
Step 3 & 4 (Landlord inputs property details and PropertyListingSystem validates): **Pipe-and-Filter** pattern within the validation process (validateProperty ()) to ensure data integrity through multiple checks.
Step 5 (Landlord uploads media): **Pipe-and-Filter** pattern, allowing sequential processing and checking of media uploads.
Step 6 (Sending property data to PropertyDatabase for storage): **Repository** pattern where PropertyListingSystem writes property data to the PropertyDatabase.
Step 7 & 8 (PropertyDatabase confirms storage and sends confirmation to Landlord): **Layered** and **Client-Server** patterns, enabling communication from the repository back to the Landlord through the PropertyListingSystem

Visual presentations of the TUSH System

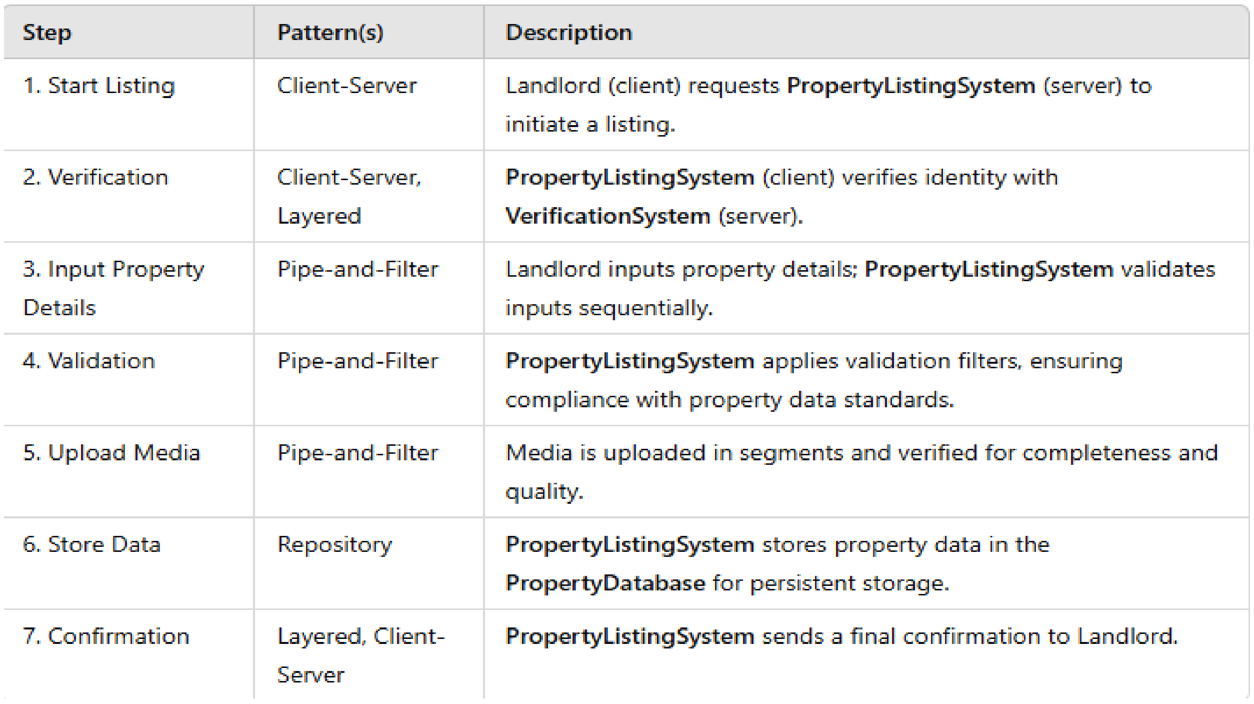
- **System Architecture for Property Listing Platform**
    - Architectural Patterns and Sequence Diagram for Landlord Property Posting Process
        - **Summary of the Architectural Patterns**
            - Brief descriptions of each architectural pattern applied
                - **Layered Architecture:** Separation of concerns across distinct layers**.**
                - **Client-Server Architecture:** Communication between the landlord and system.
                - **Repository Architecture:** Centralized storage for property details and media**.**
                - **Pipe-and-Filter Architecture:** Sequential processing in validation and media upload
        - **System Components**
            - System Components and Lifelines

- Component and roles
    - **Landlord**
    - **PropertyListingSystem**
    - **VerificationSystem**
    - **PropertyDatabase**

- **Property Posting Process and Sequence for TUSH System**
  - Process
    1. Landlord initiates **postListing ()** on **PropertyListingSystem**.
    2. **PropertyListingSystem** calls **verificationSystem** to verify.
        1. **alt fragment**: Verification success/failure conditions.
    3. Landlord inputs property details.
    4. **PropertyListingSystem** validates inputs with a loop fragment until valid.
    5. **Landlord** uploads media.
    6. PropertyListingSystem stores details and media in **PropertyDatabase**.
    7. **PropertyDatabase** confirms storage.
    8. **PropertyListingSystem** sends confirmation to Landlord.

- **Architectural Patterns in Action referencing the numbered steps above**
  - Architectural Patterns Applied to Each Step
    - Layered: **Steps 1, 4, 8** for processing and validation.
    - Client-Server: **Steps 1, 2, 6** for client-initiated interactions.
    - Repository: **Step 6** for property storage.
    - Pipe-and-Filter: **Step 4** for validation and **Step 5** for media uploads

The breakdown of how the architectural patterns apply to each step in the process



Step	Pattern(s)	Description
1. Start Listing	Client-Server	Landlord (client) requests **PropertyListingSystem** (server) to initiate a listing.
2. Verification	Client-Server, Layered	**PropertyListingSystem** (client) verifies identity with **VerificationSystem** (server).
3. Input Property Details	Pipe-and-Filter	Landlord inputs property details; **PropertyListingSystem** validates inputs sequentially.
4. Validation	Pipe-and-Filter	**PropertyListingSystem** applies validation filters, ensuring compliance with property data standards.
5. Upload Media	Pipe-and-Filter	Media is uploaded in segments and verified for completeness and quality.
6. Store Data	Repository	**PropertyListingSystem** stores property data in the **PropertyDatabase** for persistent storage.
7. Confirmation	Layered, Client-Server	**PropertyListingSystem** sends a final confirmation to Landlord.

Lifelines and Sequence of Actions

Below is the sequence of actions between these components during a property listing process. Each action represents a **message** or interaction along the **lifelines** of each component.

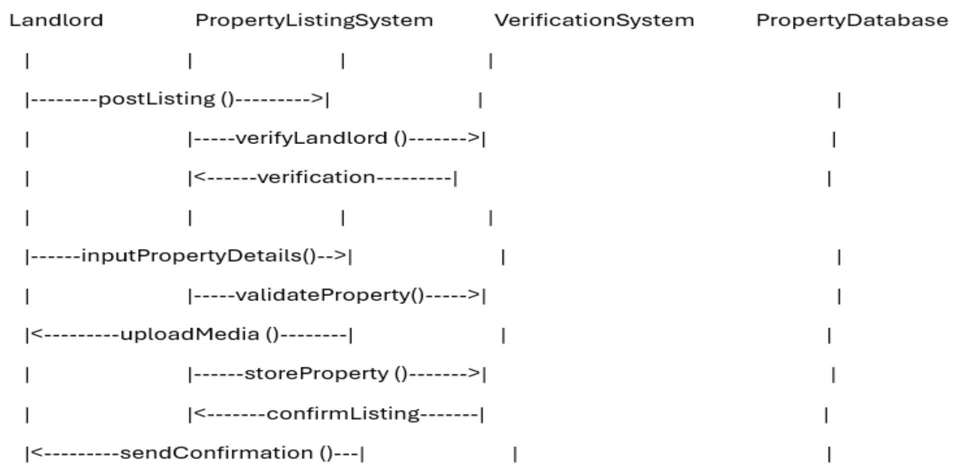
- **Landlord initiates listing**:
  - Action: postListing()
  - Lifeline Interaction: **Landlord ➔ PropertyListingSystem**

- **Verification request**:
  - Action: verifyLandlord()
  - Lifeline Interaction: **PropertyListingSystem ➔ VerificationSystem**
  - **alt fragment**:
    - **Success**: Verification successful; the process continues.
    - **Failure**: Verification fails; process terminates.

- **Property details input**:
  - Action: inputPropertyDetails()
  - Lifeline Interaction: **Landlord ➔ PropertyListingSystem**

- **Property details validation**:
    - Action: validateProperty()
    - Lifeline Interaction: **PropertyListingSystem** self-validation process.
    - **loop fragment**: Continues until all details are validated.
- **Media upload**:
    - Action: uploadMedia()
    - Lifeline Interaction: **Landlord ➔ PropertyListingSystem**
- **Store property data**:
    - Action: storeProperty()
    - Lifeline Interaction: **PropertyListingSystem ➔ PropertyDatabase**
- **Confirmation of storage**:
    - Action: confirmListing()
    - Lifeline Interaction: **PropertyDatabase ➔ PropertyListingSystem**
- **Confirmation to landlord**:
    - Action: sendConfirmation()
    - Lifeline Interaction: **PropertyListingSystem ➔ Landlord**

Diagram (High-Level) – **TUSH System**: Sequence

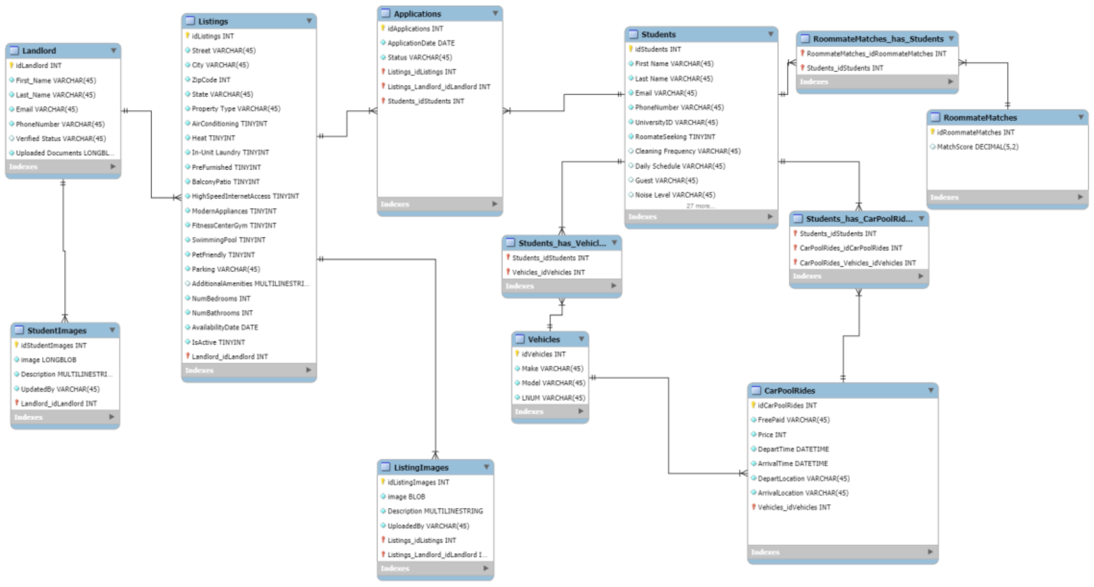


```
Landlord          PropertyListingSystem       VerificationSystem       PropertyDatabase
  |                  |                  |                  |
  |--------postListing ()--------->|                  |                                    |
  |                  |-----verifyLandlord ()------->|                                    |
  |                  |<------verification---------|                                     |
  |                  |                  |                  |
  |------inputPropertyDetails()-->|                  |                                    |
  |                  |-----validateProperty()----->|                                     |
  |<---------uploadMedia ()--------|                  |                                    |
  |                  |------storeProperty ()------->|                                    |
  |                  |<-------confirmListing-------|                                     |
  |<---------sendConfirmation ()---|                  |                                    |
```

High-level UML Diagram for TUSH Property Listing Platform

Implementation

Database: MySQL, Hosted on AmazonRDS



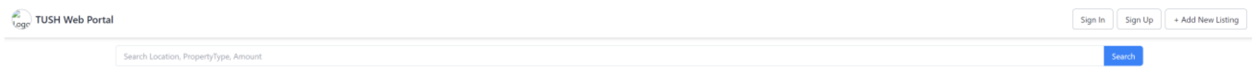
Front End/UI

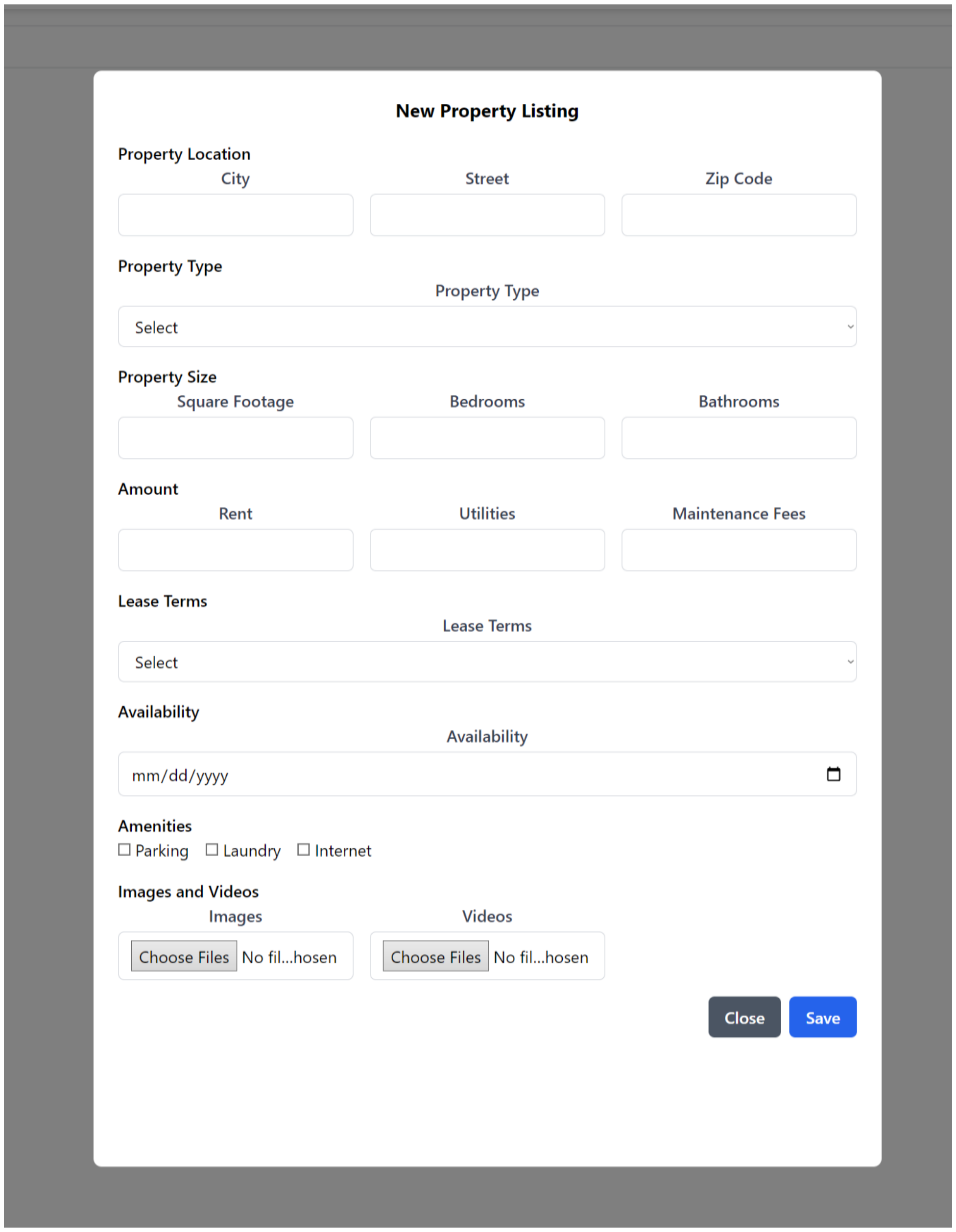
Utilized React/Javascript

Implementing Use Cases 1 and 5

Still under development, we ran into issues trying to connect the frontend to the database, if not finished by end of Sprint, i.e end of day 10/28, it will be implemented shortly into Sprint 4.

To run the application, simply cd into the application's directory and type "npm start". You may need to install node modules to run it, so while in the directory run "npm install" if that is the case. Ensure you have node.js installed as well.


Logo
TUSH Web Portal
Sign In
Sign Up
+ Add New Listing
Search Location, PropertyType, Amount
Search


New Property Listing

Property Location

City	Street	Zip Code

Property Type

Property Type

Select

Property Size

Square Footage	Bedrooms	Bathrooms

Amount

Rent	Utilities	Maintenance Fees

Lease Terms

Lease Terms

Select

Availability

Availability

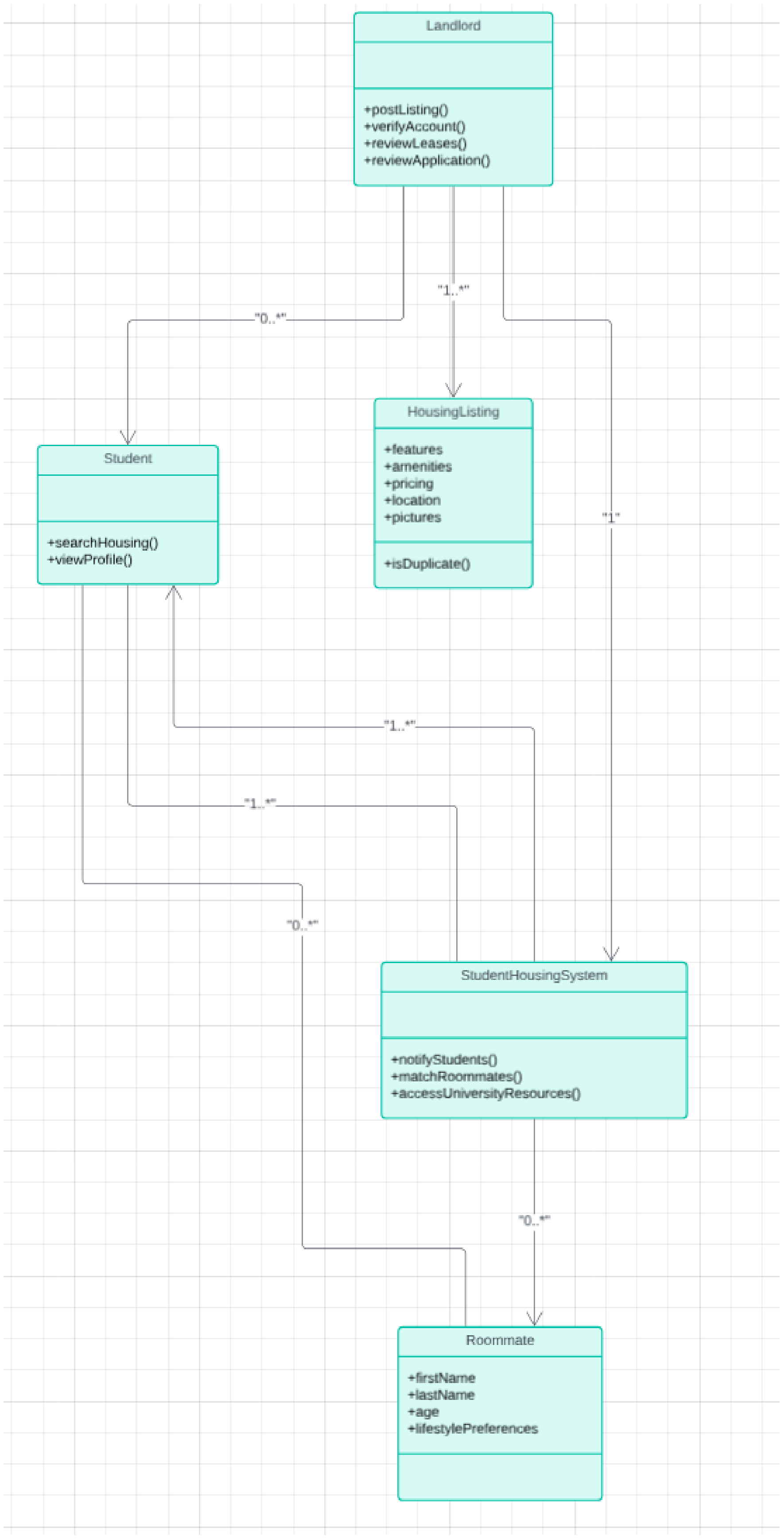
mm/dd/yyyy

Amenities

☐ Parking ☐ Laundry ☐ Internet

Images and Videos

Images	Videos
Choose Files No fil...hosen	Choose Files No fil...hosen



Landlord
+postListing()
+verifyAccount()
+reviewLeases()
+reviewApplication()
"0..*"
"1..*"
"1"
HousingListing
+features
+amenities
+pricing
+location
+pictures
+isDuplicate()
Student
+searchHousing()
+viewProfile()
"1..*"
"1..*"
"0..*"
StudentHousingSystem
+notifyStudents()
+matchRoommates()
+accessUniversityResources()
"0..*"
Roommate
+firstName
+lastName
+age
+lifestylePreferences

Testing

Test Case for Use Case 1: Landlord Posts a Property Listing

Feature: Post a property listing

Use Case: Landlord Posts a Property Listing

Test Specification

Verify Landlord Identity

Equivalence Classes for Verification:

Successful Verification (valid credentials provided)

Unsuccessful Verification (invalid credentials provided)

Test Case:

Test ID: TC1.1

Input: Valid landlord credentials

Expected Result: Verification successful, process proceeds to next step (input property details)

Input Property Details

Equivalence Classes for Property Details Validation:

Valid details (e.g., correct format, required fields filled)

Invalid details (e.g., missing or incorrect format in one or more fields)

Test Case:

Test ID: TC1.2

Input: Valid property details (e.g., address, price, description, type)

Expected Result: Details validated successfully, proceeds to upload media

Upload Media

Equivalence Classes for Media Upload:

Valid media (e.g., images and videos in supported formats)

Invalid media (e.g., unsupported file type, large file size)

Test Case:

Test ID: TC1.3

Input: Supported image and video files within the size limit

Expected Result: Media uploaded successfully, property details are sent to Property Database

Store Property in Database

Equivalence Classes for Storage Confirmation:

Successful storage confirmation

Failed storage (e.g., database connectivity issue)

Test Case:

Test ID: TC1.4

Input: All validated property details and media files

Expected Result: Property listing stored successfully, confirmation sent to landlord

Test Case for Use Case 2: Student Applies for a Property

Feature: Apply for a property

Use Case: Student Applies for a Property

Test Specification

Search Property

Equivalence Classes for Search Input:

Valid search criteria (e.g., location, price range, property type)

Invalid search criteria (e.g., empty search, invalid location)

Test Case:

Test ID: TC2.1

Input: Valid search criteria (e.g., location, budget range)

Expected Result: Search System retrieves a list of matching property listings from Property Listing System

Apply for Property

Equivalence Classes for Property Application:

Valid application (e.g., all required fields filled accurately)

Invalid application (e.g., missing information in one or more fields)

Test Case:

Test ID: TC2.2

Input: Complete and accurate application details

Expected Result: Application details are collected by Application Management System

Verify Application Details

Equivalence Classes for Application Verification:

Complete application (all required details present)

Incomplete application (missing or incorrect details)

Test Case:

Test ID: TC2.3

Input: Complete application (no missing fields)

Expected Result: Application verified as complete, proceeds to submission to the landlord

Submit Application to Landlord

Equivalence Classes for Submission:

Successful submission to landlord

Failed submission (e.g., network issue, landlord unavailable)

Test Case:

Test ID: TC2.4

Input: Verified application details ready for submission

Expected Result: Application successfully submitted to landlord

Update Application Status

Equivalence Classes for Status Update:

Application accepted (lease agreement generated)

Application rejected

Test Case:

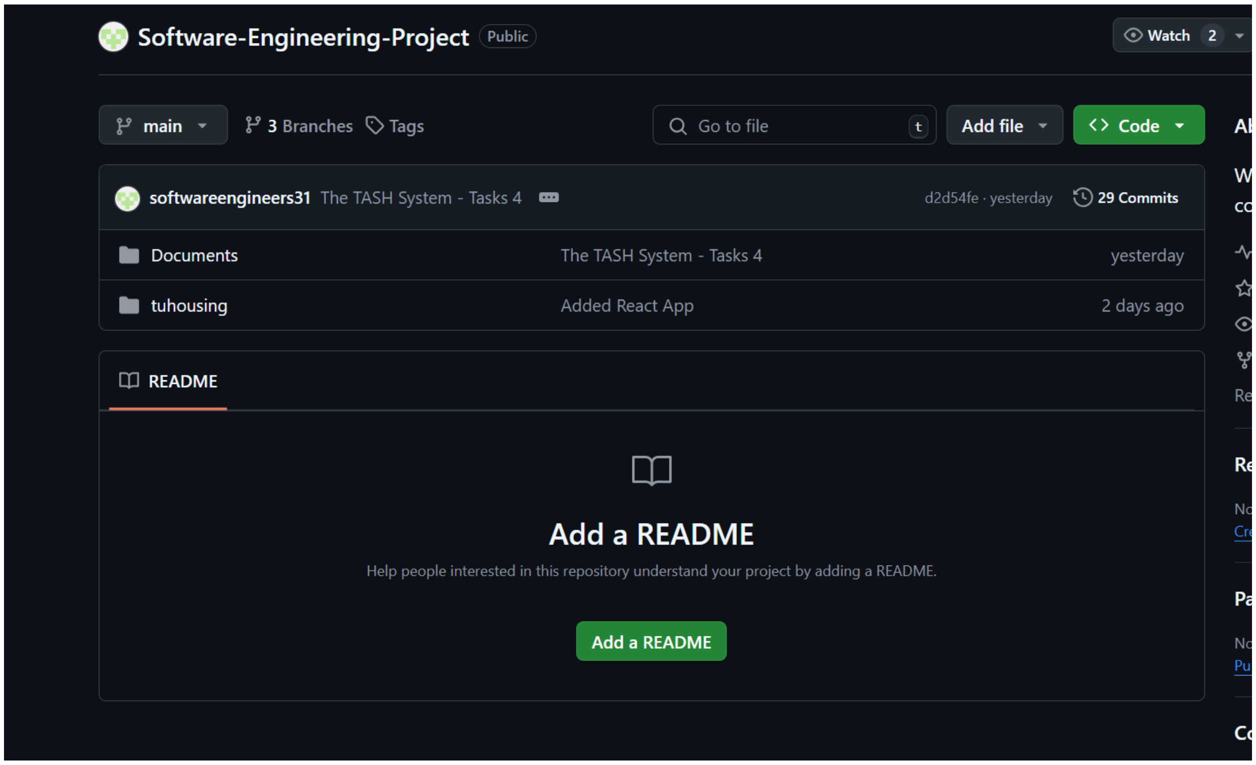
Test ID: TC2.5

Input: Landlord reviews and accepts the application

Expected Result: Application Management System updates application status to accepted, generates lease agreement

Appendix

https://github.com/softwareengineers31/Software-Engineering-Project



Software-Engineering-Project Public

Watch 2

main · 3 Branches · Tags · Go to file · Add file · Code

softwareengineers31 The TASH System - Tasks 4 · d2d54fe · yesterday · 29 Commits

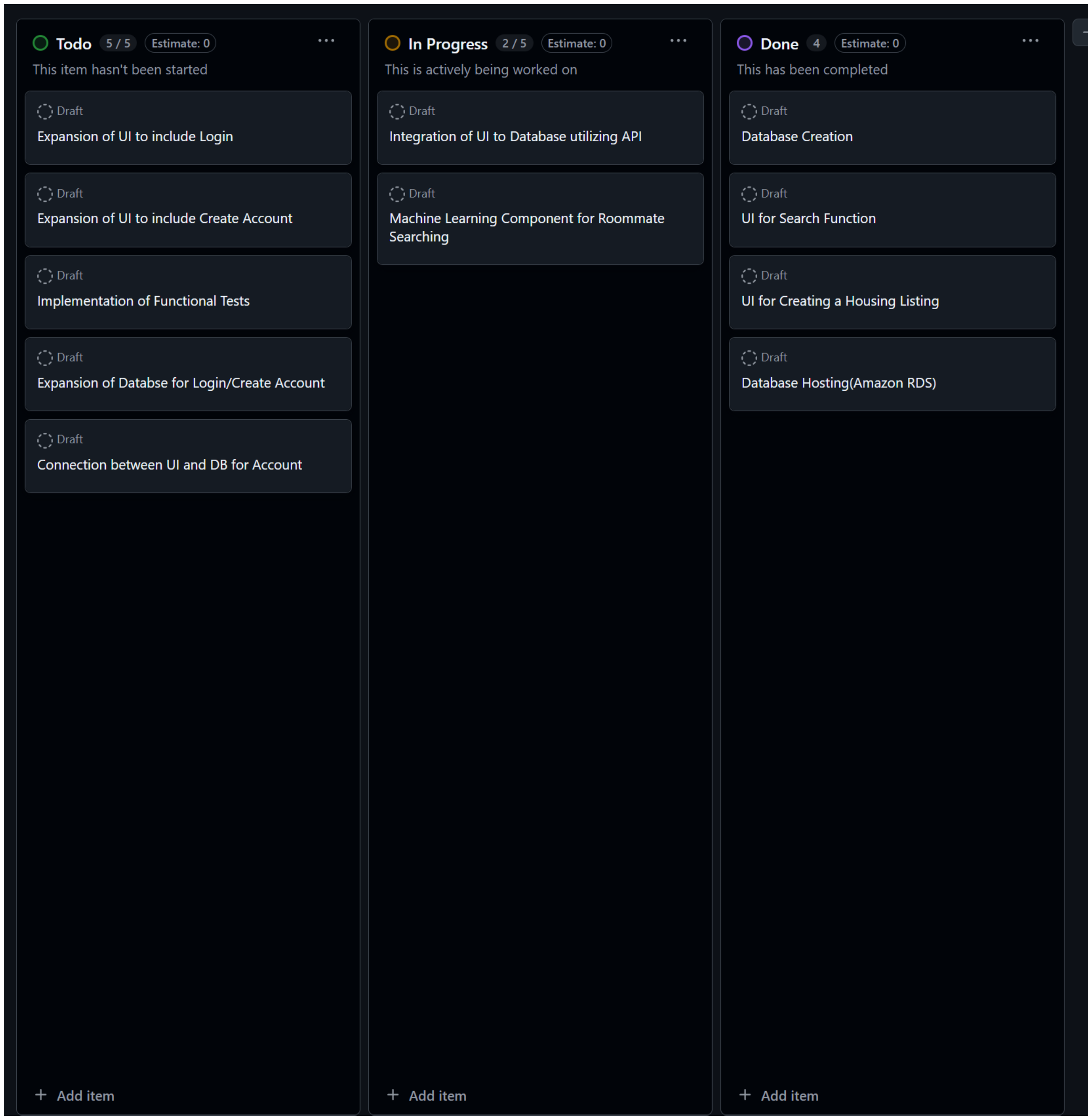
Documents	The TASH System - Tasks 4	yesterday
tuhousing	Added React App	2 days ago

README

Add a README

Help people interested in this repository understand your project by adding a README.

Add a README



Todo 5 / 5 Estimate: 0

This item hasn't been started

Draft
Expansion of UI to include Login

Draft
Expansion of UI to include Create Account

Draft
Implementation of Functional Tests

Draft
Expansion of Databse for Login/Create Account

Draft
Connection between UI and DB for Account

In Progress 2 / 5 Estimate: 0

This is actively being worked on

Draft
Integration of UI to Database utilizing API

Draft
Machine Learning Component for Roommate Searching

Done 4 Estimate: 0

This has been completed

Draft
Database Creation

Draft
UI for Search Function

Draft
UI for Creating a Housing Listing

Draft
Database Hosting(Amazon RDS)