**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 ()).**
   * **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 ()).**
   * **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](file:///C:\Users\asied\Documents\Software%20Engineering\Applicaton%20Program%20Architecture---.eddx)

A diagram of a service

Description automatically generated

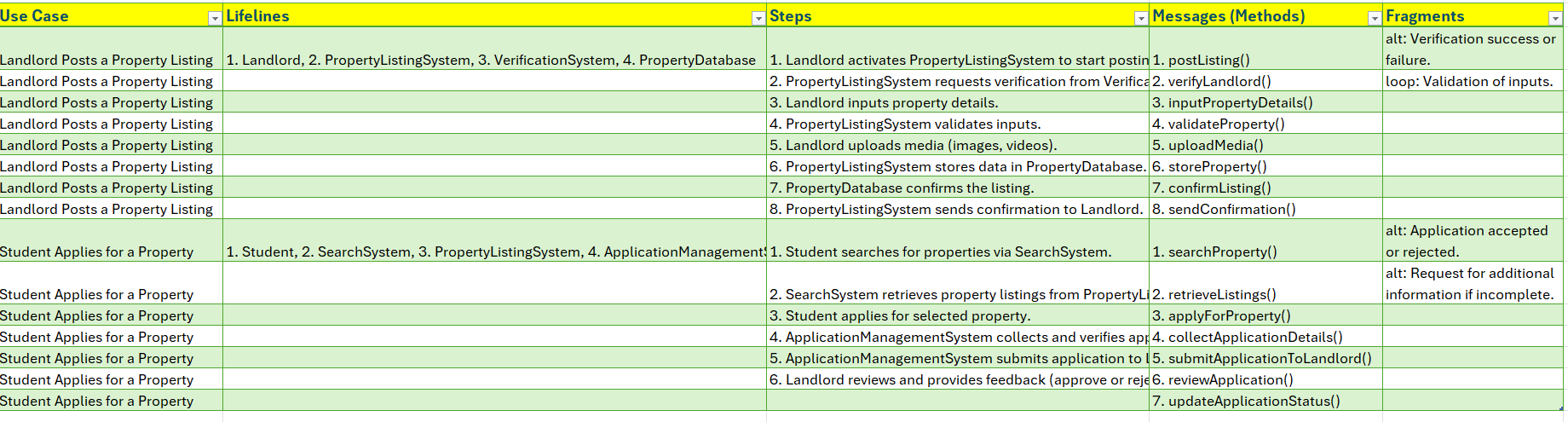
**The Layered architecture of the TASH system**

**A screenshot of a computer

Description automatically generated**

[C:\Users\asied\Documents\Software Engineering\TUSH\_Sequence\_Diagrams\_Details.xlsx](file:///C:\Users\asied\Documents\Software%20Engineering\TUSH_Sequence_Diagrams_Details.xlsx)

**TUSH: Sequence Diagrams Details Base on the Use Cases**



**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

Landlord initiates **postListing ()** on **PropertyListingSystem**.

**PropertyListingSystem** calls **verificationSystem** to verify.

**alt fragment**: Verification success/failure conditions.

Landlord inputs property details.

**PropertyListingSystem** validates inputs with a loop fragment until valid.

**Landlord** uploads media.

PropertyListingSystem stores details and media in **PropertyDatabase**.

**PropertyDatabase** confirms storage.

**PropertyListingSystem** sends confirmation to Landlord.

* + - * **Architectural Patterns in Action referencing the numbered steps above**
        + Architectural Patterns Applied to Each Step

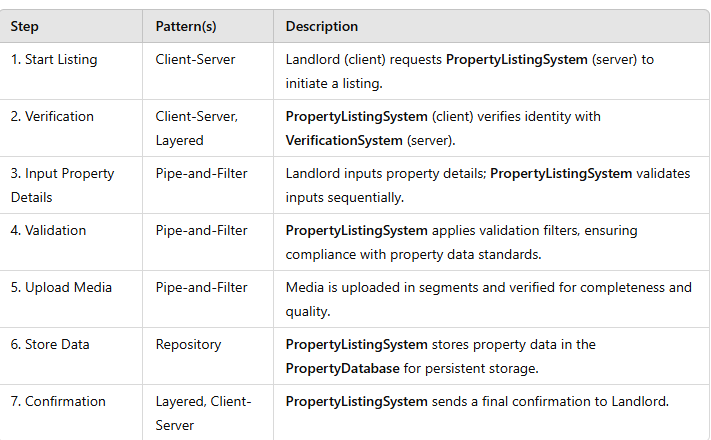
Layered: **Steps 1, 4, 8** for processing and validation.

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**

****

**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



**High-level UML Diagram** for TUSH Property Listing Platform

A model of a house surrounded by many white buttons

Description automatically generated