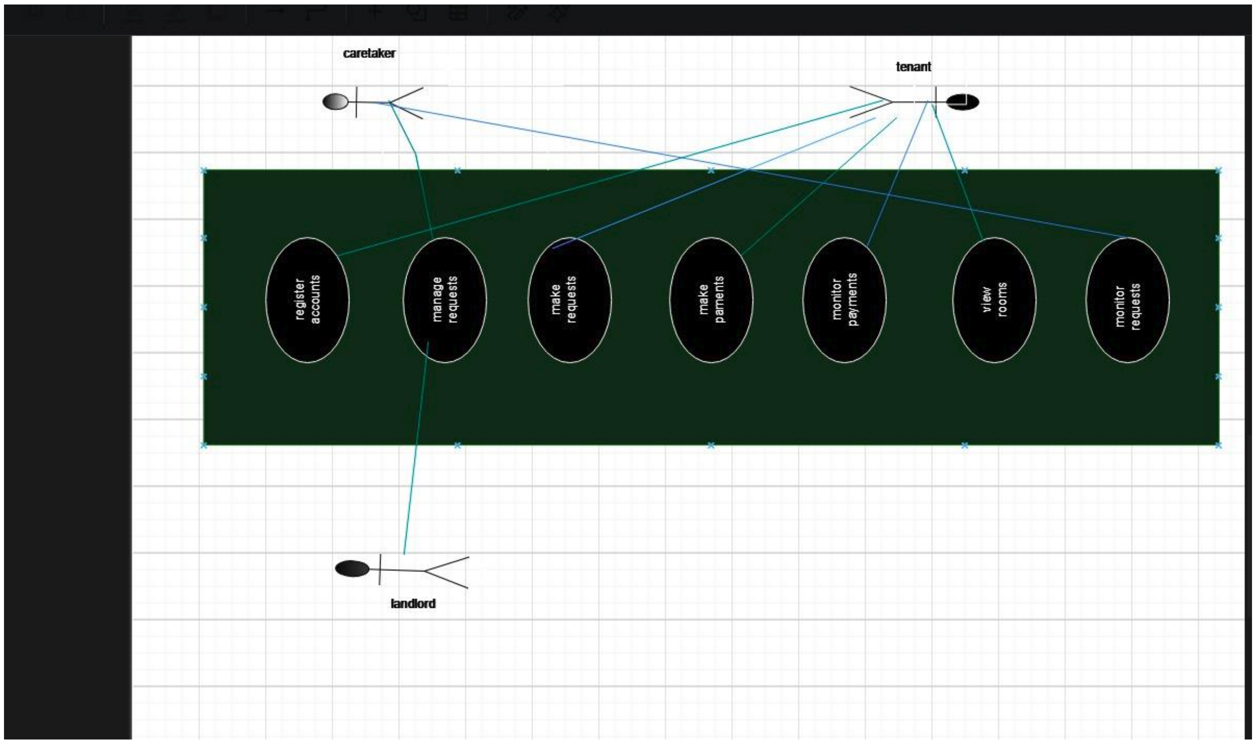
SYSTEM DESIGN WITH UML DIAGRAMS.

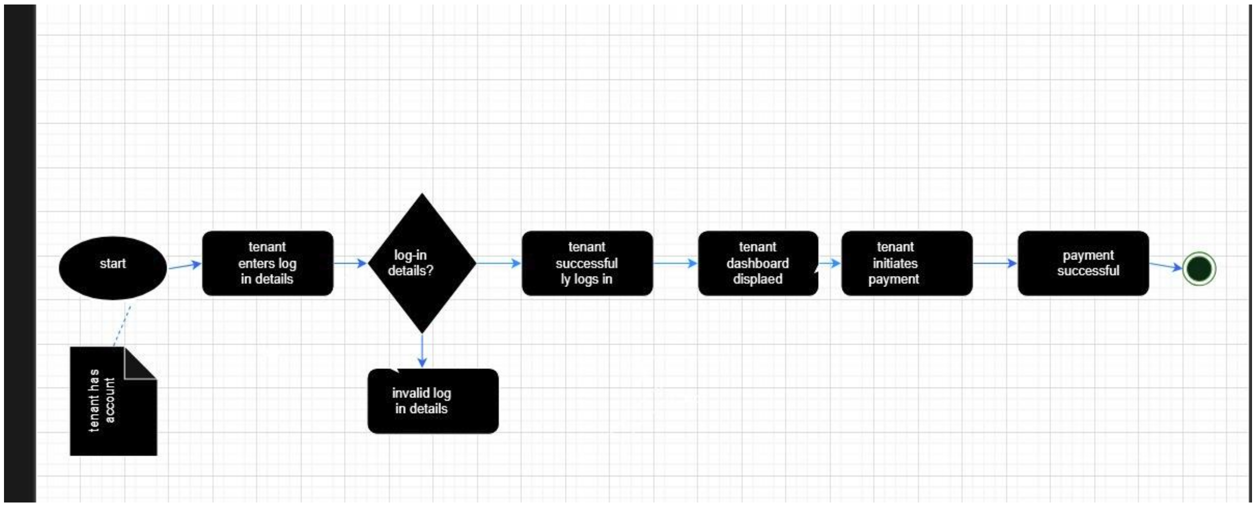
1. Use case diagram :

The diagram shows how different users interact with the hostel management system. The tenant, caretaker, and landlord are the main users. They can perform actions such as registering accounts, making and managing requests, viewing rooms, making payments, and monitoring payments or requests. Each user is connected to the system functions they are allowed to use.



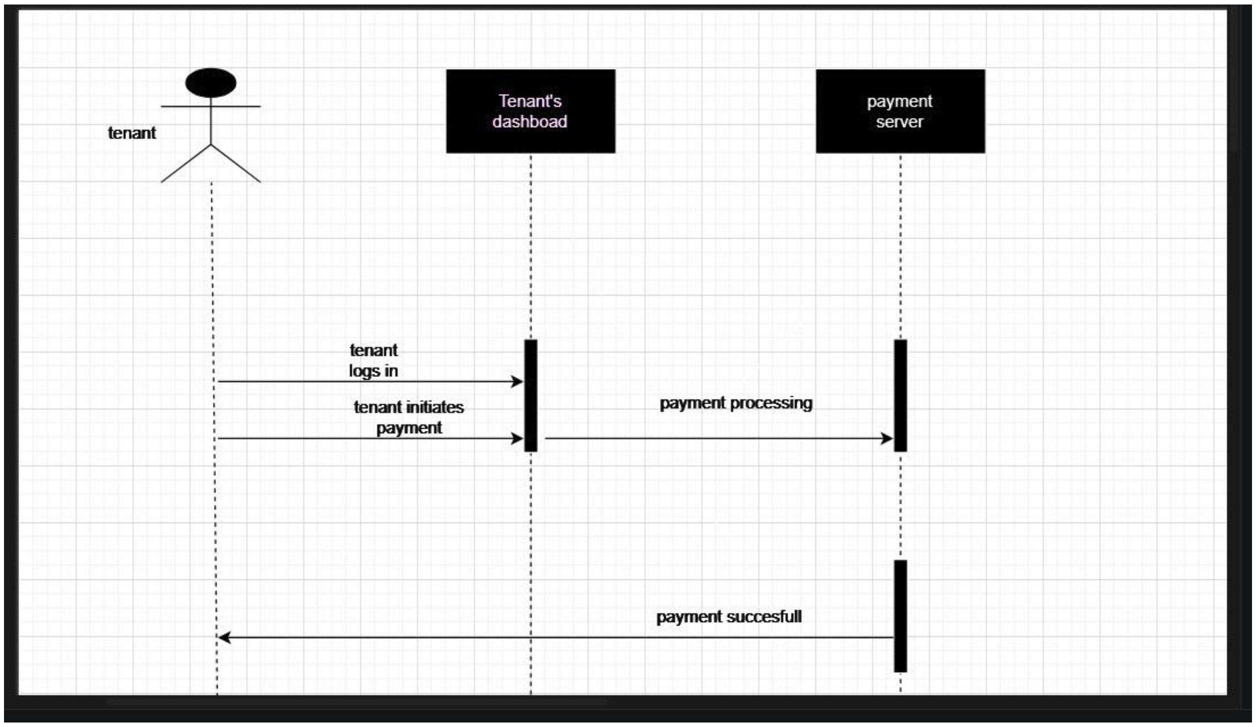
2. Activity diagram:

This diagram shows the tenant's payment process. The process starts when the tenant enters login details. If the details are correct, the tenant logs in successfully, views the dashboard, initiates a payment, and the payment is completed. If the login details are incorrect, the process goes to an invalid login stage.



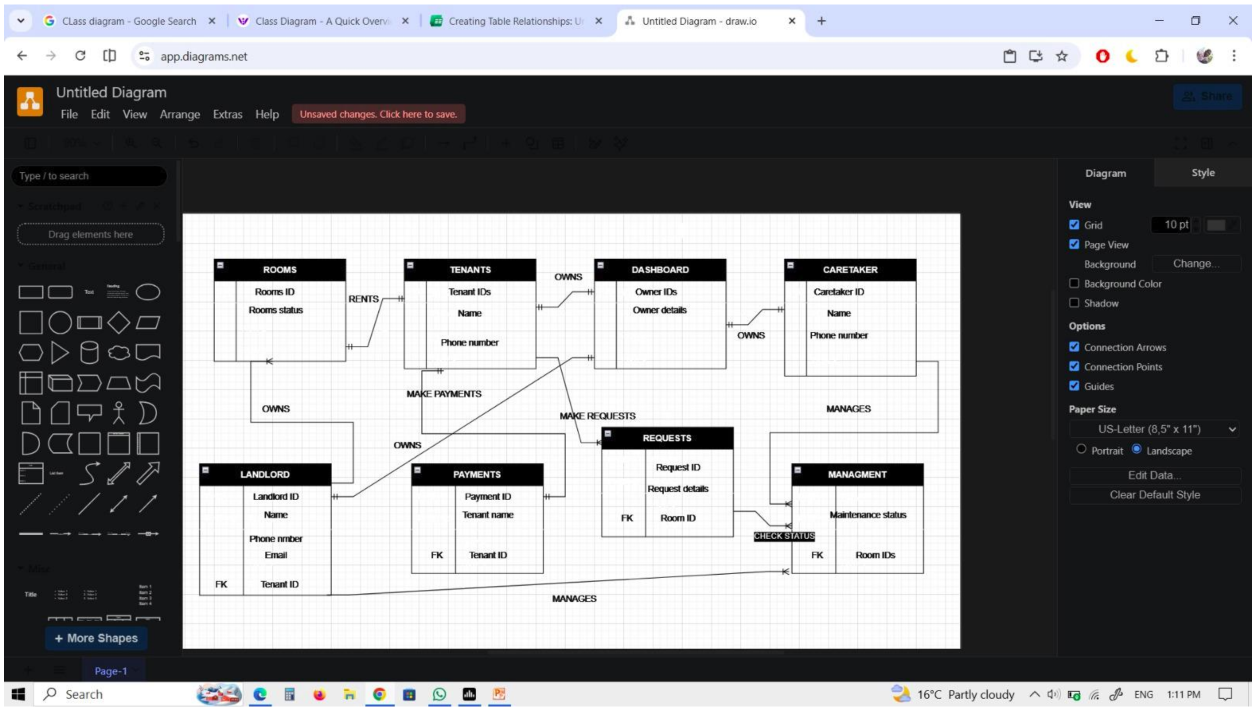
3. Sequential Diagram:

This figure illustrates a sequence diagram for the hostel fee payment process in the Hostel Management System. The diagram shows the interaction between the Tenant (student), the Tenant's Dashboard, and the Payment Server in a time-ordered manner. The tenant first logs into the system and initiates the payment through the dashboard, which then sends a payment request to the payment server for processing. After the transaction is completed, the payment server returns a payment successful response to the system, confirming the completion of the process. This sequence diagram clearly demonstrates the flow of control, system interactions, and communication involved in online hostel fee payment.



4. Class Diagram:

The class diagram represents the static structure of a rental/property management system, showing how key entities such as Rooms, Tenants, Landlord, Caretaker, Payments, Requests, Management, and Dashboard interact. It defines essential attributes with primary and foreign keys to maintain data integrity, for example linking tenants to rooms, payments, and requests, and rooms to maintenance status. The diagram illustrates associations like tenants renting rooms, making payments, and submitting requests; aggregation where landlords own multiple rooms; and composition where maintenance status depends on requests. Cardinalities (1, 1.., *0..*) clearly indicate how many instances of one class relate to another, ensuring a clear and structured representation of the system's data relationships.



View
- [✓] Grid 10 pt
- [✓] Page View
- Background Change...
- [ ] Background Color
- [ ] Shadow

Options
- [✓] Connection Arrows
- [✓] Connection Points
- [✓] Guides

Paper Size
US-Letter (8,5" x 11")
Portrait [✓ Landscape]

Edit Data...
Clear Default Style