School of Electronic Engineering and Computer Science

Final Year Undergraduate Project 2017/18

FINAL REPORT

PROJECT OF STUDY: COMPUTER SCIENCE

PROJECT TITLE: HOUSING ACCOMMODATION PROGRAM

Student Name: Musab Adam

SUPERVISOR: DR REBECCA LYNNE STEWART

DATE: 20/04/2018

Abstract

The aim of this project is to develop a working software system that allows housing accommodation companies such as “The Student Housing Company” and “Unipol” to be able to manage various aspects of housing accommodation such as managing tenant’s accommodation, managing properties and their rents and expenses and finally dealing with day to day maintenance issues. The project is divided into two subsystems where one system is a desktop java application for the housing companies to use and the other system is a web application that allows tenants of the housing accommodation to log in and report issues and the builders that work for the housing accommodation to be able to log in and see what jobs have been assigned to them. The web application is using the Django framework and is hosted on an OpenShift server.

Table of Contents

Introduction

My project is aimed at housing accommodation companies that lease properties and then rents it out to tenants. These companies must pay monthly rents to the landlords that they have leased with the possibility of rents being deducted (rent being reduced for the month) to cover maintenance costs. There will also be times where properties must be returned to its landlord, therefore companies need to move tenants to another property that they currently lease. Companies will also manage maintenance issues where an issue at a property is reported by a staff member or a tenant themselves and is then dealt with, using in-house builders or external maintenance companies.

Currently, I work with a housing accommodation company called Nationwide Accommodation Services(NACCS). They deal with renting many properties and work with the Home Office and London councils to house tenants who either can't afford accommodation or/and migrants that have recently moved into the country. NACCS currently do not have a dedicated system that deals with all the tasks that a housing company deals with. Instead, they have multiple spreadsheets where each employee is responsible for a specific spreadsheet. For instance, the accountant would be responsible for the spreadsheets that tracks expenses and rent payments and the housing manager would be responsible for the spreadsheet that manages bed spaces in properties.

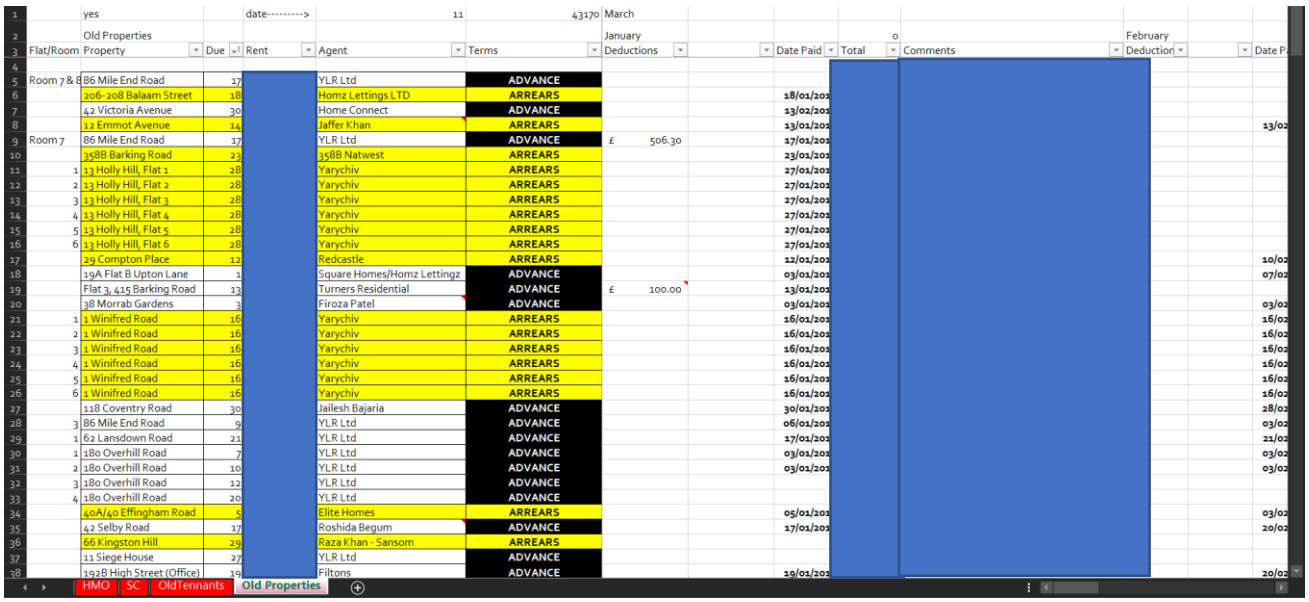


Figure 1 showing the landlord rent spreadsheet that is used by NACCS

The issue with spreadsheets is that spreadsheets cannot read data with other spreadsheets seamlessly. The only way to get past this is to use a formula that points a "cell" to another "cell" in

another spreadsheet which is required to be running as well, which leads to another problem which is no centralised data. This means that data that is needed, is fragmented into different spreadsheets, for example, if the accounts team wanted to know the profit and loss margins of the month they would have to manually gather all the data from the expense spreadsheet and the rents spreadsheet thus making it an inconvenience for the accounts team. Another problem with spreadsheets is the reliability and robustness of the files. Spreadsheets tend to corrupt over time and once they become corrupted, there are very limited recovery options for them. Also, every year, the spreadsheets are refreshed for the new year, this means that certain data such as expenses and rent payments would need to be reset for the new year. So over time, there would be many spreadsheets stored in the company's server meaning increase disk usage.

Another issue that occurs at NACCS is that tenants must ring the company's support line to report any issues in the property they live at. When the issue is reported and recorded, these tenants have no way of knowing the status of the issue and how it's being dealt with, so it simply ends up with tenants constantly calling back to acquire more information and eventually lose patience with the company and the company having to deal with more phone calls in a day. Similarly, when an issue is dealt with by a builder and the builder finds a new issue or haven't completely resolved the current issue, often they would forget to tell the area managers (staff members who do regular safety inspection of properties in a certain area) when they get back to the office of these findings causing further issues in the future.

Existing Systems

Occam System

I did some research into programs that are already on the market and found a student accommodation program called “Occam” that manages student’s accommodation. Based on the brochure of the application (since there was no demo to use), it had the ability to track tenants (students) and their payments, for instance, the program would track when the students stay at the accommodation would end and what their balance would be. However, there was no feature related to maintenance and expenses therefore if this was used in the company I work with, they would have to resort to spreadsheets to compensate for the missing feature and find a way of exporting valuable data from the student accommodation program thus making it more of an inconvenience for employers looking to gain something out of the data. (Occam, n.d.)

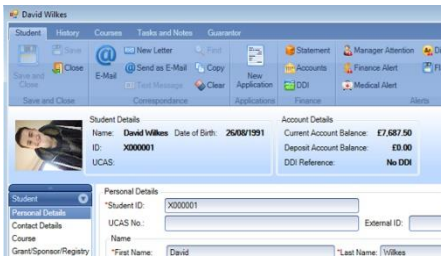


Figure 5 showing an interface of Occam system

Localpad System

Another program I found online was called “Localpad”. Localpad is an online solution that deals with social housing accommodation rather than student accommodation like Occam. From reading their solutions and benefits, there were features that manage tenants and tracks how many days tenants were living at the property. Another feature that I found very useful was that it allowed users to see what properties are vacant for incoming new tenants. However, a feature still missing was property maintenance, therefore, has the same issue as the Occam software. Also, since it’s a web application, there is a high chance that the software will be sluggish and have compatibility issues with different browsers. (solution, n.d.)

My project addresses these issues. It has a storage system in a form of a database that allows data to be centralised into one place and be accessible by different programs at the same time by using MySQL DBMS on a dedicated server. Also, since MySQL is in use, the chance of data being corrupted is very low and if there was some sort of corruption in the DBMS, the server can restore its state to a time thus allowing data to be recovered. My program also has the standard features that the two currently existing systems have which are tenant management and property

management but is not an online application, rather it is a java application thus keeping performance intact. Finally, with property maintenance issues, users using the program can track issues using the java application and dispatch builders whilst tenants and staff members can use a web application to report issues thus allowing tenants currently living at the accommodation to report their issues without having access to the java application.

Background Research

Before starting the project, I decided to research into the different programming languages and systems that I would be using during my project. Some of these tools, I already have a good amount of experience with and there are some that I have very little experience with. The research would help me with understanding the functions that would be available to me and how I would integrate the web application with the desktop application

Java 8 Platform

Java is a type of object-oriented programming language that was developed by Sun Microsystems. It is used to create applications on desktop computers, mobile devices such as Android smartphones and smartwatches and gaming consoles such as the PlayStation 3. Developed in 1996, the programming language makes use of a compiler which takes the codes and converts it into a bytecode and a virtual machine called Java virtual machine (JVM) that converts the bytecode into a code format that is readable to the host systems processor.

The use of Java also ensures the stability of the host system. This is because Java does not make any references to real memory address thus ensuring that when compiling and running the java code, no data would be corrupted and there would be no damage to the integrity of the system since the code would run in a virtual machine.

Java also has a popular feature that simplifies programming with the use of class libraries. These libraries are pre-coded classes that have a variety of functions when imported into a class and are reusable. The latest stable version is Java 8.0.161.12 which is the most stable and most secure Java, however, there is Java v9 which is still in beta and consist of new functions and code syntax that I may not be familiar with. (Perry, 2010)

Python

Python is another type of high-level object-oriented programming language that was developed by Guido Van Rossum in 1991. It is a widely used programming language that is used in many IoT devices such as raspberry pi and is also a popular programming language in web-based applications as a backend application system.

Like Java, python makes use of simplifying programming by utilising class libraries. These class libraries when imported would provide extra functions to the program that have been pre-coded by other developers and packaged into a class. It also uses a simpler syntax system where variables do not need to be initiated and their type does not need to be set compared to Java where variables must be initiated along with what type of variable it is i.e. integer, string. Many developers would say that python syntax comes very close to pseudocode which is the closest to human readable code as possible. (Developers.Google, 2016)

Django Framework

Django is a very popular web framework that is used by wide range of companies around the world. It is built for rapid development which means that developers can quickly make web applications with python and test web application by compiling python code then automatically setting up a local server and lastly deploying the application to the local server within seconds which suited my needs as my desktop application took majority of my project time thus less development time for my web application.

Django framework is a very powerful tool and helps developers by relieving the more complex work of the development when it comes to building web applications. For instance, when it comes to dealing with databases, Django will analyse the database and convert the database into objects that can be called upon by the controller. Django also maintains the integrity of the web application and aids developers in keeping the application secured by negating SQL injections (injecting unauthorised SQL code into the browser), cross-site scripting and cross-site request forgery by making use of automatic escaping in HTML coding. These security features will be a vital asset to my project as the data I handle will be very sensitive to tenants and the company itself since all companies that deal with sensitive data must comply with the Data Protection Act. (developer.mozilla, 2018)

JavaScript

JavaScript is a high-level scripting language that allows the manipulation of behaviour, visual looks of HTML pages and CSS files that linked to the HTML pages. It is regarded as a core part of web-based applications and gives websites the ability to be interactive for instance playing media on a website or inputting data into a website. JavaScript code is running in client side which means that all JavaScript code is running on the client's device and not on the server side, therefore, JavaScript code is executed instantly, creating a smoother web experience for the user.

JavaScript doesn't require additional software/plugins installed on the client's device therefore nearly all browsers natively support JavaScript if they comply with the latest JavaScript version. The latest version of JavaScript is EMCA 2017 which consist of all the latest functions and latest compatibility with all versions of HTML including HTML5. (Mozilla, 2016)

MySQL

MySQL is a type of relational database management system that is commonly used by developers and large companies. It uses the usual SQL syntax that most database developers are familiar with and is ACID compliant which means data that is going from and to the database has guaranteed validity, therefore, it would be near impossible for a database transaction to go wrong in terms of data. (Stephens, 2010)

MySQL also can work concurrently which means that the database can manage multiple queries that are incoming at the same time and return a result back to each of the user without fail. This would ideal for my project as tenants would be reading and writing into specific parts of the database whilst the users from the housing company would simultaneously be reading and writing into other parts of the same database.

Amazon Web Service(AWS)

AWS is a web hosting service that allows developers and companies to host various applications on their Amazon's server. These applications can range from hosting databases to hosting web applications such as Django applications. To join AWS, the must developer must choose a subscription tier that is suitable for them. For beginners, one of the tiers is the free tier which allows the developer to have any type of relational database management system up and running

for 750 hours per month with full support by Amazon. The tier also allocated 20gb of storage that is shared across all the applications deployed by the developer on the AWS server which will be more than enough for the database that will be built.

The support consists of the ability to periodically run database backups whilst the DBMS is running and in the event of a catastrophic error, the service allows the DBMS to revert to a specific backup. The service also developers to configure the security of the application by determining who can connect and use the hosted application and who can read or write into the application by filtering IP subnet which means that developers can choose to allow a certain group of connections to have access to the hosted application. (Amazon, n.d.)

OpenShift

OpenShift is another type of web hosting service that allows developers and companies to host various applications on their servers. Like AWS, OpenShift can host websites and databases on their servers. Currently, Queen Mary University of London has a custom server that is available to EECS students for free but can only host one web application that can be partnered with a MySQL database management system. When an application is deployed to OpenShift, the server enables an SSH tunnel that allows the developer to access the partition that consists of the application to modify and update files for the application as well as the database that is partnered with the application.

GitHub

GitHub is a development platform that allows developers to collaborate in building applications and enables software version control. GitHub gives groups and individuals to upload their project containing raw code and files onto the cloud thus acting as a cloud storage. Any changes to code and files are done locally on the developer's device and once the developer and their team are satisfied with the updated code, they can commit and push the code onto the cloud storage. GitHub can also act as a backup to the project where if a circumstance arose where the code became corrupted or there was a fatal bug that could not be resolved, the developer can simply revert to the last commit that was made to the project thus undoing any changes that were made from the last commit to now. (GitHub, 2018)

Requirements

Functional requirements

- Login Screen
    - Staff members using the system should be able to log in to the system using the correct combination of username and password.
    - A correct combination of username and password will take the user to the main menu screen
    - Incorrect log in will deny access to the home screen and will notify the user of the combination being wrong
- Home screen menu
    - Be able to access different features of the program
    - Be able to come back to the home screen to navigate to another part of the program
- Manage service users/tenants (SUs)
    - Be able to select tenants and marked them as old tenants that are no longer managed by the housing accommodation
    - Generate a username and password for the tenant for access to the maintenance website
    - Be able to delete a tenant that was created by mistake
    - Add a new tenant into the system and at the same time, assign them to the property they accommodate in
    - Move a tenant from one property to another
    - Be able to calculate the number of days per month they have stayed at the property
- Manage properties
    - Be able to add a new property as well as storing vital information linked to the property (rent payments, landlord, contract start and end date)
    - Be able to close off a property that is no longer being rented and close off any accounts in the utility section
    - Make changes to a property
    - Calculate rent that needs to be paid by taking in possible rent deduction and reducing the rent by the rent deduction

    - Give a notification to the user to alert them that the property they are viewing has rent that is still due
    - Record the rent paid for the property as well as recording vital information such as rent paid date and any deduction that was applied to the rent
- Manage expenses paid towards a property
    - Remove expenses that were made by mistake by the user
    - Record an expense as well as recording the quantity, price, category of the expense and assign the expense to the property that the expense was used for
    - Record utilities paid as well as keeping track of account numbers for the utilities i.e. gas, electric, water and council tax accounts
    - Import bank statements from online banking and automatically place each transaction into its correct category and correct property
- Manage company's finance
    - Track profit/loss per property by taking all the expenses recorded and the tenants living at the property as well as how long they have been living at the property
    - Track amount of expense paid per property
    - Track amount of income gained from each property
- Manage house maintenance issues
    - Allow tenants to log into a website using email and password
    - Allow tenants to report a problem through an online system
    - Add a new maintenance issue from the desktop application
    - Track an issue from the desktop application
    - Give tenants the ability to track the issue that they have reported
    - Add a new builder to the system and automatically generate a username and password
    - Assign an engineer/builder that will attend to the reported issue
    - Give Builders the ability to see what jobs have been assigned to them
    - Allows the builders to add comments to the currently assigned job
    - Allow the builders to mark the job assigned to them as completed with any additional comments added to the job

Non-Functional Requirements

- Clear GUI-The GUI should be clear in the sense that the program should require minimum training and try to be self-explanatory. The controls should stand out in the way that users shouldn't be looking for the controls.
- Easy to use-The GUI should be easy to use and have a feedback system which will allow the user to see that the system has registered their response.
- Include validation
    - Check if tenants already exist when adding a new tenant
    - Check if the property already exists when adding a new property
    - Check if username and password is not empty when logging in
    - Check if the correct type of values is entered in fields that require numeric values
    - Check if all the necessary values have been entered
- Performance-Must be responsive and swift and require minimum memory to ensure other programs can be used whilst the housing application is running
- Stability- The program should not crash or freeze at any point due to a bug, there should also always be a reliable on-demand connection to the database system that is hosted online. There should also be a backup system that periodically backs up the database at least once a week so that an event of loss of data can be quickly recovered
- Upgradable-The method of coding should allow the possibility of implementing extra features in the feature without requiring changing any current code

Design and Implementation- Desktop Application

Design

Styling

I have decided to follow a material theme in my program which to aid me in this, I have researched a custom JavaFX library called Jfoenix which was designed with a material design theme in mind. The material design theme was developed by Google in 2004 and makes use of bold simple colour to make elements stand out. It also makes use of a feedback system that is subtle but clear to the user, for instance, buttons would have a ripple animation when clicked to notify the user that they have clicked on the button they were pointing at. The material design also makes uses of simple shapes that are familiar to a human being i.e. square and circle shaped elements. (Google, 2018) My desktop application makes full use of the library to integrate a material type design as much as possible since the design itself is now common with most Android devices and Google applications such as Google drive and Google maps.

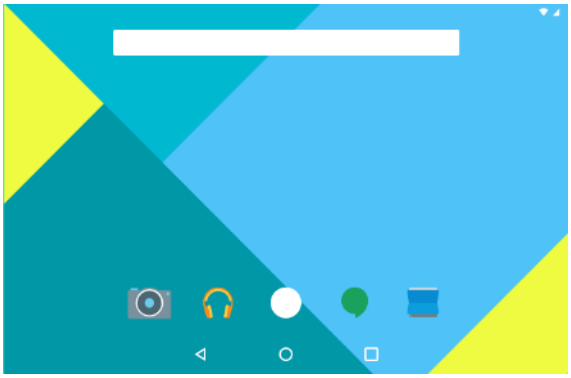


Figure 9 showing an Android home screen that has implemented a material design language

The library had to be downloaded from their site and imported manually into the java project library. Since JavaFX uses scene-builder (a GUI editor) to allow the developer to drag and drop elements onto a blank stage. Scene-builder also required the custom library to be imported into the GUI editor for the custom elements to be used in the editor. These custom elements have been styled to look like button used on an android application and have a predefined CSS with animations that make it feel like an android application. To apply functions on the custom elements, I had to resort to Jfoenix documentation which teaches you how each element work with the controller and an example of how it used. Since the custom library is coded differently, I was constantly referring to the documentation so that the elements are working as intended. (JFoenix, n.d.)

High-Level Design

The desktop program follows a model-view-controller in combination with the observer design pattern. This combination allowed me to code more efficiently and be able to add new features in the future without any risk of negatively affecting the current codes. Each main feature would have its own package where the FXML file and the FXML's controller would be stored. All the model files would be stored in its own package

Model

Model's represent real and logical entities. Each model would contain variables that would represent an attribute of the entity, for example, the tenant model would contain the attributes "First Name" and "Last Name" that can be called by the controller class. Each model instance is initiated by the controller class when the controller itself retrieves data from the database and each row is stored as an instance of the required model class. For instance, when the controller retrieves all the tenants and their data from the database, each tenant is then stored as an instance of the tenant model class with their attributes assigned via the constructor method that is in the model class

View

The view files (FXML) consist of all the UI elements that are required. The view file would determine the size and location of the element and how it looks in terms of colour and alignment. These view files would determine what the user sees and how the user will see it. Each view file has an assigned controller class that will take data from the view and output data back to the view file. When a user is navigating through the application, the initiated class will load the view file as well as the controller that controls the view file.

Controller

The controller class is the brains behind the view files where data is processed and then outputted to the relevant element in the view file. It is also responsible for creating instances of model classes and then storing all the model instances into an observable list that is constantly monitored for any change by the controller itself. The controller will also handle the data going from and to the database system that is stored on an external dedicated server as well as manipulating any elements in the view file such as disabling elements or populating combo boxes. (IBM, 2016)

Desktop Application model and screen flow

The application consists of 7 main screens that will each represent one main functional feature. One screen will always remain running which is the main menu, to allow users to navigate from one feature to another feature without having to restart the program. Once the user successfully logs into the desktop application, the main menu will begin running and provide options for the user to pick from.

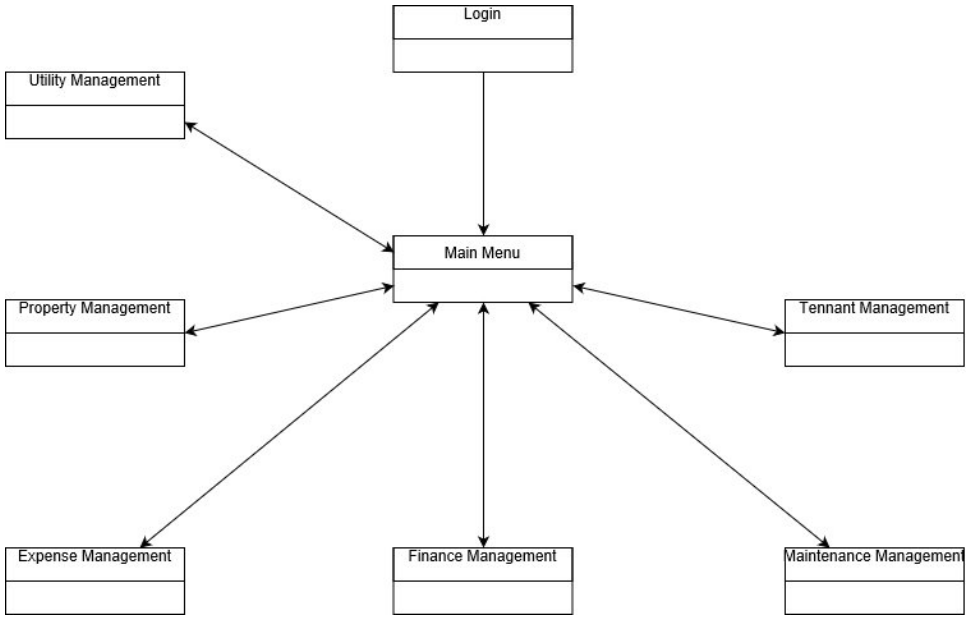


Figure 10 showing a basic class diagram of the overall structure of the desktop application

Implementation

Database package

The database package consists of all the model classes that are required for the controllers to function as intended. Each model class contains getters and setter methods that set the variables with data from the database and sends variables back to the controller when required. The package will also contain the connection class which is responsible for connecting to the MySQL database that is running on the Amazons web server and then returning the connection to the class that has called the method in the connection class.

Tenant model

The tenant model consists of all the attributes that are related to the tenant. The model would mirror the attributes that exist in the database. They are First Name, Last Name, DOB, the property

ID that links to a property, the date they moved into the property and the date they left the property.

Property model

The property model consists of the following attributes. Door number, Flat letter, the first line of the address, postcode, the date the contract started, the rent cost, the day the rent is due, the landlord ID that links to a landlord, the contract start date and contract end date.

Landlord model

The landlord model consists the following attribute: The Landlord's full name, ID number, bank account number and sort code.

Expense model

The expense model holds the expense name, quantity and cost and the category ID of a specific expense. It would also hold the sum of all the expense grouped by months per property that would be later used by the controller to display an overview of total expenses spent per month for a property.

Utility model

The utility model represents the utility bills that are needed to be monitored for a property. The model consists of the electric, gas, water and council tax account numbers as well as the company being paid and the monthly cost per utility

Issues model

The issues model represents an issue that has been reported on a property. The model consists of the issue description, the date the issue was reported, the status of the issue which can be one of the three choices: Open which means the issue has been recorded, Ongoing which means that a job has been initiated and Closed which means that the issue has been resolved. The model also has the severity of the issue which can be one of the three choices which are high, medium and low where high means a high severity issue and low is a low severity issue. The model will also hold the builder that has been assigned to the job, the date the job started, the date the job was completed, and any comments made by the builder.

Builders model

The builder model represents a builder that can be assigned to a job. The model would store the builders First name, Last name and their ID.

Login package

- Login controller- The login controller deals with checking the username and the password that has been inputted by the user. Once the user clicks on the submit button, the controller connects to the database and query the User table for a record that matches the username and the password. If it returns a result, then a match was found and the controller proceeds to start the home screen view and controller. However, if a result is not returned, then a match was not found therefore the combination is incorrect and will wait until the user attempts again.
- Login View- The login view consists of a very simple GUI of which there are two input text boxes in which the user would enter their username and password then followed by a button that submits the data entered by the user.

Home screen package

- Home screen controller- The home screen controller is the central hub of the desktop application. The controller enables the user to navigate to different parts of the application with ease and is always running in the background or foreground. If another controller is running, the home screen moves to the back of the screen until the user decides to quit the current screen which will cause the home screen to migrate back to the front to allow the user to navigate to a different part of the program.
- Home screen view- The home screen uses a “tile” system called masonry that houses various buttons and elements. These graphical elements, when added onto the masonary pane, can auto arrange itself whenever the window size is changed by the user and auto enables the ability to scroll if there are too many elements for the window size. The tile system is maintainable and upgradable which means that in the future if a new screen was implemented, the user would simply add a new button to the masonry pane and link the button to the controller.

Tenant viewer package

- Tenant management controller- The tenant management controller deals with the retrieval of tenant-related data from the database and into the tenant model. Once it retrieved and assigns all the tenants data into the model, the controller then adds every instance of the tenant model into an observable list for the table view to output. The controller also deals with data that is selected in the table view by the user and enables the ability to delete the selected tenant or relocate the tenant. It should also work out the number of days per month, the tenant has accommodated at the property by using the date that they moved into either the date they moved out or the current date. The controller also allows the user to start the adding tenant controller and the relocate tenant controller if the user chooses to.
- Tenant management view- The tenant management view consists of a large table that takes the majority of space to allow the user to see every data relevant to the tenant without any risk of truncating the data. On the right of the table, there is a list of buttons of various functions where the delete and relocate tenant buttons are disabled by default by the view file. On the bottom of the table houses the overview of the number of days a selected tenant has stayed for and is grouped by months.
- Adding tenant controller- The tenant controller simply deals with taking in data that has been entered by the user that is required for the database when adding a new tenant into the database and sends an INSERT signal with the entered data into the database system. The controller also constantly checks the data being inputted into the first name and last name field to ensure that only text and certain special characters are being inputted and if not, the controller removes the invalid character.
- Adding tenant view-The adding tenant view is a small view that runs on top of the tenant management view. The view acts as a form that the user must fill out using the provided text fields. The view also gives the user the ability to choose what property the tenant will live at using the drop-down box consisting of the current active properties that are populated by the controller as well as the date the tenant moved in using the date picker which brings out a fully working calendar in which the user can select the date.
- Relocate tenant controller-The relocate tenant controller is responsible for handling the transfer of tenant between two properties. The controller first populates the table view

with all the existing tenants that are currently in the database. When the user selects a tenant from the table-view, the controller enables the select tenant button in which once clicked on, should place the first name and last name of the tenant into the text fields and enable the relocate button in which the system updates the database on where the tenant lives based on the selected property on the drop-down list.

- Relocate tenant view- The relocate tenant view is a small view that runs on top of the tenant management view. It consists of a table view that takes up the majority of the view and two non-editable text fields and two buttons that are disabled by default on the right of the table view.

Property viewer package

- Property management controller- The property management controller deals with the retrieval of property-related data from the database and into the property model. Once it retrieved and assigns all the property data into the model, the controller then adds every instance of the property model into an observable list for the table view to output. The controller also deals with data that is selected in the table view by the user and enables the ability to decommission the selected property or paying the rent of the selected the property. The controller also allows the user to navigate to an additional screen where they can add a new property into the system or navigate to a screen where they can manage area managers that exists in the database.
- Property management view- The property management view consists of a large table that takes the majority of space to allow the user to see every data relevant to the property without any risk of truncating the data. On the right of the table, there is a list of buttons of various functions where the delete, return property and pay rent buttons are disabled by default by the view file.
- Adding property controller- The adding property controller deals with taking in data that has been entered by the user that is required for the database when adding a new property into the database and sends an INSERT signal with the entered data into the database system. The controller also constantly checks the data being inputted into property rate and rent cost field to ensure that only decimal values are inputted and that the decimal is in a two-decimal point format and if not, the controller clears the field if invalid data is detected. The

controller also gives the ability to add a new landlord if the landlord they want to select is not in the system, this brings a new screen and initialises the adding landlord controller.

- Adding property view- The adding property view is a small view that runs on top of the property management view. The view acts as a form that the user must fill out using the provided text fields. The view also gives the user the ability to choose what landlord the property belongs too by using the drop-down box consisting of the active landlords that are populated by the controller as well as the date the contract starts and ends by using the date picker which brings out a fully working calendar in which the user can select the date.
- Adding landlord controller- The adding landlord controller deals with taking in data that has been entered by the user that is required for the database when adding a new landlord into the database and sends an INSERT signal with the entered data into the database system.
- Adding landlord view- The adding landlord view is a small view that runs on top of the adding property view. The view acts as a form that the user must fill out using the provided text fields.
- Rent paying controller- The rent paying controller is responsible for handling the rent payment recording of the selected property. The controller first populates the table view with all the existing rents paid on the selected property that is currently in the database. The controller automatically retrieves the rent that is set for the property and set the rent text field with the rent due cost. The controller then takes the deductions and minuses the rent due with the deduction entered in and uses the result of the calculation as the total rent due. The controller then takes each of the values in the text fields and sends an INSERT signal with the data from the text fields into the database and refreshes the table view with the recently paid rent.
- Rent paying view- The rent paying view is a small view that runs on top of the property management view. It consists of a table view that takes up the majority of the view and two non-editable text fields and one editable text field for the deductions and a button that submits the rent that has been paid.
- Manage area manager controller- The manage area controller is responsible for handling the current area managers in the database. The controller first populates the table view with all the existing area managers that are currently in the database. When adding a new area manager, the controller takes each of the values in the text fields and sends an INSERT

signal with the data from the text fields into the database and refreshes the table view with the newly added area manager. The controller can also edit an area manager when the user selects an area manager from the table view and then clicks on the edit button, the controller sets the text fields with the data from the selected area manager on the table view. Once edited and the update button is pressed, the controller sends an UPDATE signal with the updated values to the database.

- Manage area manager view- The manage area manager view is a small view that runs on top of the property management view. It consists of a table view that takes up the majority of the view and two editable text fields four button that adds, deletes, selects and updates area managers.

Expense viewer package

- Property expense management controller- The property expense management controller deals with the retrieval and calculation of expenses spent per month from the database and into the property model. Once retrieved and calculated, the controller adds each sum of the months of the specific property into its own expense model instance. The controller then adds every instance of the expense model into an observable list for the table view to output. The controller also deals with data that is selected in the table view by the user and enables the ability to view the expenses brought for the selected property. The controller also allows the user to navigate to an additional screen where they can import expenses from bank statements.
- Property expense overview view- The property expense overview view consists of a large table that takes the majority of space to allow the user to see every data relevant to the property without any risk of truncating the data. On the right of the table, there is a list of buttons of various functions where the manage expense buttons are disabled by default by the view file.
- Expense importer controller-The expense importer controller is responsible for handling importing expenses into the database. The controller first populates the table view with all the existing expense that are currently not assigned to any proprieties that are currently in the database. The controller also allows the ability to import expenses paid from a CSV bank statement that is automatically added to the database. Before importing the expenses, the controller asks the user on what bank is the bank statement downloaded from since every

bank uses different column layout and then use the layout as a way of picking out the relevant data from the CSV file. Whilst importing, the controller checks the utility account numbers from the database with the description of the expense to see if the description contains the account number, if there is then the controller sets a property that has used the expense on that expense being imported and the category of the expense automatically. Once imported, the controller updates the observable list for the table view. The controller also allows editing an expense when the user selects an expense item from the table view and then clicks on the edit button, the controller sets the text fields with the data from the selected expense item on the table view. Once edited and the update button is pressed, the controller sends an UPDATE signal with the updated values to the database.

- Expense importer view- The expense importer view is a small view that runs on top of the Property expense overview view. It consists of a table view that takes up the majority of the view and various editable text fields as well as a drop-down list and a date picker. It also consists of four buttons that imports, deletes, selects and updates expenses.
- Manage expense controller- The expense importer controller is responsible for handling adding expenses into the database. The controller first populates the table view with all the existing expense that is currently assigned to the selected propriety. When adding a new expense, the controller takes each of the values in the text fields and sends an INSERT signal with the data from the text fields into the database and refreshes the table view with the newly added expense. The controller can also edit an expense when the user selects an expense from the table view and then clicks on the edit button, the controller sets the text fields with the data from the selected expense on the table view. Once edited and the update button is pressed, the controller sends an UPDATE signal with the updated values to the database.
- Manage expense view- The manage expense view is a small view that runs on top of the Property expense overview view. It consists of a table view that takes up the majority of the view and various editable text fields as well as a drop-down list and a date picker. It also consists of four buttons that add, deletes, selects and updates expenses.

Utility viewer package

- Utility viewer controller- The utility viewer controller deals with the retrieval of utilities and account numbers from the database and into the utility model. Once retrieved, the

controller adds the gas, electric, water and council tax name and account number and the cost per month of each property into its own utility model instance. The controller then adds every instance of the utility model into an observable list for the table view to output. The controller also deals with data that is selected in the table view by the user and enables the ability to view and edit the utility for the selected property.

- Utility viewer view- The utility viewer view consists of a large table that takes the majority of space to allow the user to see every data relevant to the property's utility without any risk of truncating the data. On the right of the table, there is a button that initiates the manage utility controller. Also, the manage utility button is disabled by default by the view file.
- Manage utility controller- The manage utility controller is responsible for handling adding utility details into the database. The controller first populates the table view with all the existing utility details that are currently assigned to the selected property. When editing a utility, the user first selects a utility from the table view and then clicks on the edit button, the controller sets the text fields with the data from the selected utility on the table view. Once edited and the update button is pressed, the controller sends an UPDATE signal with the updated values to the database.
- Manage utility view- The manage utility view is a small view that runs on top of the utility viewer view. It consists of a table view that takes up the majority of the view and various editable text fields as well as a drop-down list. It also consists of four buttons that add, deletes, selects and updates utilities.

Finance viewer package

- Finance management controller-- The Finance management controller deals with the retrieval and calculation of expenses spent per month and the income based on the number of days the tenant has been living at the property multiplied by the rate of the property from the database and into the property model. Once retrieved and calculated, the controller adds each sum of the months of the specific property into its own finance model instance. The controller then adds every instance of the expense model into an observable list for the table view to output and checks whether the result of the sum is in positive or negative and if positive, sets the cell as green to represent profit whilst negative sets the cell as red to represent a loss. The controller also deals with data that is selected in the table view by the user and enables the ability to view the overview of the selected property.

- Finance management view- The Finance management view consists of a large table that takes the majority of space to allow the user to see every data relevant to the property's finance without any risk of truncating the data. On the right of the table, there is a button that initiates the detailed finance controller. Also, the detailed finance button is disabled by default by the view file.
- Detailed finance controller- The detailed finance controller is responsible for retrieving expenses and financial data based on the selected property. The controller then outputs a pie chart consisting of an overview of what was spent on the property on the selected month and year. The controller also populates the table view with the total income and total expense spent for the selected month and year.
- Detailed finance view- The detailed finance view is a small view that runs on top of the utility viewer view. It consists of a pie chart that takes up the majority of the view, a small table view on the bottom right and two drop-down boxes that allows the user to select the year and month.

Maintenance viewer package

- Maintenance management controller- The maintenance management controller deals with the retrieval of maintenance related data from the database and into the issues model. Once it retrieved and assigns all the maintenance data into the model, the controller then adds every instance of the issues model into an observable list for the table view to output. The controller also deals with data that is selected in the table view by the user and enables the ability to initiate the selected issue or adding, deleting and editing an issue. The controller can also filter the types of issues that are being listed on the table view, for instance, the controller can change the observable list so that it consists of only completed jobs or initiated jobs and finally jobs that have not been initialised. The controller also colour-codes each issue based on the severity of the issue. If the severity is high then the cell becomes red, if it is medium then orange and finally low will be yellow.
- Maintenance management view- The maintenance management view consists of a large table that takes the majority of space to allow the user to see every data relevant to the issues without any risk of truncating the data. On the right of the table view, there are various buttons where some of the buttons would bring up a new screen and some buttons would determine what type of issues to list in the table view.

- Manage builder controller- The manage builder controller is responsible for handling the current builders in the database. The controller first populates the table view with all the existing builders that are currently in the database. When adding a new builder, the controller takes each of the values in the text fields and sends an INSERT signal with the data from the text fields into the database and refreshes the table view with the newly added builder. The controller can also edit a builder when the user selects a builder from the table view and then clicks on the edit button, the controller sets the text fields with the data from the selected builder on the table view. Once edited and the update button is pressed, the controller sends an UPDATE signal with the updated values to the database.
- Manage builder view- The manage builder view is a small view that runs on top of the maintenance management view. It consists of a table view that takes up the majority of the view and two editable text fields and four button that adds, deletes, selects and updates builders.
- Adding issue controller- The adding issue controller deals with creating new maintenance issues and adding the issues to the database. The controller first populates the property drop down box with all the current properties of the database. When filling the details of the new issue, the controller populates the table view with all the jobs that are currently active with the selected property from the drop-down box as well as populating the tenant drop down box of the tenants that currently live at the selected property. Once all the relevant fields have been filled out, the controller sends an INSERT signal to the database with the data inputted by the user and adds a new entry into the issues table.
- Adding issue view- The adding issue view is a small view that runs on top of the maintenance management view. It consists of a table view that takes up the majority of the view and text fields, drop-down boxes, date picker and four button that adds, deletes, selects and updates builders.
- Edit issue controller- The edit issue controller deals with editing selected maintenance issues and updating the selected issue into the database. The controller first sets all the fields with the data from the selected issue. The controller then populates the property drop down box with all the current properties of the database. When editing the details of the selected issue, the controller populates the table view with all the jobs that are currently active with the selected property from the drop-down box as well as populating the tenant

drop down box of the tenants that currently live at the selected property. Once all the relevant fields have been filled out, the controller sends an UPDATE signal to the database with the data inputted by the user and updates the selected entry into the issues table.

- Edit issue view- The edit issue view is a small view that runs on top of the maintenance management view. It consists of a table view that takes up the majority of the view and text fields, drop-down boxes, date picker and four button that adds, deletes, selects and updates builders.
- Initiate job controller- The initiate job controller handles the initiating of a selected “Open” issue. The controller first populates the drop-down box with builders from the database in which the user can select the builder to assign the issue to. Once the user selects the drop-down box, the controller populates the table view with all the jobs the builder is currently assigned to. Once the user completes filling in the required form, the controller updates the issue on the database as well as insert a new entry onto the initiated and completed table with the issue ID and the selected builder’s ID.
- Initiate job view- The initiate job view is a small view that runs on top of the maintenance management view. It consists of a table view that takes up the majority of the view and text fields, drop-down boxes, date picker and one button that initiates the job.

Design and Implementation- Web Application

Web Application Styling

For the web application, I have been using a template that contains a CSS file that has been preconfigured to work with the template file. The template consists of a simplistic but professional design where there is be a menu bar on top and then a body in the middle that consists of the vital data that is required by the user.

High-Level design

For my web application, I used Django's framework to build my web application. The engine of the Django framework consists of many modules where each module is responsible for a certain task and if one fails then the entire application fails.

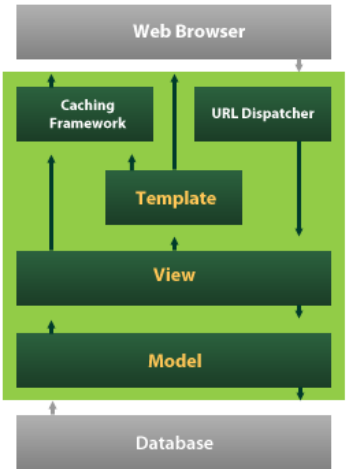


Figure 11 showing an overview of how the Django framework works

URL Dispatcher

The URL dispatcher's role is to monitor the URL that is typed on to the browser and then maps a function from a specific view within the server side (Elman & Lavin, 2014). The URL dispatcher also filters out any characters inputted by the user by checking the URL with a specified regular expression in the URL configuration file. Regular expressions are a sequence of characters that a system searches for. If the regular expression was "[a-zA-Z]" then the regular expression will only allow letters in the input whether it is lowercase (a-z) or upper case (A-Z). The URL dispatcher can also take data from the URL for instance, one of the configure URL map is "*'^(?P<type>[A-Z-a-z]+)/(?P<pk>[0-9-]+)/home?'*" where data added within <type> will be taken as an argument for a certain view so if "builder/2/home" is detected in the URL then "builder" and "2" would be taken as arguments for the mapped view and for the <type> argument, only letters will be accepted and for <pk> only integers will be accepted.

URL Patterns	Description
`url(r'^$' , views.loginPage)`	Will invoke the loginPage view E.g: http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/

`url(r'Login',views.getIndex)`	Will invoke the view called Login E.g: http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/Login
`url(r'^(?P<type>[A-Z-a-z]+)/(?P<pk>[0-9-]+)/home?',views.getIndex)`	Will invoke the view called getIndex and take the type and pk as parameters of the view E.g: http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/tennant/7/home
`url(r'^(?P<builderpk>[0-9-]+)/(?P<pk>[0-9-]+)/jobDetail?',views.getJobDetail)`	Will invoke the view called getJobDetail and take the builderpk and pk as parameters for the view E.g: http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/2/22/jobDetail
`url(r'(?P<pk>[0-9-]+)/jobUpdate?',views.jobUpdate)`	Will invoke the view called jobUpdate and take the pk as parameter for the view http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/22/jobUpdate
`url(r'(?P<pk>[0-9-]+)/reportIssue?',views.getReportPage)`	Will invoke the view called getReportPage and take pk as parameter for the view http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/7/reportIssue
`url(r'(?P<pk>[0-9-]+)/sendReport?', views.reportIssue)`	Will invoke the view called reportIssue and take pk as parameter for the view http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/7/sendReport
`url(r'(?P<pk>[0-9-]+)/(?P<issueID>[0-9-]+)/issueDetail?', views.getissue)`	Will invoke the view called getIssue and take issueID as parameter for the view http://housingmaintanance-musab.apps.devcloud.eecs.qmul.ac.uk/7/issueDetail

View

The view's role is to execute functions that are called upon and returns a certain result. These results could be a certain format of responses such as JSON response or HTML response which sends back an HTML type data back to the browser or a JSON object back to the browser.

View Functions

- index- The "index" function when called upon by the URL dispatcher first checks if the user is logged in by accessing the client's cookies to see if a request session exists. If it does then the function returns an index template for the browser to output for the user. If the session cookie does not exist, the function passes the request to the "login page" function which

returns a result back to the current view that can be used to send back to the browser as a response.

- Login page- The “login page” function is a simple function that retrieves and returns the login.html template back to the browser to output to the user. The “login page” function also returns a Boolean variable “user logged in” as false to notify the HTML page that the user has not logged in and not to show certain menu items in the menu bar.
- Login—the “login” function takes the POST request that is sent to the server with a form that is contained within the POST request payload. It takes out the username and password that was inputted into the login page’s text box and checks the combination with the user object which retrieves user data from the database and creates a user object. If a user object exists, then the combination is correct, and a session cookie is stored into the client’s browser and lastly, the function responds back a JSON object for the login page’s JS script to process. However, if a user object does not exist then the combination is incorrect, and the function does not return anything back to the login page.
- Get Index- The “get Index” function takes in the type of user and the user’s ID as an argument and retrieves data that is relevant to the type of user as well the user itself. If the user type is a builder, then the function retrieves all the jobs that are currently assigned to the builder based on the user’s ID. If the user type is a tenant, the function retrieves all the issues that the tenant has reported based on the user’s ID. Once the relevant data has been retrieved, the data is then packed into a JSON object and sent with the index HTML page back to the browser where the browser outputs the index page and the data that was sent with the HTML page. However, if the function detects that the session cookie does not exist then the function immediately responds with the login page. This is to stop unauthorised users from simple typing the type of user and a user’s ID into the URL to gain access to their data.
- Get Job Detail- The “get job detail” function retrieves a specific job that is specified by the parameters of the function called “pk”. The function retrieves the issue and the property the issue is reported on. Once all the data is gathered, the function packages the data as a JSON object which is then sent with the job Detail HTML page for the browser to render to the user and output the data within the packaged JSON object.

- Job Update- the “job update” function is responsible for updating the job that has been assigned to the builder who is currently logged in to the application. The update can be a simple addition of a new comment of the current job or can be the completion of the current job which changes the status of the job to complete. The function receives a request packet that contains the form data used to update the job. The form data consist of the type of update which could be one of the two options: complete or incomplete. If complete, the function takes today’s date as the date the job was completed as well as the job’s comments that are contained within the form data and update the entry in the initiated and completed jobs table in the database. Once updated, the function retrieves the details of the job that was completed and responds back, with the user’s object from the database that is packaged as a JSON object.
- Get Issue- the “get issue” function retrieves the selected reported issue by the tenant. The function gathers the issue details and if the issue has been initiated then the function also retrieves the job and its details. Once the relevant data has been gathered, the function returns an issue detail template back to the browser for rendering the gathered data as well as a JSON packaged object.
- Get Report Page- the “get report page” function is a simple function that retrieves the issue reporting template from the server and responds with the template for the browser to render.
- Report Issue- The “report issue” function deals with the creation of a new issue into the database. The function receives a POST packet that contains a form data payload. The form data consists of the issue that was inputted by the user on the web page. The function creates a new issue object and adds the issue to the issue object as well as the user’s ID and the current date into the issue object. Once the issue object has been filled with the relevant data, the function saves the object into the issues table in the database and respond back to the browser with an HTTP response to let the browser know that it was a success.
- Logout- the “logout” function simply flushes the client’s cookies and sets the login flag as false which directs any requests from the client to the login page.

Design and Implementation- Database System

The database is responsible for storing data that is vital to the desktop application and the web applications core features. Without the database, both types of applications would not work.

Entity-Relationship Model

I have designed a database schema which I have now used to generate the models in my database. I initially wrote out the schema on a word document and carefully added relations to each model with a brief description of how it is related as a way of planning the schema. The schema has also been carefully designed so that there is no abundant data and that each model would store data that is relevant to the table. The database I planned has 11 entities and 10 relationships.

- Tenant -> Property
    - Tenant will have a many to one relationship where many tenants can live in one property and one property can have many tenants.
- Issues-> Property
    - Issues will have a one to many relationships with properties where one property can have many issues and many issues can be for one property
- AreaManager -> Property
    - AreaManager will have a one to many relationships where one property can have one area manager and an area manager looks after many properties
- Landlord-> Property
    - Landlord will have a one to many relationships with properties where many properties are owned by one landlord and one landlord can own many properties
- OldTenants -> Property
    - OldTenants will have a many to one relationship where many tenants can live in one property and one property can have many OldTenants.
- Utilities -> Properties
    - Utilities will have a one to one relationship with Property where one property will have one set of utilities and one set of utilities is for one property
- Expense->Properties
    - Expense will have a many to one relationship with property where many expenses are brought for one property and one property will have many expenses
- Expense -> Expense Category
    - Expense will have a one to one relationship with expense category where one expense will have one expense category.
- Tenant, Builder -> Users

- Tennant and builder will have a one to one relationship with user where one tenant or builder will be one user and one user can be one tenant or builder

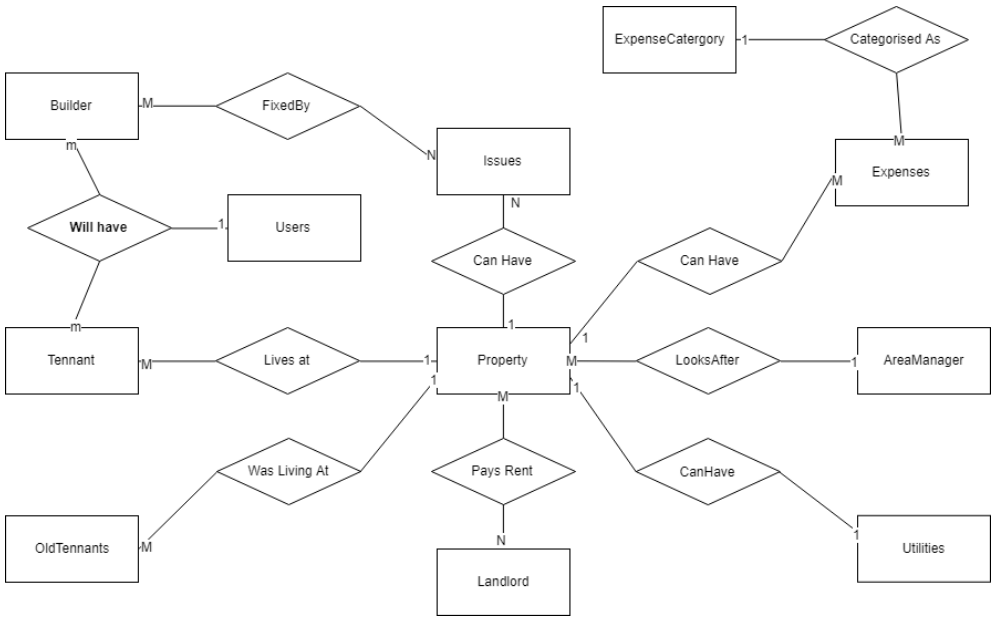


Figure 12 showing the ERM diagram of the housing accommodation database scheme

Relational Schema

- Tennant (*tennantID*, FirstName, LastName, DOB, *LivesAt,* LivingFrom)
  - Description: This model is responsible for storing active tenants (tenants that are currently living at a property). The model stores the personal details of the tenants which includes their full name and date of birth. The model also stores the date the tenant moved into the property and if the tenant has moved out, it stores the date they moved out.
  - Foreign key: “LivesAt” is linked to the model “Property” which sets the property the tenant lives at
- Property (*PropertyID*, FlatNo, DoorNo, Firstline, PostCode, Town, Borough, *AreaManager,* ContractEnded, ContractStarted, Cost, DayOfMonth, *Landlord*, Rate)
  - Description: This model is responsible for storing information about a property that is being rented by the company using the database. The model stores the Flat no (can be blank if the property is not an apartment, flat or studio), the door number, first line of the address, postcode, the town and the borough it resides in. The model also stores property contract details such as the contract start and end date, the rent that needs to be paid per month, the day the rent should be paid in every month and the rate of the property of which the tenant will be charged per night. The model is

also given the ID of the area manager that looks after the property. The model also sets the landlord that holds the contract of the property

    - Foreign key: “AreaManager” is linked to the model “AreaManager” which will set the area manager of the property.

- landlord (***LandlordID, landlordName***, bankAccountNo, sortCode)
    - Description: This model stores the information related to the landlord. This includes the full name of the landlord as well as the bank details of the landlord that will be used to pay rent to the landlord.
- PaidRent (***PropertyID, rentID***, DatePaid, Deduction)
    - Description: This model stores the information that is related to the rents paid to landlords. This would include the property the rent was paid for, the amount that was deducted off the rent for reasons such as issues that the landlord has agreed to pay for and the date the rent was paid.
    - Foreign Key: PropertyID is a foreign key to Property.
- Utilities (*utilID*, *PropertyID*, CTaxBorough, CTaxAccntNo, CTaxCost, EnegComp,EnegAcntNo, EnegCost, GasComp, GasAcntNo, GasCost, WaterComp, Water AcntNo, WaterCost)
    - Description: This model stores the information that is related to the utilities of each property. The model stores the company, account number and the cost per month of the water, gas, electric and council tax bills of the property.
    - Foreign Key: PropertyID is a foreign Key to Property
- Issues (*IssueID*, Description, *reportedBy*, Date, Status, Level of Urgency, PropertyID)
    - Description: This model stores the data that is related to issues reported by tenants and the desktop users. The model would store the issue that was described by both the users and the property the issues has occurred. The model also stores the severity of the issue and the status of the issue.
    - Foreign Key reportedBy is a foreign Key to Tennant and PropertyID is a foreign key to Property
- Builder (*BuilderID*, FirstName, LastName)
    - Description: This model stores the builders that currently work with the accommodation company. The model stores the builder’s full name.
- initiatedandCompletedJobs (*IssueID*, *BuilderAssigned*,JobStarted, JobCompleted,Comments)

    - Description: This model stores the details of the job for resolving a specific issue. The model would store the date the job started and finished and any comments regarding the job. The model stores the issue that is linked to the job as well as the builder that has been assigned to the job.
    - IssueID is a foreign key to Issues AND BuilderAssigned is a foreign key to Builder
- Area Manager (*staffID*, FirstName, LastName)
    - Description: This model stores the area manager that currently work with the accommodation company. The model stores the area manager's full name.
- Old tenant (*tennantID*, FirstName, LastName, DOB, *LivesAt,* LivingFrom)
    - Description: This model stores the information related to tenants who are no longer with the housing accommodation. This model acts as an archive for tenants, so users can look back at tenants who have not paid their rent. The model stores the exact data as the tenant model.
    - Foreign key: "LivesAt" is linked to the model "Property" which sets the property the tenant lives at
- Expenses (*idExpenses*, expenseName,amount,quantity, totalCost, Catergory, propertyID, datePaid)
    - Description: This model stores the information related to any expense brought for a certain property or for the company. The model would store the name of the expense brought, the quantity of the expense, the total cost of the expense and the date the expense was paid. The model also stores the category id of the expense and the property id of the property the expense was bought for.
    - Foreign key: "Catergory" is linked to the model "expenseCatergory" and "propertyID" is linked to the model Property
- expensescatergory(*idexpenseCatergory*,catergoryName)
    - Description: This model stores the name of the category for expenses.

Algorithms

Number of days per month algorithm

One of the drawbacks of using just the start date of the tenant accommodation and the end date is that there is no direct way of knowing the breakdown of each month the tenant lived at the property. To get tackle this problem, I created an algorithm that gets the total number of days spent per year and then breaks it down by subtracting the number of days from January 1st *1970 (in integer form, java will work out the number days from January 1st, 1970 to the date entered)*to the current month by the number of days the tenant has lived at the property and then storing the result in an array. This will iterate through each month unless it meets one of the two conditions:

- Tenant moved in after today's date
    - Since the number of days spent will be zero, all months will be set as 0 and will not iterate any further
- Tenant moved in after January
    - Start iteration on the month the tenant moved in and set the previous months as 0

For example, if the tenant moved in last year on 1 January 2017 and the user wanted a breakdown of days stayed in months. The following process will happen:

First iteration

1. No of days from January 1st, 1970 to January 31st, 2017 =17562 days=Y
2. No of days tenant stayed at the property from January 1st to the day they moved in = 17532=X
3. Difference in days=Y-X -> (17562-17532)
4. Total number of days gone through+=difference in days-> (0+31)
5. Month List [iteration 1] =difference in days

1	2	3	4	5	6	7	8	9	10	11	12	
31												
total number of days gone through ->												31

Figure 13 showing month list after the first iteration

Second iteration

1. No of days from January 1st, 1970 to February 28st, 2017 =17591 days=Y
2. No of days tenant stayed at the property from January 1st to the day they moved in = 17532=X
3. difference in days=(Y-X)- total number of days gone through -> (17591-17532)-31
4. total number of days gone through+=difference in days-> (31+27)
5. Month List [iteration 2] =difference in days

1	2	3	4	5	6	7	8	9	10	11	12	
31	27											
total number of days gone through ->												58

Figure 14 showing month list after the second iteration

Third iteration

1. No of days from January 1st, 1970 to March 31st, 2017 =17621 days=Y
2. No of days tenant stayed at the property from January 1st to the day they moved in = 17532=X
3. difference in days=(Y-X)- total number of days gone through -> (17621-17532)-58
4. total number of days gone through+=difference in days-> (58+31)
5. Month List [iteration 2] =difference in days

1	2	3	4	5	6	7	8	9	10	11	12	
31	27	31										
total number of days gone through ->												89

Figure 15 showing month list after the third iteration

This process will repeat for 12 iterations leading the array to look like this:

1	2	3	4	5	6	7	8	9	10	11	12	
31	28	31	30	31	30	31	31	30	31	30	31	
total number of days gone through ->												365

Figure 17 showing month list after 12 iterations

```
cal.set(cal.get(Calendar.YEAR), i, 1);
//set as last day of the month
cal.set(Calendar.DAY_OF_MONTH,cal.getActualMaximum(Calendar.DAY_OF_MONTH));
Date lastDay = cal.getTime();//get date format of calender object
;//difference in days=(Y-X)- total number of days gone through
long diff = TimeUnit.DAYS.convert(cal.getTimeInMillis() -
tennantMovedIn.getTimeInMillis(), TimeUnit.MILLISECONDS) - days;
days += diff;//add to total number of days
dayspentList[i] = diff;//add to arraylist
```

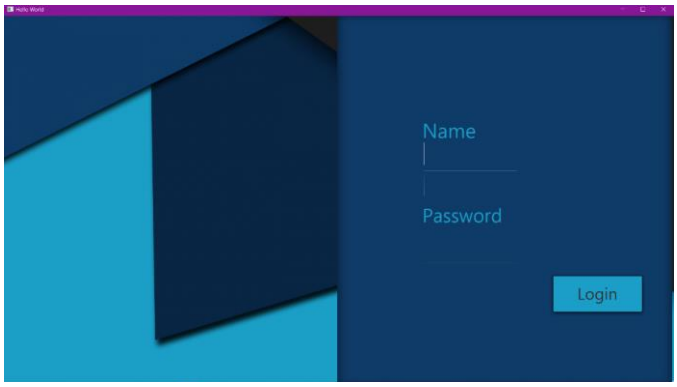
Figure 16 showing snippet of code implementing the algorithm

Application Walkthrough

In this section, I will be doing a walkthrough of my application and showing each main interface.

Desktop Walkthrough

The first screen that the user will see is the login screen. The user will have to first log in to the application by entering their set name and their correct set password. Once the user clicks on the button “Login”, the application will check the login combination and will initiate the home screen if successful and close the login screen.



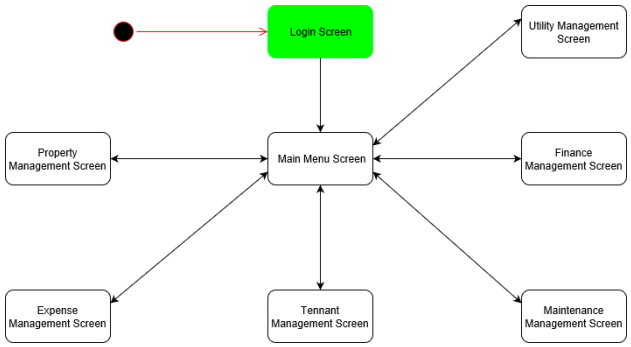


Figure 18 showing the login screen

Once successfully logged in, the application will open the main menu screen where the user can navigate through the various parts of the application by clicking on one of the many buttons on the main menu screen. For instance, if the user clicked the “View Tenants” button then the tenant management interface will be initiated, and the main menu will go into the background whilst the new interface is running.

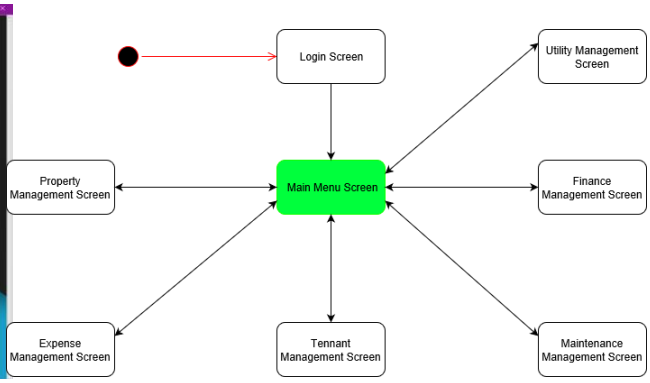


Figure 19 showing the home screen

If the user selects to view tenants, then the tenant management interface will come into the foreground. The tenant management interface mainly consists of a large table that will list all the tenant’s details such as date of birth and the property they currently accommodate in.

The table is interactive in a way that the user can click on a row and the application will then treat that as selecting a tenant in which the user can then delete the tenant (removes any existence of the tenant), remove the tenant (moves the tenant into an archive table) or relocate the tenant. The application will also calculate the number of days the selected tenant has been living at the property and will also divide the number of days into separate months and display on the bottom the table. The user can also add a new tenant by clicking on the “add tenant” button which will bring a mini interface consisting of a form that the user must fill out.

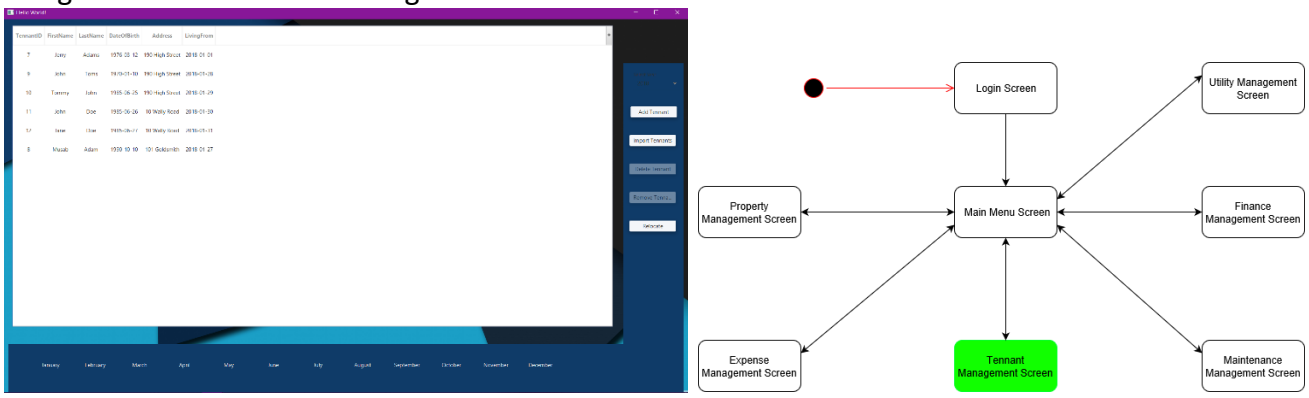


Figure 20 showing the tenant management screen

If the user selects to view properties, then the property management interface will come into the foreground. The property management interface mainly consists of a large table that will list all the properties currently in the database as well as the property details of each property such as the contract start and end date and the area manager that manages the property.

Like the tenant management interface, the table is interactive and allows the user to select a property in which the “return property”, “delete property” and pay rent options to become available. The add property button, pay rent and manage area manager buttons when selected, will open a small interface on top of the current interface in which more functions become available to the user.

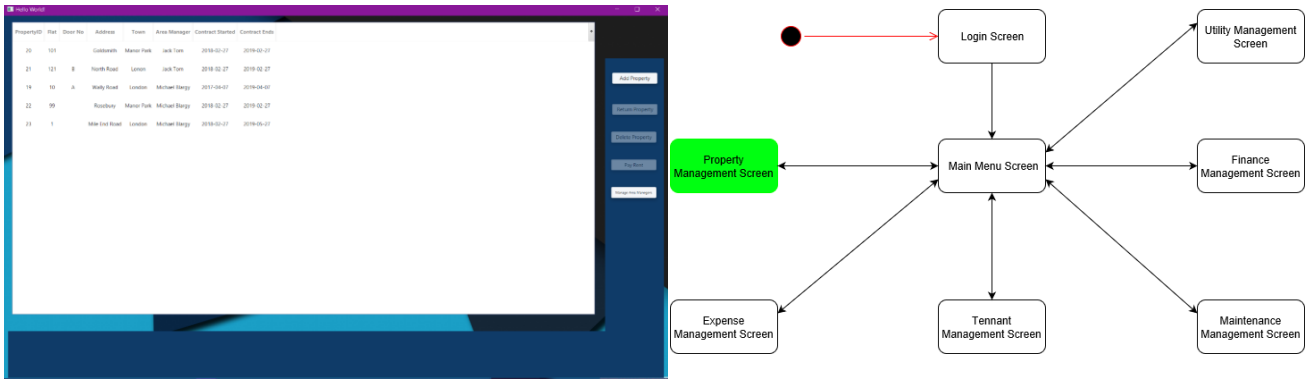


Figure 21 showing the property management screen

If the user selects to view expenses, then the property expense management interface will come into the foreground. The property expense management interface consists of a large table that will list each property and a breakdown of the expense spent on the property per year. The table will also list the breakdown of other expenses that have been spent on the company itself.

The table is interactive and allows the user to select a property which will cause the manage expense button to become enabled. The table can be ordered into the lowest to highest expense total for a certain month, for example, the user can click on the January header which will cause the table to order the data so that the property with the highest total expense spent in January will be at the top. If the user selects “manage expense” button or “import expense” button, the application will initiate a small interface and place it on top of the current interface.

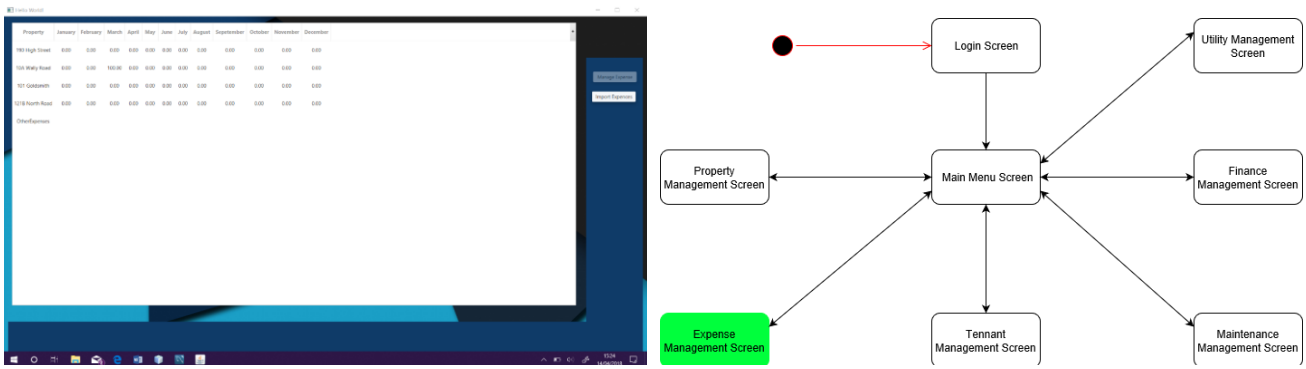


Figure 22 showing the expense management screen

If the user selects to view utilities, then the utility management interface will come into the foreground. The utility management interface consists of a large table that will list all the properties in the system and each type of utility and the details of the utility type which includes the utility supplier, the account number of the account with the supplier and the cost per month(P/m).

The table is interactive and will allow the user to select a property by clicking on the row containing the intended property. If the user selects a property, the button “manage utility” will become

available in which when clicked, will bring up a small interface in which the user can add, edit the utilities of the selected property.

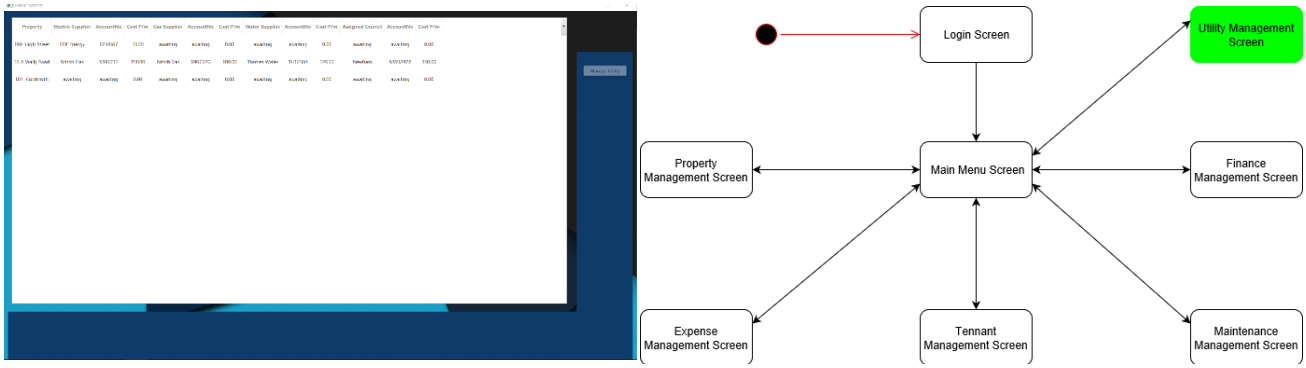


Figure 23 showing the utility management screen

If the user selects to view finance, then the finance management interface will come into the foreground. The finance management interface consists of a large table that will list all the properties in the system and a monthly income per property.

The table is interactive and allows the user to select a property by clicking on the row that contains the property. The table can also dynamically change the colour of each cell based on the income of the property in the month where if the property is/was making a loss in the month then the cell will become red but if the property makes a profit then the cell becomes green. If the user selects a property and then clicks on the button “detailed finance”, a small interface will start which will consist of an overview of the selected property including the income and outgoing costs and a pie chart showing which category of expenses is being brought the most for the property.

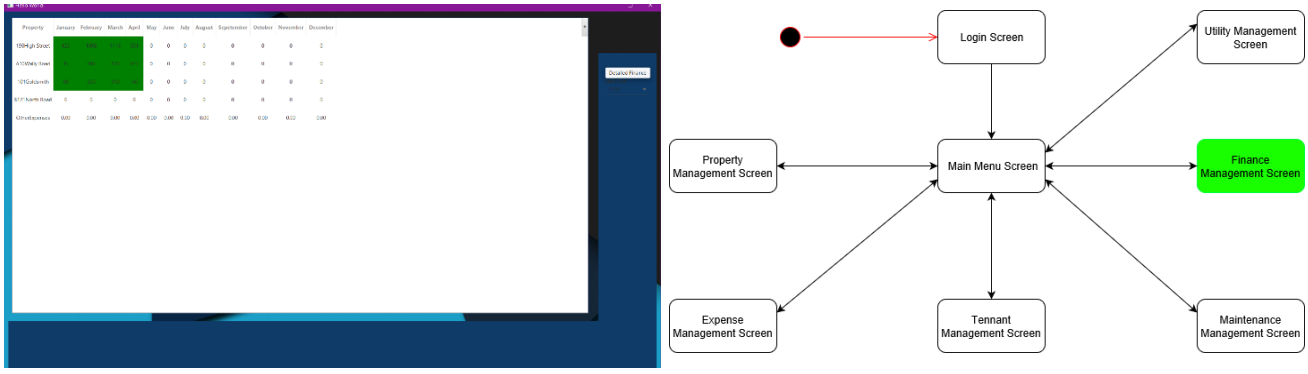


Figure 24 showing the finance management screen

Finally, if the user selects to view maintenance issues, then the maintenance management interface will come into the foreground. The maintenance management interface consists of a large table that initially lists all the issues that have not been assigned to a builder yet. The table can also change the data it is showing by clicking on one of the two buttons that is dedicated to manipulating the table which are “ongoing jobs” button that will show jobs that have been assigned

to a builder but not yet complete and “completed jobs” that will show jobs that have been assigned to a builder but are complete.

The table is interactive and allows the user to select an issue in which depending on the status of the issue, is able to select the “add issue”, “edit issue”, “delete issue” and “initiate issue” button. If the user selects “add issue” or “edit issue” or “initiate issue” button, then a small interface will pop up in front of the current interface.

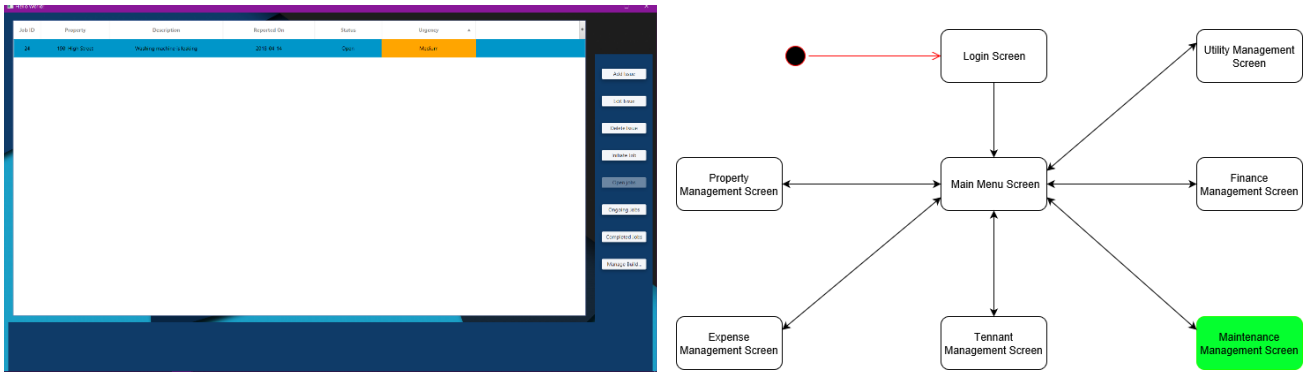


Figure 25 showing the maintenance management screen

Web application walkthrough

When the user first accesses the website, the login screen will appear in which the user will have to enter the username and password into the two text boxes. Once inputted, the web application will check with the server if the username and password combination is correct and will then determine if the user is a builder or a tenant and respond back with the relevant home page.

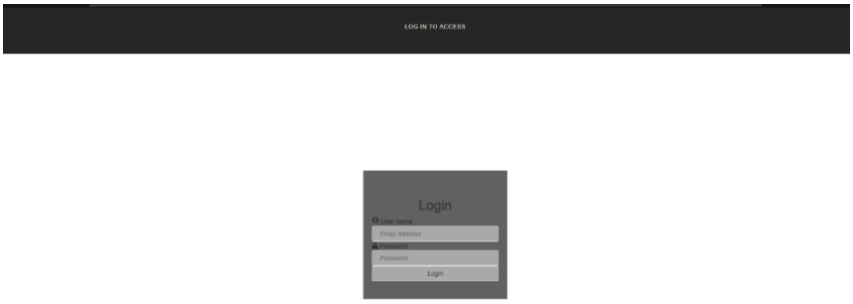


Figure 26 showing the login screen of the web application

If the user is determined as a tenant, the tenant home page is loaded that lists all the issues that have been reported at the property the tenant currently lives at. This is to ensure that the tenant does not report the same issue twice. Each issue is a link to a web page that will provide a progress report on the selected issue.

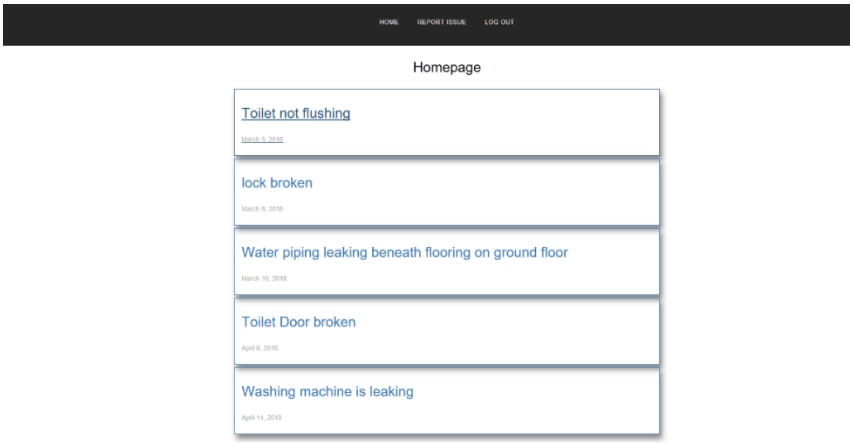


Figure 27 showing the tenants home page

If the tenant selects an issue, the server will redirect the tenant to a web page that will a detailed progress report of the reported issue and the details of the issue itself. The progress log is dynamic and will change when the issue is assigned to a builder and when the builder marks the job as complete.

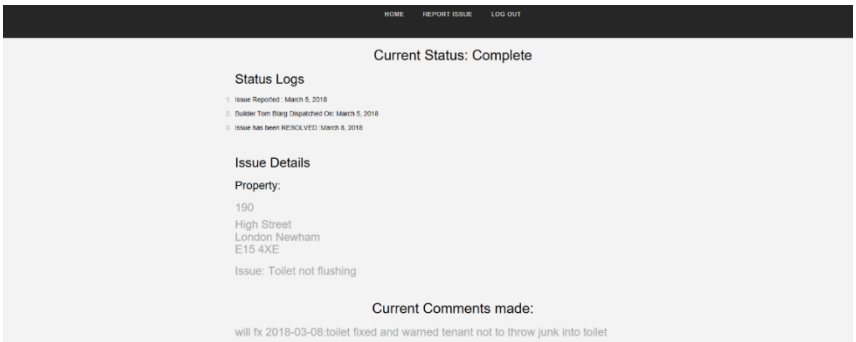


Figure 28 showing the issue status page

If the user is a builder, then the server will respond back to the browser with the builder's home page. This home page will list all the jobs that have been assigned to the builder logged in. The builder can click on the listed job that will then direct the builder to a web page that will provide a detailed report on the issue.

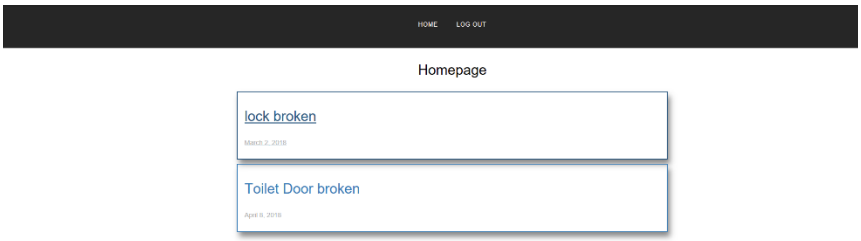


Figure 29 showing the builder's home page

Once the builder clicks on one of the listed jobs in the home page, the server will direct the builder to web page consisting of the jobs details including the issue description, the severity of it, the property that is having that issue and finally a comment box and a complete or update button. If the builder completes the job, they will have to click on the button "Job complete" which will notify the server and change the status of the issue as complete. If the job is not complete after the builder visits the property, then the builder can update the comment section of the job by adding his/her own comments to the issue using the comment box and then clicking on the "Incomplete" button.

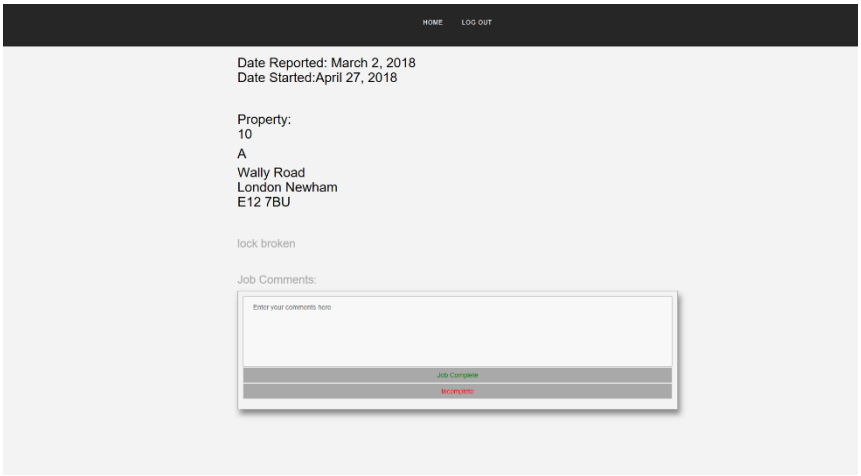


Figure 30 showing the detailed job page

Testing

During the development of each feature, the feature was thoroughly tested to ensure that the application maintained good stability. The tests were done after the all the main features were implemented where the test consisted of testing the security, stability, speed and whether the feature did what it intended to do.

Unit Testing

Unit testing consists of a test plan that contains many tests for the majority of a program. Each test in the test plan would have a set data that will be inputted into the program and will also give an expected outcome of the test when the defined data is inputted. Depending on the feature, some tests will have different subtests that will check the validation of the feature and see how it copes with data that should not be inputted. Overall the tests were a success except one where the JVM compiler sent out a number format exception for an input that was invalid. The bug was found, and a new input validation feature was implemented to ensure that if the same scenario occurred, the application can handle the error and ensure the user becomes aware of the error that has occurred.

Stability Testing

Stability is part of the non-functional requirements of my applications. This part of the testing ensured that whatever the user does with the application, does not crash or force exits the application in any way and that any errors with the user are handled by the application rather than terminating the application. During the testing of each feature, the application did not crash or freeze at any point, however during testing the features of maintenance management, when switching between uninitiated jobs and initiated jobs, the ability to initiate a job was still an option, therefore, users were able to initiate a job that was already initiated thus causing the JVM to give out an exception error that has not been handled. The solution that was applied after testing was to disable the initiate job button based on if the application was showing initiated jobs, therefore, avoiding the potential problem.

With the web application, the server was able to handle multiple requests from multiple devices without any issue at all. The web application was tested from a mobile device, a laptop and another laptop from another location at the same time and the web server was able to respond to each of the devices with the correct response without crashing the server.

Finally, with the database system, the DBMS server on Amazon's web server was able to concurrently handle data from both the web application and desktop applications as well as from multiple users from the desktop application without data being mixed up during the simultaneous connection.

Security Testing

Security is a vital feature in my application where the sensitive data is being handled by all three systems (desktop application, web application and database). All three systems were programmed to ensure that there is no unauthorised access to the data. For the desktop application, the application was able to detect the correct combination of username and password, then progress to the next part of the application and was also able to detect an invalid combination then decline the user and not progress further into the application. The web application also kept its security standards throughout testing. The web app was able to grant and deny logins successfully with the appropriate action. The web application also tracked if the user had logged into the website or not by making use of Django's session cookies that would save the username and password of the logged in user and would remember if the user is logged in to web app if the user decided to manually input a specific URL to access a certain view. If the user has not logged in and the user entered the URL for that allowed builder to mark a specific job as complete, the application correctly identified the unauthorised attempt and redirected the user to the login page. To summarise, both applications were able to deal with unauthorised access correctly and were able to resist any method of gaining unauthorised access.

Performance Testing

Another non-functional requirement was performance testing. The test was to ensure that all three systems were able to output information quickly without any signs of sluggishness. The test was done in the following test environment :

- Device: Microsoft Surface Pro 2 running Windows 10 v1803
- CPU: i5-4300U, 1.9 GHz, Turbo up to 2.9 GHz, 3 MB cache
- Memory: 4GB Dual Channel DDR3
- Network: Wifi 802.11N
- Download Speed: 40Mbits/second
- Upload Speed: 10Mbits/second
- Ping: 10-20Ms
- Browser: Microsoft Edge

Testing the desktop application revealed a very small amount of sluggishness when clicking buttons that required the application to bring up a new window, the program initiating the new screen was most likely the cause of the small lag. The issue did not concern the user and therefore not an urgent issue to fix. Another small issue picked up during testing was the half a second lag that always occurs whenever the application was opening a new window consisting of a table view. This lag was due to the program connecting to the database, querying the database, retrieving the result from the database and finally outputting the data onto the table view. All of this happens within 0.5 seconds, therefore, the micro lag is inevitable in which during the testing, the tester did notice but wasn't too concerned. The desktop application was also monitored with Window's task manager and after testing through each feature of the app, I found no spikes of high usage in CPU and RAM usage averaging 120MB concluding that the app is efficient in RAM and CPU usage.

For the web application, the testing device was able to load the web page within half a second based on the internet speed that was available during testing. The server able to upload issues to the user's web browser instantaneously as if the data already existed on the device and was also able to process data from the user and give an appropriate response very quickly with no delays.

Finally, the DBMS system on Amazon's web server was consistently quick in dealing with incoming connections from both the web application and desktop application. As mentioned before, the desktop application was able to connect and retrieve data from the database within half a second which is very fast. Similarly, with the web application, the web application was able to check the login details and generate a response to the user based on the data found on the database thus concluding that the web application's connection to the database was able to transfer data instantaneously.

User acceptance testing

For the user acceptance testing, I asked 4 of my colleagues at the housing accommodation company I work with to test the program and to see if it meets the functional requirements that I have set for this project and the non-functional requirements of the project. Overall the feedback was positive, and some new features were mentioned to could further enhance some features. For instance, being able to filter a specific tenant from the tenant management controller or for builders to be able to add building materials brought for a property as an expense.

- Tester 1: “The desktop application was quick, and I was able to do things that I normally do on my spreadsheet much faster”
- Tester 2: “Was able to see an overview of the finances clearly and was also able to view and pay rents with ease but a feature I would like is to have something that tracks assets of a property that is now closed but still visible to the company”
- Tester 2: “Adding properties was very easy to do but something that could make it better is to have the ability to add maintenance issues for the new property that have issues existing from the previous owner”

Summary

Overall the application has passed all the tests with good results. There is still room for improvement where minor tweaks are needed to make the performance slightly better but it’s not something that is currently critical and will be looked after the project finishes. The testing did pick up minor bugs which were swiftly resolved by adding preventive measures that prevent the bug from reoccurring.

Evaluation and Conclusion

In this section, I will conclude the project and reflect upon the methods and decisions made to reach the completion of my project.

What I learnt and my achievements

Looking back at the project, I feel proud of what I have created. The ability to create effectively three systems that are communicating with each other is a huge achievement and is something that at first thought it was an ambitious feature. I am also proud of the huge number of features that I had originally planned for the project that was implemented into the project without any major assistance. In total 35 sub features were implemented and only a few features were not implemented due to time constraints. The knowledge gained from some of the computer science modules such as object-oriented programming, database and web programming module had a huge input in my project as it saved a significant amount of time since there was less time spent figuring out as to how to implement this feature and more time spent implementing the feature.

I have also learnt many factors during the project that aided me in the project. One developing feature I learnt that was useful was the use of external libraries. Originally before the project, I had only used the libraries that come with java so being able to use libraries from other developers other than java libraries allowed me to try out new features that would enhance my desktop application. For instance, being able to use the jfoenix library as stated earlier in the report, gave me more access to different UI elements that made the application look and feel better compared to using the standard java FX library and the openCSV library that allowed me to load CSV files and export data from it. Another tool was the use of hosted servers. Usually any application I create are hosted locally on my laptop, so to see my application become accessible anywhere by using hosted servers such as Amazon web servers and still be fully functioning as how I programmed it is a feat that I am proud off and is something that I know, can be applied to software engineering jobs .

Another skill gained was analytical thinking. Analytical thinking is the process of turning a problem into smaller manageable components and in the context of my project, the problem was set of functionalities required for my house management system. For this project to be executed successfully, I developed a time plan that entailed what the system should do by breaking up the system into a category of main features and sub-features. This relieved the process of thinking

about what features to implement next during development and allowed me to plan ahead better and create more effective contingency plans if an issue came up during implementation which as I will mention in the next section, issues did occur in the project.

Obstacles during the project

During the project, there were setbacks that meant that my original time plan had to be revised to accommodate the setback. Timing was a reoccurring obstacle throughout the project. During the first half of the time plan, there were many delays that meant that the time plan had to be pushed back more than two weeks to accommodate for other module course works and exams. Also, since I had a part-time job that I was committed with, that meant that I had less time during the week to work on my project and make progress. During the 3-week Christmas holiday, to try and get back on track with the time plan, I deployed a strategy that I created called the 1 feature per day where each day, I would aim to complete at least one sub-feature no matter how difficult or complex the feature is. Once the sub-feature had been completed, I can then have the option to enjoy the rest of my day in the holiday. This strategy allowed me to get back on track with the time plan and gave confidence and momentum in completing the project.

Another obstacle I faced was trying to have the web application and the database under open shift. Originally the plan was to use one server for both the web application and the database, therefore the desktop application would connect to OpenShift server to retrieve data from the database. However, at the time of trying to configure a connection from the desktop application to the database, there was no clear method of connecting to OpenShift server other than creating a temporary port that enables a connection but only temporary. Since I didn't want to delay the project in trying to get the desktop application to connect to OpenShift servers, I decided to simply use the development server that I have been using to host my MYSQL database but keep the web application on OpenShift since I already had it up and running and did not want to further delay the project.

What would I do different or change

Reflecting on the project. There are things that If I still had time, would have tried to implement into the application to enhance the functionality of the application or the visual aesthetics of both applications. These features are likely to be implemented in my free time after the final exam to create a fully replaceable working system for the housing company that I work for.

Image uploading and viewing

One of the features that I had to cancel due to time constraints was being able to upload an image to the server and be able to view the image from the desktop application and the web application. This would allow builders and company employers to have a better idea of the issue that has been reported by the tenant and thus quicken the response of the issue as it would allow the builder to bring the necessary equipment to resolve the issue.

Tennant Invoicing

Another feature that I would have preferred to implement in my project is the ability to generate invoices and automatically send the invoice to the tenant via email. This will quicken the whole process of end of month invoicing to tenants where companies without a dedicated system must create invoices manually and send it manually to each tenant. This feature would decrease the chances of human error and ultimately take significantly less time to complete

Better Rent alert system

Currently, the rent feature uses a subtle alert system that is shown if the user clicks on the pay rent button on a specific property. In the future, a feature that I want to implement is to make use of the home screen menu tiles where a tile would be added to show what property rents are due/overdue. This would allow users to immediately know/reminded what rents are due as soon as they log into the desktop application.

Splash screen and data loading animation

Throughout the desktop application, there are many times where the application must go through the process of retrieving data from the database where a connection is first established then a query is sent to the database in which the database will then process the query and respond with the data found from the

Figure 31 Example of a loading animation that animated in a loop

query. This causes the application to stutter as it is waiting for the data to come back from the database which may lead the user to think that the application is freezing and experiencing problems. A solution would be to have the desktop application to show a loading animation when the application is either processing data or retrieving data. This will notify the user that the system is processing data and will require the user to wait until the process has finished.

References

Amazon, n.d. *Tutorial: Create a Web Server and an Amazon RDS Database.* [Online]
Available at:
http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html

developer.mozilla, 2018. *Django Introduction.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

Developers.Google, 2016. *Python Introduction.* [Online]
Available at: https://developers.google.com/edu/python/introduction

Elman, J. & Lavin, M., 2014. *Lightweight Django.* s.l.:s.n.

GitHub, 2018. *GitHub.* [Online]
Available at: https://github.com/

Google, 2018. *Material Design.* [Online]
Available at: https://material.io/guidelines/material-design/introduction.html#

IBM, 2016. *IBM Knowledge Center.* [Online]
Available at:
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm

JFoenix, n.d. *Documentation.* [Online]
Available at: http://www.jfoenix.com/documentation.html

Mozilla, 2016. *JavaScript.* [Online]
Available at: https://developer.mozilla.org/bm/docs/Web/JavaScript

Occam, n.d. *Features benefits.* [Online]
Available at: http://www.occam.co.uk/features-benefits/

Oracle, n.d. *Java™ Platform, Standard Edition 8.* [Online]
Available at: https://docs.oracle.com/javase/8/docs/api/

Perry, J. S., 2010. *Introduction to Java programming, Part 1.* [Online]
Available at: https://www.ibm.com/developerworks/java/tutorials/j-introtojava1/

solution, n.d. *solution.* [Online]
Available at: http://www.localpad.co.uk/solution

Stephens, R., 2010. *Beginning Database Design Solutions.* s.l.:John Wiley & Sons, 2010.

Appendix

Test Plan

Test Case	Data and method used	Expected Result	Actual Result
Login Authentication	Test data 1: • Username: Musab • Password: eliter54 Test data 2: • Username: Musab • Password: abc123	Test 1: the Desktop application should grant access to the user and bring up the home screen Test 2: the desktop application should deny access and remain on the login screen	Test 1: Successful- Application detected correct login combination and a new screen consisting of the home page appeared. Test 2: Successful- Application detected incorrect login combination and remained on the login with an error message
Add New property and new landlord	Test data 1: • Door No: 101 • First line: Goldsmith • Post Code: E15 2YE • Town: Manor Park • Borough: Newham • Rate p/Day: 10.00 • Rent Cost: 1250.00 • Contract start: 27/02/18 • Contract end: 27/02/19 • Landlord Name: Tom • Sort Code:203010 • AccountNo:62837954 • Select Landlord Top • Select Area Manager Jack Test data 2: • Door No: 101 • First line: Goldsmith • Post Code: E15 2YE • Town: Manor Park • Borough: Newham • Rate p/Day: abc • Rent Cost: 1250.ab • Contract start: 27/02/18 • Contract end: 27/02/19 • Landlord Name: Tom	Test 1: The desktop Application should successfully add a new landlord and appear in the landlord drop down box for selection, the app should then insert the new property into the database and appear on the table view on the main screen. Test 2: The application should clear any data that is invalid when appropriate thus leading to no property being added to the system	Test 1: application successfully added the new landlord and was able to select the newly added landlord from the drop-down box, the application also added the new property successfully and appeared on the table view immediately. Test 2: the application successfully rejected all the invalid data by clearing the text box whenever the wrong type of data was inputted and since text boxes became empty, the application did not allow me to add a new property

	• Sort Code:20as • AccountNo: abc • Select Landlord Top • Select Area Manager Jack		
Return a property	Return 10a Wally Road	The desktop app should update the property with the new contract end date which will be set as today's date. The property should also disappear from the table view	The test was successful, and the database showed the new contract end date that was the same as today. Also, the property disappeared from the table view.
Delete a property from the system	Delete the property 101 Goldsmith	The desktop app should delete the property from the database and should disappear from the table view	The test was successful as the selected property for deletion was removed from the table view meaning that it was deleted from the system
Pay rent to a property	Test 1 • Select Wally road • Deduction: 500.00 • Date: 2018-04-07 Test 2: • Select Wally Road • Deduction: abc • Date: 2018-04-07	Test 1: the application should recalculate the total cost and once the rent has been inserted, the rent should appear in the table view Test 2: The application should clear the invalid data	Test 1: The application successfully added the new rent record and it appeared on the table view Test 2: The application successfully cleared the invalid data as soon as the deduction text box became inactive and resetting the rent total.
Add new Tennant	Test 1: • First Name: Adam • Last Name: Blarg • Lives At: High Street • D.O.B: 1981-05-20 • Moved in: 2018-04-07 Test 2: • First Name: 1234 • Last Name: 4567 • Lives At: High Street • D.O.B: 1981-05-20 • Moved in: 2018-04-07	Test 1: Application should insert the new tenant into the database and the new tenant should appear in the table view Test 2: Application should clear any invalid data to refuse to add tenant record that contains empty details	Test1: Application successfully inserted new tenant into DB and appeared in the table view Test 2: Application successfully cleared any invalid data and gave an error message if any details still had blank spaces.

Relocate Tennant	Move Jerry Adams to 101 goldsmith	The application should successfully set the tenant "Lives at" key to the ID of 101 goldsmith and should appear in the table view	Application successfully changed the lives at the field of the specified tenant and updated the table view to reflect that
Add a new expense	Test 1: • Expense Name: House Taps • Cost: 20.50 • Quantity: 2 • Category: Building Material • Property: Goldsmith • Date Paid: 27/10/18 Test 2: • Expense Name: House Taps • Cost: abcx • Quantity: xyz • Category: Building Material • Property: Goldsmith • Date Paid: 27/10/18	Test 1: The application should successfully add the new expense and should immediately appear in the table view Test 2: The application should refuse to add the new expense and an error message should appear to make user know of the error	Test 1: Successful, application added the new expense and it appeared on the table view Test 2: Successful, application denied the user from adding the invalid data by clearing any invalid fields and causing the program to not accept empty fields
Set the utilities of a new property	Test 1: • Select Electric for edit • Company: EDF Energy • Cost/Month: 100.00 • Account No: acb123 Test 2: • Select Electric for edit • Company: EDF Energy • Cost/Month: abc123 • Account No: acb123	Test 1: Application should set the details of the electric utility type and the changes should reflect in the table view Test 2: Application should refuse to set the electric utility type	Test 1: Application successfully edited the electric utility type and the table view automatically updated to reflect the change Test 2: Application failed to stop the invalid data and produced a number format exception Solution: Added a function that listens to each of the boxes and if any invalid characters are detected that will clear the text box, also added a function that checks if all the necessary fields have been filled.
Add a new issue (desktop app)	Test 1: • Select property: Goldsmith • Date Reported: 2018-04-08	Test 1: Application should add the new issue to the database and update the	Test 1: Application successfully added the new issue and the table view

	• Priority: Select High • Reported By: Musab Adam • Description: Kitchen Taps leaking water	table view with the newly added issue	updated automatically with the newly added issue
Initiate a job	• Builder: Select Tom Blarg • Job Started: 2018-04-08 • Comment: Need to replace the door	The application should add a new job that links to the selected issue and the issue should disappear from the open jobs table view and appear onto the ongoing jobs table view	Application successfully initiated the job and disappeared from the open jobs and appeared into the ongoing jobs.
Report an Issue (Web app)	Test 1 • Username: JerryAdams • Password: Adams19760312 • Report Issue: Toilet Door broken	The web application should show a success message and redirect back to the index page where the newly reported issue should appear, the issue should also appear in the desktop application under open jobs	The test was successful, the issue was seen on the index page as well as the open jobs table view in the desktop application.
Complete an issue (Web app)	Test 1 • Username: tomblarg • Password: password • Report Issue: Toilet Door broken	The web application should show the job that has been assigned to tom blarg and when the complete job is clicked, the assigned job should disappear from the index page as well as be visible under the completed jobs table view in the desktop application.	The test was a success, the assigned job appeared on the builder's home page and was able to mark the job as completed as well adding a comment which was visible from the completed job table view in the desktop application.