JAVA WEB-BASED STUDENT ACCOMMODATION SYSTEM FOR UNIVERSITY OF GREENWICH

Final Year Project

Abstract

This web-based student accommodation system for the University of Greenwich provides a portal for students to meet a wide range of landlords who are interested in renting their properties to the students. This project includes the requirements' prioritisation using the MoSCoW method, design using UML, technologies and tool selection, implementation and testing phases.

This research will depict how each phase in the software development lifecycle is executed to complete this project successfully. Concluding recommendations of solutions are provided with justification for tool and technology selection.

Table of Contents

List of Tables

List of Figures

1.0. Introduction

The education sector in the UK is expanding especially with the influx of many international students. There is a high demand for students' accommodation as a result of this influx. Most universities have their accommodations, however, they usually do not meet the demand of the university populace. Currently, the University of Greenwich provides accommodation for its students within its premises only. Consequently, this project involves the creation of an accommodation system for students of the University of Greenwich, which allows all the landlords within the area who wish to rent out their properties to register and advertise them. This platform assists the students by providing the landlords' information, available properties and also to make a booking. This system can assist the students with a wide range of accommodations to satisfy their requirements.

1.1. Aim and Objectives

1.1.1. Aim

This project primarily aims at implementing or developing an error or fault free student accommodation system for students from the University of Greenwich, allowing landlords to list properties they are willing to rent to university students. Moreover, the accommodation system would be web-based so all the university students and intending students can access it from any part of the world.

1.1.2. Objectives

The objectives of the project are

Objective 1: To identify the system functional requirements and non-functional requirements

Objective 2: The identified functional and non-functional requirements should be implemented.

Objective 3: The system should be a web-based application. In this project, it can run on the localhost.

Objective 4: The system should go through several testing which are unit testing, integration testing, system testing, and acceptance testing.

Objective 5: The system should have user and technical documentation for future modification

1.2. Scope

The student accommodation system ought to have the essential functionalities stated in section 3.1, and it would exclude the platform to take payment from the Landlord to list a particular property. The manager of the system receives and confirms payment outside the system and allows the property to go live.

The Assumptions

The initial assumptions taken to implement the web-based student accommodation system are provided below:

- The payment for the listing of property by the landlords is done outside the system.
- The system assumed that the landlords are the people that are listing the properties in the system.
- The landlord and his or her properties are verified outside the system by the university in order to the authenticity of the property.
- In this project specification, the system will only allow the landlords to list properties such as Bungalow, Terraced House, Single Room, Flat, Duplex and Detached House.

1.3. Report Structure

- Section 2.0. contains the literature review and it discusses the student accommodation system available in the University of Greenwich alongside other universities.
- Section 3.0 is the methodology section which provides the step by step process approach taken during the project
- Section 4.0 provides the functional and non-functional requirements identified for the student accommodation information system.
- Section 5.0 provides the UML diagrams such as class diagram, ER diagram, activity diagram, use case diagram and sequence diagram
- Section 6.0 provides the system design which comprises of the application type, MVC design pattern and Technology used.
- Section 7.0 provides the screen printout and the explanation of the student accommodation information system.
- Section 8.0 provides the detail description of the testing conducted on the system
- Section 9.0 provides the evaluation section
- Section 10.0 provides the technical and user documentation

- Section 11.0 provides the conclusion and future work section

2.0. Methodology

The development of the web-based student accommodation system for the University of Greenwich was performed using the System Development Life-Cycle (SDLC) methods. The steps involved in the SDLC methods are provided below:

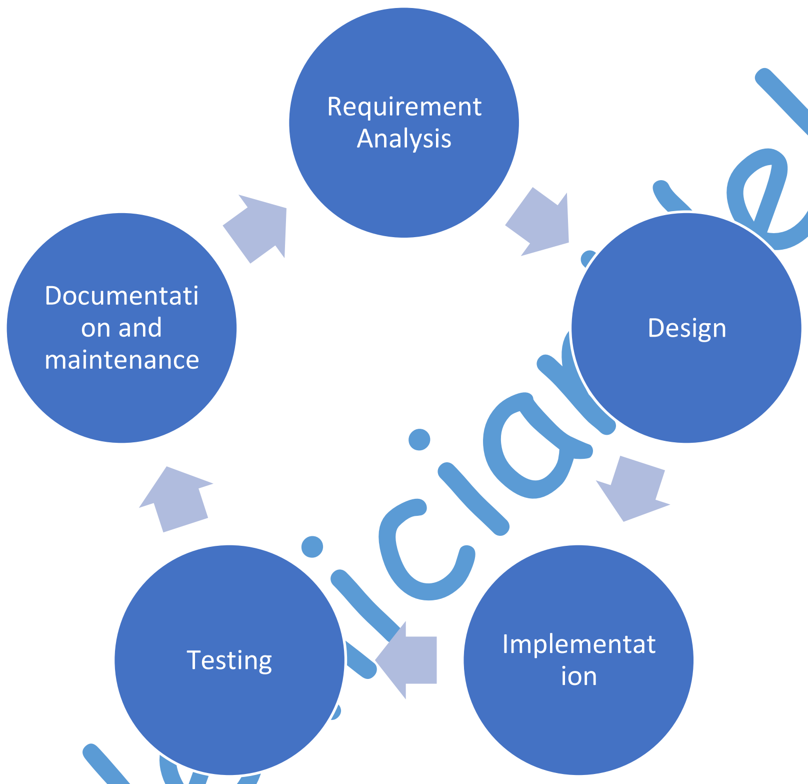


Figure 1: System Development Life Cycle for the Student accommodation system

The steps followed in the SDLC process is provided below:

Step 1: Requirement Analysis

The list of functional requirements and non-functional requirements are identified for the student accommodation system. Moreover, the MoSCoW method of prioritisation is used to prioritise the functional and non-functional requirements in order to develop the system.

Step 2: Design

The Unified Modelling Language (UML) Design such as use case diagram, use case specification, class diagram, activity diagram, sequence diagram and entity relationship diagram are drawn to verify the functionalities of the system

Step 3: Implementation

The student accommodation system was developed using the following technologies CSS, JSP, Java, Spring, Hibernate, MySQL and Maven

Step 4: Testing

There are several types of testing performed in student accommodation system which are unit testing, integration testing, system testing and security testing. The brief description of the testing is provided below:

- **Unit testing:** Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinised for proper operation. Unit testing can be done manually but is often automated.
- **Integration testing:** Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.
- **System testing:** It is a level of the software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. ISTQB's definition of system testing states that it is the process of testing an integrated system to verify that it is according to specification.
- **Acceptance testing:** Acceptance Testing is a level of the software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.
- **Security testing:** Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended.

Step 5: Documentation

The user and technical documentation for the student accommodation system were created.

3.0. Feasibility Study

Four major operational feasibility study was conducted for this project, and they include operational feasibility, technical feasibility, schedule feasibility, and economic feasibility (McCarty, 1969).

3.1. The operational feasibility study of the project

The operational feasibility study of the project is provided in table 9.

Table 1: Operational Feasibility Study

Factors for an operational feasibility study	Responses
Were the roles for the project created?	Yes
How will the application be implemented?	The application will be implemented in units or modules
How will the development process be documented?	The documentation goes hand in hand with the application development
What is the process taken to educate users on how to use the application?	The clear user manual will be provided.

3.2. The technical feasibility study of the project

The technical feasibility study of the project is provided in table 10.

Table 2: a Technical feasibility study

Factors for a technical feasibility study	Responses
What platform is the application going to use?	Web browser
Which software would be used to develop the application?	Eclipse
What is the database used?	MySQ
What is the major programming language used for the application development?	Java
Are required skills available in the group?	Yes
Does the group have the clear idea about the future roadmap of the project?	Yes

3.3. The economic feasibility of the project

The economic feasibility study of the project is provided in table 11.

Table 3: AneEconomic feasibility study

Factors for an economic feasibility study	Response
Do you have a software for development that has a proper license?	Yes. (Software is free)

3.4. The schedule feasibility of the project

The operational feasibility study of the project is provided in table 12.

Table 4: A Schedule feasibility study

Factors for a schedule feasibility study	Response
Is the time enough to do the development?	Yes. Only limited functionalities are implemented.
Is there time allocations for unexpected changes?	Yes

4.0. Requirement Gathering

4.1. Functional Requirements

The list of functional requirements of the Student accommodation system is provided below:

- The system should have three major users such as Administrator, student and landlord **(FR 1).**
- The system user student should be able to register on the system **(FR 2).**
- The students should receive an email notification with their username and password **(FR 3).**
- The student should be able to book for viewing the property **(FR 4).**
- The student should be able to login to the system to book for viewing **(FR 5).**
- The system user landlord should be able to register on the system **(FR 6).**
- The landlord should be able to login to the system **(FR 7).**
- The landlord should be able to list the property to rent in the system **(FR 8).**
- The landlord should be able to edit the already listed properties **(FR 9).**
- The landlord should be able to delete the already listed properties **(FR 10).**
- The landlord should be able to add profile pictures and other gallery pictures **(FR 11).**
- The administrator should be able to login to the system **(FR 12).**
- The administrator should be able to view the listed properties **(FR 13).**
- The administrator should be able to update the payment for the listed properties **(FR 14).**
- The listed property by the landlord will only be visible to the students when the payment is confirmed by the administrator of the system **(FR 15).**
- The administrator will be able to view the paid properties reports **(FR 16).**
- The administrator will be able to view the unpaid properties reports **(FR 17).**
- The administrator will be able to view the vacant properties reports **(FR 18).**
- The administrator will be able to view the occupied properties reports **(FR 19).**

4.2. Non-Functional Requirements

The list of non-functional requirements of the Student accommodation system is provided below:

- The system will be user-friendly to its users **(NFR 1).**

- The system should have proper authentication protocol to maintain security **(NFR 2).**
- The system should have consistency between the system web pages **(NFR 3).**

4.3. MoSCoW method of Prioritisation

The MoSCoW method is a prioritisation technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement.

- **Must have:** Requirements labelled as 'Must have' are critical to the current delivery timebox in order for it to be a success. If even one 'Must have' requirement is not included, the project delivery should be considered a failure (note: requirements can be downgraded from 'Must have', by agreeing with all relevant stakeholders; for example, when new requirements are deemed more important). MUST can also be considered an acronym for the Minimum Usable SubseT.
- **Should have:** Requirements labelled as 'Should have' are important but not necessary for delivery in the current delivery timebox. While 'Should have' requirements can be as important as 'Must have', they are often not as time-critical, or there may be another way to satisfy the requirement so that it can be held back until a future delivery timebox.
- **Could have:** Requirements labelled as 'Could have' are desirable but not necessary, and could improve user experience or customer satisfaction for little development cost. These will typically be included if time and resources permit.
- **Won't have:** Requirements labelled as 'Won't have' have been agreed by stakeholders as the least-critical, lowest-payback items, or not appropriate at that time. As a result, 'Won't have' requirements are not planned into the schedule for the delivery timebox. 'Won't have' requirements are either dropped or reconsidered for inclusion in later timeboxes.

The MoSCoW method of prioritisation for functional requirements and non-functional requirements are provided in table 5 and 6.

Table 5: MoSCoW method of prioritisation for functional requirements of student accommodation system

No	Functional Requirement	MoSCoW method of prioritisation
FR 1	The system should have three major users such as Administrator, student and landlord	Must have

No	Functional Requirement	MoSCoW method of prioritisation
FR 2	The system user student should be able to register on the system	Must have
FR 3	The students should receive an email notification with their username and password	Must have
FR 4	The students should be able to book for viewing the property	Must have
FR 5	The students should be able to login to the system to book for viewing	Must have
FR 6	The system user landlord should be able to register on the system	Must have
FR 7	The landlord should be able to login to the system	Must have
FR 8	The landlord should be able to list the property to rent in the system	Must have
FR 9	The landlord should be able to edit the already listed properties	Should have
FR 10	The landlord should be able to delete the already listed properties	Should have
FR 11	The landlord should be able to add profile pictures and other gallery pictures	Could have
FR 12	The administrator should be able to login to the system	Must have
FR 13	The administrator should be able to view the listed properties	Must have
FR 14	The administrator should be able to update the payment for the listed properties	Must have
FR 15	The listed property by the landlord will only be visible to the students when the administrator of the system confirms the payment	Must have

No	Functional Requirement	MoSCoW method of prioritisation
FR 16	The administrator will be able to view the paid properties reports	Should have
FR 17	The administrator will be able to view the unpaid properties reports	Should have
FR 18	The administrator will be able to view the vacant properties reports	Should have
FR 19	The administrator will be able to view the occupied properties reports	Should have

Table 6: MoSCoW method of prioritisation for non- functional requirements of student accommodation system

No	Non-functional Requirement	MoSCoW method of prioritisation
NFR 1	The system will be user-friendly to its users	Must have
NFR 2	The system should have a proper authentication protocol to maintain security	Must have
NFR 3	The system should have consistency between the system web pages	Must have

5.0. Unified Modelling Language Diagrams

5.1. Use Case diagram

The use case diagram is a graphical representation of the intersection between the user and the system. Moreover, the use case diagram is used to identify the types of users involved in the system as well as the different use cases (Holt, 2004). The use case diagram represents the student accommodation information system's interaction between its users such as Administrator, landlord and student. The use case diagram is mainly used to identify the system users and how they interact with the system (Sommerville, 2000).

The use case diagram for the student accommodation system is provided in figure 1.

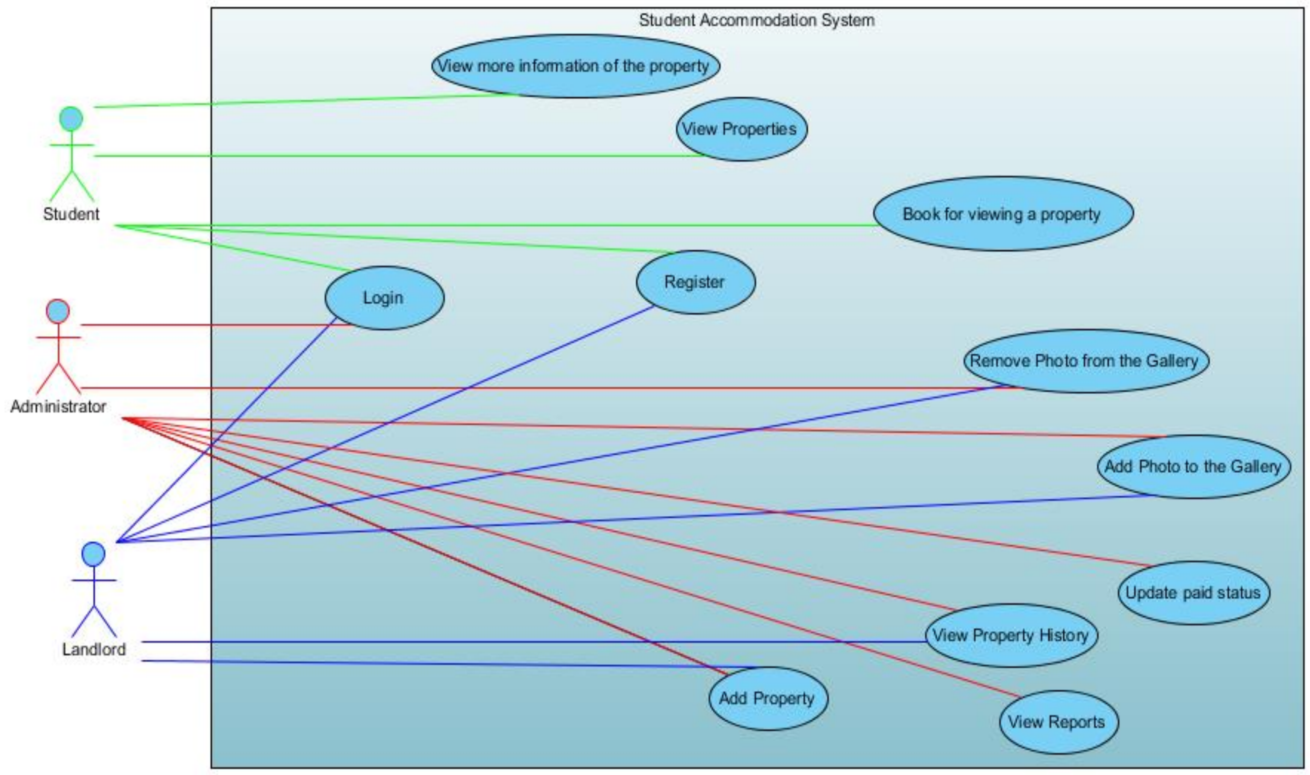


Figure 2: Use Case Diagram for Student Accommodation System

The Use Case IDs are provided below:

Table 7: Use Case IDs

Use Case ID	Use Case
UCID 1	Register by the Student
UCID 2	Login by the Student
UCID 3	View Property by the Student
UCID 4	View more information on the property by the Student
UCID 5	Book for viewing a property by the student
UCID 6	Register by the Landlord
UCID 7	Login by the Landlord
UCID 8	Add Property by the landlord
UCID 9	View Property History by the landlord
UCID 10	Add Photos to the gallery by the landlord
UCID 11	Remove Photo from the gallery by the landlord
UCID 12	Login by the Administrator
UCID 13	Add Property by Administrator

Use Case ID	Use Case
UCID 14	View Property History by Administrator
UCID 15	Add Photos to the gallery by Administrator
UCID 16	Remove Photo from the gallery by Administrator
UCID 17	View reports by Administrator
UCID 18	Update paid status of the property by Administrator

5.2. Use Case Specifications

Table 8: Use Case Specification for Register by the Student

Use Case ID: **UCID 1**	Use Case: **Register by the Student**
Description:	The Student can register into the system to view property information and book for viewing
Primary actor:	**Student**
Secondary actors:	None
Preconditions:	There is no precondition for the student to register with the University of Greenwich web-based student accommodation system
Main flow:	1. The student selects the register tab 2. Enters the required information 3. Presses OK
Postconditions:	The username and password will be sent to the email, and with that, the student can log in to the University of Greenwich web-based student accommodation system
Alternative flow:	None

Table 9: Use Case Specification for login by the student

Use Case ID: **UCID 2**	Use Case: **Login by the Student**
Description:	The Student can log in using his or her username and password
Primary actor:	**Student**
Secondary actors:	None
Preconditions:	The student should already be registered with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select login tab from top left 2. Insert username and password into the field 3. Click on the login button
Postconditions:	If the username and password are correct, then the system will allow the user into the system to do advance functionalities. Else, the error message is given, and he/she need to try again.
Alternative flow:	None

Table 10: Use Case Specification for View property by the student

Use Case ID: **UCID 3**	Use Case: **View Property by the Student**
Description:	The Student can view the properties
Primary actor:	**Student**
Secondary actors:	None
Preconditions:	The student can be logged in or not with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select sView Property 2. Views the list of properties listed in the system

Postconditions:	The logged in student can view the property 'more information and book for viewing'
Alternative flow:	None

Table 11: Use Case Specification for view more information on the property by the student

Use Case ID: **UCID 4** Use Case: **View more information on the property by the Student**	
Description:	The students can view more information about the property that interests them
Primary actor:	**Student**
Secondary actors:	None
Preconditions:	The students should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor can press the more information 2. View the detail information of the property
Postconditions:	The students can book for viewings
Alternative flow:	None

Table 12: Use Case Specification for Book for viewing a property by the student

Use Case ID: **UCID 5** Use Case: **Book for viewing a property by the student**	
Description:	The Student can book for viewing for the properties that they are interested.
Primary actor:	**Student**

Secondary actors:	None
Preconditions:	The student should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor can select the book viewing button to make the booking
Postconditions:	
Alternative flow:	None

Table 13: Use Case Specification for Register by the landlord

Use Case ID: **UCID 6**	Use Case: **Register by the landlord**
Description:	The Landlord can register into the system to list a property that they want to rent it to the students of the university.
Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	There is no precondition for the landlord to register with the University of Greenwich web-based student accommodation system
Main flow:	1. The students select the register tab 2. Entire the required information 3. Press OK
Postconditions:	The username and password will be sent to the email and using that the landlord can log in to the University of Greenwich web-based student accommodation system
Alternative flow:	None

Table 14: Use Case Specification for login by the landlord

Use Case ID: **UCID 7**	Use Case: **Login by the landlord**
Description:	The Landlord can log in using his or her username and password
Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	The landlord should already be registered with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select login tab from top left 2. Insert username and password into the field 3. Click on the login button
Postconditions:	If the username and password are correct, then the system will allow the user into the system to do advance functionalities. Else, the error message is given, and he/she need to try again.
Alternative flow:	None

Table 15: Use Case Specification for add property by the landlord

Use Case ID: **UCID 8**	Use Case: **Add property by the landlord**
Description:	The landlords can add their property to the system
Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	The landlords should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select add a property 2. Provide the required information

	3. Press submit
Postconditions:	The listed property will not go live for students to view immediately. It will only go live when the administrator confirms the payment received in the system
Alternative flow:	None

Table 16: Use Case Specification for view property history by the landlord

Use Case ID: **UCID 9**	Use Case: **View property history by the landlord**
Description:	The landlord can view the property histories for the properties he or she listed in the system.
Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	The landlord should have already logged in with the University of Greenwich web-based student accommodation system. Also, he or she should have listed a property in the system
Main flow:	1. The actor selects view property history 2. Views the history information of the properties listed by him or her.
Postconditions:	
Alternative flow:	None

Table 17: Use case specification for Add photos to the gallery by the landlord

Use Case ID: **UCID 10**	Use Case: **Add Photos to the gallery by the landlord**
Description:	The landlord can add photos to the gallery of the property listed

Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	The landlord should have already logged in with the University of Greenwich web-based student accommodation system. Also, he or she should have listed a property in the system
Main flow:	1. The actor selects add photos to the gallery 2. Selects the photo from the local machine 3. Confirms the photo selected
Postconditions:	
Alternative flow:	None

Table 18: Use Case Specification for Remover photos from the gallery by the landlord

Use Case ID: **UCID 11**	Use Case: **Remove photos from the gallery by the landlord**
Description:	The landlord can remove the photo from the gallery of the property listed
Primary actor:	**Landlord**
Secondary actors:	None
Preconditions:	The landlord should have already logged in with the University of Greenwich web-based student accommodation system. Also, he or she should have listed a property in the system. Besides, the gallery should have images uploaded already.
Main flow:	1. The actor selects the image he/she wants to delete 2. Press Delete 3. And Confirm

Postconditions:	The gallery image will be deleted and will not show on the system
Alternative flow:	None

Table 19: Use Case Specification for login by the administrator

Use Case ID: **UCID 12**	Use Case: **Login by the Administrator**
Description:	The Administrator can log in using his or her username and password
Primary actor:	**Administrator**
Secondary actors:	None
Preconditions:	The Administrator should already be registered with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor selects login tab from top left 2. Inserts username and password into the field 3. Clicks on the login button
Postconditions:	If the username and password are correct, then the system will allow the user into the system to do advance functionalities. Else, the error message is given, and he/she need to try again.
Alternative flow:	None

Table 20: Use Case Specification for Add Property by the administrator

Use Case ID: **UCID 13**	Use Case: **Add property by administrator**
Description:	The administrators can add their property to the system
Primary actor:	**Administrator**
Secondary actors:	None

Preconditions:	The administrator should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select add a property 2. Provide the required information 3. Press submit
Postconditions:	The listed property will not go live for students to view immediately. It will only go live when the administrator confirms the payment received in the system
Alternative flow:	None

Table 21: Use Case Specification for view property by administrator

Use Case ID: **UCID 14**	Use Case: **View property history by Administrator**
Description:	The administrator can view the property histories for the properties listed in the system.
Primary actor:	**administrator**
Secondary actors:	None
Preconditions:	The administrator should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor select view property history 2. View the history information of the properties listed by him or her.
Postconditions:	
Alternative flow:	None

Table 22: Use Case Specification for Add Photos to the gallery by the administrator

Use Case ID: **UCID 15**	Use Case: **Add Photos to the gallery by the administrator**
Description:	The administrator can add photos to the gallery of the property listed
Primary actor:	**Administrator**
Secondary actors:	None
Preconditions:	The Administrator should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor selects add photos to the gallery 2. Selects the photo from the local machine 3. Confirms the photo selected
Postconditions:	
Alternative flow:	None

Table 23: Use Case Specification for Remover Photo from the gallery by the administrator

Use Case ID: **UCID 16**	Use Case: **Remove photos from the gallery by the Administrator**
Description:	The Administrator can remove the photo from the gallery of the property listed
Primary actor:	**Administrator**
Secondary actors:	None
Preconditions:	The Administrator should have already logged in with the University of Greenwich web-based student accommodation system. Besides, the gallery should have images uploaded already.
Main flow:	1. The actor selects the image he/she wants to delete 2. Press Delete

	3. And Confirm
Postconditions:	The gallery image will be deleted and will not show on the system
Alternative flow:	None

Table 24: Use Case Specification for View reports by the administrator

Use Case ID: **UCID 17**	Use Case: **View reports by the Administrator**
Description:	The administrator can view the reports for the information in the system
Primary actor:	**administrator**
Secondary actors:	None
Preconditions:	The administrator should have already logged in with the University of Greenwich web-based student accommodation system
Main flow:	1. The actor selects view reports 2. Selects the type of report required 3. Views the report generated
Postconditions:	
Alternative flow:	None

Table 25: Use Case Specification for update paid status of the property by administrator

Use Case ID: **UCID 18**	Use Case: **Update paid status of the property by administrator**
Description:	The administrator can update the property paid status for the properties listed by him or by the landlord
Primary actor:	**Administrator**

Secondary actors:	None
Preconditions:	The administrator should have already logged in with the University of Greenwich web-based student accommodation system. Besides, there should be property listed in the system.
Main flow:	1. The actor selects the property 2. Selects update information 3. Selects the paid status 4. Clicks on the confirm button
Postconditions:	
Alternative flow:	None

5.3. Class Diagram

The class diagram is a pictorial representation of the system by showing the classes with its attributes and methods and the relationship between them. The class diagram has several relationships which are one-to-one, one-to-many and many-to-many (Holt, 2004). The class diagram represents the classes that will be used in implementing the student accommodation information system. Moreover, the class diagram includes the attributes and operations of each class within them (Faulkner, 2009). Also, the relationship classes such as many to many, one to one and many to one are specified within the class diagram (Faulkner, 2009).

The full overview of the class diagram for the student accommodation system is provided in figure 2.

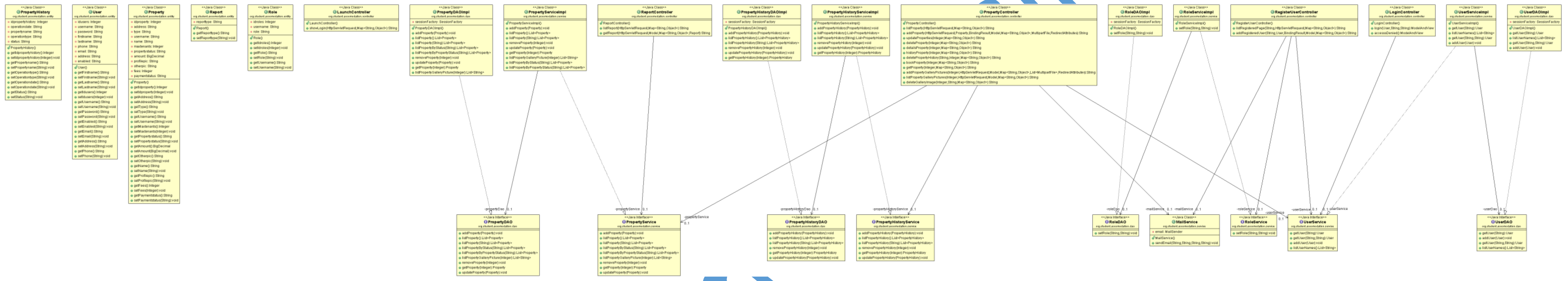
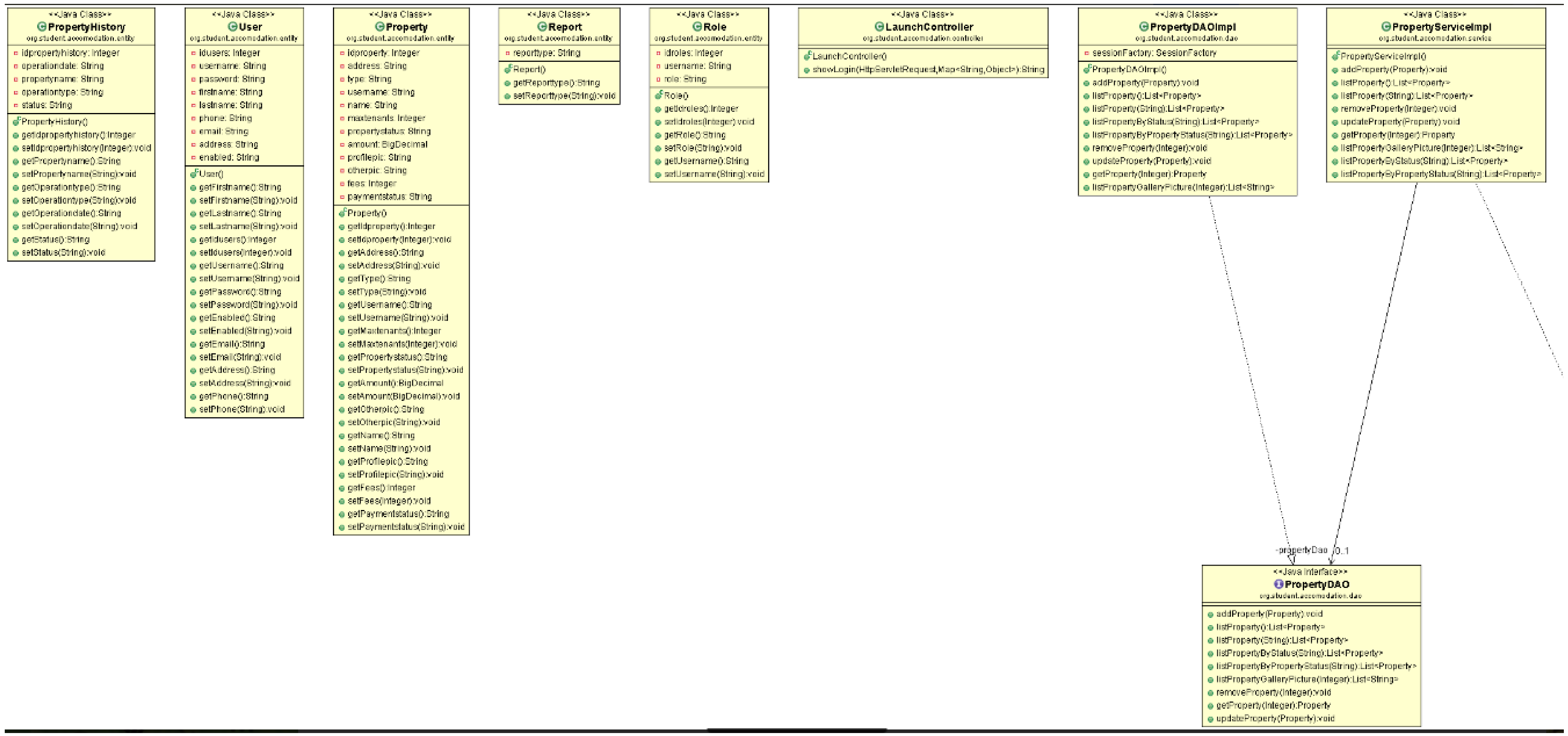


Figure 3: The complete overview of the class diagram

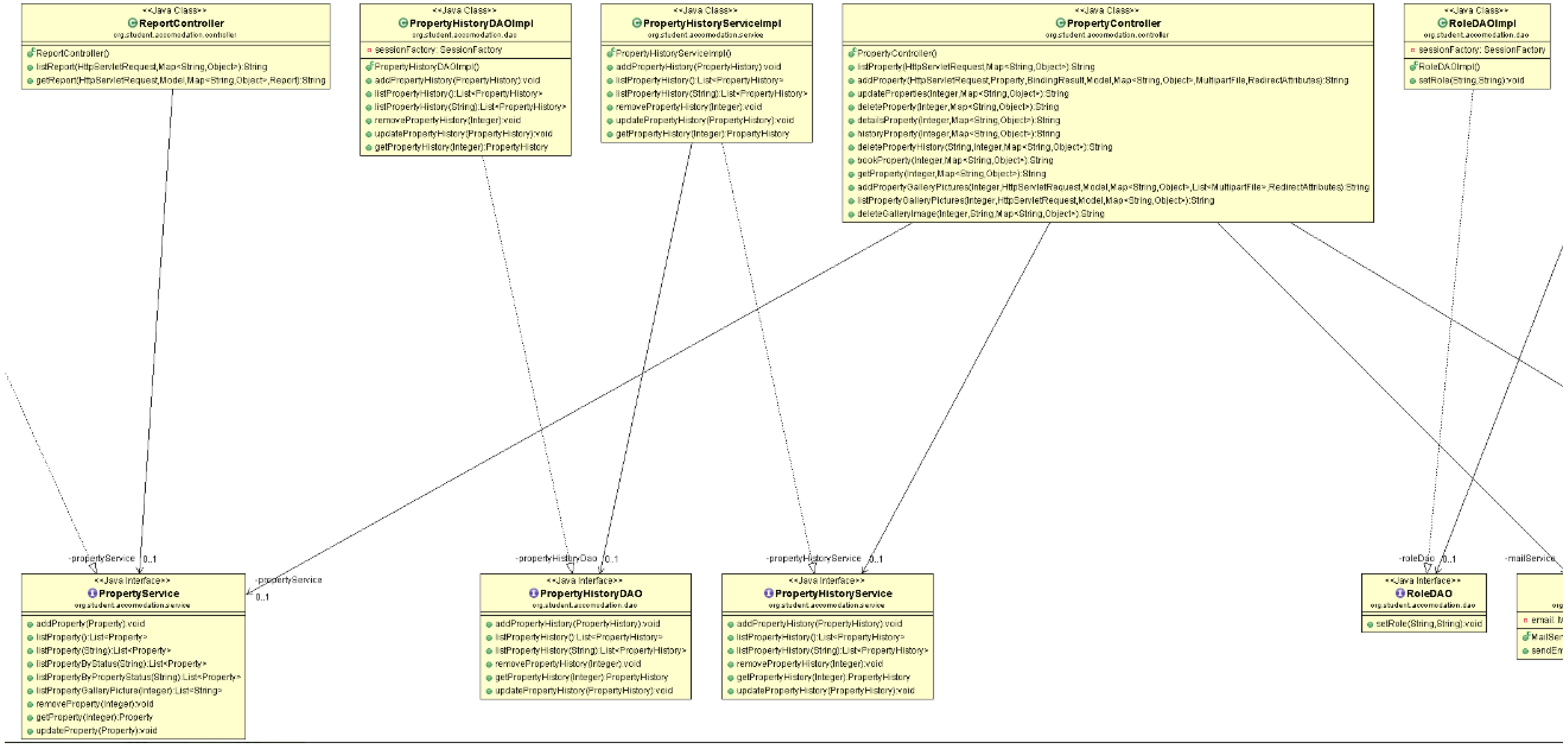
The part 1 of the class diagram is provided in figure 3.



<<Java Class>>
PropertyHistory
org.student.accomodation.entity
idpropertyhistory: Integer
operationdate: String
propertyname: String
operationtype: String
status: String
PropertyHistory()
getIdpropertyhistory():Integer
setIdpropertyhistory(Integer):void
getPropertyname():String
setPropertyname(String):void
getOperationtype():String
setOperationtype(String):void
getOperationdate():String
setOperationdate(String):void
getStatus():String
setStatus(String):void
<<Java Class>>
User
org.student.accomodation.entity
idusers: Integer
username: String
password: String
firstname: String
lastname: String
phone: String
email: String
address: String
enabled: String
User()
getFirstname():String
setFirstname(String):void
getLastname():String
setLastname(String):void
getIdusers():Integer
setIdusers(Integer):void
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
getEnabled():String
setEnabled(String):void
getEmail():String
setEmail(String):void
getAddress():String
setAddress(String):void
getPhone():String
setPhone(String):void
<<Java Class>>
Property
org.student.accomodation.entity
idproperty: Integer
address: String
type: String
username: String
name: String
maxtenants: Integer
propertystatus: String
amount: BigDecimal
profilepic: String
otherpic: String
fees: Integer
paymentstatus: String
Property()
getIdproperty():Integer
setIdproperty(Integer):void
getAddress():String
setAddress(String):void
getType():String
setType(String):void
getUsername():String
setUsername(String):void
getMaxtenants():Integer
setMaxtenants(Integer):void
getPropertystatus():String
setPropertystatus(String):void
getAmount():BigDecimal
setAmount(BigDecimal):void
getOtherpic():String
setOtherpic(String):void
getName():String
setName(String):void
getProfilepic():String
setProfilepic(String):void
getFees():Integer
setFees(Integer):void
getPaymentstatus():String
setPaymentstatus(String):void
<<Java Class>>
Report
org.student.accomodation.entity
reporttype: String
Report()
getReporttype():String
setReporttype(String):void
<<Java Class>>
Role
org.student.accomodation.entity
idroles: Integer
username: String
role: String
Role()
getIdroles():Integer
setIdroles(Integer):void
getRole():String
setRole(String):void
getUsername():String
setUsername(String):void
<<Java Class>>
LaunchController
org.student.accomodation.controller
LaunchController()
showLogin(HttpServletRequest,Map<String,Object>):String
<<Java Class>>
PropertyDAOImpl
org.student.accomodation.dao
sessionFactory: SessionFactory
PropertyDAOImpl()
addProperty(Property):void
listProperty():List<Property>
listProperty(String):List<Property>
listPropertyByStatus(String):List<Property>
listPropertyByPropertyStatus(String):List<Property>
removeProperty(Integer):void
updateProperty(Property):void
getProperty(Integer):Property
listPropertyGalleryPicture(Integer):List<String>
<<Java Class>>
PropertyServiceImpl
org.student.accomodation.service
PropertyServiceImpl()
addProperty(Property):void
listProperty():List<Property>
listProperty(String):List<Property>
removeProperty(Integer):void
updateProperty(Property):void
getProperty(Integer):Property
listPropertyGalleryPicture(Integer):List<String>
listPropertyByStatus(String):List<Property>
listPropertyByPropertyStatus(String):List<Property>
-propertyDao
0..1
<<Java Interface>>
PropertyDAO
org.student.accomodation.dao
addProperty(Property):void
listProperty():List<Property>
listProperty(String):List<Property>
listPropertyByStatus(String):List<Property>
listPropertyByPropertyStatus(String):List<Property>
listPropertyGalleryPicture(Integer):List<String>
removeProperty(Integer):void
getProperty(Integer):Property
updateProperty(Property):void

Figure 4: Part 1 of the class diagram

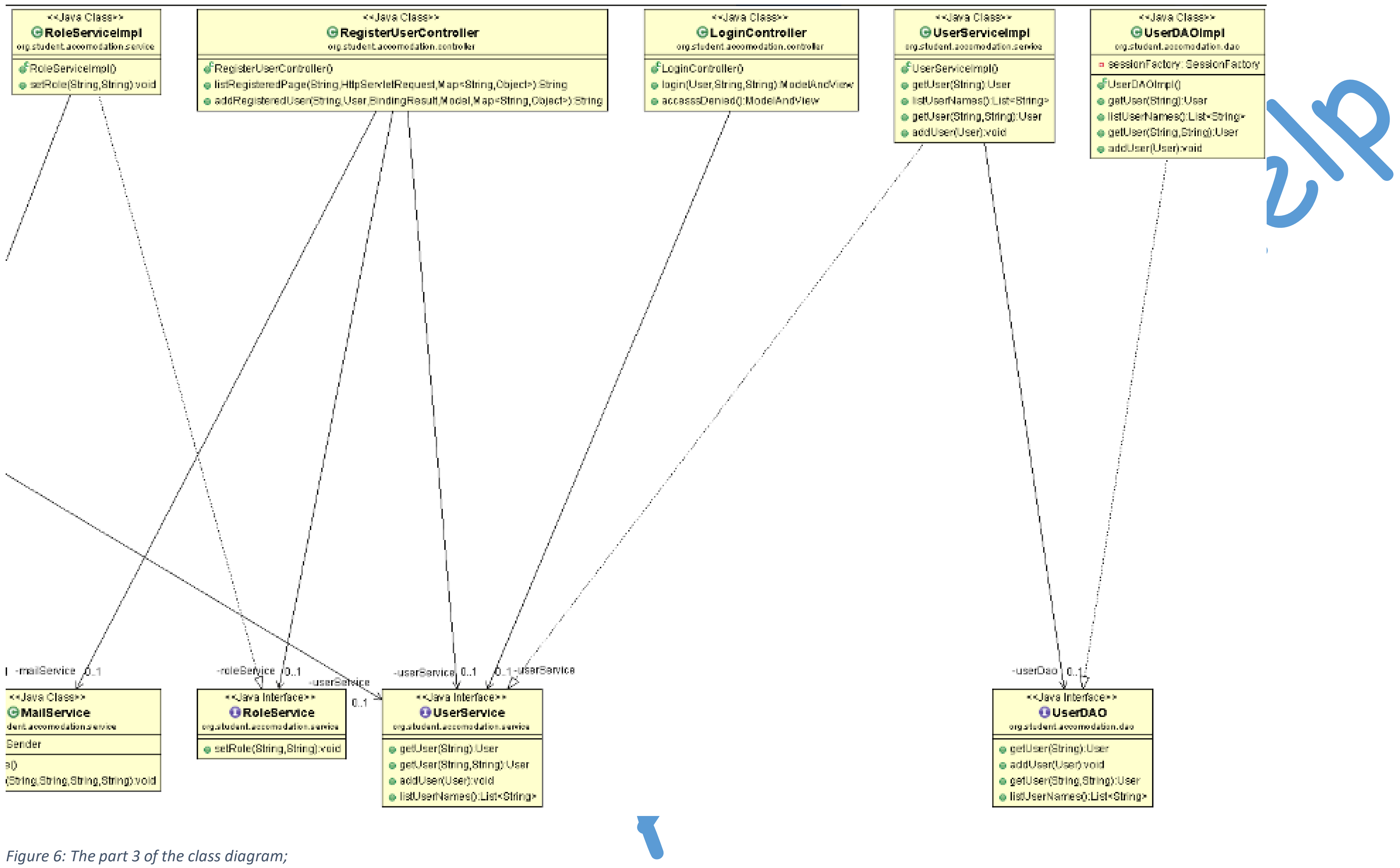
The part 2 of the class diagram is provided in figure 4.



<<Java Class>>
ReportController
org.student.accomodation.controller
ReportController()
listReport(HttpServletRequest,Map<String,Object>):String
getReport(HttpServletRequest,Model,Map<String,Object>,Report):String
<<Java Class>>
PropertyHistoryDAOImpl
org.student.accomodation.dao
sessionFactory: SessionFactory
PropertyHistoryDAOImpl()
addPropertyHistory(PropertyHistory):void
listPropertyHistory():List<PropertyHistory>
listPropertyHistory(String):List<PropertyHistory>
removePropertyHistory(Integer):void
updatePropertyHistory(PropertyHistory):void
getPropertyHistory(Integer):PropertyHistory
<<Java Class>>
PropertyHistoryServiceImpl
org.student.accomodation.service
PropertyHistoryServiceImpl()
addPropertyHistory(PropertyHistory):void
listPropertyHistory():List<PropertyHistory>
listPropertyHistory(String):List<PropertyHistory>
removePropertyHistory(Integer):void
updatePropertyHistory(PropertyHistory):void
getPropertyHistory(Integer):PropertyHistory
<<Java Class>>
PropertyController
org.student.accomodation.controller
PropertyController()
listProperty(HttpServletRequest,Map<String,Object>):String
addProperty(HttpServletRequest,Property,BindingResult,Model,Map<String,Object>,MultipartFile,RedirectAttributes):String
updateProperties(Integer,Map<String,Object>):String
deleteProperty(Integer,Map<String,Object>):String
detailsProperty(Integer,Map<String,Object>):String
historyProperty(Integer,Map<String,Object>):String
deletePropertyHistory(String,Integer,Map<String,Object>):String
bookProperty(Integer,Map<String,Object>):String
getProperty(Integer,Map<String,Object>):String
addPropertyGalleryPictures(Integer,HttpServletRequest,Model,Map<String,Object>,List<MultipartFile>,RedirectAttributes):String
listPropertyGalleryPictures(Integer,HttpServletRequest,Model,Map<String,Object>):String
deleteGalleryImage(Integer,String,Map<String,Object>):String
<<Java Class>>
RoleDAOImpl
org.student.accomodation.dao
sessionFactory: SessionFactory
RoleDAOImpl()
setRole(String,String):void
-propertyService
0..1
-propertyService
0..1
<<Java Interface>>
PropertyService
org.student.accomodation.service
addProperty(Property):void
listProperty():List<Property>
listProperty(String):List<Property>
listPropertyByStatus(String):List<Property>
listPropertyByPropertyStatus(String):List<Property>
listPropertyGalleryPicture(Integer):List<String>
removeProperty(Integer):void
getProperty(Integer):Property
updateProperty(Property):void
-propertyHistoryDao
0..1
<<Java Interface>>
PropertyHistoryDAO
org.student.accomodation.dao
addPropertyHistory(PropertyHistory):void
listPropertyHistory():List<PropertyHistory>
listPropertyHistory(String):List<PropertyHistory>
removePropertyHistory(Integer):void
getPropertyHistory(Integer):PropertyHistory
updatePropertyHistory(PropertyHistory):void
-propertyHistoryService
0..1
<<Java Interface>>
PropertyHistoryService
org.student.accomodation.service
addPropertyHistory(PropertyHistory):void
listPropertyHistory():List<PropertyHistory>
listPropertyHistory(String):List<PropertyHistory>
removePropertyHistory(Integer):void
getPropertyHistory(Integer):PropertyHistory
updatePropertyHistory(PropertyHistory):void
-roleDao
0..1
<<Java Interface>>
RoleDAO
org.student.accomodation.dao
setRole(String,String):void
-mailService

Figure 5: The part 2 of the class diagram

The part 3 of the class diagram is provided in figure 5.



<<Java Class>>
RoleServiceImpl
org.student.accomodation.service
RoleServiceImpl()
setRole(String,String):void
<<Java Class>>
RegisterUserController
org.student.accomodation.controller
RegisterUserController()
listRegisteredPage(String,HttpServletRequest,Map<String,Object>):String
addRegisteredUser(String,User,BindingResult,Model,Map<String,Object>):String
<<Java Class>>
LoginController
org.student.accomodation.controller
LoginController()
login(User,String,String):ModelAndView
accessDenied():ModelAndView
<<Java Class>>
UserServiceImpl
org.student.accomodation.service
UserServiceImpl()
getUser(String):User
listUserNames():List<String>
getUser(String,String):User
addUser(User):void
<<Java Class>>
UserDAOImpl
org.student.accomodation.dao
sessionFactory: SessionFactory
UserDAOImpl()
getUser(String):User
listUserNames():List<String>
getUser(String,String):User
addUser(User):void
-mailService 0..1
-roleService 0..1
-userService 0..1
-userService 0..1
0..1 -userService
-userDao 0..1
<<Java Class>>
MailService
dent.accomodation.service
Sender
s()
(String,String,String,String):void
<<Java Interface>>
RoleService
org.student.accomodation.service
setRole(String,String):void
<<Java Interface>>
UserService
org.student.accomodation.service
getUser(String):User
getUser(String,String):User
addUser(User):void
listUserNames():List<String>
<<Java Interface>>
UserDAO
org.student.accomodation.dao
getUser(String):User
addUser(User):void
getUser(String,String):User
listUserNames():List<String>

Figure 6: The part 3 of the class diagram;

5.4. Sequence Diagram

The sequence diagram provides a detailed description of the objectives and how they interact with each other and in which order (Holt, 2004). The sequence diagram of the student accommodation information system provides the sequence of interaction between the objectives (Bjørner, 2006).

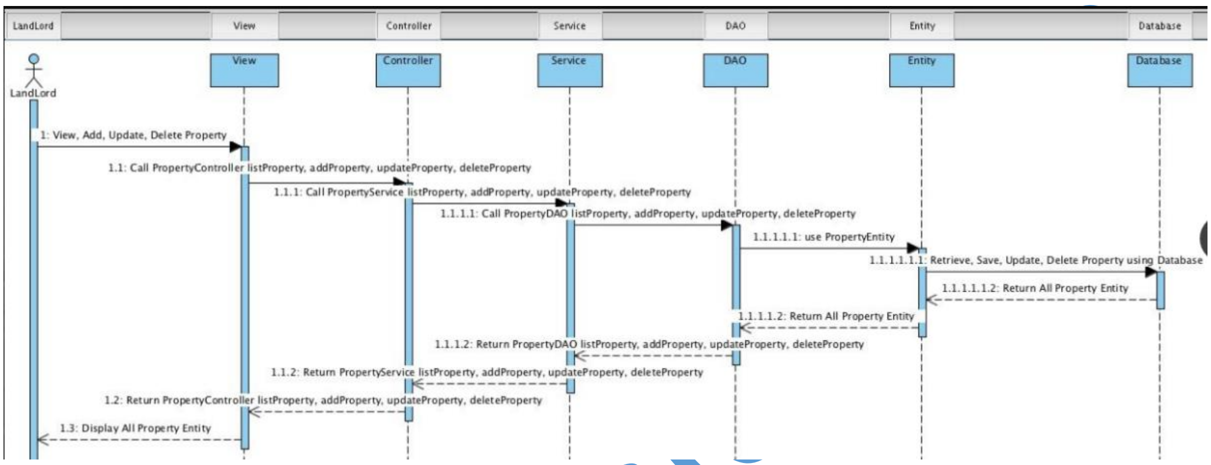

LandLord
View
Controller
Service
DAO
Entity
Database
1: View, Add, Update, Delete Property
1.1: Call PropertyController listProperty, addProperty, updateProperty, deleteProperty
1.1.1: Call PropertyService listProperty, addProperty, updateProperty, deleteProperty
1.1.1.1: Call PropertyDAO listProperty, addProperty, updateProperty, deleteProperty
1.1.1.1.1: use PropertyEntity
1.1.1.1.1.1: Retrieve, Save, Update, Delete Property using Database
1.1.1.1.1.2: Return All Property Entity
1.1.1.1.2: Return All Property Entity
1.1.1.2: Return PropertyDAO listProperty, addProperty, updateProperty, deleteProperty
1.1.2: Return PropertyService listProperty, addProperty, updateProperty, deleteProperty
1.2: Return PropertyController listProperty, addProperty, updateProperty, deleteProperty
1.3: Display All Property Entity

Figure 7: View, add, delete and update property sequence diagram for the landlord

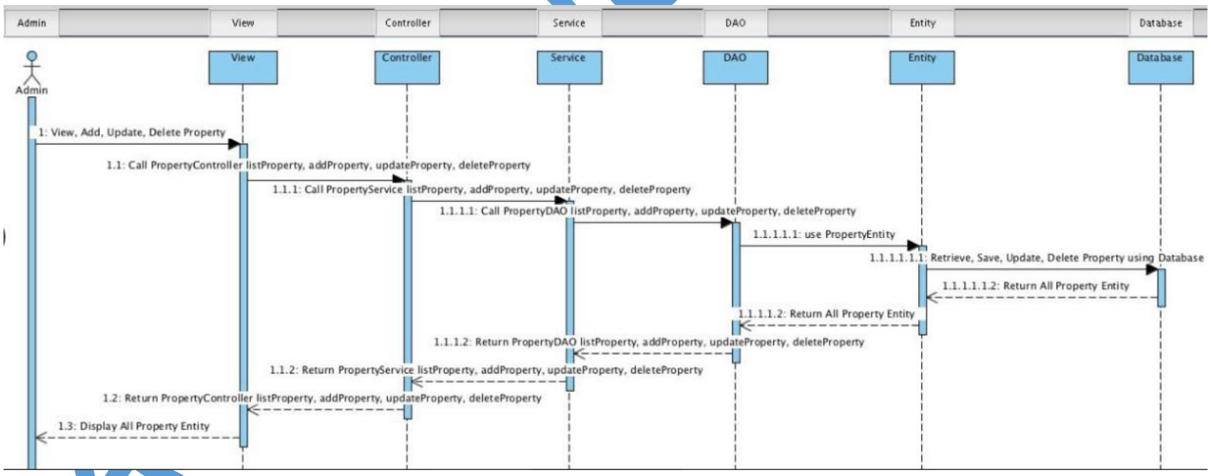

Admin
View
Controller
Service
DAO
Entity
Database
1: View, Add, Update, Delete Property
1.1: Call PropertyController listProperty, addProperty, updateProperty, deleteProperty
1.1.1: Call PropertyService listProperty, addProperty, updateProperty, deleteProperty
1.1.1.1: Call PropertyDAO listProperty, addProperty, updateProperty, deleteProperty
1.1.1.1.1: use PropertyEntity
1.1.1.1.1.1: Retrieve, Save, Update, Delete Property using Database
1.1.1.1.1.2: Return All Property Entity
1.1.1.1.2: Return All Property Entity
1.1.1.2: Return PropertyDAO listProperty, addProperty, updateProperty, deleteProperty
1.1.2: Return PropertyService listProperty, addProperty, updateProperty, deleteProperty
1.2: Return PropertyController listProperty, addProperty, updateProperty, deleteProperty
1.3: Display All Property Entity

Figure 8: view, add, delete and update property sequence diagram for admin

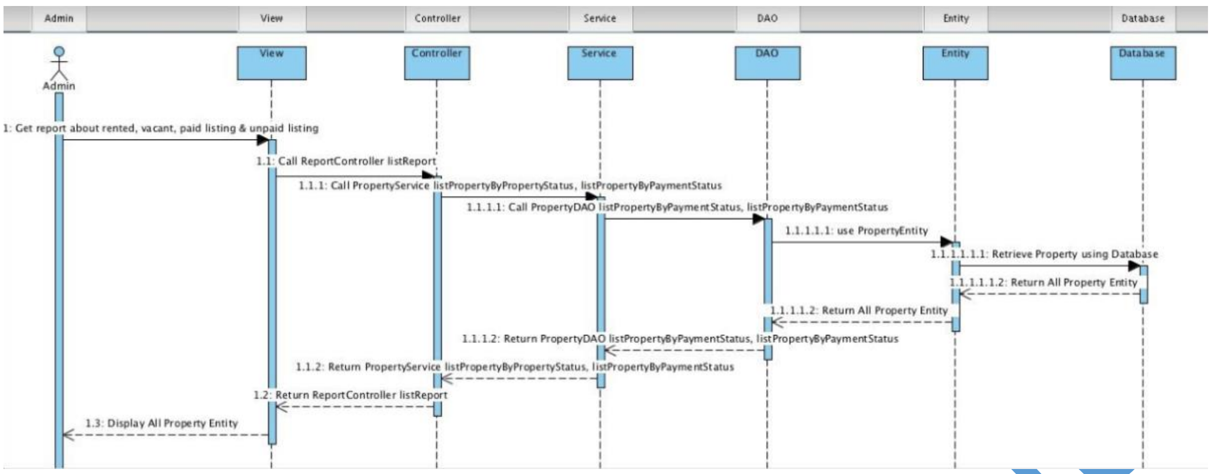

Admin
View
Controller
Service
DAO
Entity
Database
1: Get report about rented, vacant, paid listing & unpaid listing
1.1: Call ReportController listReport
1.1.1: Call PropertyService listPropertyByPropertyStatus, listPropertyByPaymentStatus
1.1.1.1: Call PropertyDAO listPropertyByPaymentStatus, listPropertyByPaymentStatus
1.1.1.1.1: use PropertyEntity
1.1.1.1.1.1: Retrieve Property using Database
1.1.1.1.1.2: Return All Property Entity
1.1.1.1.2: Return All Property Entity
1.1.1.2: Return PropertyDAO listPropertyByPaymentStatus, listPropertyByPaymentStatus
1.1.2: Return PropertyService listPropertyByPropertyStatus, listPropertyByPaymentStatus
1.2: Return ReportController listReport
1.3: Display All Property Entity

Figure 9: report sequence Diagram for admin

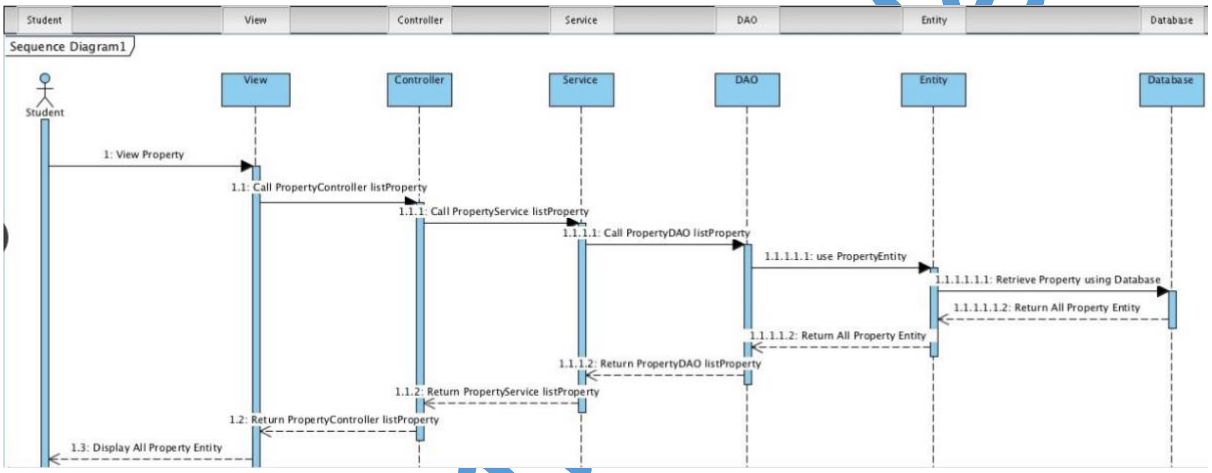

Student
View
Controller
Service
DAO
Entity
Database
Sequence Diagram1
1: View Property
1.1: Call PropertyController listProperty
1.1.1: Call PropertyService listProperty
1.1.1.1: Call PropertyDAO listProperty
1.1.1.1.1: use PropertyEntity
1.1.1.1.1.1: Retrieve Property using Database
1.1.1.1.1.2: Return All Property Entity
1.1.1.1.2: Return All Property Entity
1.1.1.2: Return PropertyDAO listProperty
1.1.2: Return PropertyService listProperty
1.2: Return PropertyController listProperty
1.3: Display All Property Entity

Figure 10: View Property for student sequence diagram

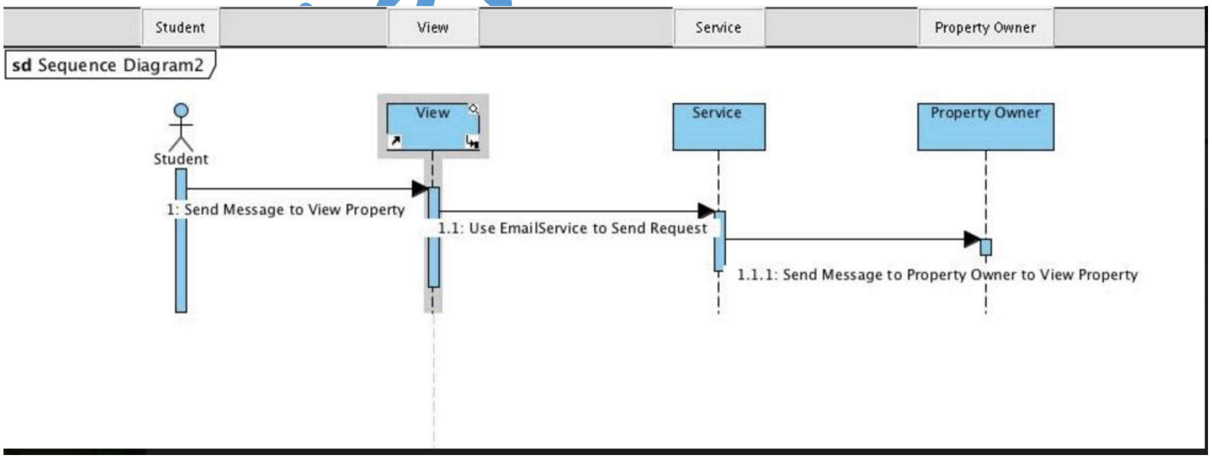

Student
View
Service
Property Owner
sd Sequence Diagram2
1: Send Message to View Property
1.1: Use EmailService to Send Request
1.1.1: Send Message to Property Owner to View Property

Figure 11: Send a message sequence diagram

5.5. Activity Diagram

The Activity diagram is a flowchart that shows the flow of activities in the system (Holt, 2004).

The activity diagram for the student accommodation system is provided in figure 11.

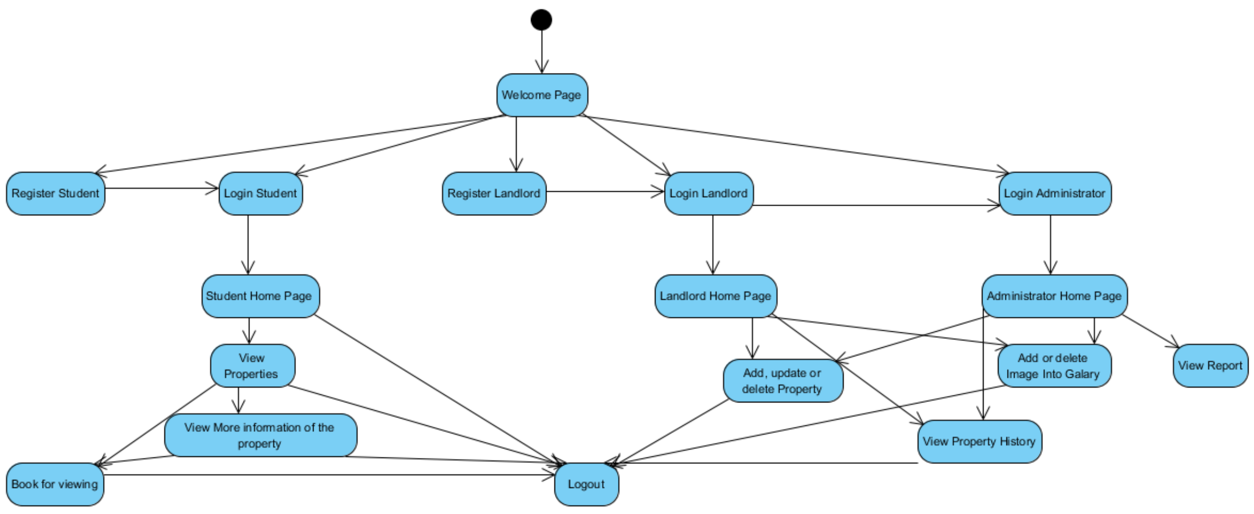


Figure 12: Activity Diagram for Student Accommodation System

5.6. Entity-Relationship Diagram

The Entity-Relationship (ER) diagram depicts the database tables , its fields and its relationship. Moreover, the primary key for the tables and the foreign key of the tables are displayed in the diagram (Holt, 2004).

The entity-relationship diagram for students' accommodation system is provided in figure 12.

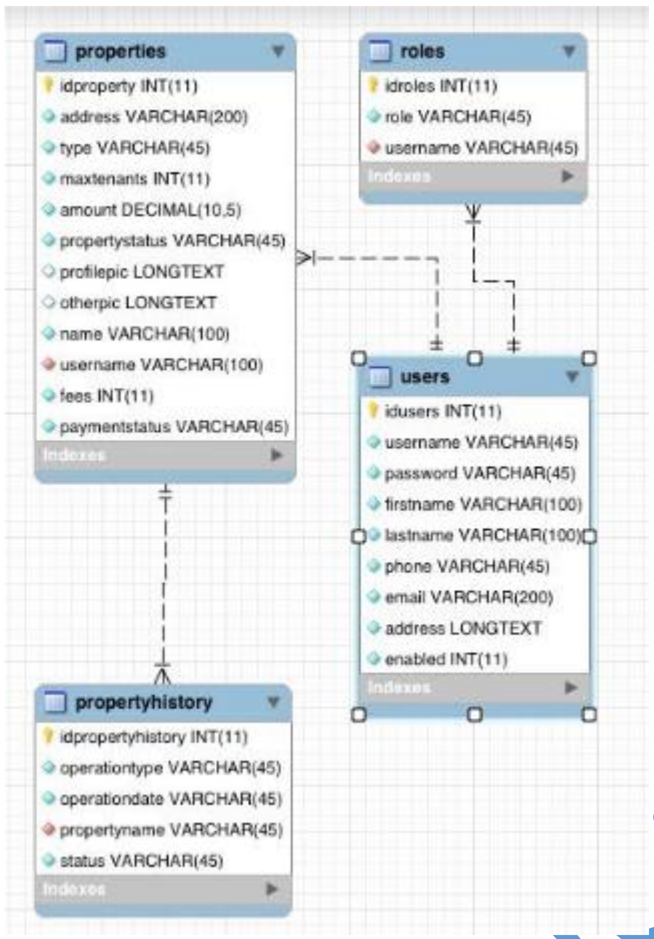


Figure 13: Entity Relationship Diagram for Student Accommodation System

6.0. System design

6.1. Application Type

The student management system should be a web application because it needs the following advantages:

- Ability to be accessed anywhere, anytime.
- Ability of the system to be accessed on different platforms
- It is easy to install
- It is easy to do maintenance

6.2. Architecture Style

There are system architecture designs available, which are singleton pattern, factory pattern, abstract factory pattern, MVC and so on.

The singleton pattern is a basic pattern that ensures that a Java class has only one instance. This pattern is not used because the student accommodation system is complex, and it is not possible to restrict a class to a single instance (Theodoridis and Koutroumbas, 2006).

The factory pattern refers to the objects created using a general interface. In the student accommodation system, it is not used because, only one class is inheriting from other classes and all other classes are not inheriting from any common class to have an interface (Theodoridis and Koutroumbas, 2006).

The abstract factory pattern uses interface to create a group of similar objects. In the student accommodation system, it is not used because the class objects cannot be grouped since they are independent (Theodoridis and Koutroumbas, 2006).

The MVC patterns divide the system into three major parts which are model, view and controller. The model represents the object of the class; view represents the visualisation of the data that the object or model comprises and the controller is used as an intermediary between model and view; it controls the data flow and updates it in the view. In the student accommodation system, the MVC pattern is used to control the data flow within the object (Theodoridis and Koutroumbas, 2006). The MVC architecture is provided below:

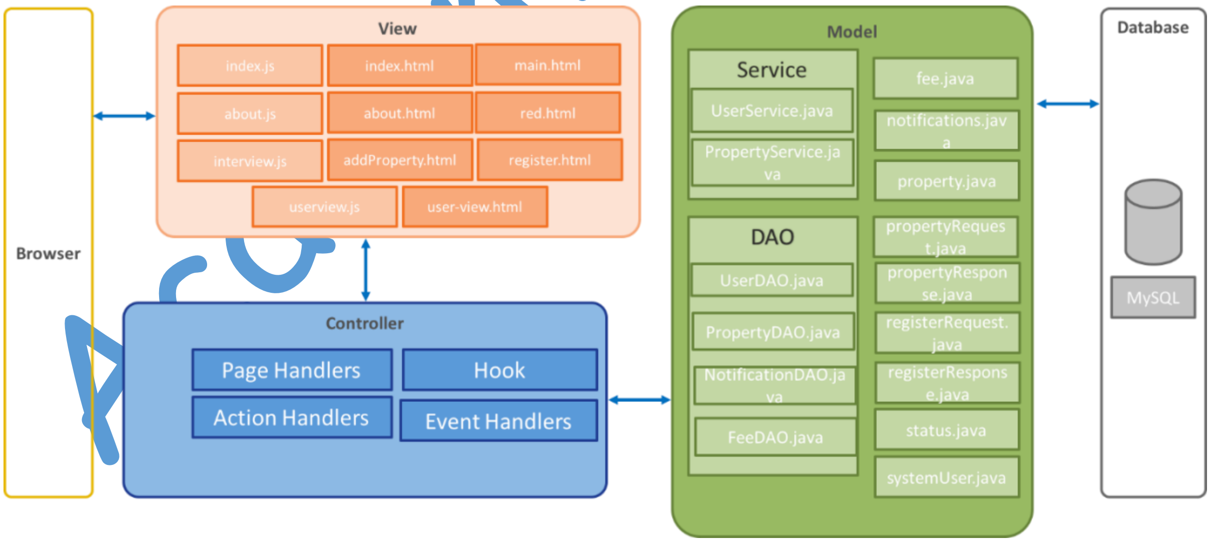


Figure 14: MVC architecture

The MVC provides general recurring solutions to the most common software design problems and also ensures that the solutions are flexible and scalable. Relationships and interactions between classes and objects are used to customise and solve problems related to general design.

There are two mechanisms of reuse in design patterns, including Class Inheritance and Object Composition. Class inheritance uses the white-box reuse method where the internals of the parent class are often visible to the subclass, where reuse mechanism can be applied by subclassing. Objection composition is also referred to as black-box reusing mechanism, where the internals of the objects are encapsulated. In object composition, by assembling and composing objects, the reusing concept is executed.

These also adhere to good object-oriented design principles as mentioned below.

- Programming to an (abstract) interface, not a concrete implementation reduces implementation of dependencies by declaring variables to be instances of an abstract class (interface).
- Favouring object composition over class inheritance where it is defined dynamically at run-time (object composition) and defined at compile-time (class inheritance). This makes each class encapsulated and reduced focus on a single task.

"Gang of Four" design patterns include 23 patterns which are categorised as below stated.

Table 26: "Gang of Four" Design Patterns

Creational Patterns (5)	Structural Patterns (7)	Behavioural Patterns (11)
The process of object creation	The composition of objects or classes	The way in which objects or classes interact and distribute responsibility
• Abstract Factory • Builder • Factory Method • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Resp. • Command • Interpreter • Iterator • Mediator • Memento • Observer • State • Strategy

		• Template • Visitor

Based on the characteristics of the project, thorough research was done to select the best-suited design pattern, and Creational patterns were selected as the project is mostly adhering to object-oriented methodology. The below table depicts a comparison between the creational patterns and how a creational pattern was selected for the current project.

Table 27: Creational patterns in more details

	Abstract Factory	Factory Method	Builder	Prototype	Singleton
When to use	Need complete flexibility- new kinds of derived classes could be created later which would be usable by the client	Need limited flexibility – choose from among a limited set of configurations	Do not need the details, need a configuration which will enable the reseracher to start working on it right away	Do not want the details, do not even need to pick what package I will get it will probably be passed to me	Need to create only one of these- could be known to the researcher or could be passed to him.
Client Knowledge	Less knowledge of type- client knows the base type, not a specific class. Knowledge of class flexible, but elements of the configuration, it has to create them explicitly	More knowledge – client knows the specific type, and knows about the internal configuration - it has to create them explicitly	Less knowledge of details, no knowledge of configuration	More knowledge of details, some knowledge of configuration - enough to make the right selection	Orthogonal to how much client has to know, but knowledge limited by the fact that most of the time, it uses an instance that already exists
Advantage/ Disadvantage	Great flexibility, Low client knowledge	Reduced flexibility, High client knowledge	Reduced flexibility, Minimal client knowledge	Good flexibility, Minimal client knowledge	Variable flexibility, Minimal client knowledge

From the creational patterns, Factory pattern was used in this project. It is one of the most used design patterns in Java. This provides the best way to create objects. It enables objects to be created without revealing the creation logic to the client and refer to a freshly created object using a common interface.

Intent: When creating an object, it defines its interface, but also provides the capability of letting the subclasses choose which class to be instantiated. Factory method allows a class vary instantiation to subclasses.

Problem: Standardising the architectural model of a framework to cater to a range of applications which is ideally a difficult task. Also, when standardising the applications, they should be able to define their domain objects for their instantiation.

Discussion: Factory method is similar to an algorithm where objects can be created as a template Method. The superclass defines all standard and general characteristics and then assigns the creation details to subclasses which are provided by the client.
This method makes a design more customisable and also makes slightly complicated. Other design patterns require new classes, but in the Factory Method, it only requires a new operation.
Some developers very often use this method to create objects, but it is not mandatory. This is necessary only if the class initiated does not change or provide subclasses, and this can easily be overridden where instantiation happens in operation. These methods are customarily defined by the architectural framework and then implanted by the developer.

:

6.3. Framework, technology and language used

The technologies used to build the student accommodation system are Java, CSS, JSP, MySQL, Spring, Hibernate and Maven.

6.3.1. Java

The java is a programming language that is used in the student accommodation system. The programming language is used because it has a lot of academic resources and materials to support during the development process. Also, the Eclipse IDE was used to write the java program.

Choosing the language is a vital part of a software development project. Based on the requirements of the coursework and the module content, the author identified that Java is the best-suited language as it is OOP based.

Five major facts are always being considered when an object oriented programming language is discussed.

- A class where operations, attributes are represented as a mono entity.
- An encapsulation where the data is protected from other external interfaces.
- An inheritance, which uses the concept of reusability, where a child class can inherit the properties of the parent class.
- A polymorphism provides the capability of handling different functionalities of functions which similar in name.
- An abstraction wherein the conceptual level details are used when presenting classes with attributes along with operations.

These above-mentioned characteristics are available in most of the languages, but in different methods. Therefore, in this scenario threading, memory handling, platform agnostic and support is being taken into consideration.

Java is a language with automatic memory management, where the developer should not worry about coding the methods of memory utilisation. The feature used in Java to utilise memory is Garbage collector. This method is used to utilise memory from the objects which are not being used. This helps the developers to focus on coding the functional requirements as they should not focus on freeing memory. This garbage collector method is not available in

C++ (Voegele, 2012). A good programming language must be capable of aiding the development and design of a scalable and a reliable software product. Therefore, error messages and warnings should be used in programming. Java is beneficial as it executes many runtime checks which is comparable with C and C++, but where C++ is better than C. Maintainability is another vital factor which should also provide sufficient documentation for the developers. Java is a language which can be maintained by any developer as for the fact that it is easily readable. Language is ideally better to be independent of the operating system as well. Java, in that case, is a platform independent language. Compared with C and C++, Java is ahead as it compiles into bytecode but C++ compiles into its native code. C# also is a platform independent language, though mainly focuses on Microsoft platforms. Concurrency is also another characteristic of a good programming language as it increases the efficiency of a program. Threading is supported by C and C++ but does not possess the benefit of in-built process or threads capability.

Furthermore, Java and C# enable threading (Wheeler 1996). When considering constructs, these languages contain similar constructs. However, Java and C++ possess identical characteristics of constructs, apart from C++. Inner classes and multiple interface inheritance are not available in C++. Header files comprise of global functions and constants in C++ (Albahari, 2000).

Platform independence is the main reason for selecting Java. It is also can be run independently from the platform. Also, security is a very beneficial factor in Java when comparing it with other programming languages. The table below will depict a comparison between the languages.

Table 28: The comparison between programming languages

Feature	C++	C#	Java
Knowledge of the author	Familiar	Familiar	Very Familiar
Encapsulation	Y	Y	Y
Inheritance	Several	Single Class or Multiple Interfaces	Single Class or Multiple Interfaces
Polymorphism	Y	Y	Y

All User-defined Types are objects	N	Y	Y
Reliability	N	Moderate	Y
Maintainability	Moderate	Moderate	Y
Machine Independence / Platform Independence	Moderate	Moderate	Y
Threading	Libraries	Yes	Yes
Garbage Collection	No	Yes	Yes
Built-in Security	No	Yes	Yes

Based on the above research author selected Java as the programming language.

6.3.2. CSS

Cascading Style Sheets (CSS) is a popular styling language that used to represent the presentation of the application.

6.3.3. JSP

Java Server Pages (JSP) is a popular server-side programming language that used to develop platform independent and dynamic web related applications.

6.3.4. MySQL

MySQL is one of the relational database management systems. However, MySQL is very famous because it is open source. This is used to create the database for the student accommodation system.

6.3.5. Spring

Spring is a light Java framework, that is used to handle the web related application forms securely.

6.3.6. Hibernate

Hibernate is a mapping tool for Java. It assists in mapping the object-oriented domain model into the relational database model. Hibernate is a free object-relational mapping (ORM) library for Java. It is primarily used to map Java objects to database tables. It provides data query and retrieval facilities so that the author need not manually handle result sets and object conversions.

Additionally, Hibernate supports all the popular database servers, which means that the author could easily replace MySQL with a more enterprise level server such as Oracle if such a need arose in the future.

6.3.7. Maven

The maven technology is used in the Java program to manage the build process of the web application as well as the jar.

6.3.8. JAX-RS

The framework should be developed as a web service. There are two types of web services, namely SOAP-based web services and RESTful web services. Of these, the soap-based services are more popular and widely used. However, the author found in his research that using soap-based services would have a performance impact on external devices. This is because soap messages are XML-based, and converts these into XML and back from a device.
Therefore, the author decided to implement the system as a RESTful web-service, which uses simple HTTP requests to send and receive data. JAX-RS (Java API for RESTful Web Services) was used for the implementation used for building RESTful web services. It provides annotations to configure the resources of a web service easily. It also provides automatic conversion between XML and Java objects and between JSON and Java objects. Another advantage of using this is that it allows the service to be built in such a way that the client can interact with the server using either XML or JSON, without the author having to handle it manually.

6.3.9. Application Server and Database Server

The framework is a web application requiring access to a database. This application could only be run on a servlet container which was provided by most commonly known and freely available application servers such as Apache Tomcat, JBoss, IBM Websphere and Glassfish. Out of these application servers, the Apache Tomcat server is the one that meets the minimum requirement of a servlet container. The others were bundled with additional components such as EJB containers, which would be of no use to the application under consideration and hence, would be having an overhead on the hardware resources being used.
Therefore, the Apache Tomcat Application Server was chosen, also considering its simplicity in

configuration and seamless integration with the Eclipse IDE. The choice of database servers was among Microsoft‟s SQL Server, Oracle Server and MySQL server. Both MSSQL and Oracle servers have more features when compared with the MySQL server. Also, MSSQL lacks cross-platform support, and Oracle servers are generally used for large-scale enterprise applications, which would be too much in the case of this project. Additionally, none of the two is freely available. MySQL server is not commercial, it is free and also easy to use, which supports many platforms. These above considerations resulted in the author in choosing MySQL as the best-suited database server.

6.4. Three-tier Organisation

The three-tier organisation was maintained in the student accommodation system. The three-tire organisation architecture is provided in figure 13.

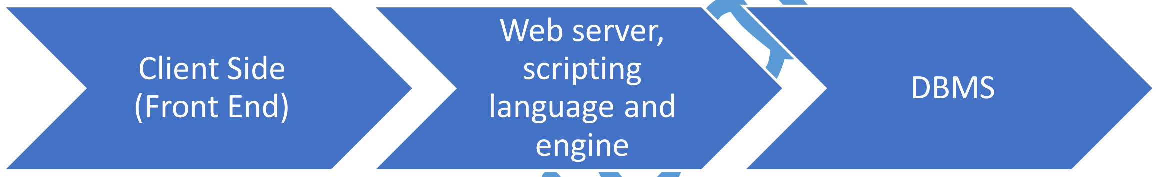


Figure 15: Three-Tier Organisation of the student accommodation system

- The tier 1 (Client Side – Front end) allows the user to interact with the system.
- The tier 2 (Web server, scripting language and engine) Is a logical part of the system that assists in communicating between the tier 1 and tier 3.
- The tier 3 (DBMS) provides the database structure and it is used to store the data.

The more detailed 3-tier architecture is provided below:

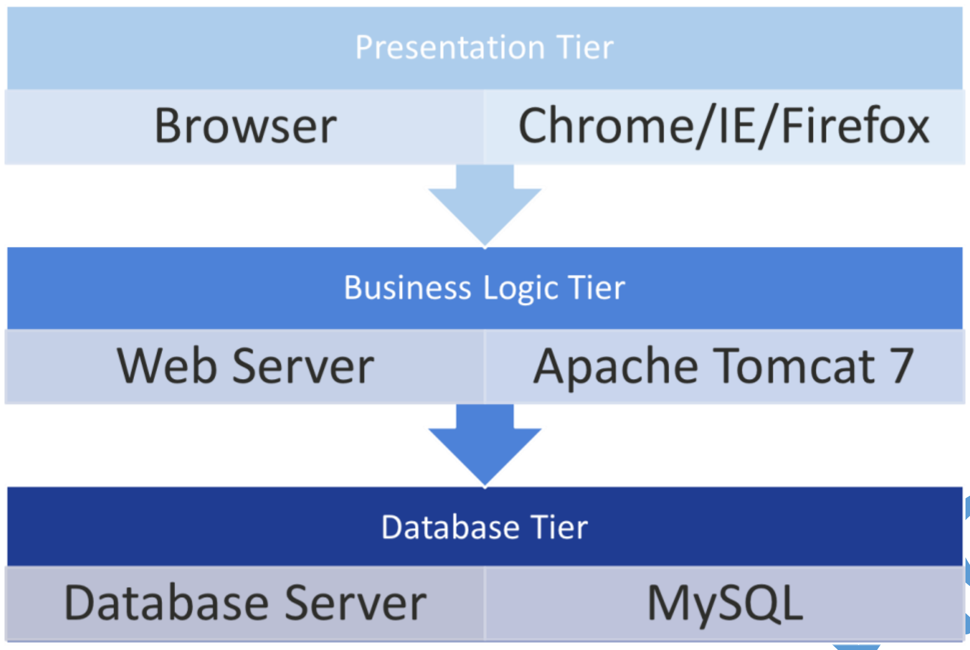


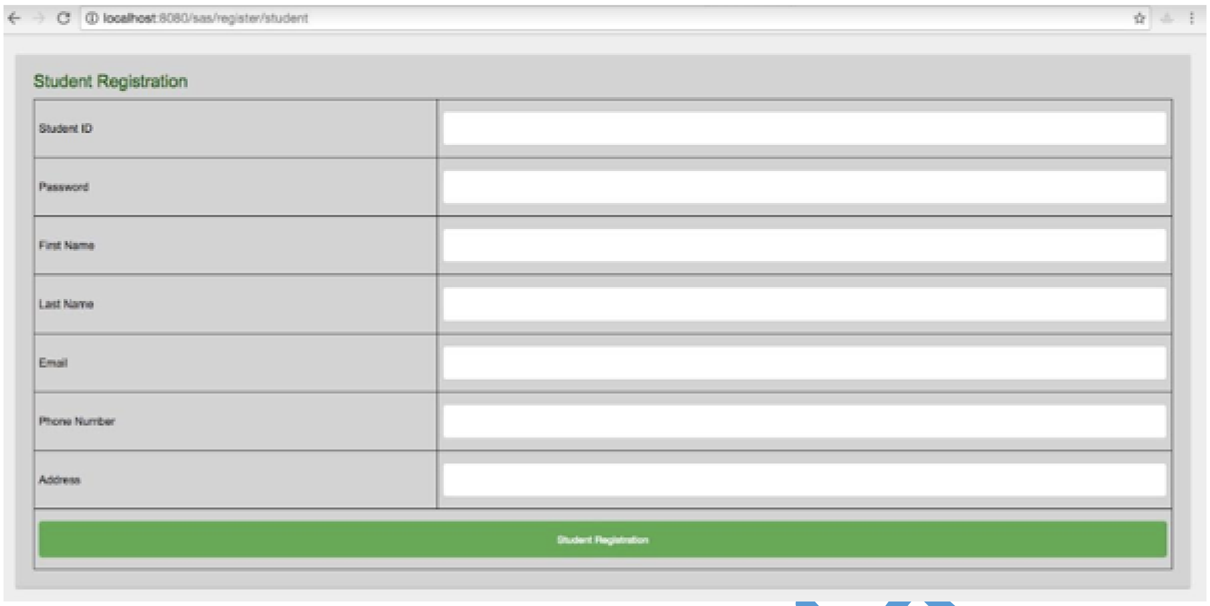
Figure 16: Detailed 3-Tier Architecture

7.0. Deliverables

7.1. The student Web pages

7.1.1. Register Page

Figure 14 shows the register Page of the student. The Student ID is a field that suggests the student provide a desired username for the future login purposes. There are other required information such as Password, first name, last name, E-mail address, Phone number and Address to register as a student in the portal. When a student provides valid information in the registration fields, then the system will send an email to the student with the provided username and password to log in to the system. See Figure 14, for the sample of email received by the student after the registration.



localhost:8080/sas/register/student

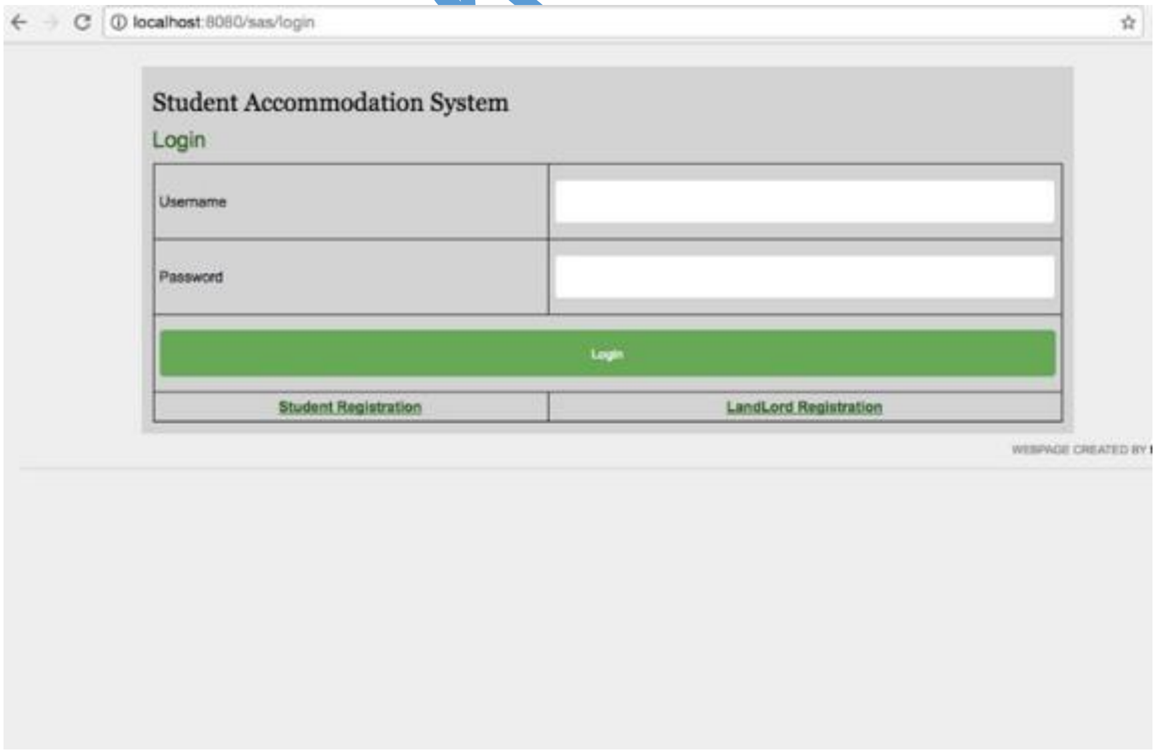
Student Registration

Student ID	
Password	
First Name	
Last Name	
Email	
Phone Number	
Address	

Figure 17: The student Registration Page

7.1.2. Log in Page

Figure 15 provides the screen printout of the student login form. The username and password that was received in the email after the registration will be used to log in to the system. Once the correct credentials are provided, the student is required to press the login button to log in to the system.



localhost:8080/sas/login

Student Accommodation System

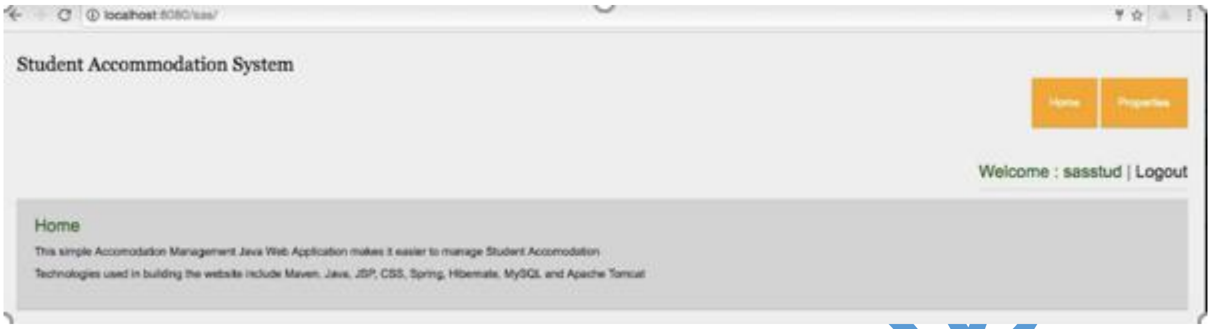
Login

Username	
Password	

Figure 18: Student Login Page

7.1.3. Logged-in Home Page

Figure 16 provides the home page of the student. When a student successfully logs in to the system, the home page will be displayed. In the home page, it allows the student to view properties and also to log out of the system.

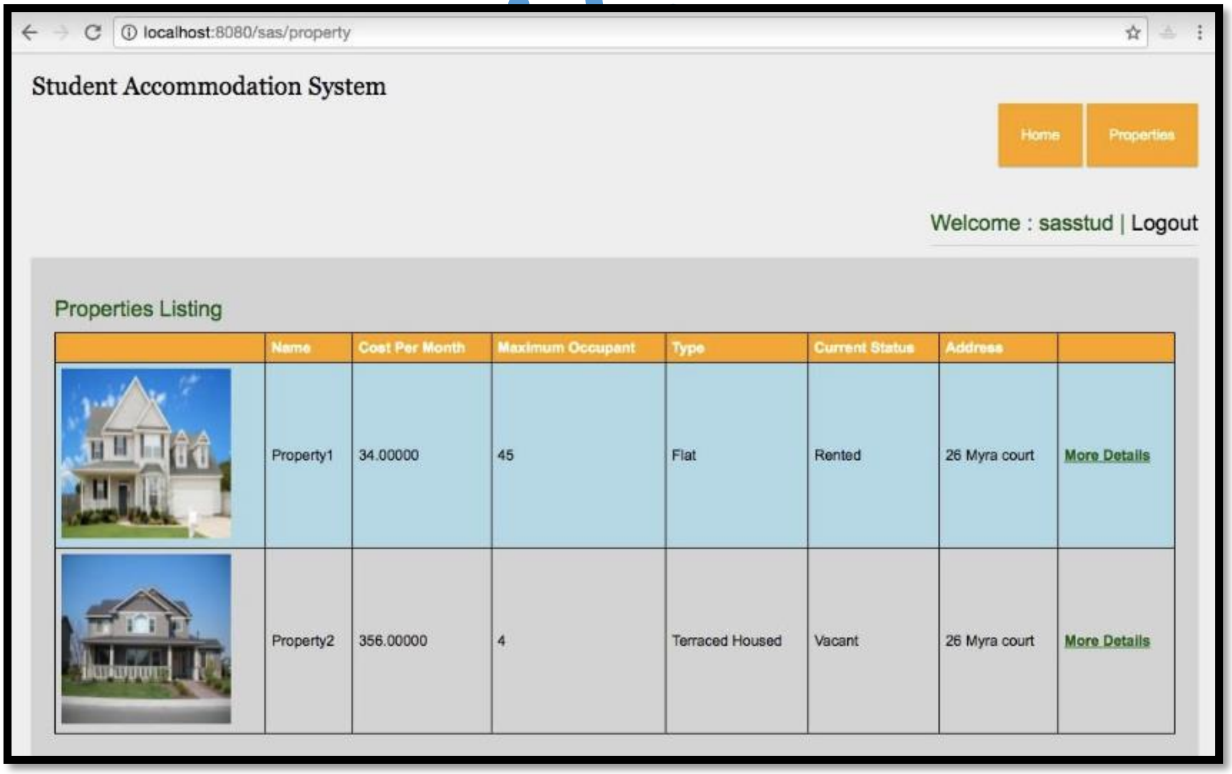


This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation
Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

Figure 19: Student Logged-in Home Page

7.1.4. Property Page

Figure 17 provides the property page screen printout. The student can view all the properties listed in the system. By clicking on the More Details link in the same row of the property. It is possible to view more information and photos about the property.



localhost:8080/sas/property
Student Accommodation System
Home
Properties
Welcome : sasstud | Logout
Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	
	Property1	34.00000	45	Flat	Rented	26 Myra court	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	More Details

Figure 20: Student Property Page

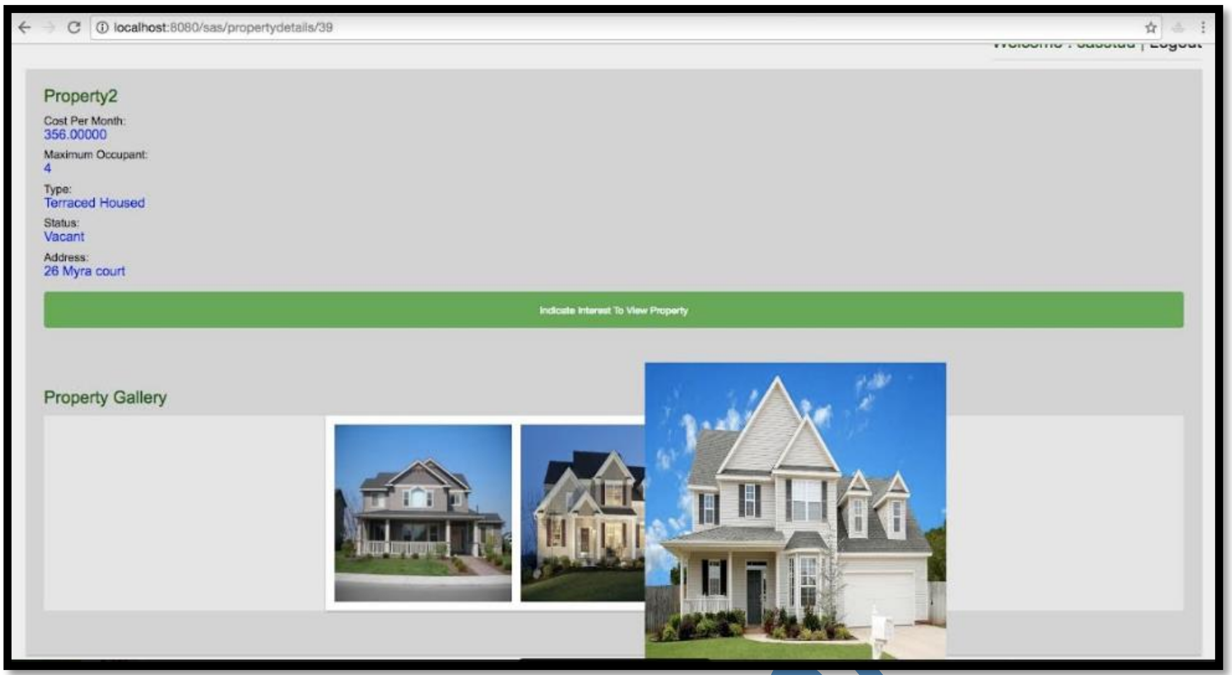
7.1.5. Property Details Page

Figure 18 provides the property details page of the student accommodation system. The page provides the detail information about the selected property such as the cost per month, maximum occupant, type, status and address.

In the property gallery, the uploaded different images of the property by the landlord will be displayed. The Student can move the mouse to the specific image, and the image will be zoomed in for the easy use.

The CSS code that is used to zoom the image when the mouse is on it, is provided below:

```
.gallery > div {
  position: relative;
  float: left;
  padding: 5px;
}
.gallery > div > img {
  display: block;
  width: 200px;
  transition: .1s transform;
  transform: translateZ(0); /* hack */
}
.gallery > div:hover {
  z-index: 1;
}
.gallery > div:hover > img {
  transform: scale(1.7,1.7);
  transition: .3s transform;
}
```


localhost:8080/sas/propertydetails/39

Property2

Cost Per Month:
356.00000

Maximum Occupant:
4

Type:
Terraced Housed

Status:
Vacant

Address:
26 Myra court

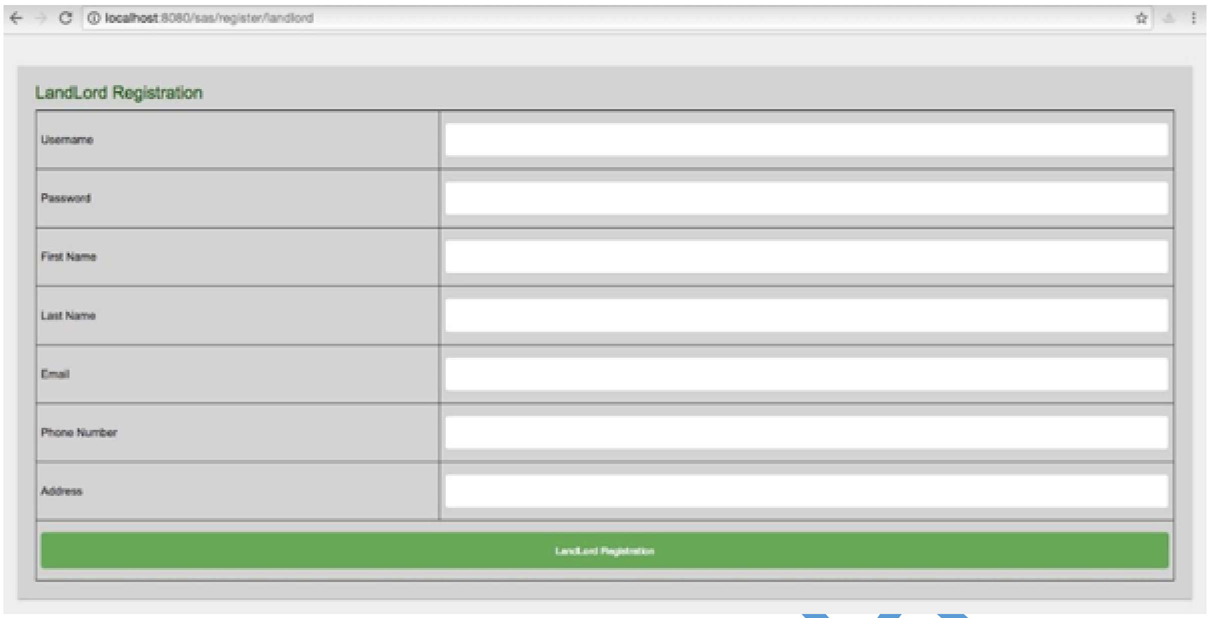
Property Gallery

Figure 21: Student Property Details Page

7.2. The Landlord Web pages

7.2.1. Registration Page

Figure 19 shows the register Page of the Landlord. The Username is a field that suggests the landlord provides a desired username for the future login purposes. There are other required information such as Password, first name, last name, E-mail address, Phone number and Address to register as a landlord in the portal. When a landlord provides valid information in the registration fields, the system will send an email to the landlord with the provided username and password to login to the system. See Figure 19, for the sample of email received by the landlord after the registration.



localhost:8080/sas/register/landlord

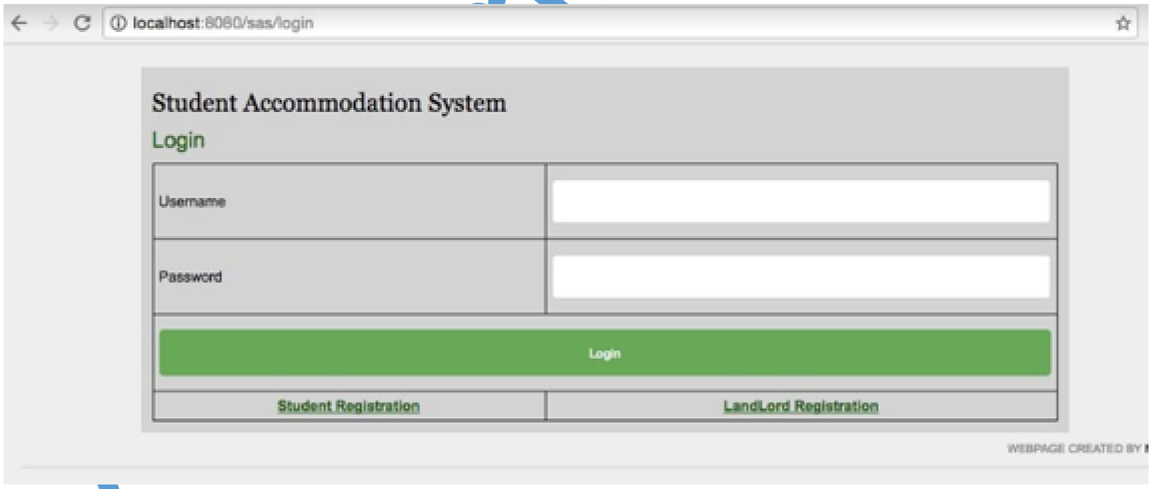
LandLord Registration

Username	
Password	
First Name	
Last Name	
Email	
Phone Number	
Address	

Figure 22: landlord Registration Page

7.2.2. Login Page

Figure 20 provides the screen printout of the landlord login form. The username and password that was received in the email after the registration will be used to login to the system. Once the correct credentials are provided, the landlord is required to press the login button to log in to the system.



localhost:8080/sas/login

Student Accommodation System

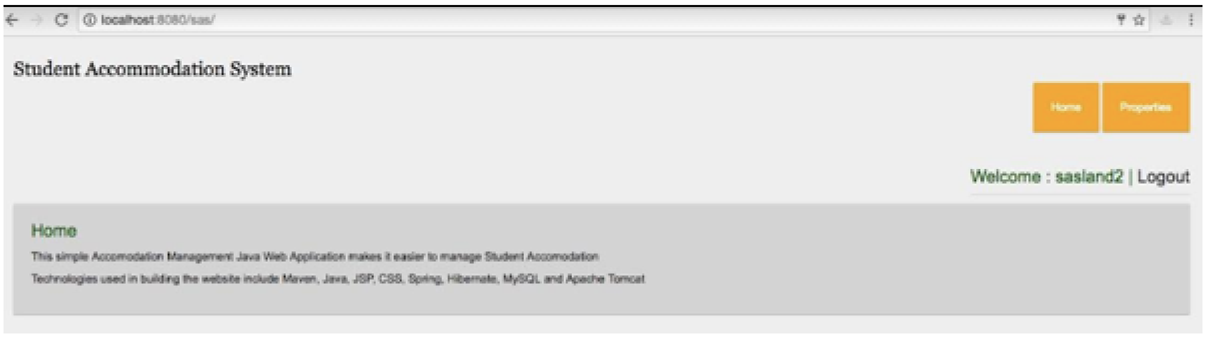
Login

Username	
Password	

Figure 23: Landlord Login Page

7.2.3. Logged-in Home Page

Figure 21 provides the home page of the landlord. When a landlord successfully logs in to the system, the home page will be displayed. In the home page, it allows the landlord to view properties and also to log out of the system.



Student Accommodation System

Welcome : sasland2 | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

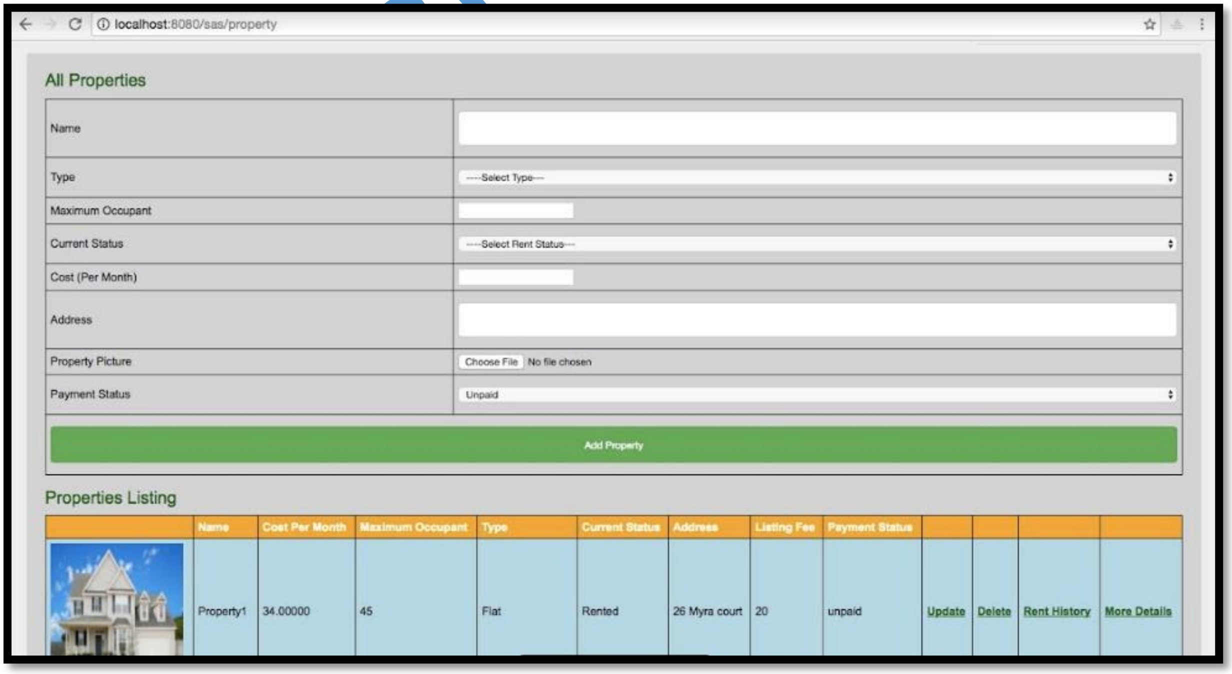
Figure 24: Landlord Home Page

7.2.4. Property Page

Figure 22 provides the property page of the landlord. The fields such as name, type, maximum occupant, current status, cost (per month), address, property pictures, and payment status are required to add a property to list of the system. The drop-down list values for property type field are Flat, Terraced House, Detached House, Bungalow, Duplex and Single Room.

The drop-down list values for current status which are rented and vacant. The property pictures field require the landlord to select images of type jpeg, png and GIF. If the images type differs from the mentioned format, the system will not load the image into the system.

The property will be displayed only to the landlord and admin, not for the student until the administrator confirms the payment for the listing in the back end.



All Properties

Name	
Type	----Select Type----
Maximum Occupant	
Current Status	----Select Rent Status----
Cost (Per Month)	
Address	
Property Picture	No file chosen
Payment Status	Unpaid

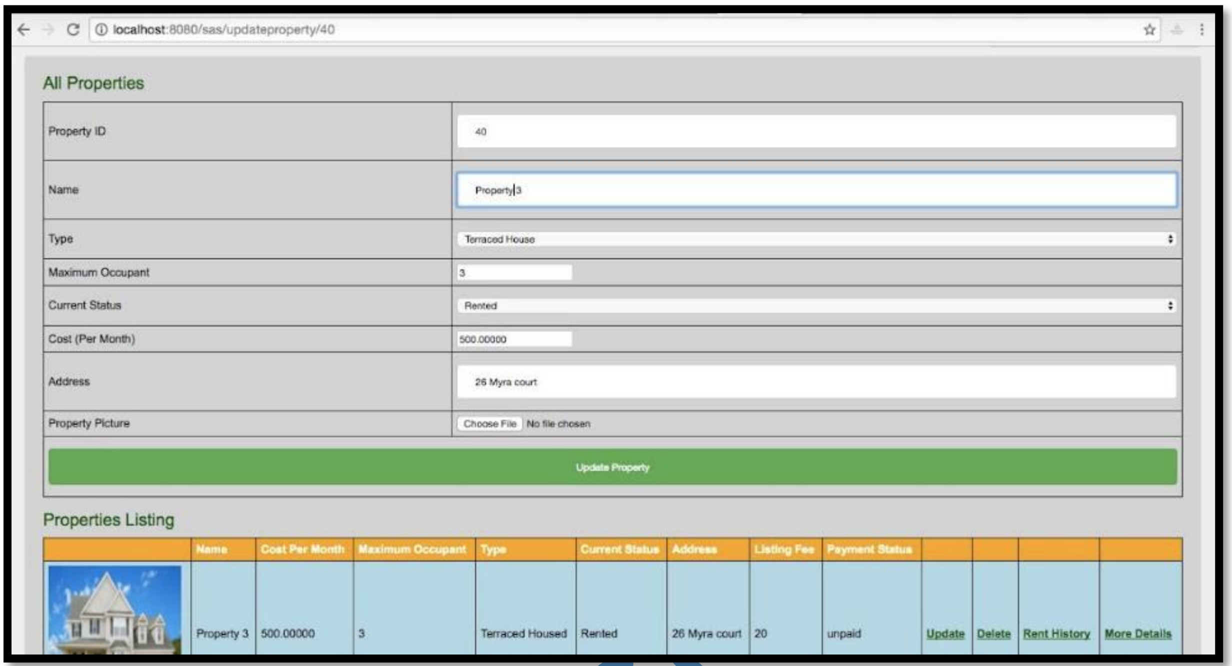
Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 25: Landlord Property Page

7.2.5. Update Property

Figure 23 provides the update property page. The landlord can select the update property from the table and update the information as shown in the figure.



localhost:8080/sas/updateproperty/40

All Properties

Property ID	40
Name	Property3
Type	Terraced House
Maximum Occupant	3
Current Status	Rented
Cost (Per Month)	500.00000
Address	26 Myra court
Property Picture	Choose File No file chosen

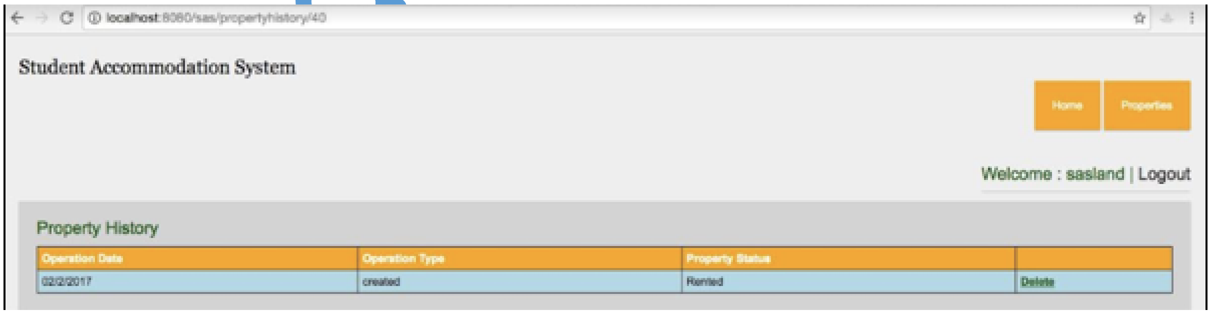
Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property 3	500.00000	3	Terraced Housed	Rented	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 26: landlord update property page

7.2.6. Property History Page

Figure 24 provides the property history page of the landlord. Operation date, operation type, and property status are the necessary information about a specific property. Also, it allows the landlord to delete the property history information.



localhost:8080/sas/propertyhistory/40

Student Accommodation System

Welcome : sasland | Logout

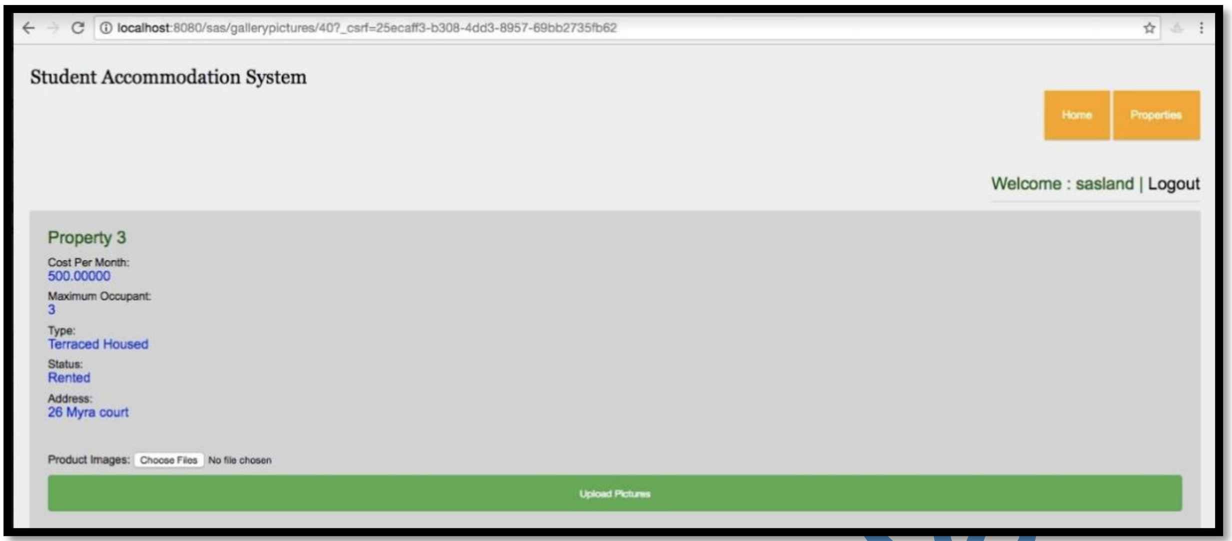
Property History

Operation Date	Operation Type	Property Status	
02/2/2017	created	Rented	Delete

Figure 27: Landlord property history page

7.2.7. Gallery Pictures

The upload image to the property gallery is provided in figure 25. The landlord can upload images as much as he can into the system for a specific property.


localhost:8080/sas/gallerypictures/40?_csrf=25ecaff3-b308-4dd3-8957-69bb2735fb62

Student Accommodation System

Welcome : sasland | Logout

Property 3

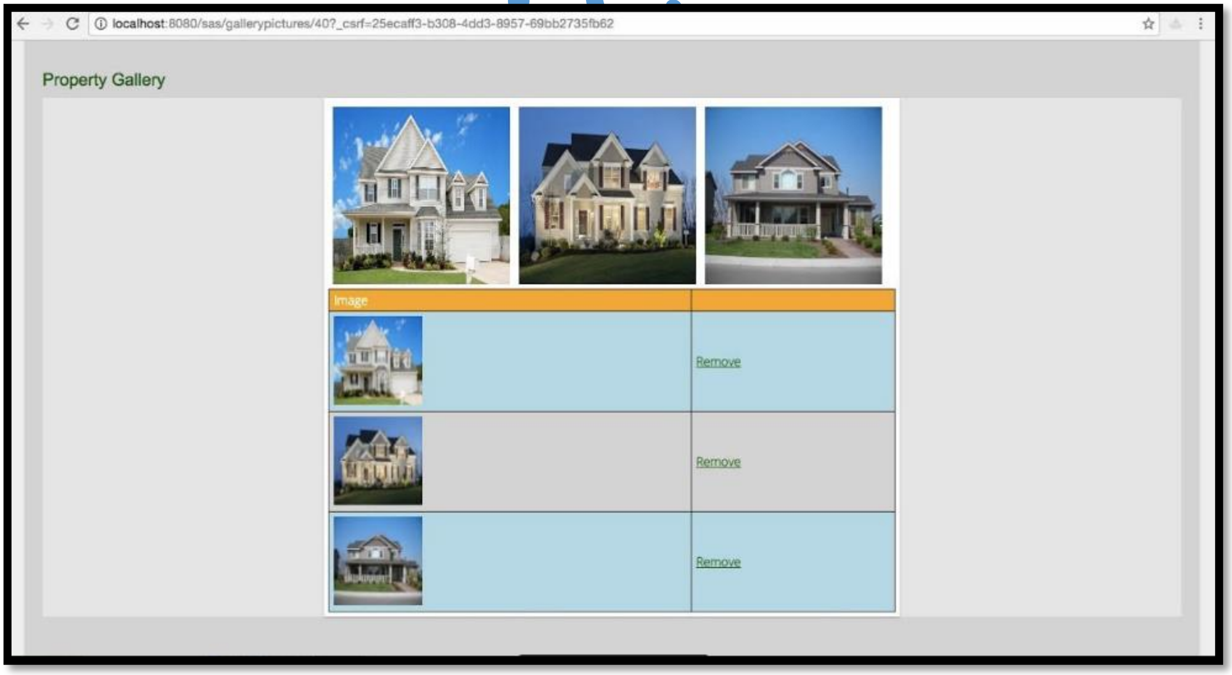
Cost Per Month:
500.00000
Maximum Occupant:
3
Type:
Terraced Housed
Status:
Rented
Address:
26 Myra court

Product Images: Choose Files No file chosen

Figure 28: Landlord upload gallery pictures

7.2.8. Remove gallery images page

The landlord can delete the images from the property gallery page (see figure 26). He can press on the Remove link to delete a specific image.


localhost:8080/sas/gallerypictures/40?_csrf=25ecaff3-b308-4dd3-8957-69bb2735fb62

Property Gallery

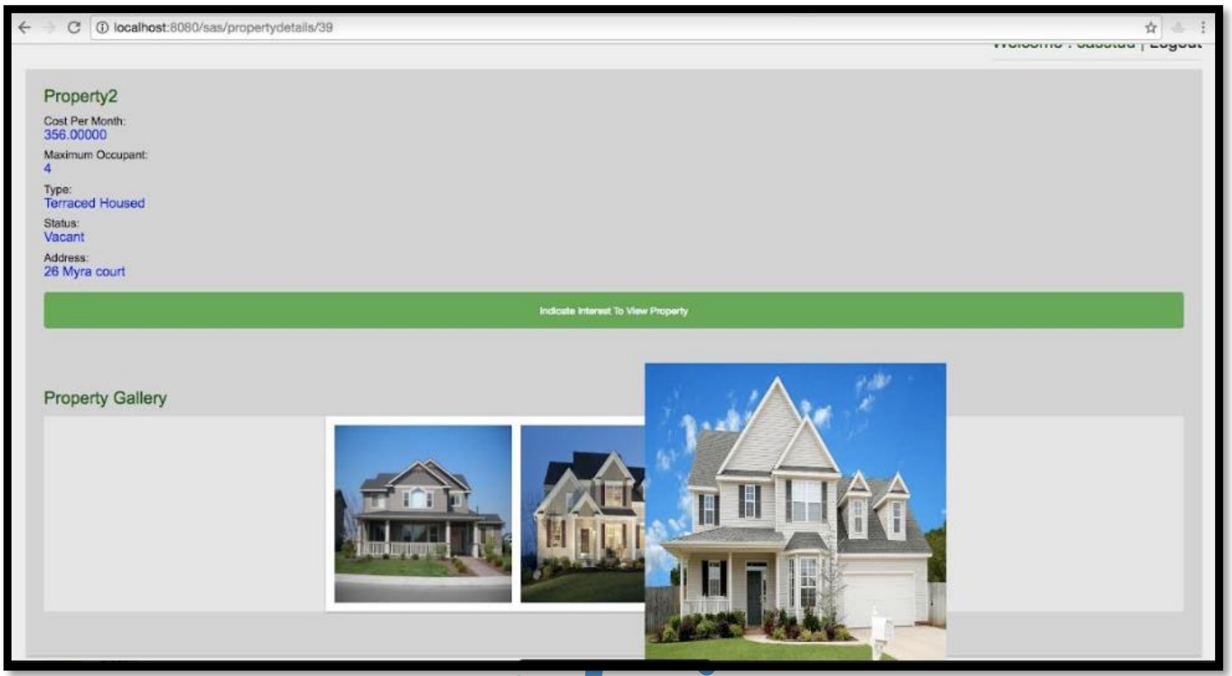
Image	
	Remove
	Remove
	Remove

Figure 29: Landlord remove gallery pictures

7.2.9. More details of the Property Page

Figure 27 provides the property details page of the student accommodation system. The page provides the detail information about the selected property such as the cost per month, maximum occupant, type, status and address.

In the property gallery, there will be a display of the uploaded different images of the property by the landlord. He can move the mouse to the specific image, and the image will be zoomed in for the easy use.



localhost:8080/sas/propertydetails/39

Property2

Cost Per Month:
356.00000

Maximum Occupant:
4

Type:
Terraced Housed

Status:
Vacant

Address:
26 Myra court

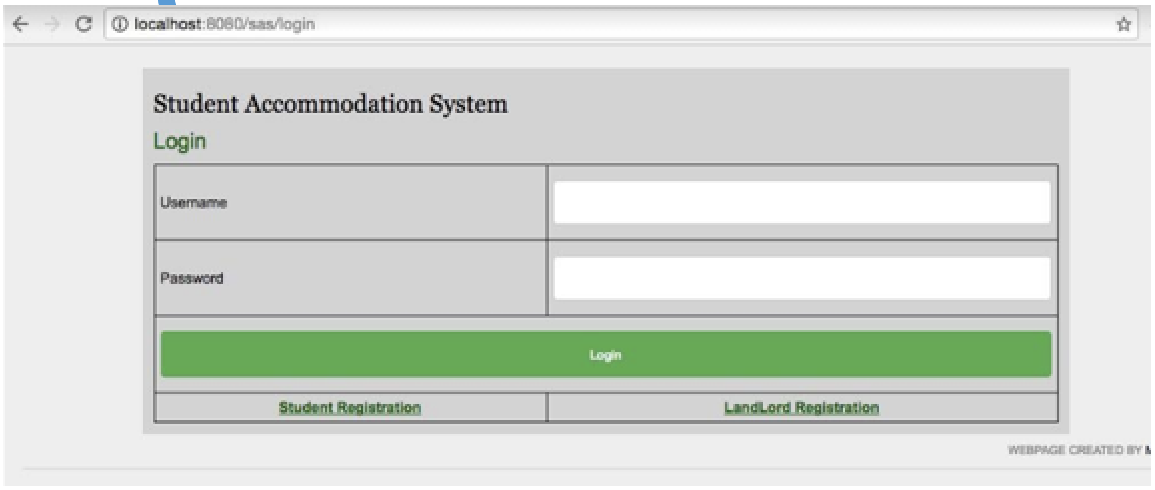
Property Gallery

Figure 30: Landlord more details of the property page

7.3. The Administrator Web pages

7.3.1. Login Page

Figure 28 provides the screen printout of the administrator login form. The username and password that was received in the email after the registration will be used to login to the system. Once the correct credentials are provided, the administrator is required to press the login button to log in to the system.



localhost:8080/sas/login

Student Accommodation System

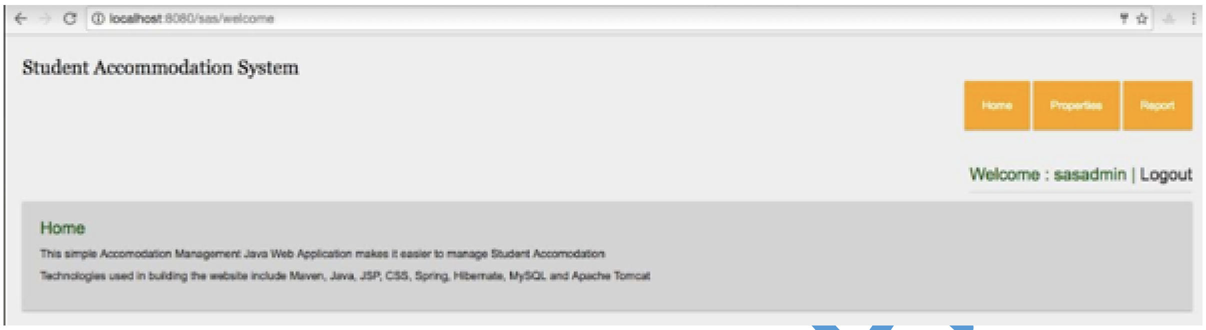
Login

Username	
Password	

Figure 31: Administrator Login Page

7.3.2. Logged-in Home Page

Figure 29 provides the home page of the administrator. When an administrator successfully logs in to the system, the home page will be displayed. In the home page, it allows the administrator to view properties and also to log out of the system.


localhost:8080/sas/welcome

Student Accommodation System

Welcome : sasadmin | Logout

Home

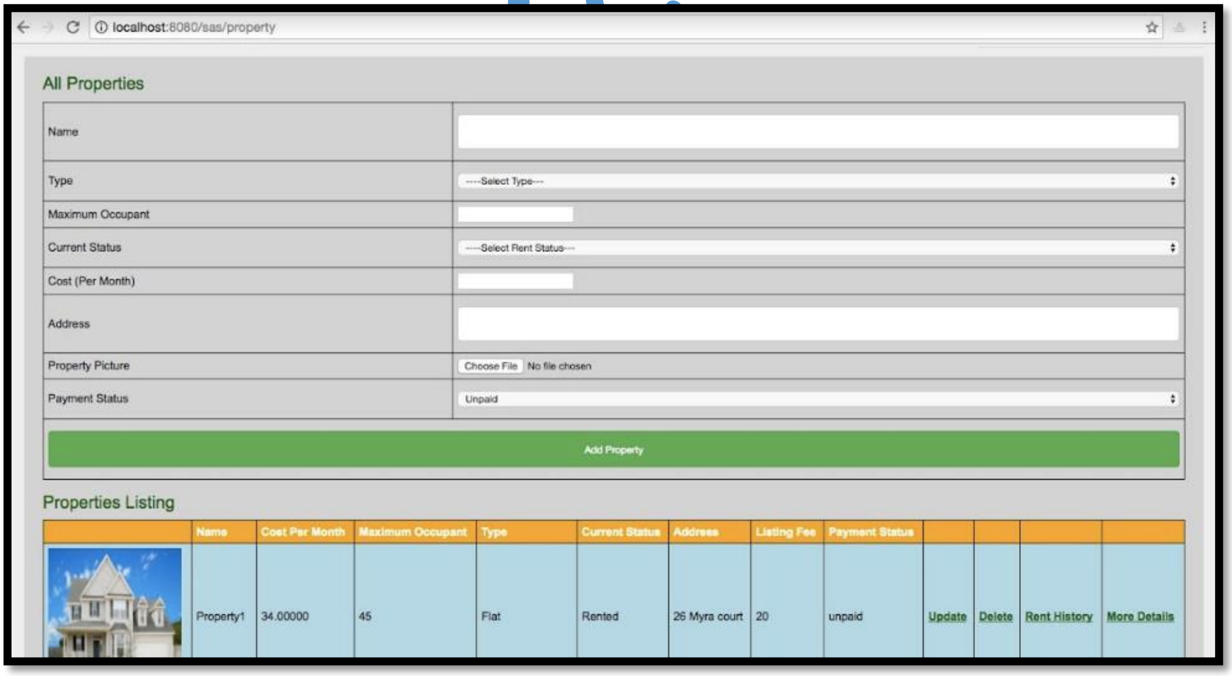
This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

Figure 32: Administrator Logged in Home Page

7.3.3. property Page

Figure 30 provides the screen printout of the property page. The administrator will be able to list the property in the system.


localhost:8080/sas/property

All Properties

Name	
Type	----Select Type----
Maximum Occupant	
Current Status	----Select Rent Status----
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen
Payment Status	Unpaid

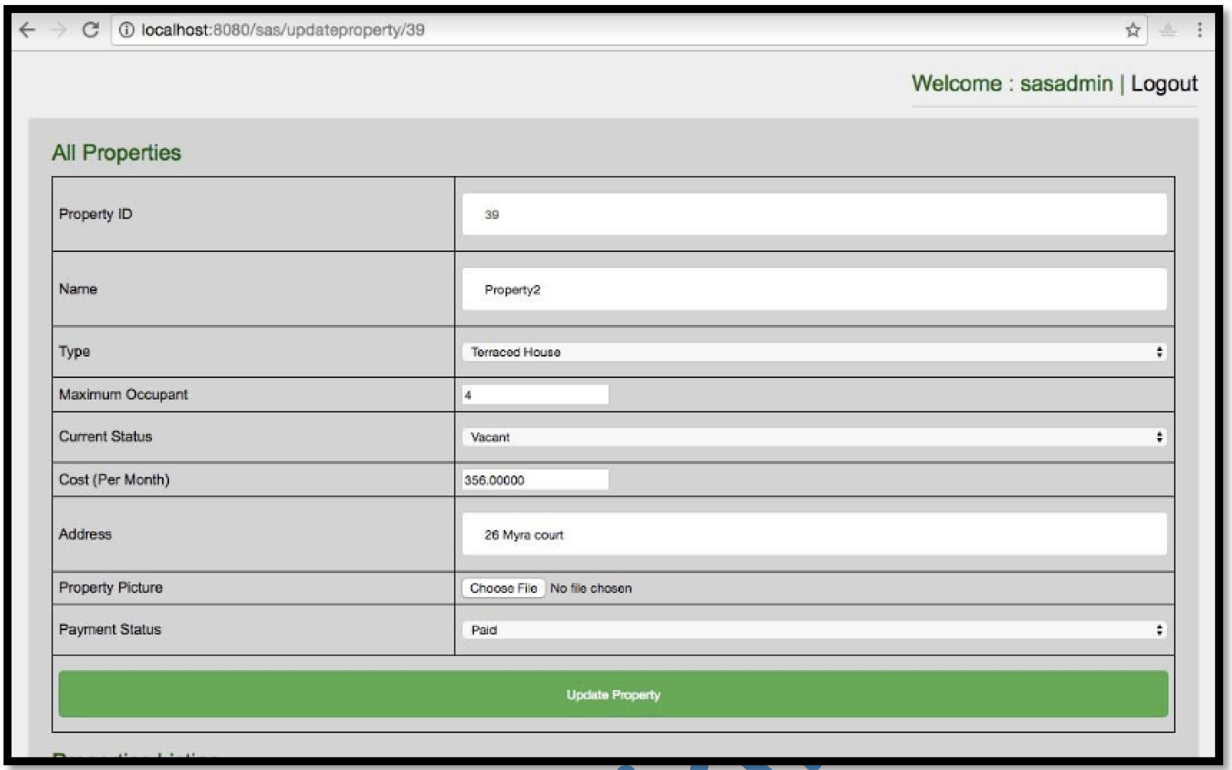
Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 33: Administrator property Page

7.3.4. Update Property Page

The below figure provides the option to update the property. The administrator should be able to update the property details as well as payment status to paid or unpaid.


localhost:8080/sas/updateproperty/39

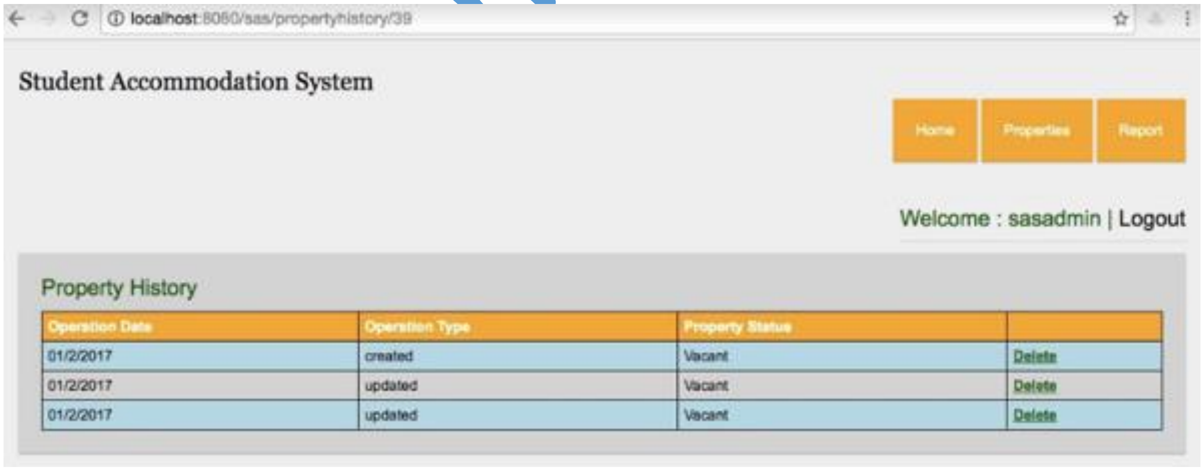
Welcome : sasadmin | Logout

All Properties

Property ID	39
Name	Property2
Type	Terraced House
Maximum Occupant	4
Current Status	Vacant
Cost (Per Month)	356.00000
Address	26 Myra court
Property Picture	Choose File No file chosen
Payment Status	Paid

7.3.5. Property History

The figure below provides the property history page of the administrator. Operation date, operation type, and property status are the necessary information about a specific property. Also, it allows the administrator to delete the property history information.


localhost:8080/sas/propertyhistory/39

Student Accommodation System

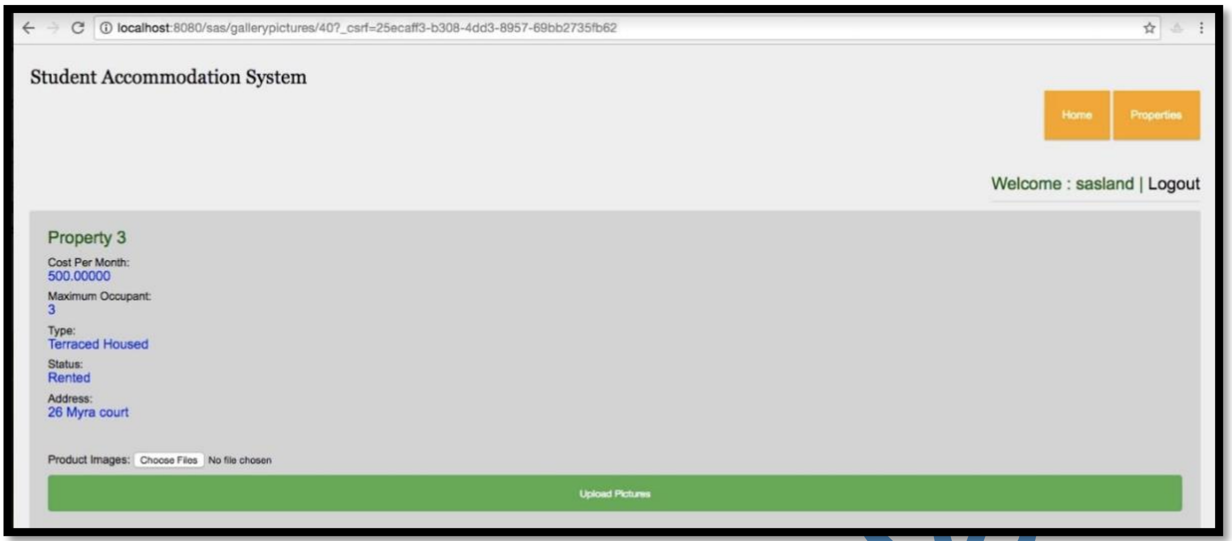
Welcome : sasadmin | Logout

Property History

Operation Date	Operation Type	Property Status	
01/2/2017	created	Vacant	Delete
01/2/2017	updated	Vacant	Delete
01/2/2017	updated	Vacant	Delete

7.3.6. Gallery Picture Page

The upload image to the property gallery is provided in the figure below. The administrator can upload images as much as he can into the system for a specific property.


localhost:8080/sas/gallerypictures/40?_csrf=25ecaff3-b308-4dd3-8957-69bb2735fb62

Student Accommodation System

Welcome : sasland | Logout

Property 3

Cost Per Month:
500.00000

Maximum Occupant:
3

Type:
Terraced Housed

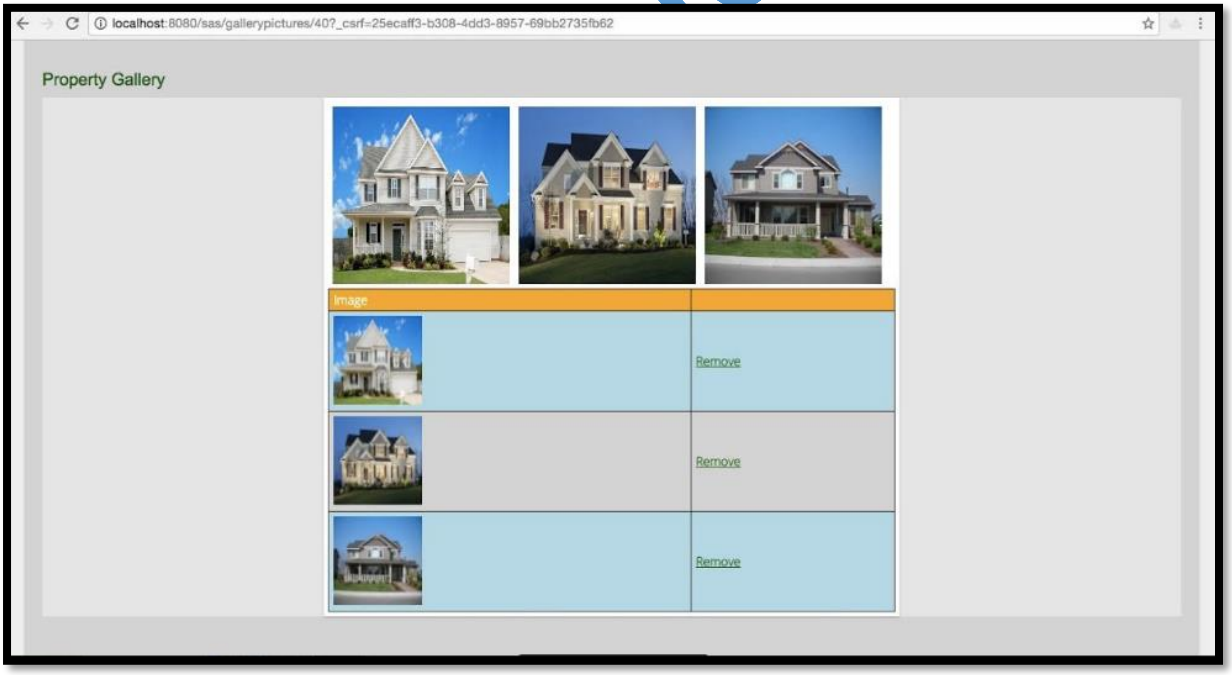
Status:
Rented

Address:
26 Myra court

Product Images: Choose Files No file chosen

7.3.7. Remove gallery

The administrator can delete the images from the property gallery page (see below figure). He can press on the Remove link to delete a specific image.


localhost:8080/sas/gallerypictures/40?_csrf=25ecaff3-b308-4dd3-8957-69bb2735fb62

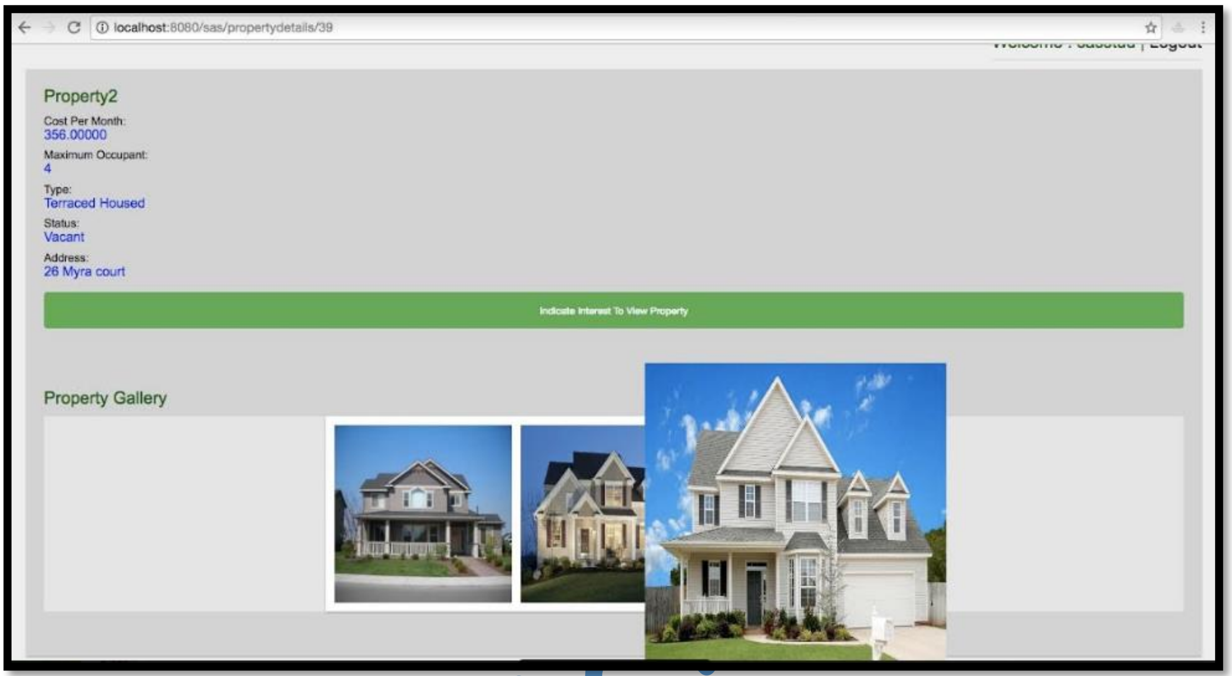
Property Gallery

Image	
	Remove
	Remove
	Remove

7.3.8. More details of the property Page

The below figure provides the property details page of the student accommodation system. The page provides the detail information about the selected property such as the cost per month, maximum occupant, type, status and address.

In the property gallery, the different images of the property that he uploaded will be displayed. The administration can move the mouse to the specific image, and the image will be zoomed in for the easy use.



localhost:8080/sas/propertydetails/39

Property2

Cost Per Month:
356.00000

Maximum Occupant:
4

Type:
Terraced Housed

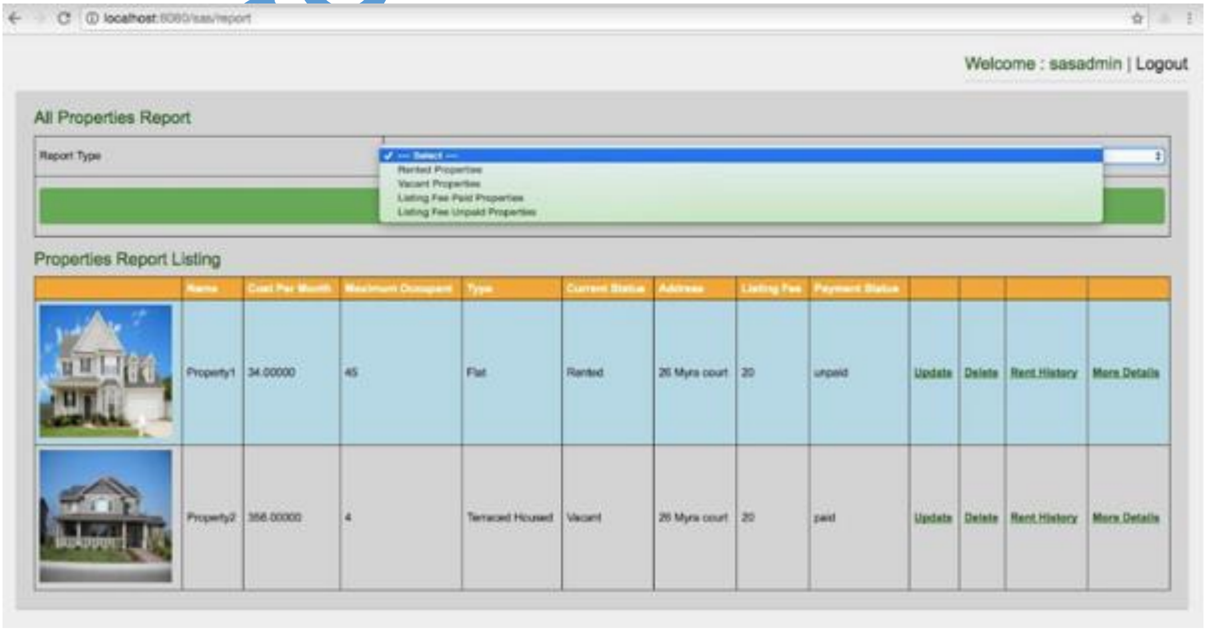
Status:
Vacant

Address:
26 Myra court

Property Gallery

7.3.9. Report Page

The below figure provides the report page. The administrator will be able to view the following report by selecting the drop-down list which include rented properties, vacant properties, the listing fee paid properties and listing fee unpaid properties.



localhost:8080/sas/report

All Properties Report

Report Type: ✓ --- Select ---
- Rented Properties
- Vacant Properties
- Listing Fee Paid Properties
- Listing Fee Unpaid Properties

Properties Report Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	20	paid	Update	Delete	Rent History	More Details

8.0. Testing

There are several testings that took place in the student accommodation system including unit testing, integration testing, system testing or acceptance testing and the security testing.

8.1. Test Environment

The testing which will be carried out in the next section will not have any value if the environments are not identical. Ideally, the functional tests, unit tests and integration tests are tested in similar environments. Most importantly the performance tests change up to some level based on the environment.

Table 29: Test Environment

Required Processor	Intel Core 2 Duo (2 Hz)
RAM	3GB DDR 2
Operating System	Windows 7
Platform	32 bit/ 64 bit
Other Tools	Java

8.2. Unit testing

There are several errors identified in the unit testing. Most of the errors are with the validation of data that the application should process only the required information. All the encountered errors were rectified immediately. Table 1 provides the unit testing conducted and its evidence.

Table 30: Unit testing

Unit to Test	Comment	Evidence
Student Login	The unit is working as anticipated	Appendix A
Administrator Login	The unit is working as anticipated	Appendix B
Landlord Login	The unit is working as anticipated	Appendix C
Student View Properties	The unit is working as anticipated	Appendix D

Student View More information about the property	The unit is working as anticipated	Appendix E
Student Registration	The unit is working as anticipated	Appendix F R
Landlord Property listing	The unit is working as anticipated	Appendix G
Landlord include more images to the property gallery	The unit is working as anticipated	Appendix H
Landlord delete images from the property gallery	The unit is working as anticipated	Appendix I
Landlord update the property listed	The unit is working as anticipated	Appendix J
Landlord view and delete property history	The unit is working as anticipated	Appendix K
Administrator list a property	The unit is working as anticipated	Appendix L G
Administrator update the listed property information	The unit is working as anticipated	Appendix M j
Administrator should be able to delete images from the property gallery	The unit is working as anticipated	Appendix N I
Administrator should be able to upload more images to the property	The unit is working as anticipated	Appendix O H
Administrator logout	The unit is working as anticipated	Appendix P
Administrator view and delete property history	The unit is working as anticipated	Appendix Q K

Landlord Registration	The unit is working as anticipated	Appendix R

8.3. Integration testing

The integration testing is mainly to focus on whether the data are transmitted from one part of the system to another part of the system. The errors were fixed. Table 2 provides the unit testing conducted and the evidence to it.

Table 31: Integration testing

Integration testing	Comment	Evidence
To confirm that the student's successful login took him to the appropriate home page	The integration is working as anticipated	Appendix S
To confirm that the Landlord's successful login took him to the appropriate home page	The integration is working as anticipated	Appendix T
To verify that the administrator's successful login took him to the appropriate home page	The integration is working as anticipated	Appendix U
To verify that the landlord's listed property, which is unpaid is not seen by the student	The integration is working as anticipated	Appendix V
To verify that the administrator can update tlhe landlord's listed property from unpaid to paid	The integration is working as anticipated	Appendix W
To verify that if the administrator lists a property, student can view it	The integration is working as anticipated	Appendix X

8.4. System testing or acceptance testing

The system testing or acceptance testing was done against the functional requirement identified at the beginning of the project. Table 3 verifies the functional requirements.

Table 32: System testing and acceptance testing

Functional Requirements	Comment
The system should have three major users such as Administrator, student and landlord.	Working as anticipated
The system user student should be able to register on the system	Working as anticipated
The students should receive an email notification with their username and password.	Working as anticipated
The students should be able to book for viewing the property	Working as anticipated
The student should be able to login to the system to book for viewing.	Working as anticipated
The system user landlord should be able to register on the system.	Working as anticipated
The landlord should be able to login to the system	Working as anticipated
The landlord should be able to list the property to rent in the system	Working as anticipated
The landlord should be able to edit the already listed properties.	Working as anticipated
The landlord should be able to delete the already listed properties. The landlord should be able to add profile pictures and other gallery pictures	Working as anticipated
The administrator should be able to login to the system.	Working as anticipated

The administrator should be able to view the listed properties.	Working as anticipated
The administrator should be able to update the payment for the listed properties.	Working as anticipated
The listed property by the landlord will only be visible to the students when the administrator of the system confirms the payment.	Working as anticipated
The administrator can view the paid properties report	Working as anticipated
The administrator can view the unpaid properties report	Working as anticipated
The administrator can view the vacant properties report	Working as anticipated
The administrator can view the occupant properties report	Working as anticipated

8.5. Security testing

The security of the system was tested, and the test results were provided in table 4.

Table 33: Security testing

Security test cases	Comment	Evidence
Student authentication was accurate	Working as expected	Appendix Y
Landlord authentication was accurate	Working as expected	Appendix Z
Administrator authentication was accurate	Working as expected	Appendix AA
When a property is listed, and the listing fee is not paid, the student cannot view the property details	Working as expected	Appendix AB

When a student tries to access the administrator, report using the URL – it should not allow	Working as expected	Appendix AC

9.0. Evaluation

Several ethical issues can be faced by the development of the student accommodation information system, they include: ownership, privacy, and phishing scam

The detail description of the ethical issues and how the student accommodation system overcomes the issues are provided below:

Ethical Issue 1: Ownership

It is important that the landlord who is listing the property should be the actual and absolute property owner. It is possible that fraudulent people can open account pretending to be a landlord and defraud the students.

The student accommodation system provides full control to the administrator, and the administrator can remove or modify any information related to the property. Therefore, if a complaint comes regarding the property or landlord, then the administrator can remove the account information from the system.

Ethical Issue 2: Privacy

It is crucial that the personal information of the landlord and student as well as the property details be kept secured in the database. The unauthorised users should not be able to view the information. For example, student phone number, address and email address are personal and sensitive information, therefore, this information should be modified, or lost.

The student accommodation system provides the secured website authentication to view the information. It is only student that is logged in with the authorised credentials that can view his or her personal information.

Ethical Issue 3: Phishing Scam

The fraudulent email will be sent to the user of the system to get their credit card information.

The student accommodation system does not involve any online payment, therefore, it is not possible for its users such as landlord and student to give their credit or debit card details.

Usage of Web Services

The server design includes web services which increase flexibility and platform and language agnosticism. SOAP and RESTful web services are the two types of web services.

SOAP Web Services

SOAP web services rely on XML based messages to transfer data. The advantage of a SOAP web service is that since it adheres to a contract, there is rigid type checking. It is also versatile enough to allow for the use of different transport protocols. Additionally, it has more widespread support from development tools, and contains more resources online. The disadvantage with SOAP is that it contains much overhead due to the use of XML and is slower than REST as a result. XML is wrapped with a response and request using an XML wrapper, thereby consuming more bandwidth. It also makes the client-side more complicated by requiring client stubs. On the security front, it can be argued that since SOAP uses POST method for all communications, it has more security threat than with REST, since each SOAP message would need to be considered.

RESTful Web Services

Every URL is a blueprint of an object in RESTful web services. There are four methods in HTTP that are used by RESTful web services. They are used for communication such as GET, PUT, POST and DELETE. An advantage of a REST web service is that it is lightweight and has less overhead when compared with SOAP. Therefore, it uses less bandwidth for requests and responses. Another advantage is that the results given by the service are in human readable format.

Also, since it uses HTTP for communication, it will not be hindered by firewall setups. On the security front, it can be considered to be more secure than SOAP in the sense that, GET calls and it can be immediately cleared since it only retrieves data, whereas with SOAP all communications usthe the POST. Based on the above research, the author selected RESTful web services as the best-suited service type.

Security

Since RESTful services make use of HTTP, there is not much security when transferring data. Securely transferring data is an important factor in the system since we are dealing with real-time data regarding users‟ locations. To provide more security to the system, the author decided to use HTTPS communication for all requests and responses. It provides encrypted communication to prevent eavesdropping and secure identification of a network web server, to know the exact web server that the researching is discussing with. When connecting to a server, HTTPS makes it possible to know whether the researcher is talking to the right server and protects it from passive and active network attacks such as Man-in-the-middle attacks. During a session, it can protect against eavesdropping and tampering with the information the researcher sends to the server.

Coding Standards Adhered

- All variable and class names in Java named in camel case.
- Standard Java coding standards were used for Java development
- Official coding standards released by Adobe (Adobe Systems Inc.,2009) were used as guidelines for flex development.
- HTML was formatted according to the guidelines established by W3C (W3C,2009).
- All methods and computations are adequately commented

10.0. Documentation

10.1. User documentation

10.1.1. The Administrator Manual

The login information that can be used to login to the student accommodation system is provided below:

- **Username: sasadmin**
- **Password: sasadmin123**

10.1.2. The Student Manual

The user can register as a student or use the following login information to log in to the student accommodation system.

- **Username: sasstud**
- **Password: sasstud123**

10.1.3. The landlord Manual

The user can register as a landlord or use the following login information to log in to the landlord accommodation system.

- **Username: sasland**
- **Password: sasland123**

10.2. Technical documentation

Step 1: Install Eclipse for the computer

Step 2: import the java project into the Eclipse

Step 3: Start the MySQL

Step 4: Import the database using PHPMyAdmin or MySQL workbench

Step 5: Right click on the application -> Run As -> Run Configuration. In the Goals text field type Clean install tomcat7:run

Step 6: Type the following link in the browser.

http://localhost:8080/sas/

11.0. Conclusion and Future work

11.1. Conclusion

The student accommodation system allows the landlords to list their properties and students can book to view these properties. The system was developed mainly using Java programming language, Hibernate, Spring, Maven and MySQL.

11.2. Future work

The list of future works for the student accommodation system is provided below:

- An online payment should be provided
- The landlord should upload a legal document to prove that the property belongs to him.
- The student should be able to chat with the landlord through the application
- Developing an Android and IOS application will enhance the usefulness of the application

12.0. References

Bjørner, D. (2006). Software engineering. 1st ed. Berlin: Springer-Verlag.

Cortez, R. and Vazhenin, A. (2015). Virtual model-view-controller design pattern: Extended MVC for service-oriented architecture. IEEJ Transactions on Electrical and Electronic Engineering, 10(4), pp.411-422.

Faulkner, C. (2009). Software engineering. 1st ed. Chandni Chowk, Delhi: Global Media.

Holt, J. (2004). UML for systems engineering. 1st ed. London: Institution of Electrical Engineers.

Ko, J. and Song, Y. (2011). A Study on Model Transformation Mechanism Using Graph Comparison Algorithm, Abstract Factory Pattern and Bridge Pattern. Applied Mechanics and Materials, 121-126, pp.2476-2481.

Lyon, D. and Castellanos, F. (2007). The Parametric Singleton Design Pattern. The Journal of Object Technology, 6(3), p.13.

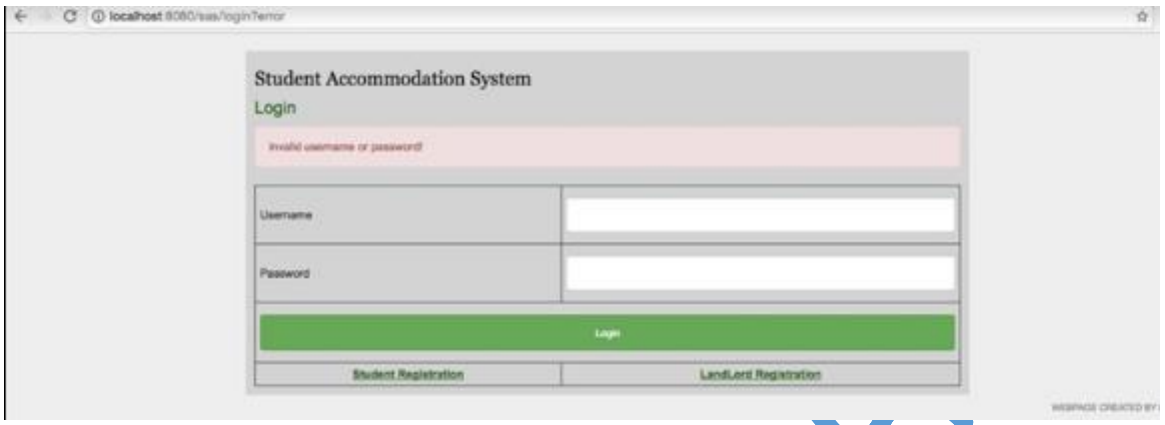
Rost, J. (2004). Is "Factory Method" really a pattern?. ACM SIGSOFT Software Engineering Notes, 29(5), p.1.

Sommerville, I. (2000). Software engineering. 1st ed. Harlow, England: Addison-Wesley.

Theodoridis, S. and Koutroumbas, K. (2006). Pattern recognition. 1st ed. Amsterdam: Elsevier/Academic Press.

APPENDIX A: Student Login Unit testing

When the username and password is wrong the following error message will be displayed in the system



localhost:8080/sas/login?error

Student Accommodation System

Login

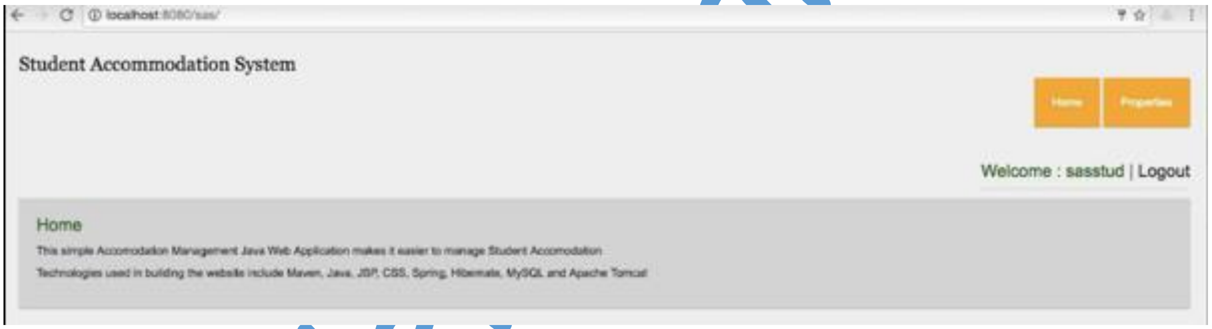
Invalid username or password!

Username

Password

Figure 34: Error Message for username or password wrong

The homepage for the student will be displayed



localhost:8080/sas/

Student Accommodation System

Welcome : sasstud | Logout

Home

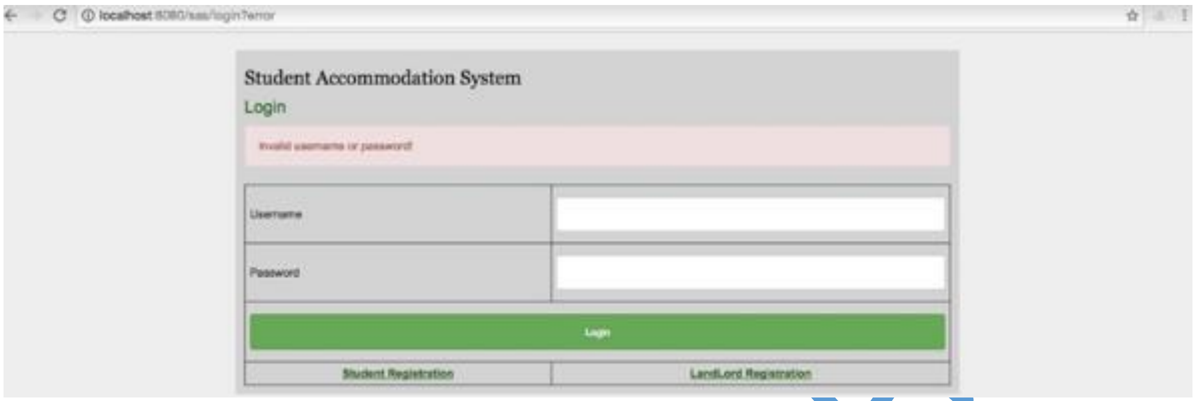
This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

Figure 35: Successfully logged in homepage

APPENDIX B: Administrator Login Unit testing

When the username and password is wrong the following error message will be displayed in the system


localhost:8080/sas/login?error

Student Accommodation System

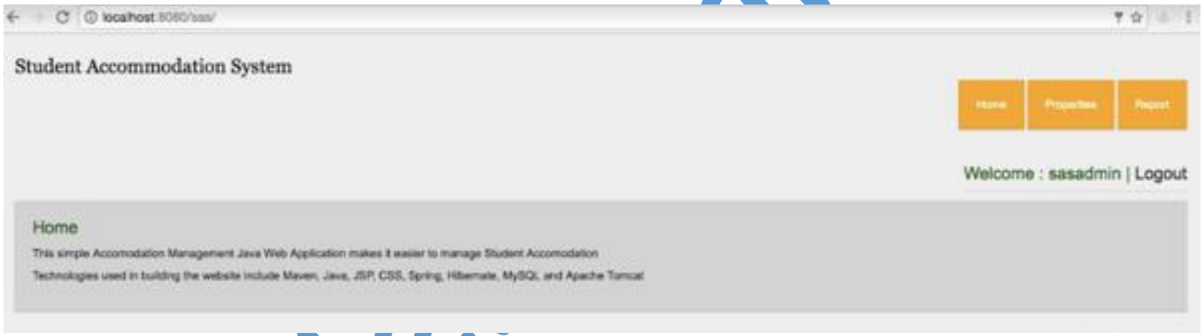
Login

Invalid username or password!

Username	
Password	

Figure 36: Error Message for username or password wrong

The homepage for the student will be displayed


localhost:8080/sas/

Student Accommodation System

Welcome : sasadmin | Logout

Home

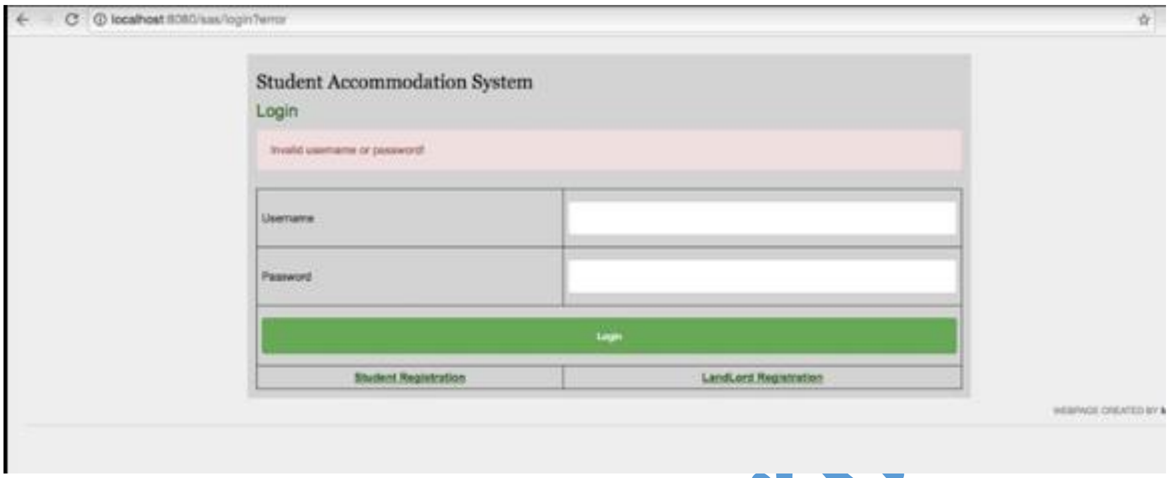
This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

Figure 37: Successfully logged in homepage

APPENDIX C: Landlord Login Unit testing

When the username and password is wrong the following error message will be displayed in the system


localhost:8080/sas/login?error

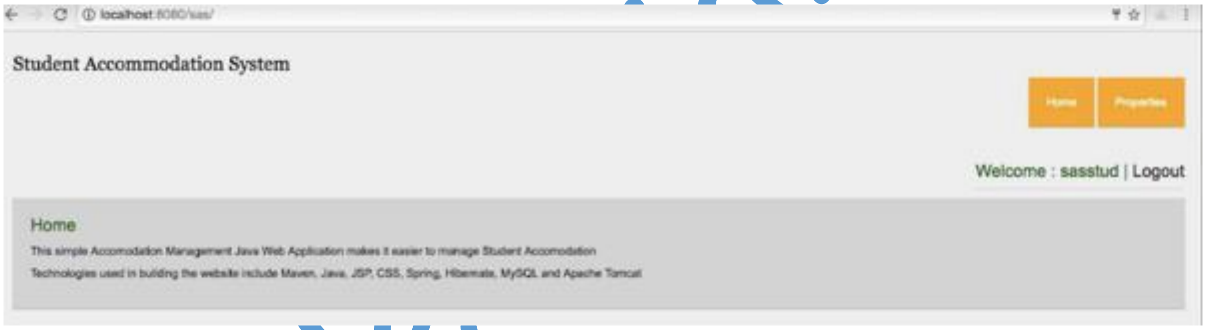
Student Accommodation System

Login

Invalid username or password!

Username	
Password	

Figure 38: Error Message for username or password is wrong


localhost:8080/sas/

Student Accommodation System

Welcome : sasstud | Logout

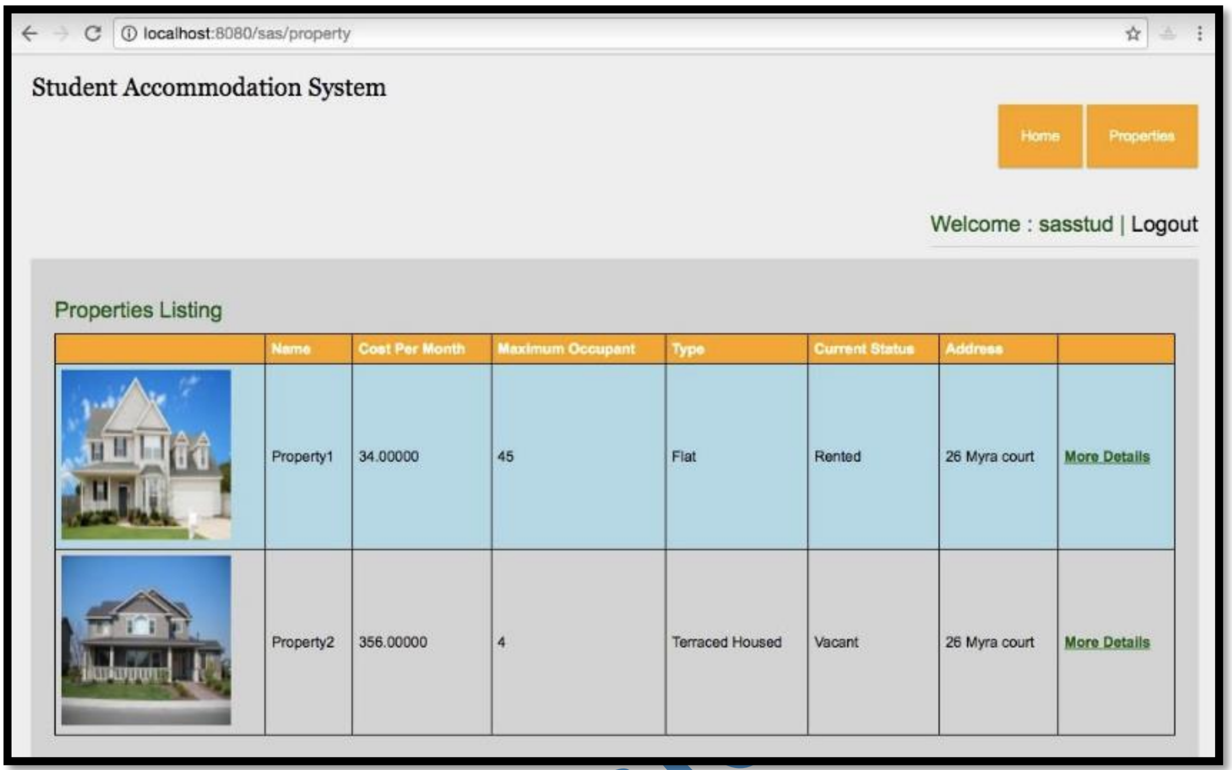
Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in bulding the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

Figure 39: Successfully logged in homepage

APPENDIX D: Student View Property Unit testing


localhost:8080/sas/property

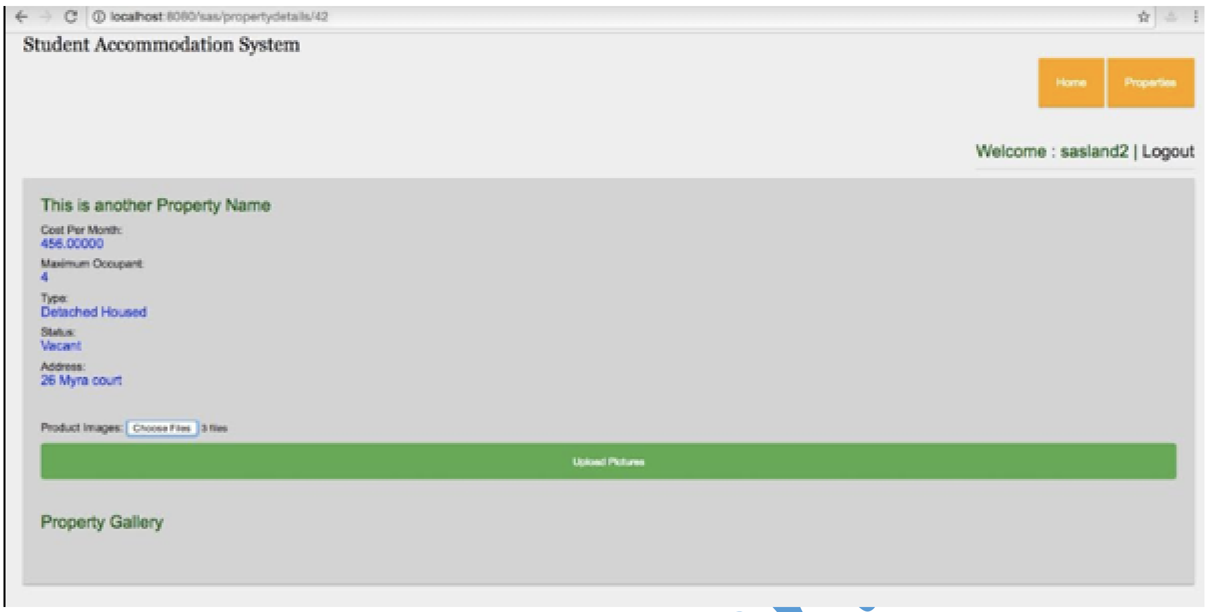
Student Accommodation System

Welcome : sasstud | Logout

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	
	Property1	34.00000	45	Flat	Rented	26 Myra court	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	More Details

APPENDIX E: Student View More information about the property Unit testing



Student Accommodation System

Welcome : sasland2 | Logout

This is another Property Name

Cost Per Month:
456.00000

Maximum Occupant:
4

Type:
Detached Housed

Status:
Vacant

Address:
26 Myra court

Product Images: Choose Files 3 files

Property Gallery

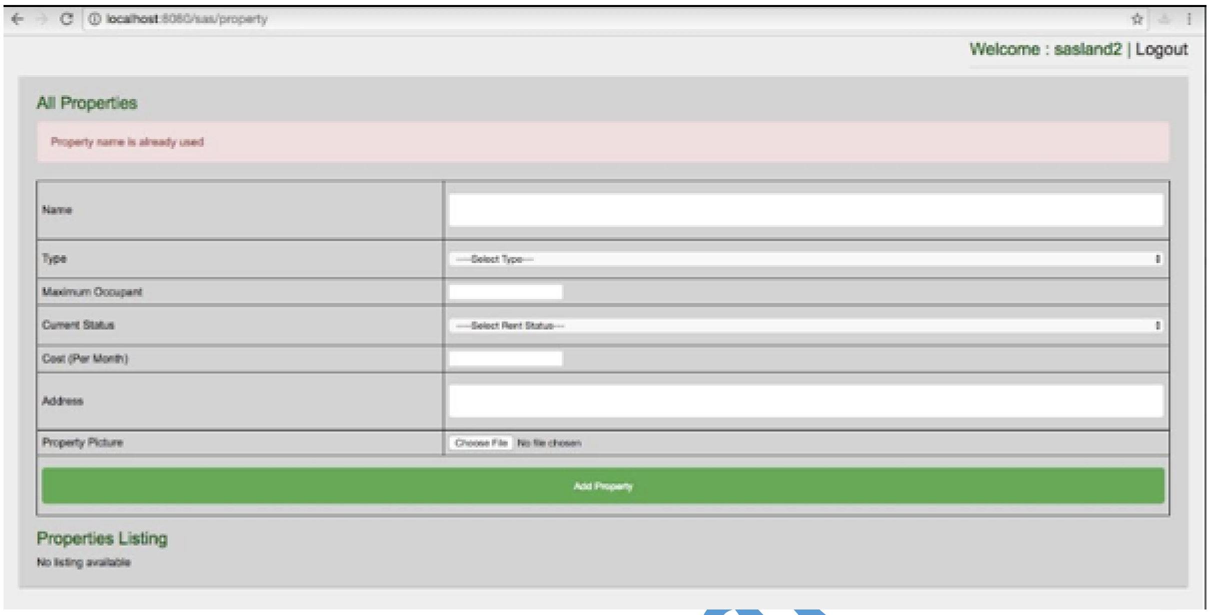
Figure 40: View more information about the property

APPENDIX F: Student Registration Unit testing

It is same as Appendix R

APPENDIX G: Landlord Property listing; Unit testing

When the property name already exists, the following error will be displayed



localhost:8080/sas/property

Welcome : sasland2 | Logout

All Properties

Property name is already used

Name	
Type	---Select Type---
Maximum Occupant	
Current Status	---Select Rent Status---
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen

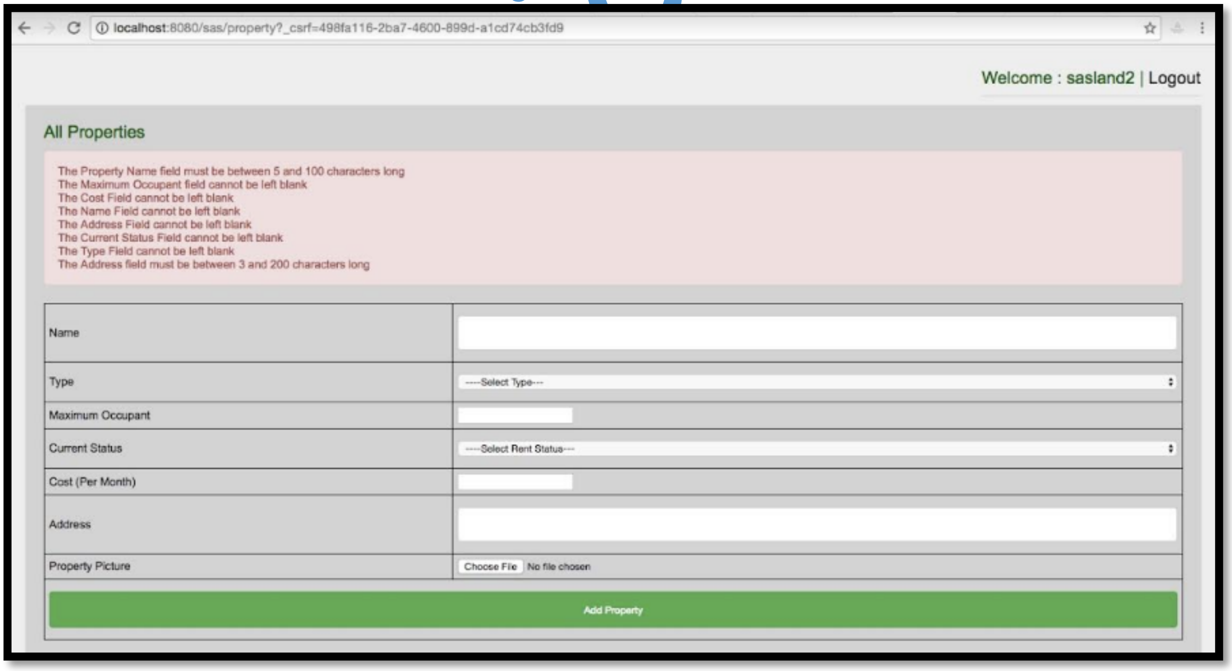
Add Property

Properties Listing

No listing available

Figure 41: When Property name exists

When the fields are left empty, the following errors will be displayed



localhost:8080/sas/property?_csrf=498fa116-2ba7-4600-899d-a1cd74cb3fd9

Welcome : sasland2 | Logout

All Properties

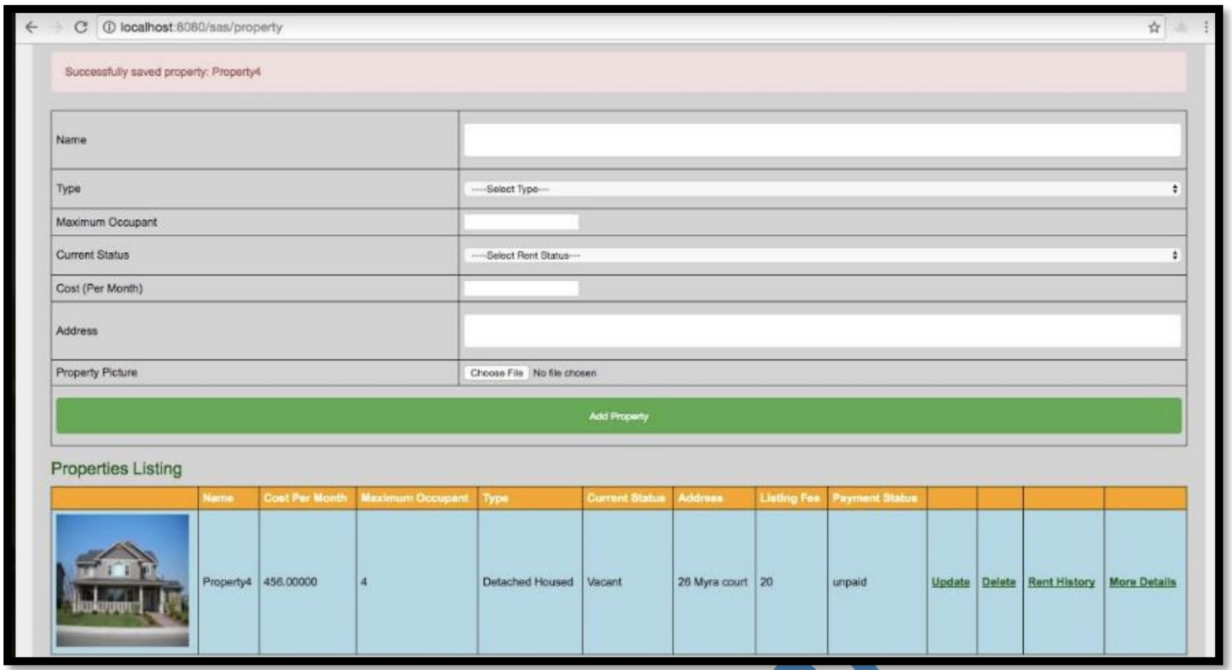
The Property Name field must be between 5 and 100 characters long
The Maximum Occupant field cannot be left blank
The Cost Field cannot be left blank
The Name Field cannot be left blank
The Address Field cannot be left blank
The Current Status Field cannot be left blank
The Type Field cannot be left blank
The Address field must be between 3 and 200 characters long

Name	
Type	----Select Type---
Maximum Occupant	
Current Status	----Select Rent Status---
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen

Add Property

Figure 42: When the fields are empty

When all the information is provided, the property will be added.



Successfully saved property: Property4

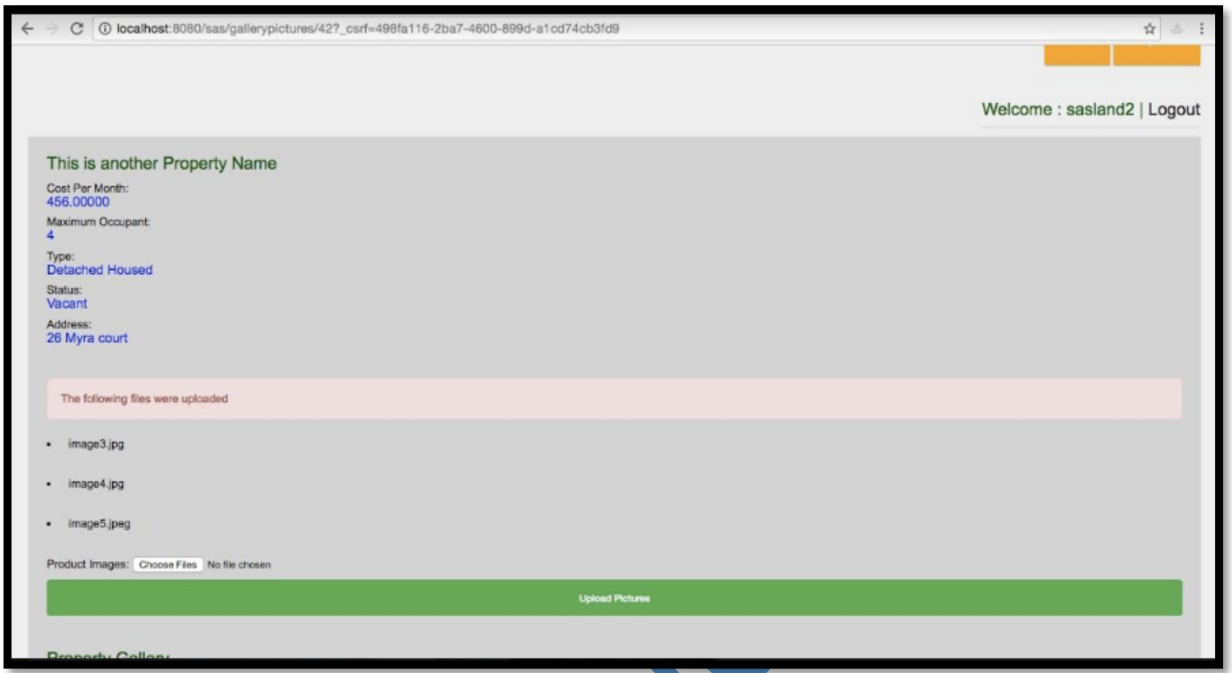
Name	
Type	----Select Type----
Maximum Occupant	
Current Status	----Select Rent Status----
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property4	456.00000	4	Detached Housed	Vacant	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 43: The property was successfully added

APPENDIX H: Landlord include more images to the property gallery Unit testing


localhost:8080/sas/gallerypictures/42?_csrf=498fa116-2ba7-4600-899d-a1cd74cb3fd9

Welcome : sasland2 | Logout

This is another Property Name

Cost Per Month:
456.00000

Maximum Occupant:
4

Type:
Detached Housed

Status:
Vacant

Address:
26 Myra court

The following files were uploaded

- image3.jpg
- image4.jpg
- image5.jpeg

Product Images: Choose Files No file chosen

Upload Pictures

Property Gallery

Figure 44: When more images are uploaded into the gallery

APPENDIX I: Landlord delete images from the property gallery Unit testing

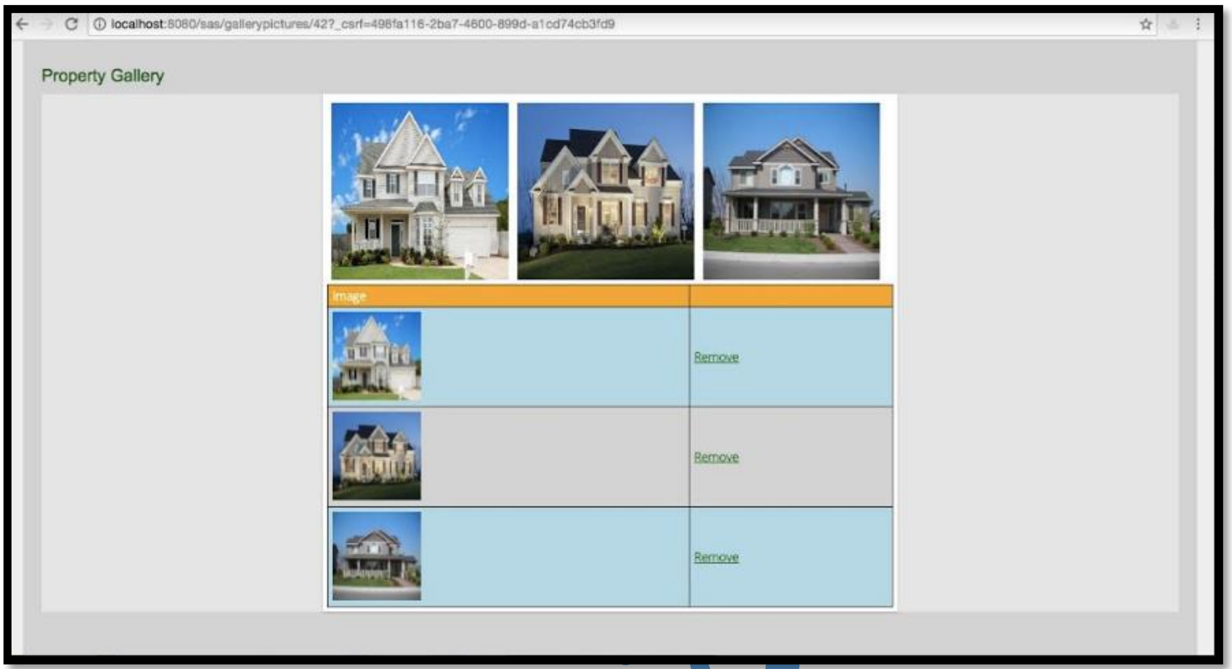

localhost:8080/sas/gallerypictures/42?_csrf=498fa116-2ba7-4600-899d-a1cd74cb3fd9
Property Gallery
Image
Remove
Remove
Remove

Figure 45: gallery before deleting

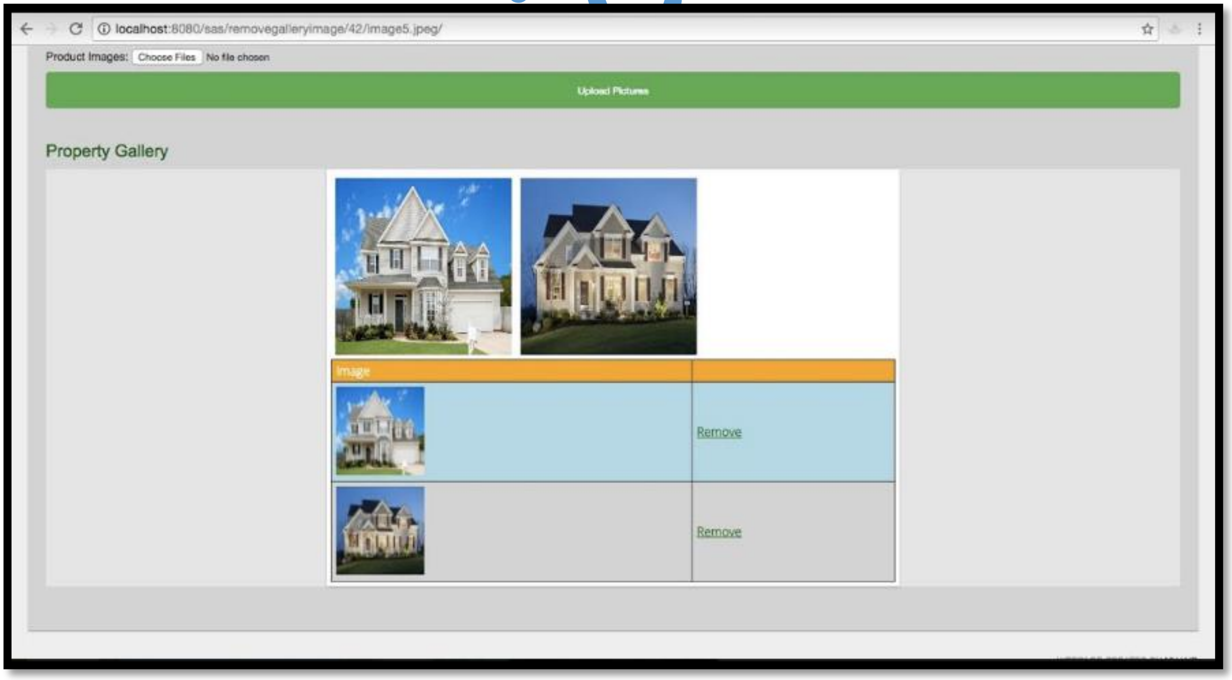
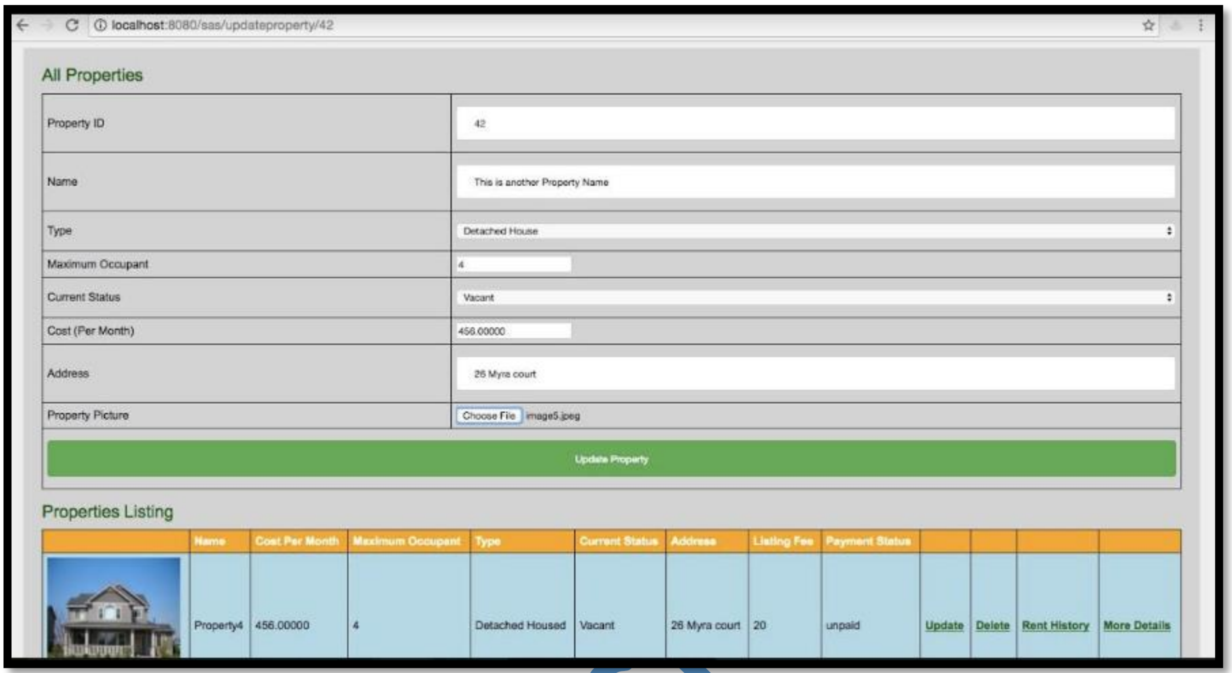


Figure 46: After deleting the image

APPENDIX J: Landlord update and delete the property listed Unit testing

When the user tries to update the property details



localhost:8080/sas/updateproperty/42

All Properties

Property ID	42
Name	This is another Property Name
Type	Detached House
Maximum Occupant	4
Current Status	Vacant
Cost (Per Month)	456.00000
Address	26 Myra court
Property Picture	Choose File image5.jpeg

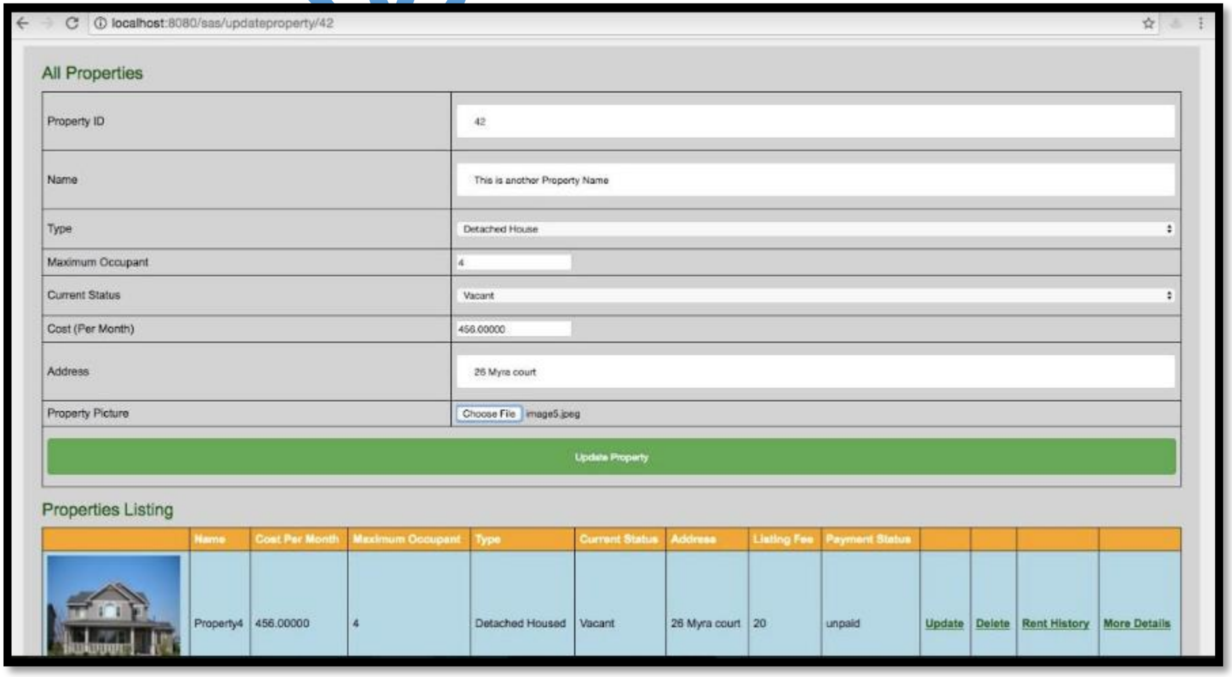
Update Property

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property4	456.00000	4	Detached Housed	Vacant	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 47: Update property information

When the user tries to delete the property without deleting the property history, the following error will occur



localhost:8080/sas/updateproperty/42

All Properties

Property ID	42
Name	This is another Property Name
Type	Detached House
Maximum Occupant	4
Current Status	Vacant
Cost (Per Month)	456.00000
Address	26 Myra court
Property Picture	Choose File image5.jpeg

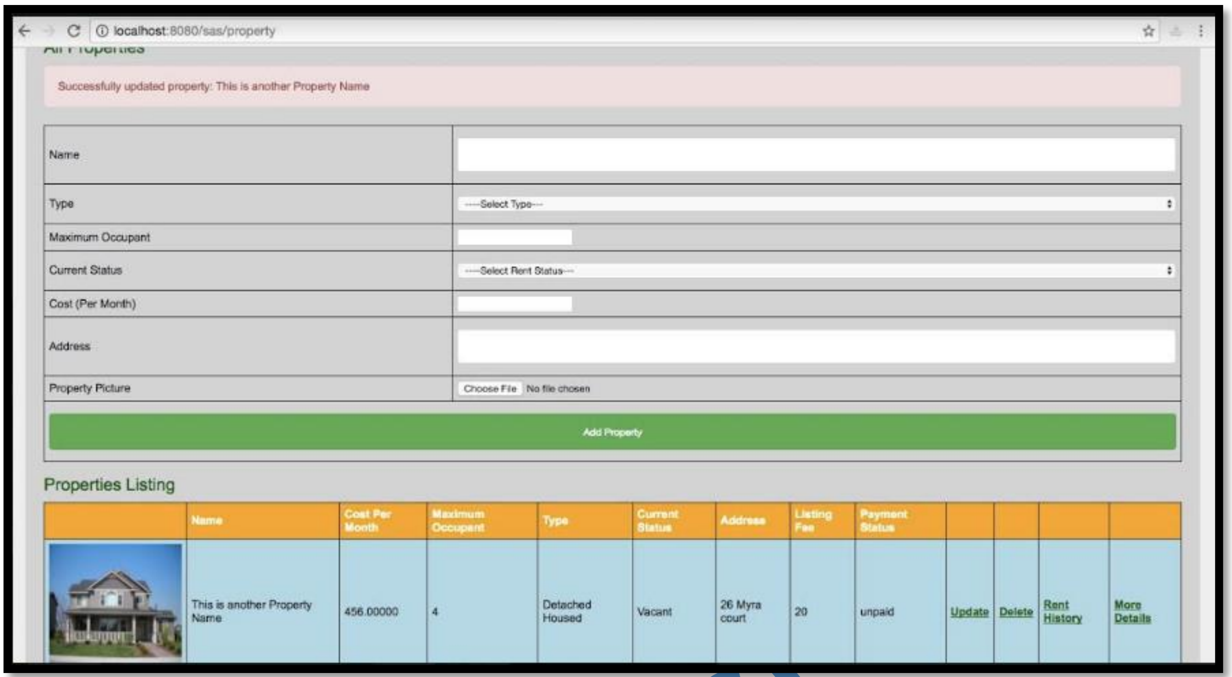
Update Property

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property4	456.00000	4	Detached Housed	Vacant	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 48: Deleting property without deleting the property history

When the property is updated



localhost:8080/sas/property

All Properties

Successfully updated property: This is another Property Name

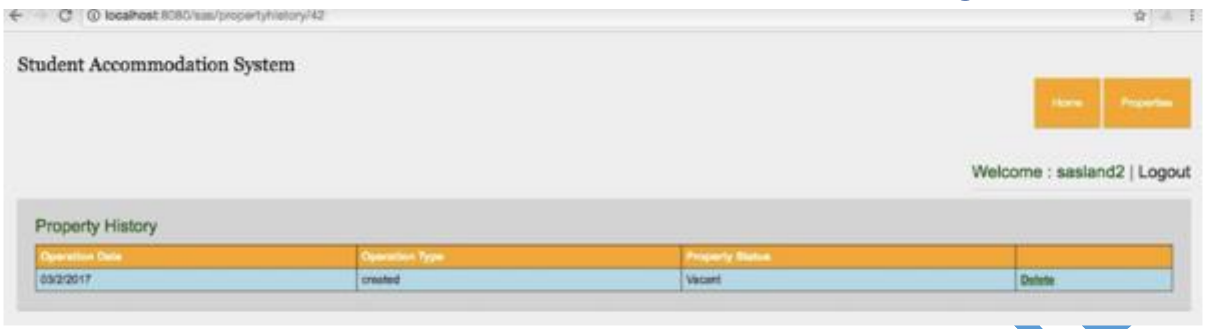
Name	
Type	----Select Type----
Maximum Occupant	
Current Status	----Select Rent Status----
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	This is another Property Name	456.00000	4	Detached Housed	Vacant	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 49: When the property is successfully updated

APPENDIX K: Landlord view and delete property history Unit testing


localhost:8080/sas/propertyhistory/42

Student Accommodation System

Home Properties

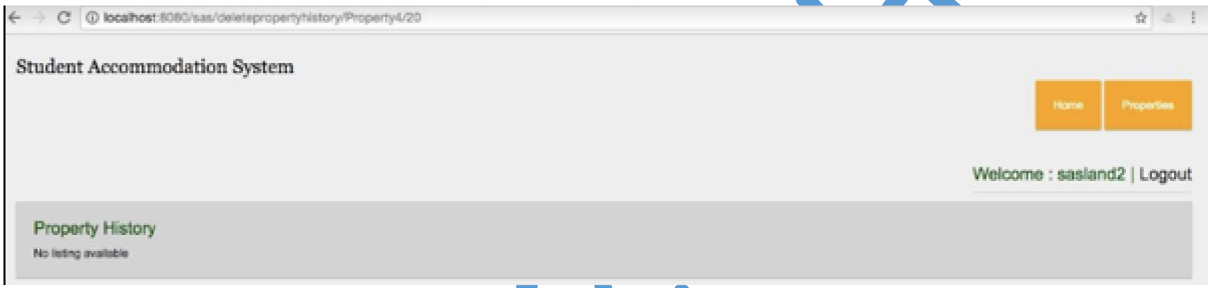
Welcome : sasland2 | Logout

Property History

Operation Date	Operation Type	Property Status	
03/2/2017	created	Vacant	Delete

Figure 50: View and delete property history

Successfully deleted property history


localhost:8080/sas/deletepropertyhistory/Property4/20

Student Accommodation System

Home Properties

Welcome : sasland2 | Logout

Property History

No listing available

Figure 51: Deleted property history

APPENDIX L: Administrator list a property Unit testing

Same as Appendix G

APPENDIX M: Administrator update the listed property information Unit testing

Same as Appendix J

APPENDIX N: Administrator should be able to delete images from the property gallery

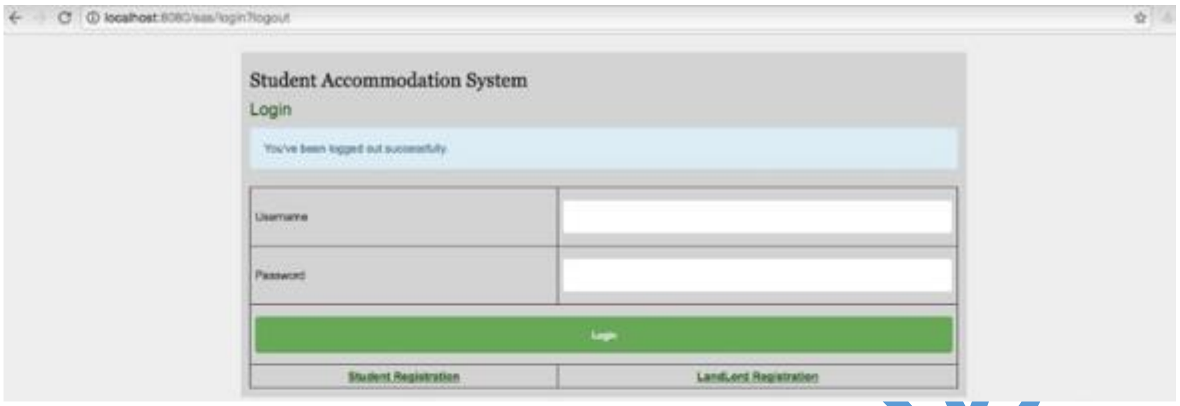
Unit testing

Same as Appendix I

APPENDIX O: Administrator should be able to upload more images to the property Unit testing

Same as Appendix H

APPENDIX P: Administrator logout Unit testing



localhost:8080/sas/login?logout

Student Accommodation System

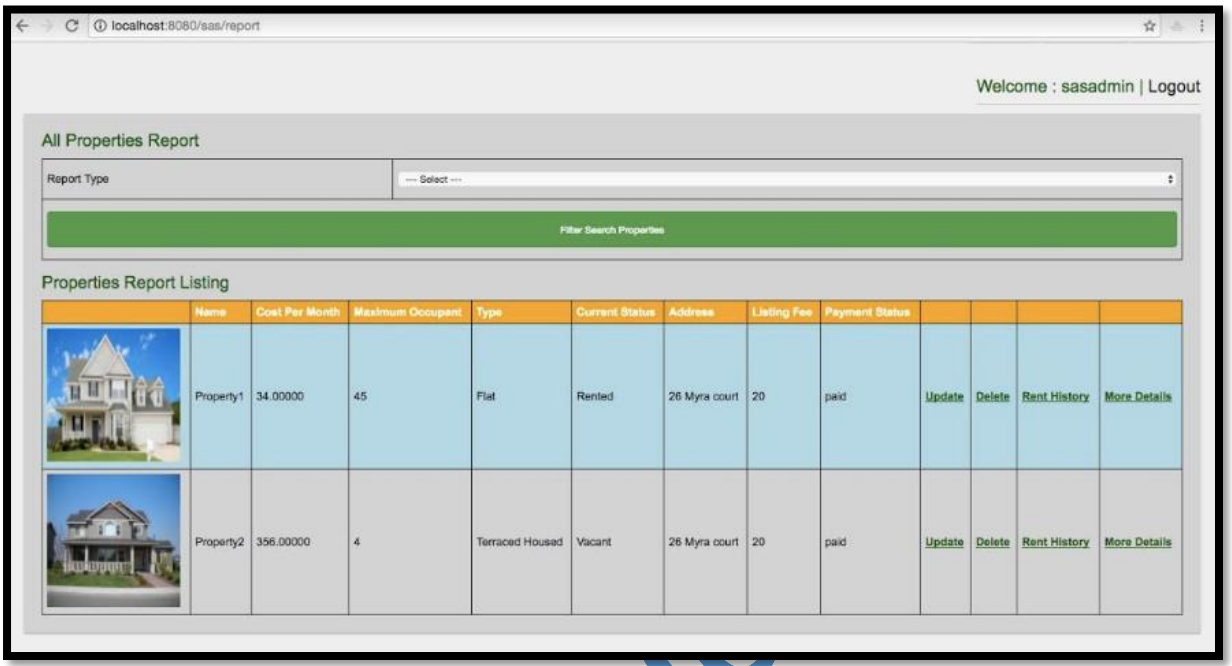
Login

You've been logged out successfully

Username	
Password	

Figure 52: Logout screen

APPENDIX Q: Administrator view property history Unit testing


localhost:8080/sas/report

Welcome : sasadmin | Logout

All Properties Report

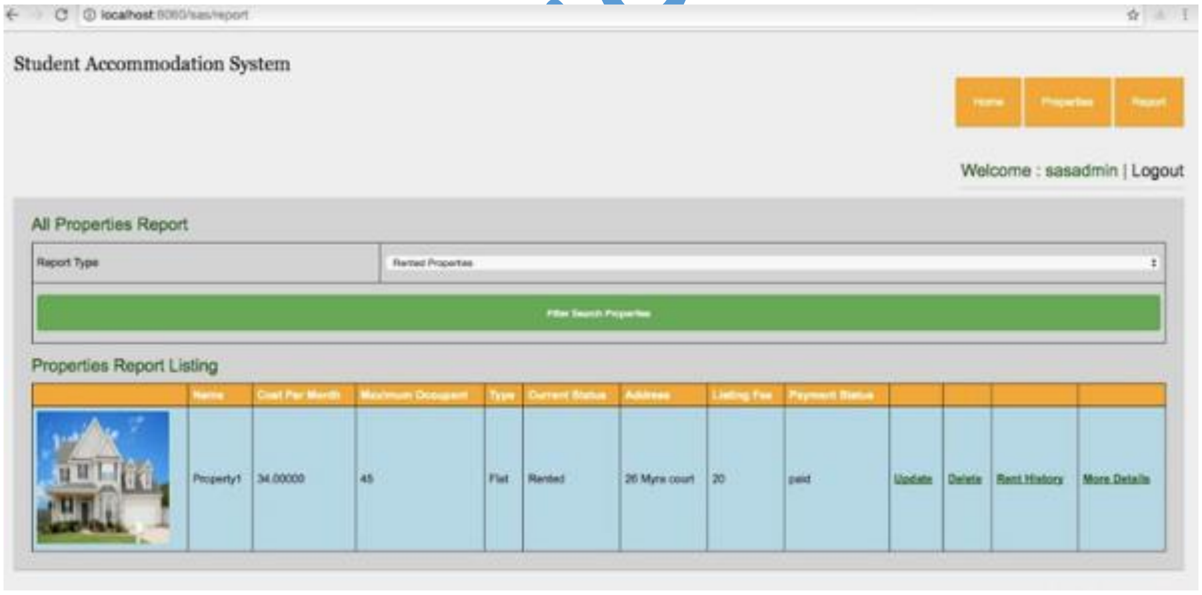
Report Type	--- Select ---

Filter Search Properties

Properties Report Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	paid	Update	Delete	Rent History	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	20	paid	Update	Delete	Rent History	More Details

Figure 53: properties report listing


localhost:8080/sas/report

Student Accommodation System

Welcome : sasadmin | Logout

All Properties Report

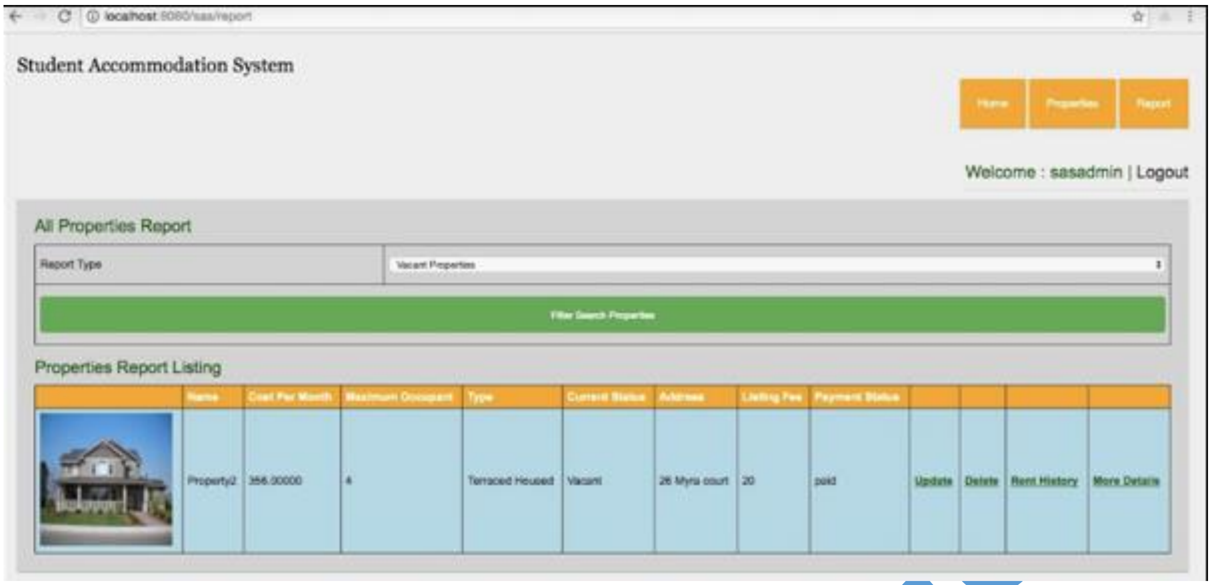
Report Type	Rented Properties

Filter Search Properties

Properties Report Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	paid	Update	Delete	Rent History	More Details

Figure 54: Rented properties report


localhost:8080/sas/report

Student Accommodation System

Welcome : sasadmin | Logout

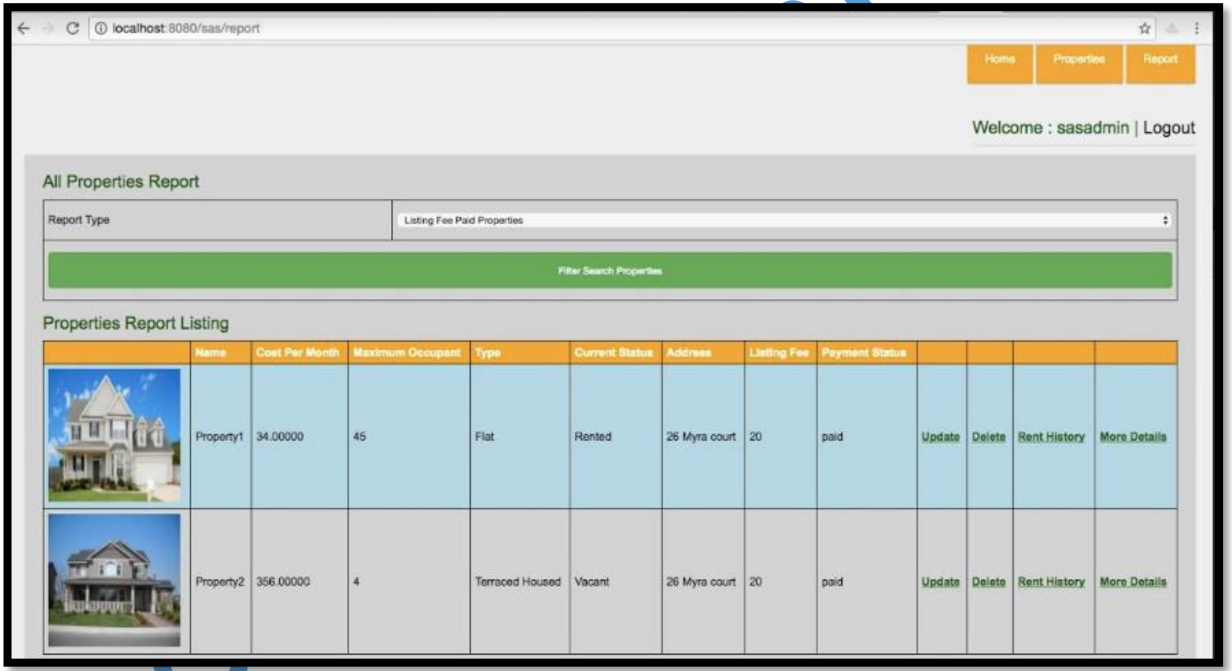
All Properties Report

Report Type	Vacant Properties

Properties Report Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	20	paid	Update	Delete	Rent History	More Details

Figure 55: Vacant Properties


localhost:8080/sas/report

Welcome : sasadmin | Logout

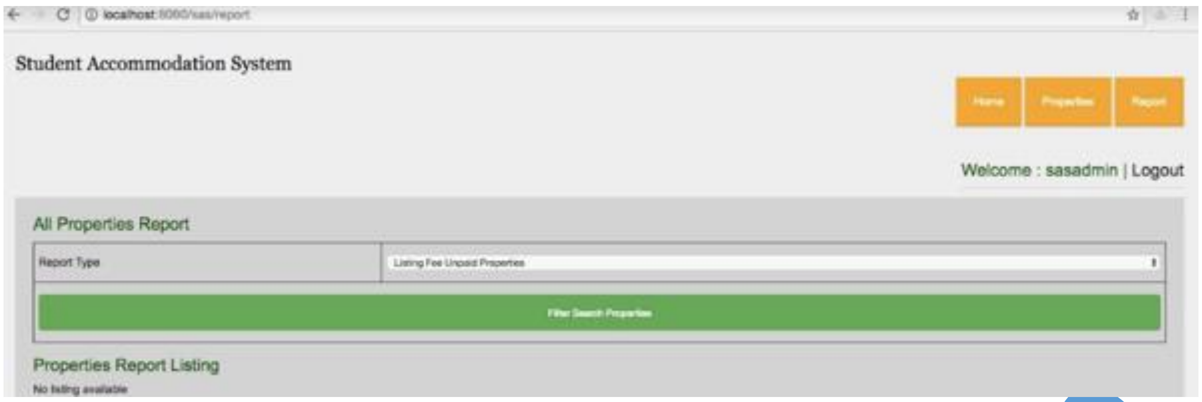
All Properties Report

Report Type	Listing Fee Paid Properties

Properties Report Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	paid	Update	Delete	Rent History	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	20	paid	Update	Delete	Rent History	More Details

Figure 56: Listing fee paid property Report


localhost:8080/sas/report
Student Accommodation System
Home
Properties
Report
Welcome : sasadmin | Logout
All Properties Report
Filter Search Properties
Properties Report Listing

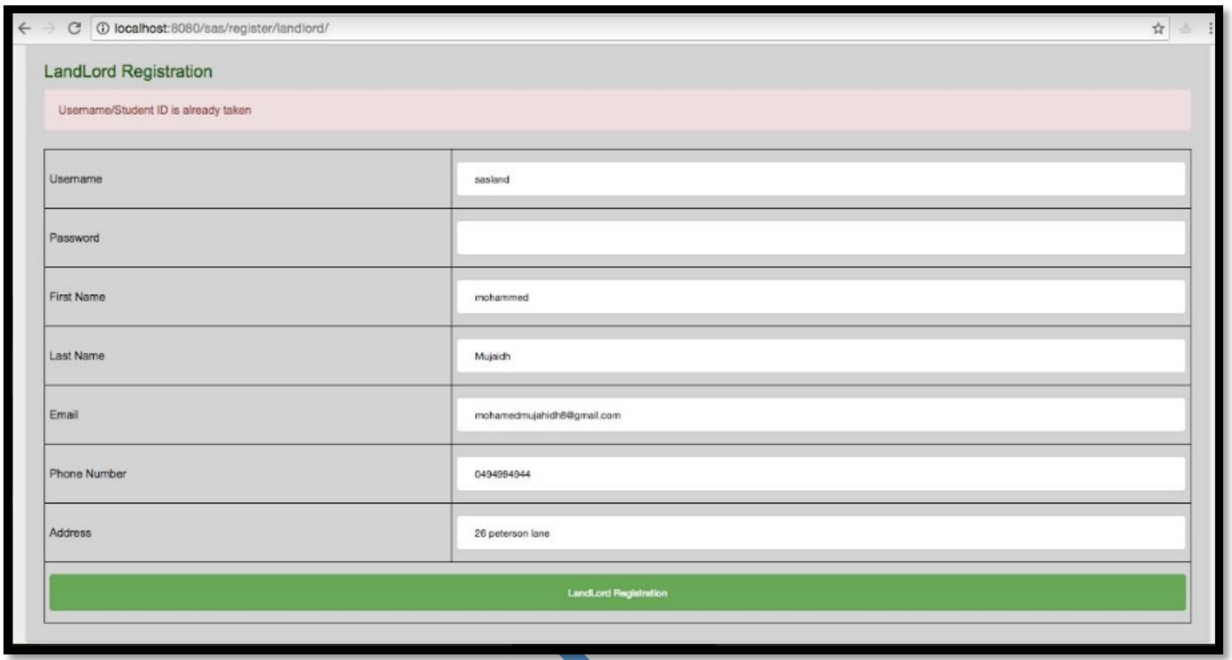
Report Type	Listing Fee Unpaid Properties

No listing available

Figure 57: Listing fee unpaid property

APPENDIX R: Landlord Registration Unit testing

When the landlord tries to give a username that already exists, the following error will be displayed



localhost:8080/sas/register/landlord/

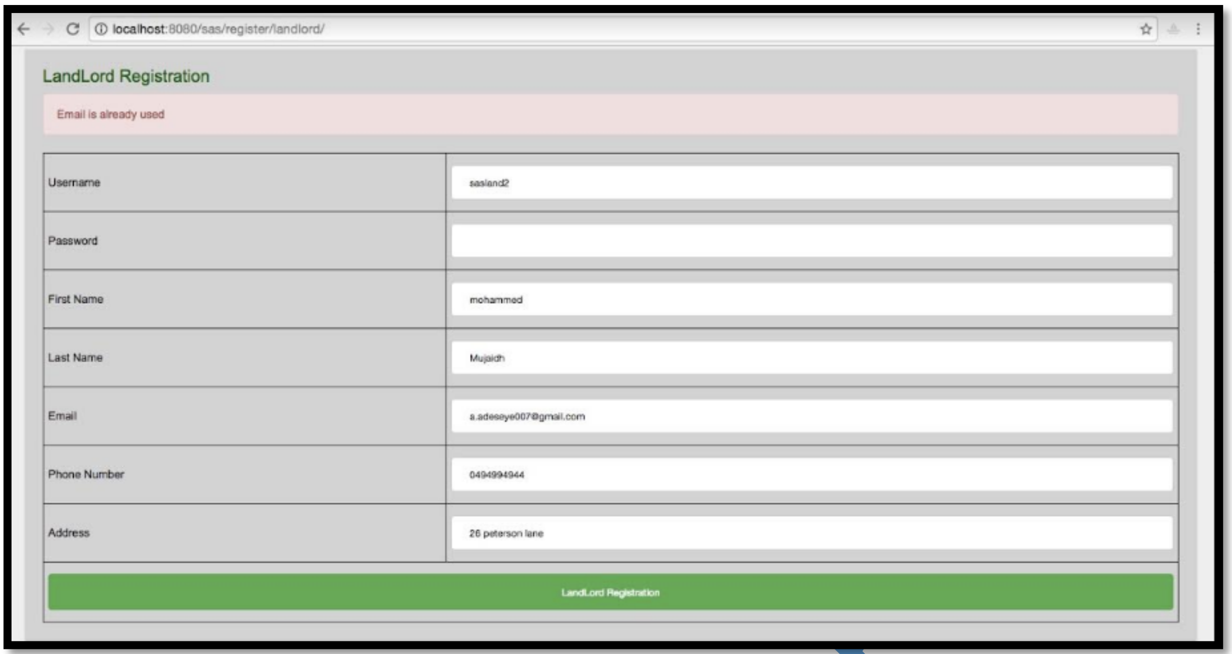
LandLord Registration

Username/Student ID is already taken

Username	sasland
Password	
First Name	mohammed
Last Name	Mujaidh
Email	mohamedmujahidh8@gmail.com
Phone Number	0494984944
Address	26 peterson lane

Figure 58: When username exists

When the landlord tries to enter an email address which is already used, the following error will be displayed



localhost:8080/sas/register/landlord/

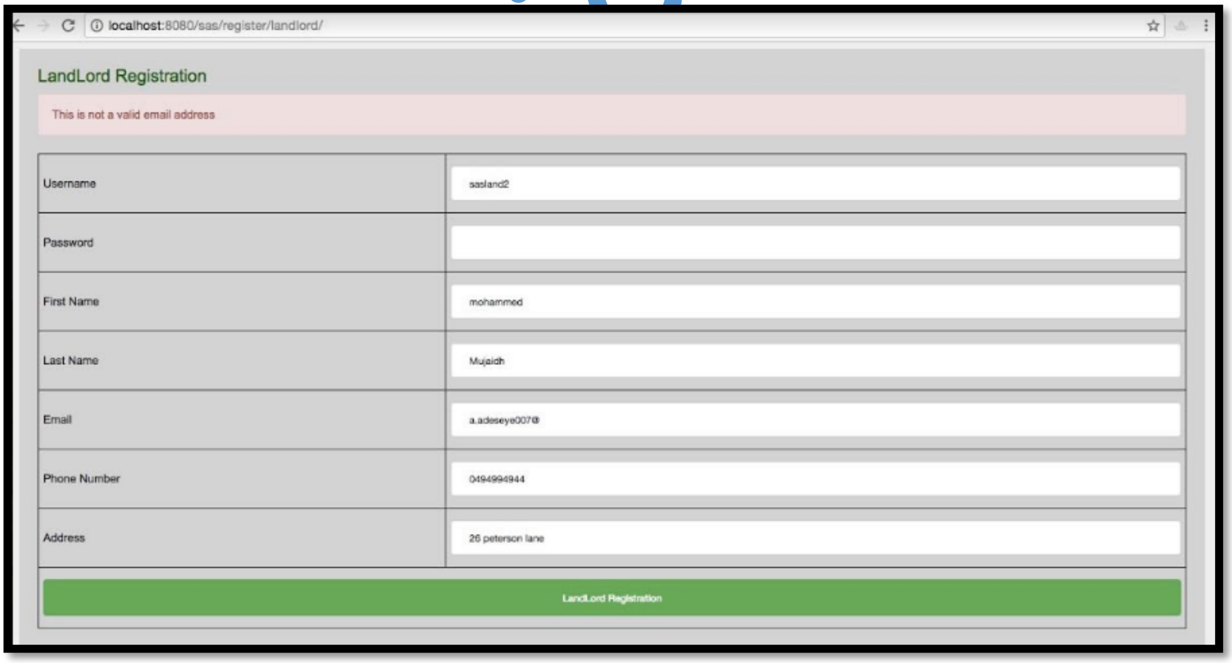
LandLord Registration

Email is already used

Username	sasland2
Password	
First Name	mohammed
Last Name	Mujaidh
Email	a.adeseye007@gmail.com
Phone Number	0494994944
Address	26 peterson lane

Figure 59: When email is already in use

When a user tries to insert a username that is not valid, the following error message will be displayed



localhost:8080/sas/register/landlord/

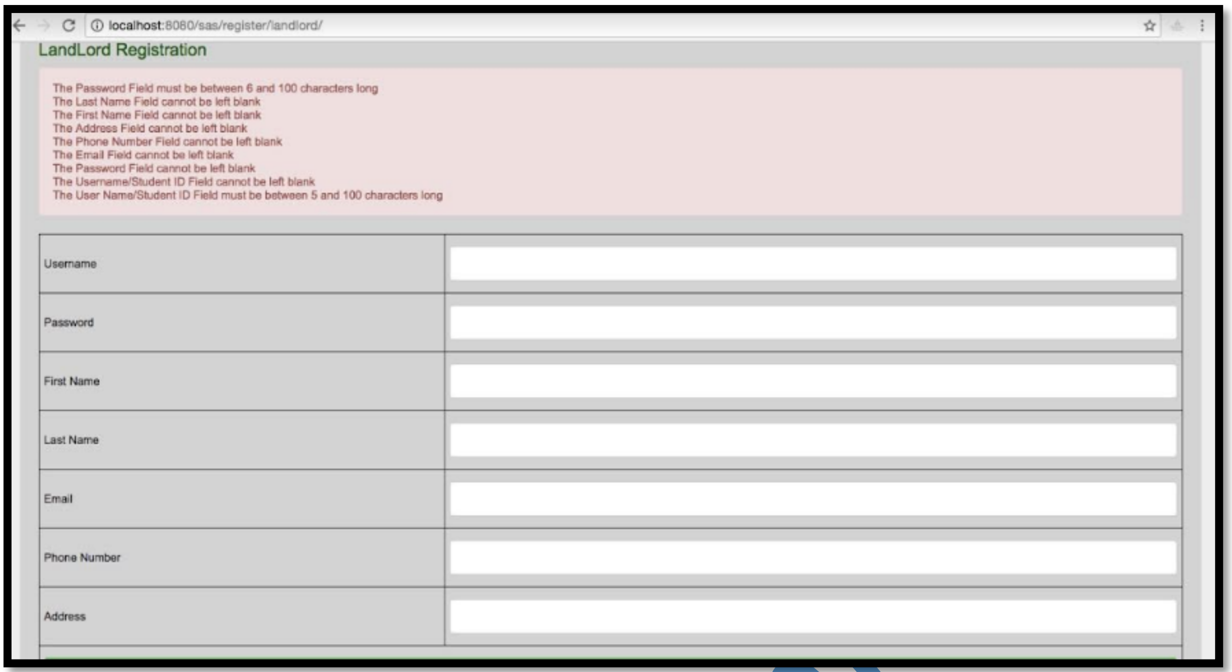
LandLord Registration

This is not a valid email address

Username	sasland2
Password	
First Name	mohammed
Last Name	Mujaidh
Email	a.adeseye007@
Phone Number	0494994944
Address	26 peterson lane

Figure 60: Invalid email address

When the registration fields are left empty, the following errors will be displayed.



localhost:8080/sas/register/landlord/

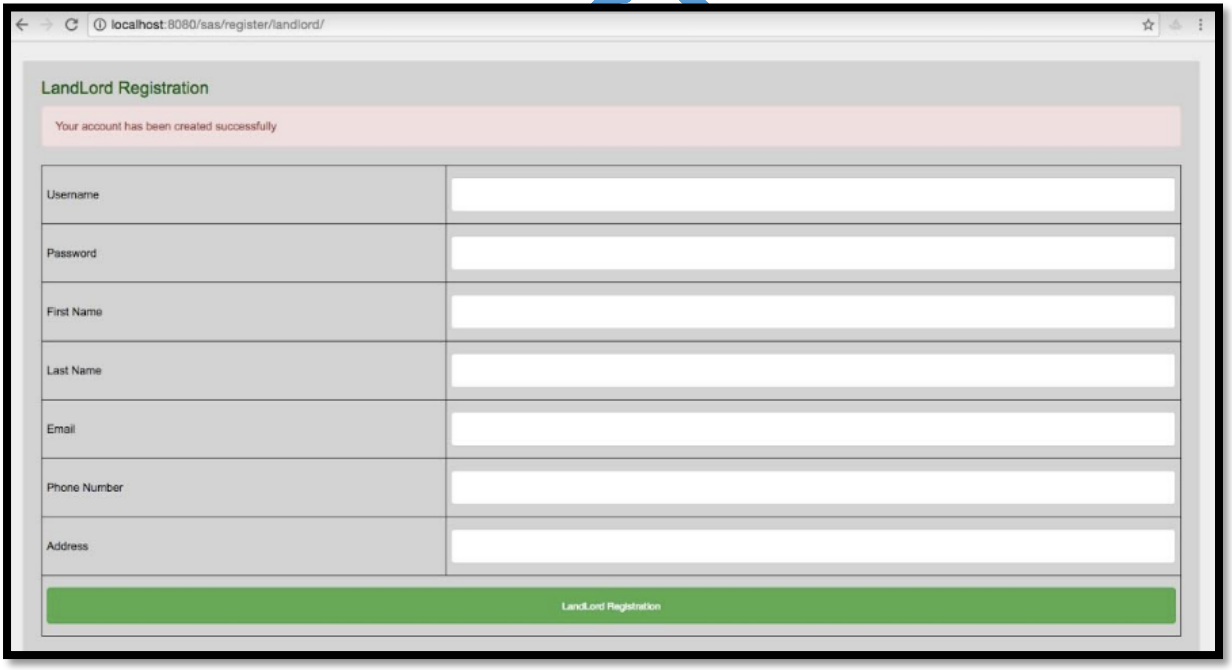
LandLord Registration

The Password Field must be between 6 and 100 characters long
The Last Name Field cannot be left blank
The First Name Field cannot be left blank
The Address Field cannot be left blank
The Phone Number Field cannot be left blank
The Email Field cannot be left blank
The Password Field cannot be left blank
The Username/Student ID Field cannot be left blank
The User Name/Student ID Field must be between 5 and 100 characters long

Username	
Password	
First Name	
Last Name	
Email	
Phone Number	
Address	

Figure 61: Fields are empty

When the successful information is provided



localhost:8080/sas/register/landlord/

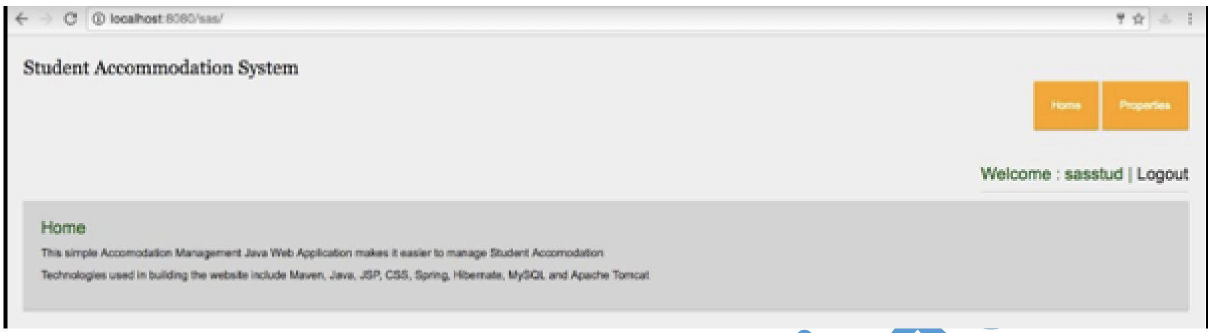
LandLord Registration

Your account has been created successfully

Username	
Password	
First Name	
Last Name	
Email	
Phone Number	
Address	

Figure 62: When the correct field information is provided

APPENDIX S: Student successful login taken to the appropriate home page Integration testing



Student Accommodation System

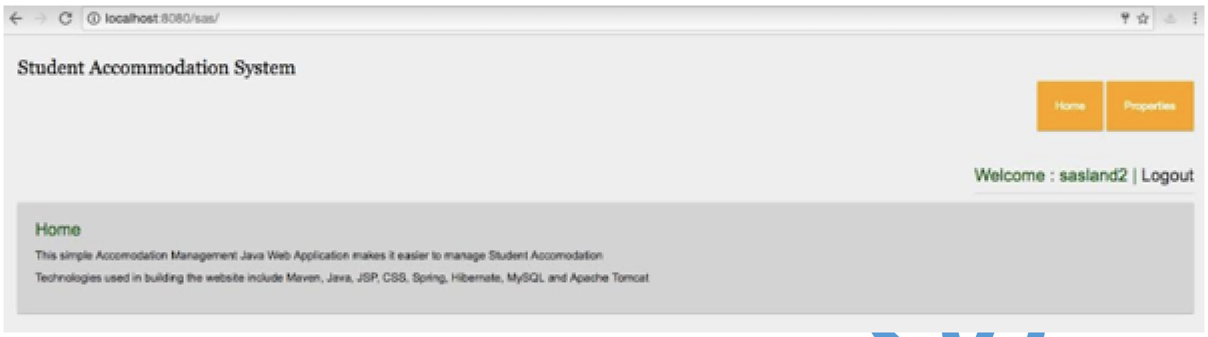
Welcome : sasstud | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX T: Successful Landlord login taken to the appropriate home page Integration testing


localhost:8080/sas/

Student Accommodation System

Home | Properties

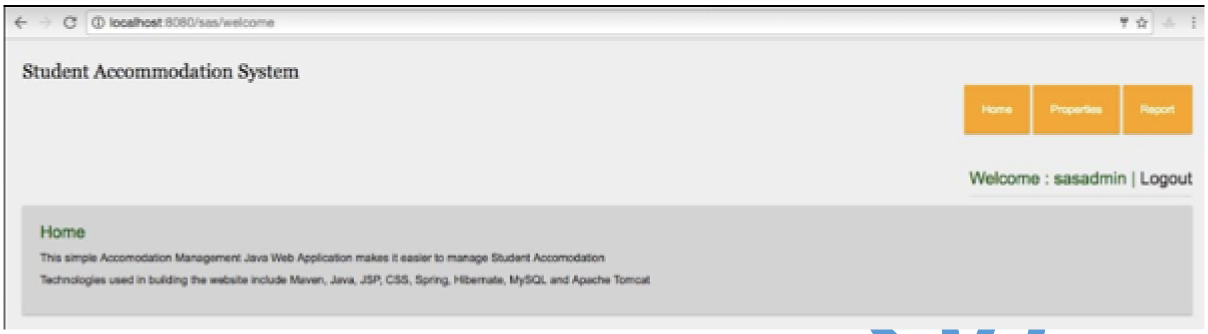
Welcome : sasland2 | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX U: Successful Administrator login taken to the appropriate home page Integration testing


localhost:8080/sas/welcome
Student Accommodation System
Home
Properties
Report
Welcome : sasadmin | Logout

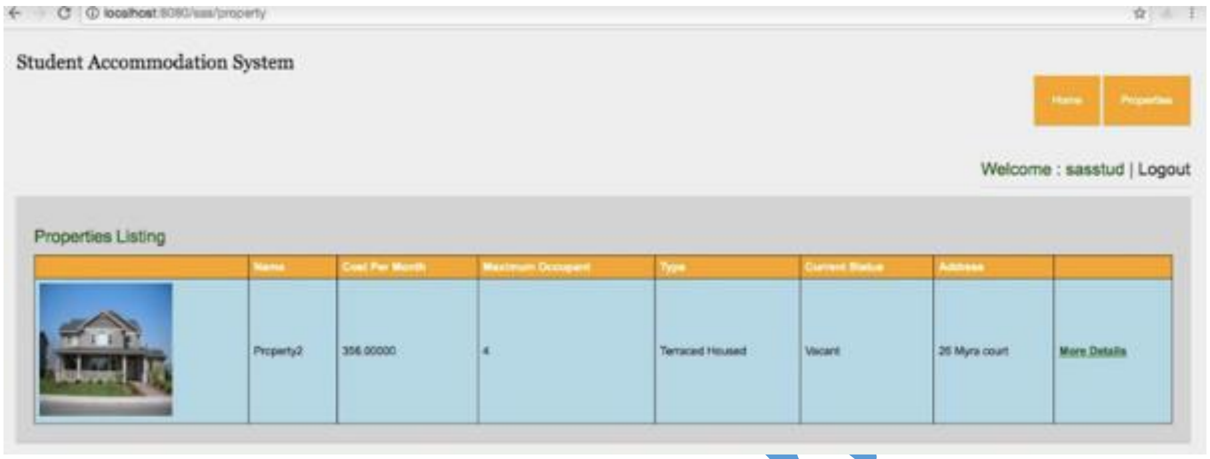
Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX V: The student Integration testing does not see landlord listed property which is unpaid

Before adding the property


localhost:8080/sas/property

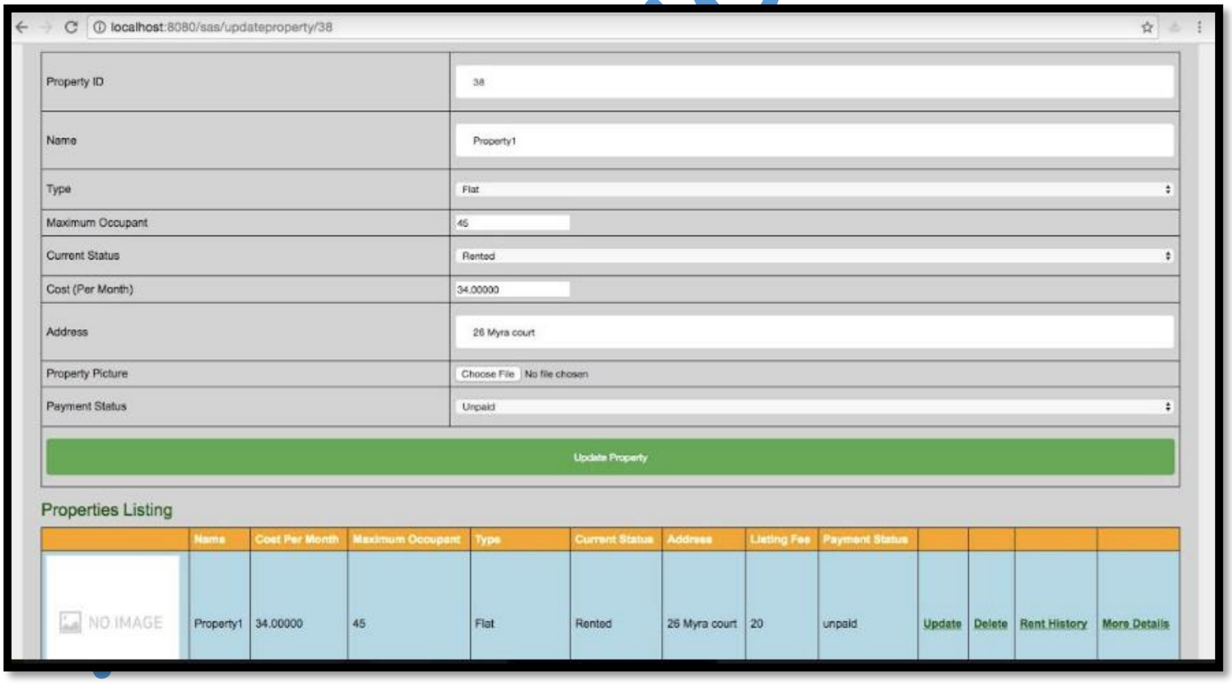
Student Accommodation System

Welcome : sasstud | Logout

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	More Details

Figure 63: Before adding the property

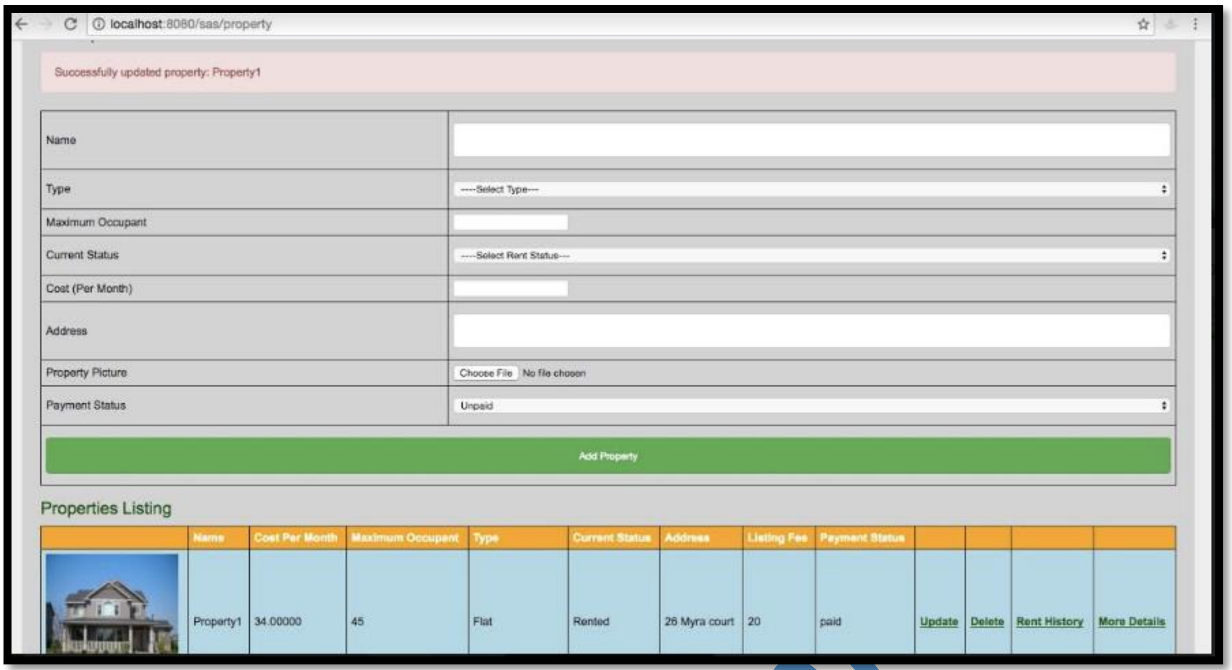

localhost:8080/sas/updateproperty/38

Property ID	38
Name	Property1
Type	Flat
Maximum Occupant	45
Current Status	Rented
Cost (Per Month)	34.00000
Address	26 Myra court
Property Picture	Choose File No file chosen
Payment Status	Unpaid

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
NO IMAGE	Property1	34.00000	45	Flat	Rented	26 Myra court	20	unpaid	Update	Delete	Rent History	More Details

Figure 64: Insert new property information


localhost:8080/sas/property

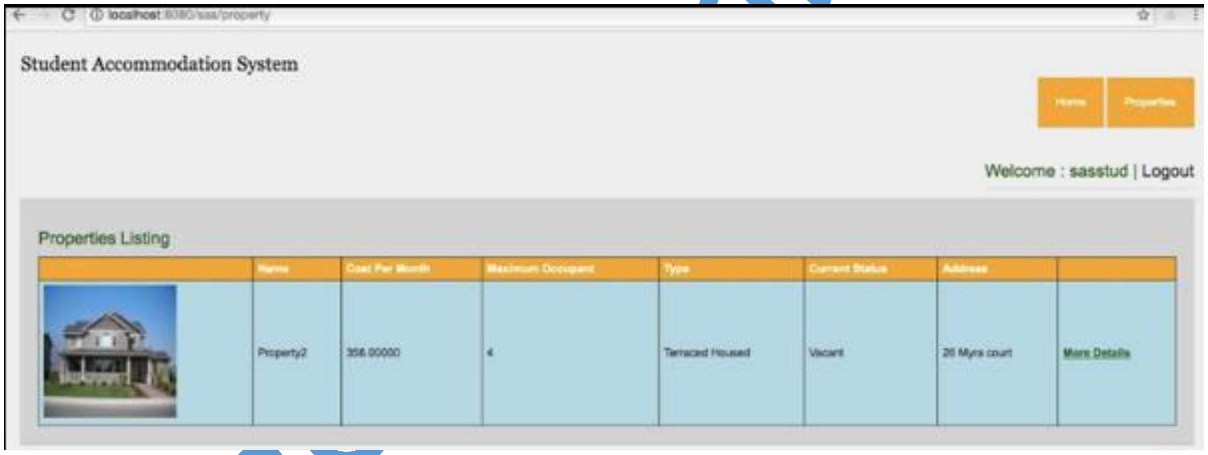
Successfully updated property: Property1

Name	
Type	---Select Type---
Maximum Occupant	
Current Status	---Select Rent Status---
Cost (Per Month)	
Address	
Property Picture	Choose File No file chosen
Payment Status	Unpaid

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	Listing Fee	Payment Status				
	Property1	34.00000	45	Flat	Rented	26 Myra court	20	paid	Update	Delete	Rent History	More Details

Figure 65: New property successfully added


localhost:8080/sas/property

Student Accommodation System

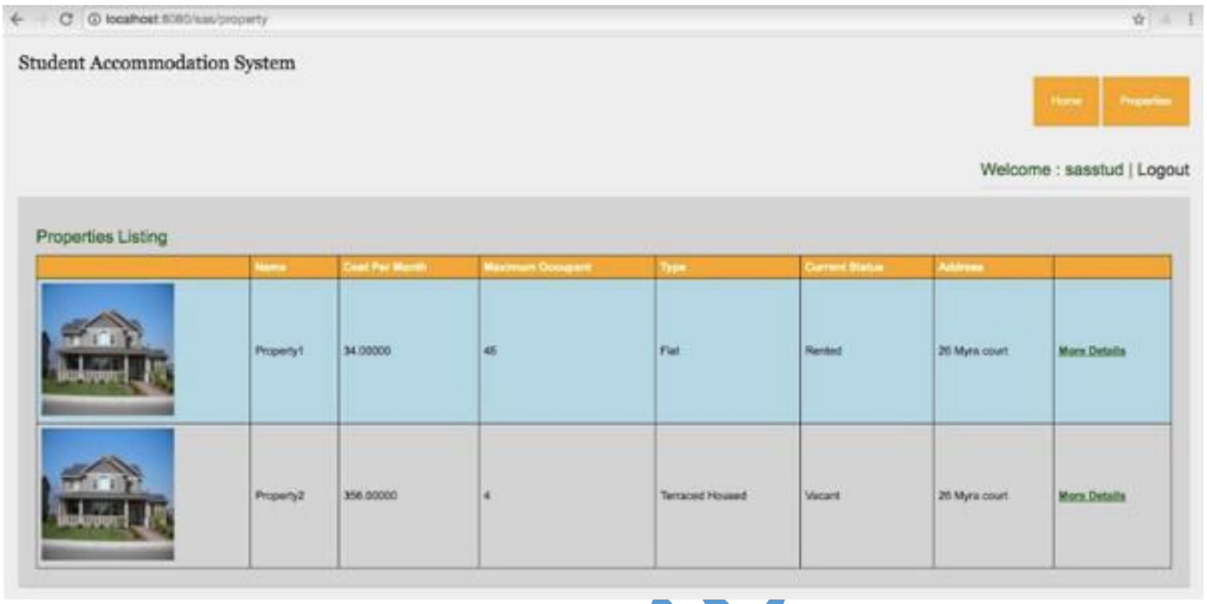
Welcome : sasstud | Logout

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	
	Property2	358.00000	4	Terraced Housed	Vacant	26 Myra court	More Details

Figure 66: When the property listing fee not paid, the student cannot see it

APPENDIX W: Landlord listed property which can be viewed updated by the administrator from unpaid to paid Integration testing


localhost:8080/sas/property

Student Accommodation System

Welcome : sasstud | Logout

Properties Listing

	Name	Cost Per Month	Maximum Occupant	Type	Current Status	Address	
	Property1	34.00000	45	Flat	Rented	26 Myra court	More Details
	Property2	356.00000	4	Terraced Housed	Vacant	26 Myra court	More Details

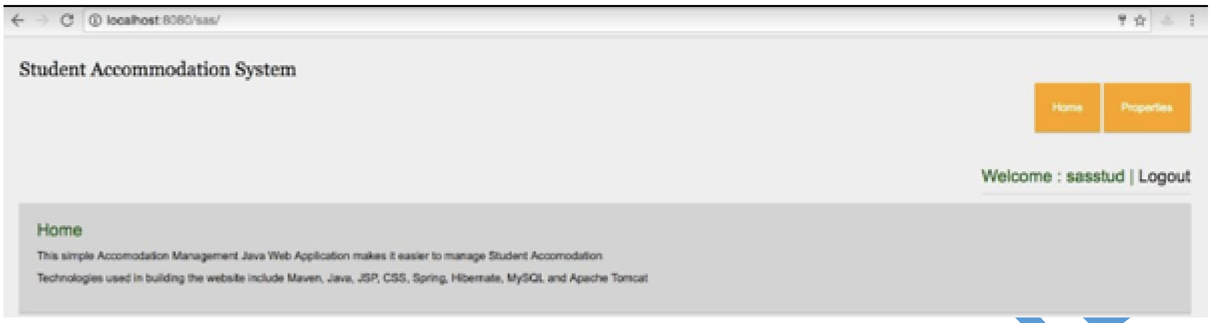
Figure 67: Can view when the listing fee is paid

APPENDIX X: Administrator list a property, student can view it

Integration testing

Same as Appendix V and W

APPENDIX Y: Student authentication was accurate



Student Accommodation System

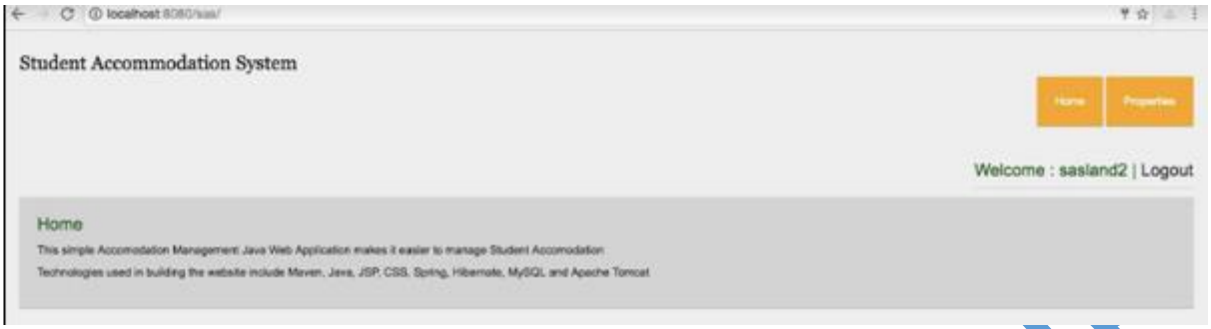
Welcome : sasstud | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX Z: Landlord authentication was accurate



Student Accommodation System

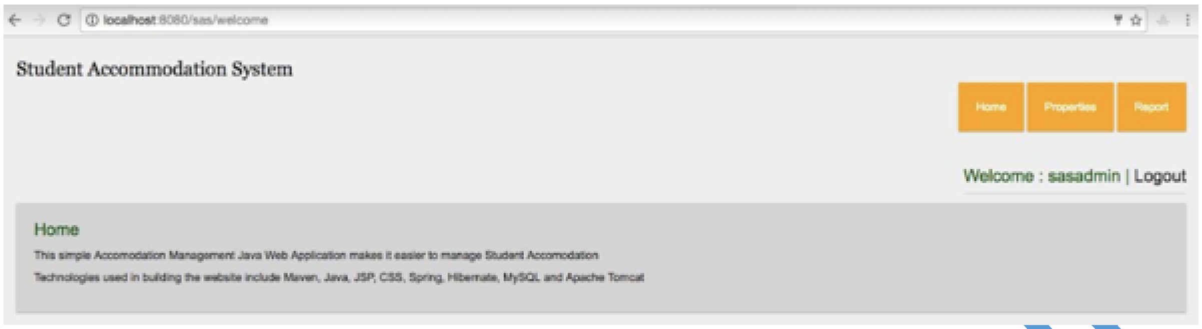
Welcome : sasland2 | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX AA: Administrator authentication was accurate



Student Accommodation System

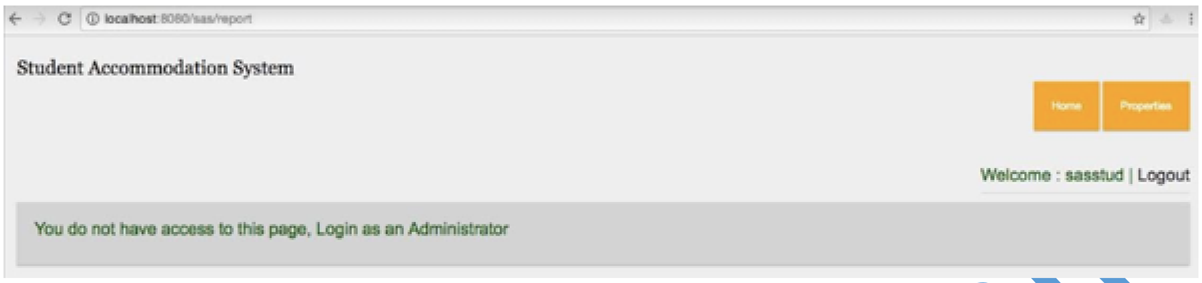
Welcome : sasadmin | Logout

Home

This simple Accomodation Management Java Web Application makes it easier to manage Student Accomodation

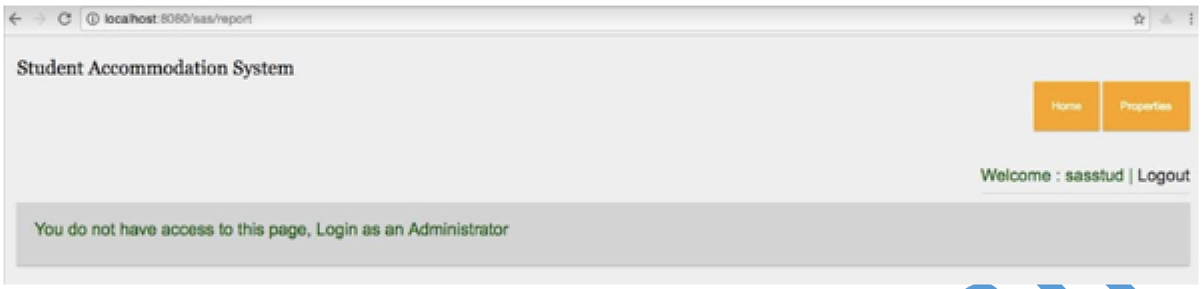
Technologies used in building the website include Maven, Java, JSP, CSS, Spring, Hibernate, MySQL and Apache Tomcat

APPENDIX AB: When a property is listed, and the listing fee is not paid, the student cannot view the property details


localhost:8080/sas/report
Student Accommodation System
Home
Properties
Welcome : sasstud | Logout

You do not have access to this page, Login as an Administrator

APPENDIX AC: When a student tries to access the administrator, report using the URL – it should not allow



Student Accommodation System

Welcome : sasstud | Logout

You do not have access to this page, Login as an Administrator