FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE

IN

Software Engineering

PROJECT REPORT

Name: Raphael Harris

ID Number: K1423138

Project Title: Property Portfolio Management System: Self-Manage Pro

Project Type: Design

Date: 29/04/2019

Supervisor: Professor Christos Politis

Kingston University London

ACKNOWLEDGEMENT

I would like to thank my parents for always supporting me throughout my time at University

PLAGIARISM DECLARATION

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date 29/04/2019

Signature

ABSTRACT

This project aims to solve the problems associated with self-managing a property portfolio as a private landlord. The project documents from start to finish, an industry-grade application which serves as an interface between tenants, landlords and their contractors; for reporting and tracking maintenance tasks. Built using Java Enterprise Edition and a MySQL database.

TABLE OF CONTENTS

1 INTRODUCTION & LITERATURE REVIEW

1.1 BACKGROUND

The problem I aim to solve with software proposed in this report is one that I personally experience as a tenant. Legislation, work overload and people involved with managing a property portfolio cause the handling of maintenance issues to be disorganised and creates unnecessary distress for both tenants and landlords alike. It is commonplace for maintenance issues to be ignored, often unbeknownst to the landlord; whether it be due to poor management systems, or poor staff, the tenants needs go without attention.

I will design and build a sophisticated ticketing system connecting landlords, tenants and contractors via an online portal, through which landlords will be able to easily register their properties; landlords will be able to find their contractors and assign them to specific properties; tenants will be able to register where they live and log maintenance issues; maintenance jobs will be sent to the landlord for review at which point contractors will have the final say about work being carried out before the job is completed, after which, the status of the job will be updated and any relevant notes will be appended to the job by either Contractor or Landlord.

1.2 MOTIVATION AND PROBLEMS

Legislation problem

According to the Telegraph[1], The heavy legislation of today's private rented sector means being a truly DIY landlord is now difficult to achieve. Between quarter four of 2016 and the first quarter of 2017, the number of landlords using a letting agent to manage their properties rose by 7% whilst at the same time, the number of self-managed properties has fallen by 10%, shown in a study carried out by the National Landlords Association (RentPro, 2017). This is a significant amount of business as letting agents will typically charge anywhere between 5 and 15 percent of rental income for managing services. [3]

As of November 2016, landlords may only self-manage their properties if they are registered and have been trained by approved bodies on up-to-date information to manage tenancies within the law. If a landlord wishes to self-manage a property they must obtain a license to run a large house in multiple occupation, or HMO.

Further to this, numerus standards and certifications must be adhered to, these include[4]

- Energy Efficiency requirements
- Gas safety certificate
- Electrical safety
- Fire safety
- Tenancy Deposit Protection
- Maintaining a safe and clean environment

Conflicting Economic Objectives

In addition to the legislation problem, according to parliamentary figures as of the 19th March 2019, 204 out of 650 representatives of the House of Commons (Almost 32%) have registered their vested interests in property.[5] This is a disproportionately large figure and creates an uneven power dynamic between tenants and their landlords when considering that those who pass the laws are also landlords with conflicting economic objectives. It is financially favourable for politicians to pass laws which strip tenants of their rights and enable landlords to do the bare minimum where maintenance is concerned. For example, a tenant in a fixed-term contract with no break clause may not vacate their tenancy early unless the landlord agrees, regardless of how the tenant has been treated for the duration. It is not uncommon for tenants, students especially, to have their maintenance issues ignored by landlords who know that there is no recourse for their negligence.

People problem

The Weakest point of any system is people, this goes not only for information security, but importantly, system functionality. People are not perfect, we are all flawed by nature and this fact of life, transposed onto the business of managing a property portfolio can cause important issues to fall through the cracks. A transparent system relates contractually to both the obligations of tenant and landlord. Tenants are also not perfect and the problem works both ways.

More and more landlords are turning to managing agents to handle the business end of their portfolio as an easy solution. This new influx of business and a general lack of preparedness for it by the managing agents is creating issues for tenants where, speaking from experience, important jobs reported can go unattended for weeks. There's numerous reasons why managing agents might not handle properties to the fullest of their abilities but they ultimately boil down to:

- Managing agents not hiring enough staff to cope with all aspects of maintenance
- Process is not fully automated and never can be due to the nature of the problem

The second part of the people problem is that landlords assume hiring a managing agent for their property means handing over complete control. In actuality, this is not the case as often, landlords will demand all maintenance issues are moderated by themselves once they realise how expensive it can be to give complete control to the agents [6]. This adds another layer of bureaucracy to an already arduous process.

1.3 AIMS AND OBJECTIVES

To create a database-driven ticket system for use by Tenants, Landlords and Contractors alike. Landlords will have information about their property portfolio in the database, including information about their contractors. Contractors will log in and be able to see any maintenance jobs allocated to them, they will then be able to accept or decline the job, add any notes and update the job status. Tenants will be able to log in and see any open jobs for their property as well as being able to create new jobs and assign contractors. Administrators should be able to create and delete landlords, tenants, contractors, properties and jobs.

This has been surmised to a list of aims and objectives:

1.3.1 Aims

- Create a website that is both responsive and intuitive
- Allow landlords to manage their day-to-day property portfolio maintenance tasks in a paperless system
- Relieve some of the workload from an very paperwork intensive system
- Let contractors easily work with landlords and tenants
- Keep tenants informed about work being carried out at their property

1.3.2 Objectives

- Create a secure login for clients
- Build a database for storing information about Landlords, Tenants, Contractors, Jobs and Properties
- Link landlords and tenants to their properties
- Deliver different content depending on the user type
- Link contractors only to properties they are assigned to
- Build a ticket centre where users can see open and closed jobs for linked properties
- Build an interface where landlords and tenants can create new tickets
- Allow users to add notes to jobs, and consequently view them

1.4 SCOPE

The scope of this project is primarily focused on making self-managing a property portfolio for landlords a more attractive prospect and building a system which encourages this, convincing them to leave behind the old ways of blindly trusting managing agents. The secondary focus is on ensuring that the needs of tenants are catered to, allowing for the easy logging of maintenance jobs and viewing their status and any supporting notes.

Contractors are not included in the considerations for this project, they are not the target audience, and however, they do stand to gain from the system since managing workload will become an easier task. It is expected that if adoption of the proposed system occurred amongst landlords and tenants, contractors would create accounts merely for the practicality of finding work.

The system will be designed for desktop and laptop computers however should be responsive and scalable to mobile phones, tablets and other devices of any size. The stakeholders for this project are private landlords, tenants & contractors (positively impacted) & managing / estate agents (negatively impacted).

1.5 METHODOLOGY

In this section, a number of different methodologies for software development are considered, with some advantages and disadvantages of each listed. Then, my chosen methodology for this project is given along with justification for my choices.

1.5.1 Waterfall

The waterfall methodology is a simple software development cycle which involves Analysis, Design, Development, Testing and Feedback/ Maintenance. It is a defined as a cycle since the feedback and maintenance step loop around to the analysis phase and is repeated until the customer is satisfied.

Its advantages include that it is simple and easy to use, deliverables are straightforward and development is linear. It is easy to order tasks and it works well for smaller projects.

The disadvantages are numerous however. There is no minimum viable product until late stages of development. As the waterfall model is linear, which can be seen as an advantage, it also means that there is no overlap between stages such as design and implementation, a way of thinking which promotes the view of project management when a far more appropriate modern idea is that of product management, where design and development co-inside with each-other and support for the product carried on beyond deployment.

1.5.2 Agile Kanban

Kanban is a Japanese system for project management, it promotes transparency of work and individual team-member skill. All aspects of development for a project are displayed on a team board where all developers / team members can see the status of individual tasks at any time.

Tools exist to aid this methodology such as Trello, an online project board user interface, making Kanban an attractive choice.

A common disadvantage of Kanban development is that issues can arise from outdated boards where tasks have not been updated as they should be by all tem members. Another common occurrence is task cards over-complicating issues when they ought to be split into a set of simpler tasks.

1.5.3 Agile Scrum

Team members are assigned one of numerous roles including product owner, testers, designers and user experience specialists. All development tasks are aggregated into a list called the product backlog, these are then prioritised and split into time-period iterations called Sprints (usually two weeks). At the start of each sprint, tasks are allocated to different team members based on ability and role within the team. At the end of a sprint, the team should have a number of deliverables which can be shown to the product owner who represents the stakeholders. Any unfinished tasks from the sprint are moved back to the backlog and the priority in relation to the project is reassessed.

Advantages include testing being done during Sprints meaning finished deliverables work as intended, and incremental delivery means a minimum viable product is produced faster than in other methodologies. Downsides include methodology adoption difficulties for large teams and daily meetings can undermine the development process.

1.5.4 Chosen Methodology and Justification

Of the three methodologies above, I have chosen to disregard waterfall due to the lack of overlap between design and implementation in favour of methodologies which are more tolerant of changes to design during development iterations.

I have chosen to cherry pick certain aspects from Kanban and Scrum. For example, I will be monitoring and tracking development progress throughout the project on a Trello board just as in the Kanban methodology, and will couple this with Sprints and a product backlog from Scrum development. Both

methodologies are very team driven, however I believe that picking techniques from each in this way will enable me to develop the proposed system effectively.

I will also be using the Incremental Design principal from Extreme programming, another Agile Framework. Incremental design allows for developers to start making a system before they have all the specifics, meaning changes to the design can be made midway through Sprints.

1.6 LITERATURE REVIEW

1.6.1 Consumer Satisfaction and System Efficiency

Karunasena, Vijerathne & Muthmala (Facilities; Bradford, 2018) found that, according to one paper they reviewed, tenant satisfaction is conceptualized as interaction-specific means of individual experience. If this is indeed the case, then it is difficult to quantitatively measure how satisfied customers are using statistics and figures. They went on to write about another paper which stated overall satisfaction is not to do with an individual interaction, rather, a cumulative culmination of services received and their product. Definitions of customer satisfaction must also take into account cost and profit analysis. A tenant being treated neglectfully and being overcharged will lead to a low overall satisfaction versus one whose needs are met and feels as if they got a good deal out of an arrangement.

It is important to satisfy customers in order to gain a good share of the market. The result of satisfying customers more than one's competitors is that consumers will develop an increasing psychological commitment to an organisation and its products. (Birks & Southan 1992) In this case, a tenant will be more likely to renew a tenancy at the end of their lease if they have been sufficiently satisfied and believe that they get a service and satisfaction level comparable to that which they would receive from a property under a managing agent. According to one study on 5,000 participants, a tenant is almost four times more likely to not renew their lease if they receive poor service. (Sanderson, D 2018)

It is, for the proposed system, important to satisfy the needs of tenants; but also those of landlords to inspire post-purchase occurrences for example shift in attitude towards self-managing properties, repeat purchase and brand loyalty (Birks & Southan 1992). A good metric for measuring satisfaction of landlords who chose to use a managing/estate agent is the level of service the agent provides, and thus, the amount of repeat purchase they inspire in tenants. The level of service received can be quantified as the number of completed jobs carried out during a tenancy as a percentage of the total number reported. Qualitatively, the completion time and scale of jobs is also considered when assessing overall satisfaction.

Having a system for reporting maintenance jobs increases convenience to not only the tenant but also the landlord and contractor. Often with private tenancies, to have an issue attended there must be a back and forth of phone calls and emails. For larger problems this can be overbearing, dealing with the stress of the issue at hand whilst simultaneously having to check all communication mediums for any update. Convenience can be perceived as the state of being able to proceed with something without difficulty, and consumers measure this by products and services which are easily accessible, time & energy- saving and frustration reducing (Rahman, Khan & Parisa 2014). By that definition, the current system is inconvenient, makes tenants unlikely to report issues, and is due for replacement. My system will offer a solution which displays everything a user may want to know, with an easy-to-read layout which combining information, simplicity and convenience in one package.

Along with keeping track of communications, there is a large amount of paperwork that comes with managing properties. Phone calls must be written down; for larger portfolios, diaries of all appointments must be kept to ensure nothing falls through the cracks, and written logs must be kept for every task. This is not always done, whether it be due to poor communication or lack of motivation. Property management system, paperless or otherwise, should be transparent. A computerised system such as this leaves very little margin for making errors since all tickets will be updated with notes comprising of the user who left the note and a description of each occurrence. If maintenance issues are left unattended, the reason will be clear from looking at the ticket, who is at fault and why the fault occurred.

1.6.2 Existing Systems

I have experienced first-hand the negligence of managing agents and have identified the need for a product such as this in the industry. Although I arrived at this idea by myself, I am not the first person to come up with similar ideas. This section serves to review the existing systems, where they succeed, their shortcomings and how the proposed system differs from these.

Re-Leased

Re-Leased is a complete solution for managing an entire property portfolio. Their system boasts "Accounts, Lease Management, Inspections & Maintenance, Compliance Management, Tenant Communication, Document Storage and Reporting" (re-leased.com, 2015). On the surface, they are a very attractive looking company offering a web-client as well as Android and iOS apps with a clean colour scheme and good layout.

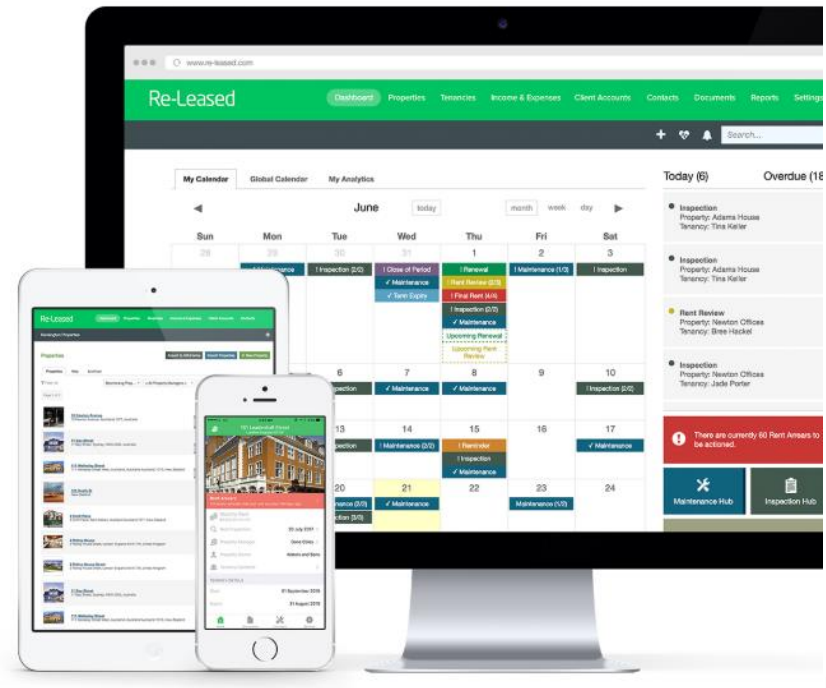


Figure 1 Re-Leased Graphic

There is though, a lot to digest when looking at the screen. It is well known that users prefer a simple user interface, and whilst Re-lease is visually attractive, in my opinion there is too much information on the screen at one time.

Their target audience is landlords and managing agents with vast commercial property portfolios. This differs from my system as I am primarily trying to capture the business of private, self-managing landlords.

Re-leased is an extremely comprehensive solution with vast expanse, however it offers a lot of unnecessary features considering the goal my system is trying to accomplish.

Rentman

Rentman is a self-management solution for the private sector. Their target audience is private property managing and letting agencies. Contractor and maintenance management is only a small part of what they offer as their services extend to property admin, marketing and finding new tenants, a 'Things to do' section for managing legislation compliance such as scheduling Gas and PAT safety certificates, Deposit protection and more (Rman.co.uk, 2019).

On the surface, Rentman is not attractive. They use an outdated user-interface, which eclipses the array of functionality they provide and dissuades potential buyers from purchasing the system.

Their system differs from mine in that they are not trying to captivate smaller, private self-managing landlords, or tenants; their system favours function over form whereas mine focuses on user experience, securing repeat business and removing stress from day-to-day tasks.

The systems I have reviewed both offer extremely comprehensive solutions for managing a property portfolio, although they sacrifice user experience. Further, their target audiences are slightly different from mine, aiming to secure the business of managing agents and large commercial portfolio holders. The system I am proposing will have far less features than the solutions available, however a lot of consideration will be put into the design for the features I will be providing; maximising user experience and likelihood of tenancy renewal to the benefit of the landlord.

1.6.3 Relevant Technologies

Zendesk

Zendesk is an all-in-one solution for the management of support-tickets. Their primary focus is allowing companies to track issues in a customisable way. Their software provides the boilerplate to set up a support-ticket system for any scenario, with easily customisable configuration options, Zendesk can be made relevant to a wide variety of industries (Zendesk, 2019).

A ticket system such as this could be utilized to perform the goal of my system, however, their out-the-box solution works better for providing immediate support and answering customer service related enquiries. It would be difficult to implement a solution which not only handles tickets between landlords and tenants but also interfaces with contractors and allows all three user types to interact in one location.

Autotask

Autotask is a piece of software made by Datto, a child company of Vista Equity Partners. It provides a ticket system API with integrated billing support supporting multiple currencies and time zones. Autotask has different billing and ticket codes relating to a number of possible scenarios which are easy to implement. Further, Autotask supports different asset and account types, automated emails and invoice generation (Datto Inc, 2018)

Despite its clear practical application for the purposes of Self-Manage pro, using Autotask would be excessive. The functionality of the API extends far beyond the scope this project requires and is much better suited for use in large scale company internal ticket management solutions; not only because of this but also due to its big business price.

1.7 SWOT ANALYSIS

SWOT analysis is a technique used industry wide for analysing the strengths and weaknesses of a project before it is undertaken and is particularly useful for planning development patterns as well as determining risk before making mission critical decisions. It is not just used in software projects however, the SWOT methodology can also be applied to teams, businesses and organisations. (Parsons, 2018) The letters in SWOT stand for:

- Strengths
- Weaknesses
- Opportunities
- Threats

SWOT is not for determining how to carry out certain tasks, rather, it is for deciding whether or not something should be carried out. SWOT matches internal strengths with external opportunities at the same time as determining internal exploitable weaknesses and external threats, usually beyond the scope of control of the project.

Figure 2 SWOT Analysis

Internal strengths and weaknesses are usually within the scope of control for the project and the outcomes can be changes. External opportunities cannot necessarily be controlled however they can be managed and predicted. The points outlined in the SWOT analysis above lay out the foundations of considerations which must be taken into account and reflected upon throughout development. It is most important to address the threats and weaknesses of the system to counter them as best as possible. SWOT is a vital form of analysis to perform in order to better understand the target market and competitors (Contributor, 2015).

1.8 PESTEL Analysis

PESTEL analysis is a tool similar to SWOT except instead of analysing strengths and weaknesses at product level; it focuses on the Political, Economic, Social, Technological, Environmental and Legal considerations of a project, the bigger picture aspects which influence decisions, businesses and the market as a whole (Mindtools.com, 2018). These factors are beyond the control of the project but can be identified and risks arisen therefrom can be accordingly managed.

Political	**Economic**	**Social**
Almost a third (31.8%) of the House of Commons representatives have vested interest in property. Threat of Brexit is likely to make the pound weak There are a lot of certificates and paperwork required to conform to legislation	Brexit is predicted to cause a housing crash Managing agents have a monopoly making the market difficult to enter Skill level of workers could vary massively due to self-employed nature Austerity economics make cheap solutions desirable Landlords may expand their portfolio as time goes on EU housing laws are ever changing but will eventually be made redundant	Some properties will have multiple landlords or stakeholders Brexit housing crash will see an influx of new buyers Cost of labour will differ depending on the contractor and their vocation Cultural norm is to have a managing agent handle a portfolio New techniques make carrying out jobs faster, cheaper and more efficient

Technological	Environmental	Legal
Online world is ever-changing and people desire modern solutions New technology has conditioned people to want instant and responsive applications Hackers are constantly adapting and utilising new techniques to penetrate systems and steal confidential data	Recent threat of global warming has inspired many to fight against climate change Online systems are paperless and damage the environment considerably less Online scheduling will allow contractors to spend less time in traffic at peak times	Gas safety, PAT, Fire Safety and Electrical Safety Certificates must be obtained frequently Landlords must have multiple types of insurance which need yearly renewal and cause a lot of paperwork Rental abodes must meet minimum acceptable living standards including clean property, secure locks on external doors, running water, hot water and heating control

Figure 3 PESTEL Analysis

According to Mark Carney, the governor of the Bank of England, in the event of a worst-case scenario Brexit, house prices could be seen to plummet as much as 35% over three years (BBC News, 2018). This would have a direct effect on the project since the market would see an influx of new buyers whom couldn't previously get on the property ladder. At the same time as this, the pound sterling may also devalue (BBC News, 2019) since the referendum in 2016 saw it take a sharp decline. This will have social and economic impact on contractors, managing agents, landlords and tenants alike. The declining economy may be a good thing for the project however, since landlords will be looking for cheaper alternatives to managing agents.

1.9 Epic Stories

An epic story is a large requirement specifying a range of functionalities which achieve a common goal, captured from the view of the stakeholder. These epics are then broken into smaller user stories which can be used as a metric for progress towards project completion and are also split into pieces of singular, achievable functionality. These functionalities / task are then arranged into a list called the product backlog, which is used to decide tasks for each sprint.

1.9.1 Epic Story 1

"As a User, I only want to see content relevant to my account"

1.9.2 Epic Story 2

"As a Landlord I want a system I can use for managing my property portfolio"

1.9.3 Epic Story 3

"As a Tenant, I want a hassle free way to report and track maintenance jobs"

1.9.4 Epic Story 4

“As a Contractor, I want all my work to be accessible in one place”

1.9.5 Epic Story 5

“As a Landlord I want to see a summary of all tenants and maintenance jobs at my properties”

1.10 User Stories

With the Epic Stories defined, these are split into user stories as follows:

1. As a user I want to log in and see info relevant to me
2. As a tenant I want to see all tickets for my home
3. As a landlord I want to see all tickets for my portfolio sorted by property
4. As a contractor I want to see all tickets assigned to me
5. As a tenant I want to be able to create new tickets when a job needs doing
6. As a landlord I want to create tickets for properties I own
7. As a contractor I want to be able to accept/decline Jobs assigned to me
8. As a landlord I want to sign off on jobs before work is done
9. As a tenant I want to see what’s going on with jobs at my property
10. As a tenant I want to see reasons for jobs being declined

1.11 PRODUCT BACKLOG

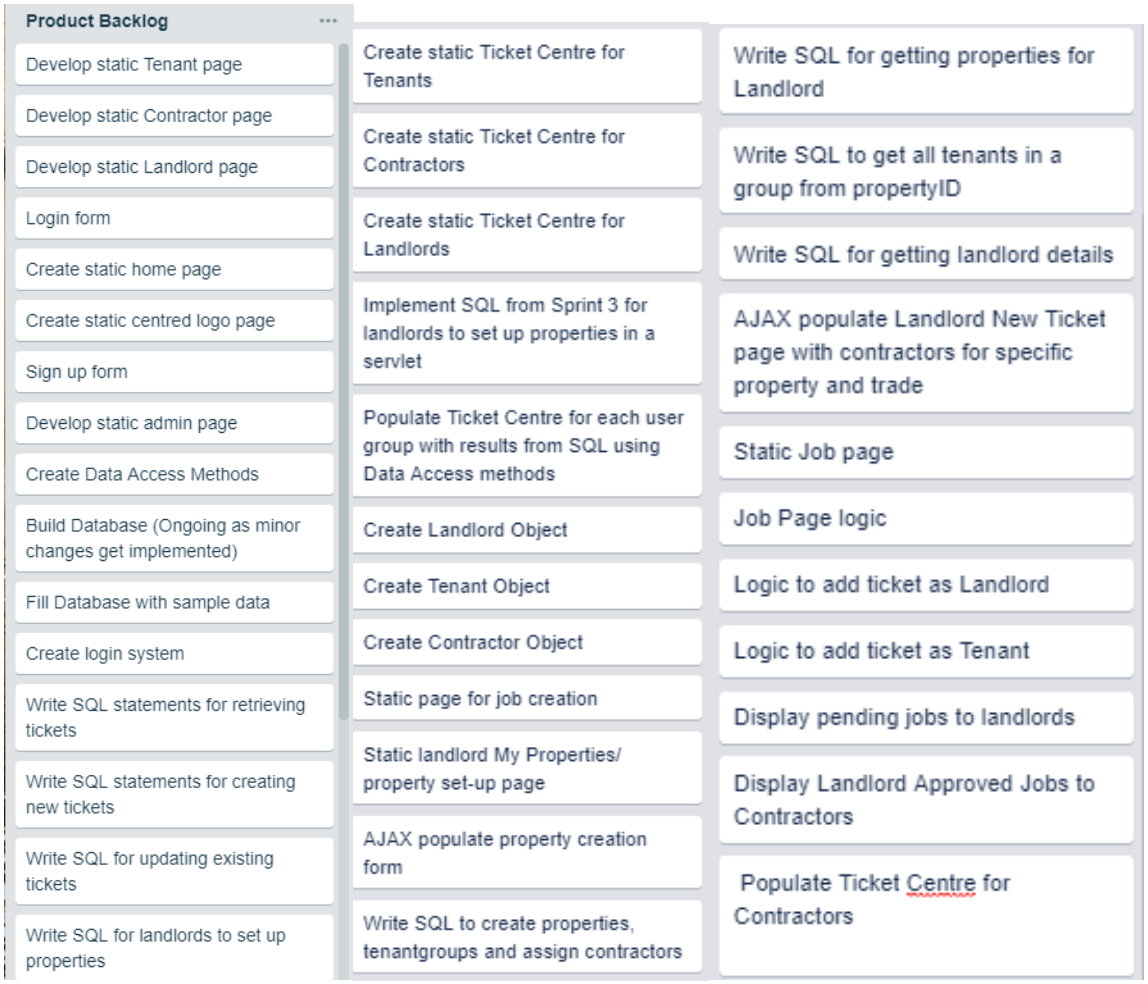


Figure 4 Product Backlog

The screenshots above show all tasks inside the product backlog, derived from the stories in the previous section, they make up all the main tasks which need to be completed to fulfil the epic stories.

1.12 REQUIREMENTS ANALYSIS

Defining requirements is an important part of any project, it should be done at the beginning to ensure the team has a clear vision and will guide the entire development process. A project without requirements has no measurable goals and is doomed from the start. It helps to first split the requirements list into two main categories, functional and non-functional. A functional requirement denotes what the system should do whereas a non-functional requirement specifies how the system should it (Eriksson, 2012).

1.12.1 Functional Requirements

- Store Tenant Details
- Store Contractor Details
- Store Landlord and Property Details
- Provide Login area for clients
- Link landlords and tenants to relevant properties
- Distinguish between user type and display the correct content

- Securely store passwords
- Link contractors only to properties they are assigned to
- Allow tenants to report Jobs for their property
- Allow landlords to report jobs for any of their properties
- Allow Contractors to view any jobs assigned to them
- Assign different statuses to jobs
- Close / reject tickets at any time as a landlord/contractor
- View details of the assigned contractor for a given job as a tenant
- View details of tenant and property for a given ticket as a contractor
- Search page for landlords to add new contractors to a property
- View details of Properties and active jobs as landlord
- Navigation bar which differentiates between user type and displays correct areas

1.12.2 Non-Functional Requirements

- Require Landlords AND Contractors to sign off on jobs before work can be carried out
- Add notes to tickets which can be viewed by any user with permission
- Intuitive UI, simply laid out and easy to use
- Responsive design (support resizing and interactive elements)
- Modal sign in minimising page loads
- Rating system for contractors
- Home Page encouraging new users to sign up
- Ensure login is secure

1.12.3 MoSCoW Method

MoSCoW is a technique which can be used to prioritise these requirements and is particularly helpful when planning sprints and the order of development for the project. MoSCoW can be split into four ranks of requirement which makes for faster planning and a more successful product; these are:

- Mo - Must have
- S – Should have
- Co – Could have
- W – Won't have

The problem with MoSCoW analysis, as pointed out in the 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (Popli, Sharma & Chauhan, 2014) , is that splitting the requirements into four categories such as this can often be a difficult task and is not as straightforward as it seems since there is often overlap in requirements.

Mo	**Must have** - System will not function without it (Minimum Viable Product)
S	**Should have** – Important for system function, but not vital for MVP
Co	**Could have** – Desirable, but not as important as should haves
W	**Won't –and would have** – Won't include for release, nice to have but have no impact

FR NO	Functional Requirements	MoSCoW
1	Store Tenant Details	Mo

2	Store Contractor Details	Mo
3	Store Landlord and Property Details	Mo
4	Provide Login area for clients	Mo
5	Salt and hash stored passwords	Mo
6	Link landlords and tenants to relevant properties	Mo
7	Distinguish between user type and display the correct content	Mo
7	View details of Properties and active jobs as landlord	Mo
8	Link contractors only to properties they are assigned to	Mo
9	Close / reject tickets at any time as a landlord/contractor	Mo
10	View details of tenant and property for a given ticket as a contractor	Mo
11	Allow Contractors to view any jobs assigned to them	Mo
18	Allow tenants to report Jobs for their properties	Mo
12	Assign different statuses to jobs	S
13	Search page for landlords to add new contractors to a property	S
14	Allow landlords to report jobs for any of their properties	S
15	Navigation bar which differentiates between user type and displays correct areas	S
16	Securely store passwords	S
17	View details of the assigned contractor for a given job as a tenant	Co
19	Allow tenants to rate a contractor and view ratings	W

NFR NO	Non-Functional Requirements	MoSCoW
1	Add notes to tickets which can be viewed by any user with permission	Mo
2	Intuitive UI, simply laid out and easy to use	Mo
3	Ensure login is secure	Mo
4	Colour scheme should be consistent	S
5	Modal sign in minimising page loads	S
6	Require Landlords AND Contractors to sign off on jobs before work can be carried out	S
7	Responsive design (support resizing and interactive elements)	Co
8	Rating system for contractors	W
9	Home Page encouraging new users to sign up	W

1.13 SECURITY ANALYSIS

According to a study by the Ponemon Institute (IBM, 2018), the average cost of a data breach on the global scale is $3.86 million, with a 27.9% likelihood of another breach occurring within the next two years. It is, therefore, more important than ever to protect the business and privacy interests of software systems and their users. This section addresses the two main security concerns when building a database driven application, what they are and suitable controls.

1.13.1 Password Storage

Storing any data in plaintext is not secure. A search on CVEDetails.com reveals that there are currently 250 known vulnerabilities for phpMyAdmin (cvedetails.com, 2019), the software being used to provide the access and make manual changes to the database. It is extremely common, although ill advised, for users to share the same password across multiple sites thus, a data breach containing plaintext passwords could be extremely damaging for customers. If the password unlocks other accounts such as an email address, the user could find themselves the victim of identity theft, fraud and much more. To that end, measures should be taken to encrypt sensitive information.

To protect passwords, they will be salted and hashed before storage; the process of appending a randomly generated number (salt) to the password and applying a message digest algorithm to produce a hexadecimal representation of the bytes making up the salted password. The hash value will be stored in the database along with the salt which was used to generate it. When a user logs in, the salt will be retrieved from the database and then combined with the user-entered password, hashed and compared to the stored hash value. If the values are the same, the user will be authenticated and allowed access to the system.

It could be argued that in the interests of security, other sensitive user data such as email addresses and security question answers should be salted and hashed as well. Retrieving data, hashing and making comparisons for more than just the passwords would have large resource requirements and effect the performance and possibly the availability of the system.

1.13.2 SQL Injection

SQL Injection is the process of inserting malicious SQL code into unprotected form input boxes on a webpage. SQL injection was first publicly disclosed in 1998 and easy implementations exist to prevent this form of attack from being successful, yet in 2017 it was the second most common type of cyber-attack; 21.6% of all attacks (Positive Technologies, 2018).

SQL injection is performed by manipulating the way statements receive variables through user input. A statement is concatenated with variables and then executed to perform a function. For example, a sign up form with input boxes for username and password, when submitted would form the following SQL:

`INSERT INTO USERS username, password VALUES` + *username Input* + *password Input*;

This could be manipulated if a malicious actor, instead of entering a password entered:

`"password; DROP TABLE USERS"`

The result of this would be:

`INSERT INTO USERS username, password VALUES` + *username Input* + `password; DROP TABLE USERS;`

Because of the way statements are constructed, the pre-built query when combined with the actors input forms two statements instead of one. The first will add a new user to the database and the following query would drop the entire Users table. SQL injection is not limited to drop statements, any SQL can be injected to perform much more sophisticated operations.

Fortunately, PreparedStatement objects can be used in java as a control for SQL Injection attacks. Instead of parameters, statements are constructed using “?” and then one of the objects setter methods is called to set the value and datatype required.

1.14 Use Case Diagram

A use case diagram is a device for showing how the end user should use a system to accomplish a goal. Use cases help to view complicated systems in easily readable terms, splitting a system into its core functionality and showing the relationship between functionality and user groups.

Shown below is the use case diagram for the proposed system.

Figure 5 Use Case Diagram

1.14.1 Signing Up

As can be seen from the diagram, when not signed in, a user will be presented with the home page if they are not signed in. They will have the option from there to either sign up or log in. If a user is signing up, they will first be asked if they are a Contractor, Tenant or Landlord. All three user types enter their email address, password, first name, last name, telephone number and date of birth. Additionally, if the user is a Landlord or Contractor, they will be asked for their address; if the user is a Contractor, they will also select all the trades for which they are qualified from a list. They can then submit the form, will be redirected back to the homepage where they can log in.

1.14.2 Creating a Property

When a landlord logs in for the first time and navigates to View Properties, they will be greeted with a message inviting them to set up a new property. If they have previously created properties they can navigate through to the ticket centre for each individual property in their portfolio.

When creating a property, the user will first be asked for details such as type of property, number of bedrooms and its address. They will then enter the email addresses of their tenant and select them from a dropdown to add them to the property. Finally, they will input the email addresses of their contractors who will have already created accounts, they will be selected from a dropdown and added to the property. The property will then be created and show up in the View Properties section.

1.14.3 Creating a Ticket

As a Landlord or Tenant, from the ticket centre for a property, tickets will be created by pressing a button at the bottom of the page to navigate through to the ticket creation form. From there, tenants will select the trade required and enter a description about the maintenance required. If the user is a landlord, they will be prompted to select a contractor of the relevant trade from their list of contractors which they added when they signed up. Pressing Create will redirect the user to the Job View page for the ticket in question. Contractors cannot create new tickets, only view and update existing ones.

1.14.4 Viewing Individual Tickets

Tenants and Contractors after signing in, will navigate to Ticket Centre where they will be displayed a list of all open and closed tickets relevant to them. They will then be able to click on the job ID for each ticket and navigate through to the View Job page which will display information about the property, the assigned contractor, a description of the job, its status and any notes which have been added. As previously mentioned, Landlords navigate to the ticket centre for individual properties through the View Properties section rather than straight from the homepage.

1.14.5 Updating Tickets

Tenant created tickets will require approval first from the landlord, and then from the contractor who is assigned to the job. This will be done in the same way as appending notes and updating the job status. From the Job View page, the user will press a button and a pop-up will appear prompting for a note to be added. If the user is a Landlord or Contractor they will also be able to set the status of the job from a dropdown. The statuses in the dropdown will be Pending, Landlord Approved, Open, Contact made, Appointment Booked, Closed. Landlord approved will only appear to landlords, and, if selected, will prompt the user to assign a contractor from a dropdown. The user will then press add note and the Job page will refresh to show the note.

2 DESIGN

Every project must have a design phase to increase its chances of success and to ensure developers have a clear understanding of how both front and back-end should be built and interlink. This section shows how the epic stories correlate with wireframes and functional requirements as well as showcasing different design tools and methods used including data dictionary and ER diagram.

2.1 DEVICES USED

Wireframes

Wireframes are used to present a clear idea of how the application will look and positioning of elements. Low-Fidelity wireframes are designed first and are used to quickly make design choices about the website, they are relatively low-cost to produce and are commonly used since they can be modified quickly and easily. After design choices have been made, high-fidelity wireframes can be produced to the low-fidelity specification, with additional content and increased attention to detail.

Data Dictionary

The data dictionary is an easy way to assess all information storage requirements. It shows developers what will be stored, how the data is split into different tables and datatypes & sizes. The data dictionary can then be used to construct an ER Diagram.

ER Diagram

An Entity Relationship diagram is used to visualize the relationships between tables in a database and see how the foreign key in one table relates the data entry to information stored in another table.

2.2 Low Fidelity Wireframes Design Narrative

2.2.1 Epic Story 1

"As a User, I only want to see content relevant to my account"

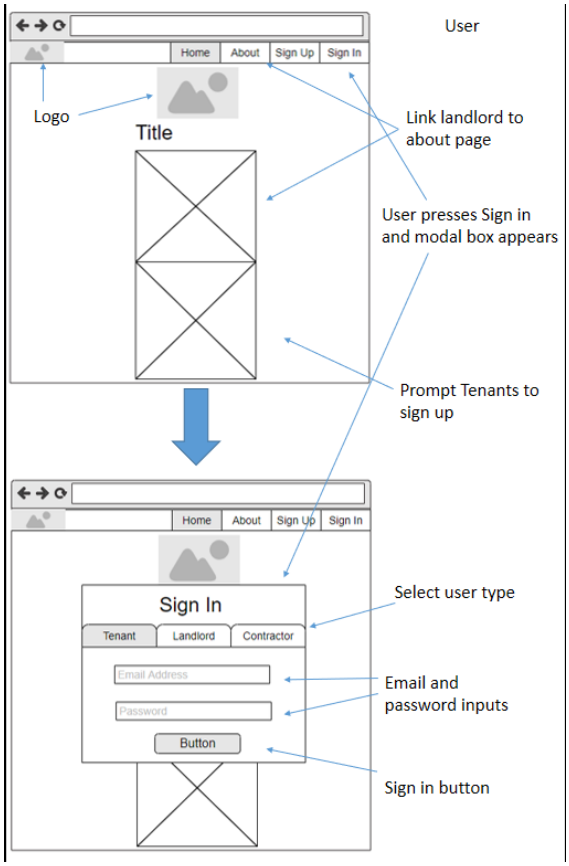


Figure 6 Low Fidelity wireframe for epic 1

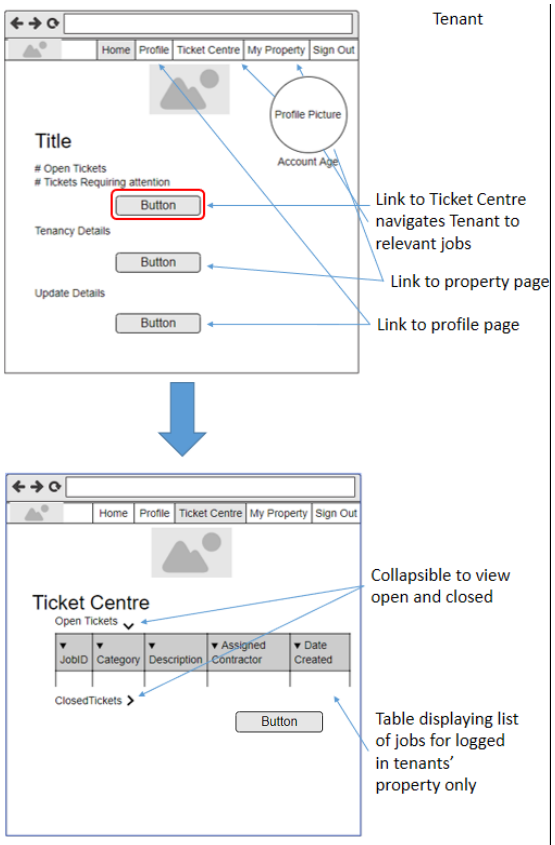
When a user first loads the page, they will see the home screen, which has two main cards. The first invites landlords to learn more about the product and view the about page.

The second invites tenants to create their account so landlords may find them.

The user then opens the log in modal by pressing the button in the top right corner

After Selecting their user type and entering their email address and password, the user can sign in and proceed to the welcome screen

Content relevant to Tenants



After a tenant has logged in, they will be displayed a brief overview of their account along with links to the ticket centre for their property, information about the property and their profile page

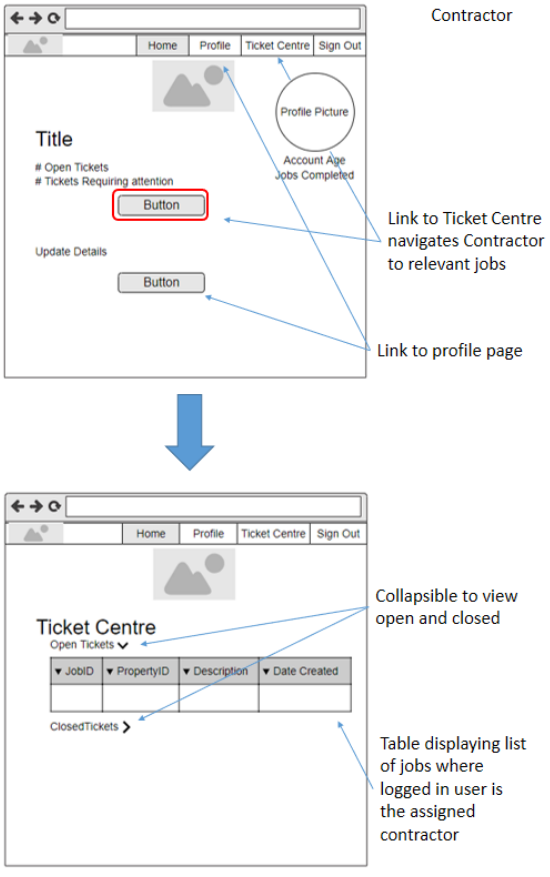
To view jobs relevant to the Tenant, the highlighted button is pressed.

This is the ticket centre where tenants can view all open and closed jobs for their property

Each job can be viewed on its own page if the ID is clicked

Figure 7 (left) Low Fidelity wireframe for epic 1

Content relevant to Contractors



This is the landing page a contractor sees after logging in

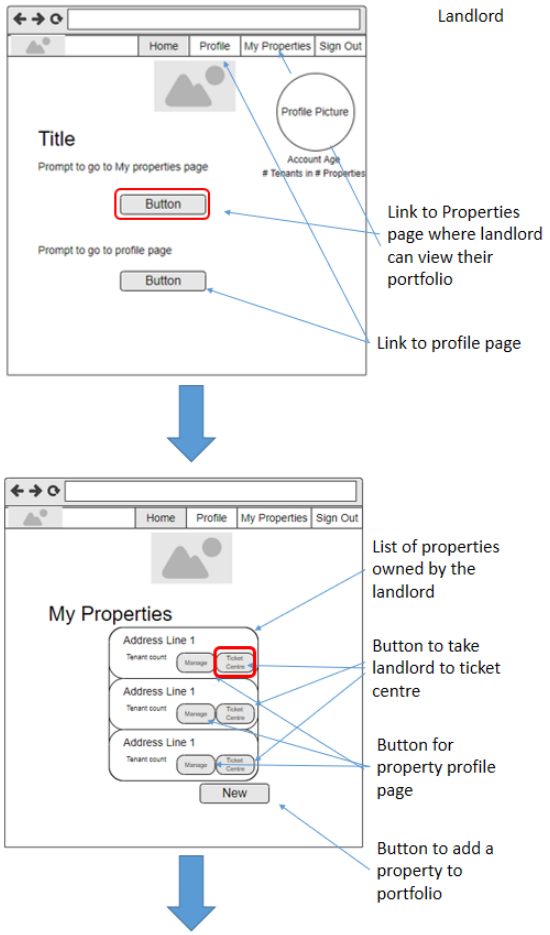
From here, they can navigate to their profile or they can choose to navigate to the ticket centre by pressing on the highlighted button

This is the ticket centre contractor view, it displays all jobs where the user is the contractor assigned by the landlord.

The user can collapse and expand the tables by pressing the arrow buttons shown left.

Figure 8 Low Fidelity wireframe for epic 1

Content relevant to landlords



This is the landing screen landlords see when they log in

It has information about the number of tenants they have and how many properties they own.

The user can go to their profile page, or they can navigate to see their properties by pressing the highlighted button

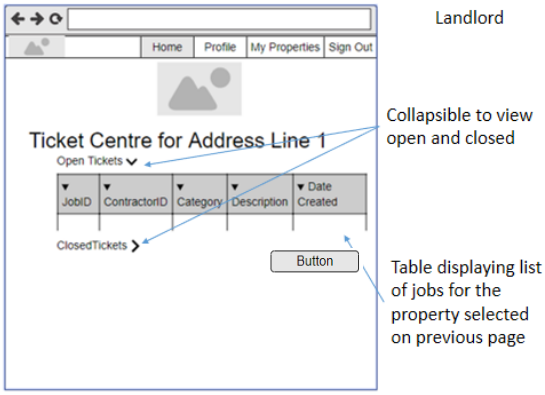
This is the landlord properties page

Shown in the middle is a card displaying a list of the properties the landlord owns

Landlords can go through to the ticket centre for individual properties by pressing the red highlighted button

Alternatively, they may add a new property by pressing the 'New' button

Figures 9 & 10 Low Fidelity wireframe for epic 1

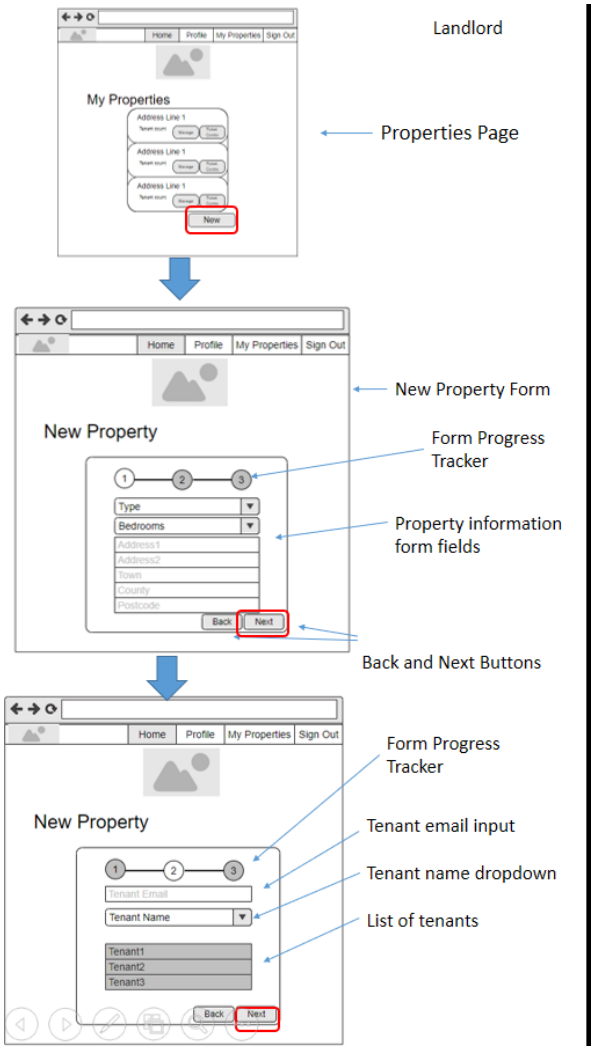


This is the ticket centre for an individual property in a landlords' portfolio

It has collapsible sections containing tables displaying all open and closed jobs for the property

2.2.2 Epic Story 2

"As a Landlord I want a system I can use for managing my property portfolio"



For this epic story, a landlord is logged in and starts, having already navigated to the previously shown Properties page

The landlord then navigates to the new property form by pressing the highlighted button

The new property form is a three step form on a singular page.

The first part of the form collects the properties address, the number of bedrooms and the type of property.

The next button (highlighted) is pressed to go show the second part of the form

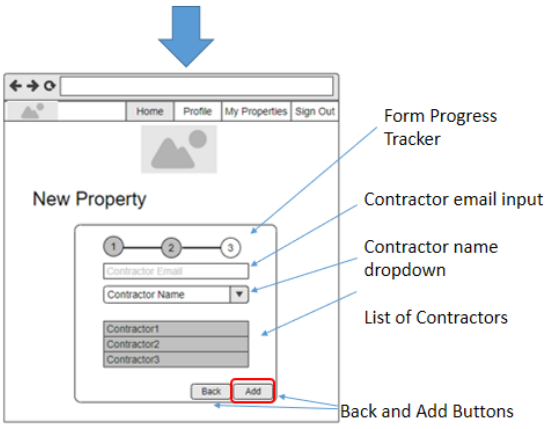
The second part of the form has an input box which takes the email address of the tenant.

When a value has been entered, the dropdown box will appear and populate with names of possible tenants

The landlord selects the correct tenants and they are displayed below.

The tenant presses next to go to the third step

Figures 11 & 12 Low Fidelity Wireframes for epic 2



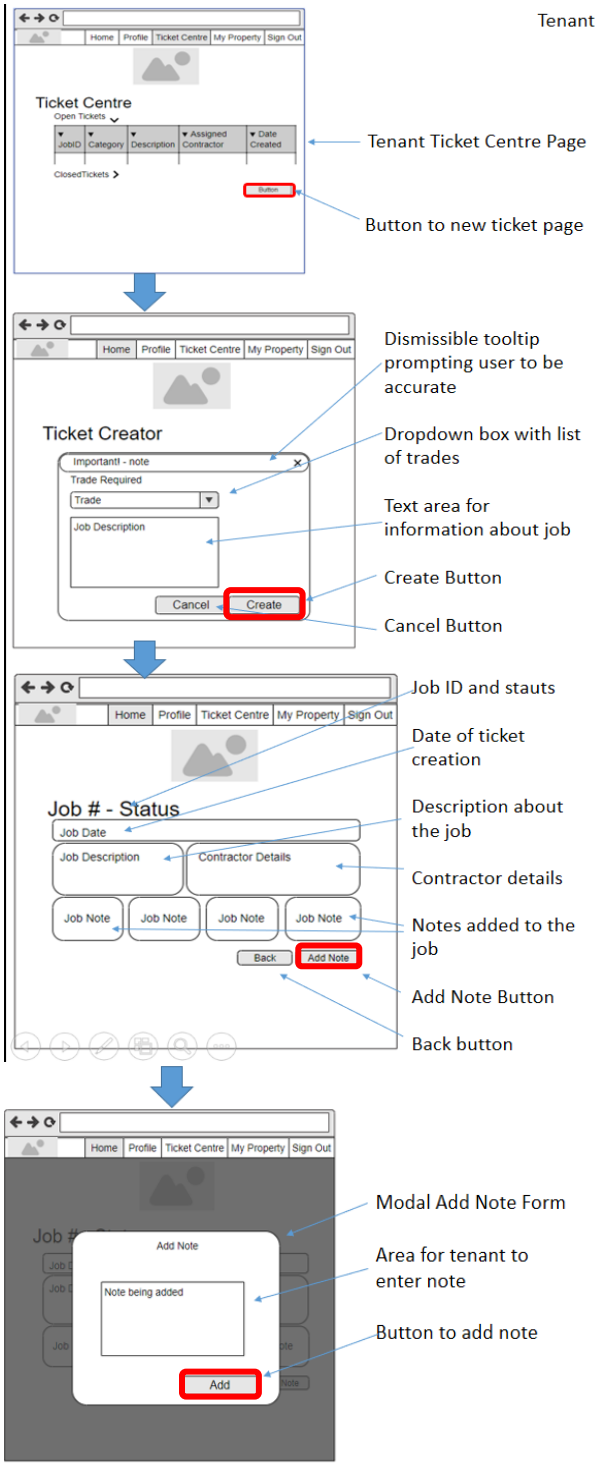
In the third step, the landlord is prompted to enter the email addresses, one at a time, of their chosen contractors

The dropdown then populates with names of matching contractors

When they are selected from the dropdown, their names will appear in the disabled input boxes below.

2.2.3 Epic Story 3

"As a Tenant, I want a hassle free way to report and track maintenance jobs"



In this epic story, the logged in tenant starts from the previously mentioned ticket centre which shows tickets relevant to their property

The user navigates to the Ticket creation page using the highlighted button

This is the Ticket Creator page for tenants. It has a tooltip displaying important information to the user

After a user has filled out the form fields with information relating to the job being reported, they can submit the job by pressing the highlighted create button

The user is then navigated to the job view page which shows details about the job which they have just reported, its current status, the assigned contractor and any notes which have been added.

If a user wishes to add a note of their own, they can do this by clicking the Add Note button highlighted

This is the add note modal form which displays over the job view page.

It has a form field which takes text from the user

The note can be added to the job by pressing the highlighted add note button

Figure 13 (top) & 14 Low fidelity wireframes for epic 3

2.2.4 Epic Story 4

"As a Contractor, I want all my work to be accessible in one place"

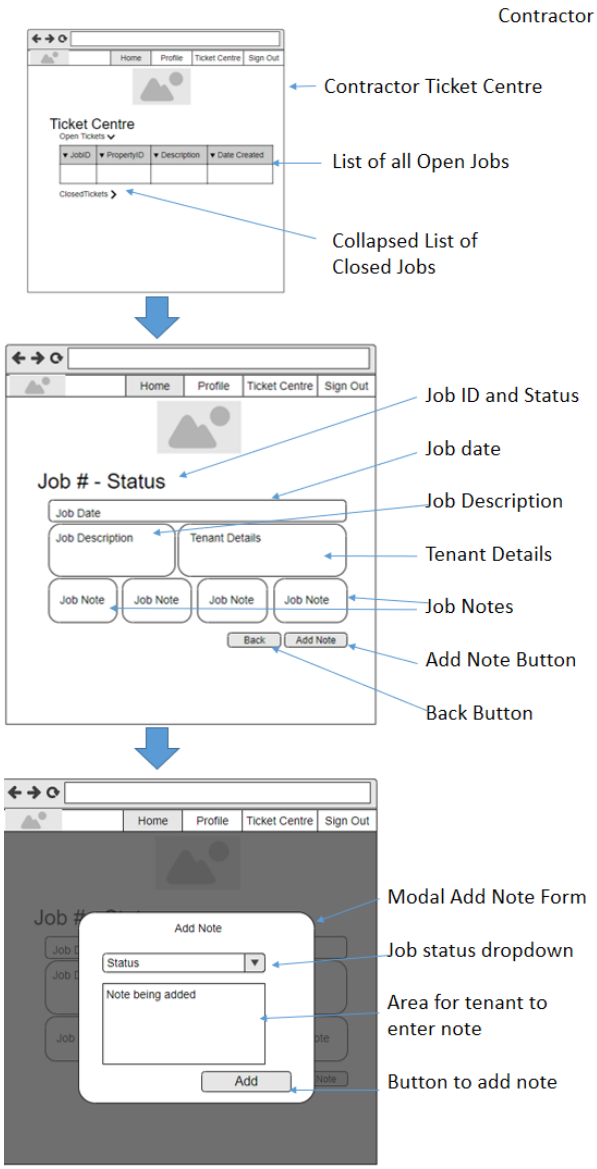


Figure 15 Low Fidelity wireframe for epic 4

This is the Contractor Ticket Centre job view, it was previously mentioned in epic story 1. This is the start point for epic story 4

All tickets assigned to a contractor will be displayed here. To go to the Job view page for a ticket, the contractor must click the ID of the job they wish to view

This is the Job View page

As with the tenant version of this page, it displays the date the ticket was created, job description and job notes added to the job.

Instead of contractor details, the user sees a tenant details card which shows the property address and the tenants contact details

The user can update the status and add notes by pressing the Add Note button

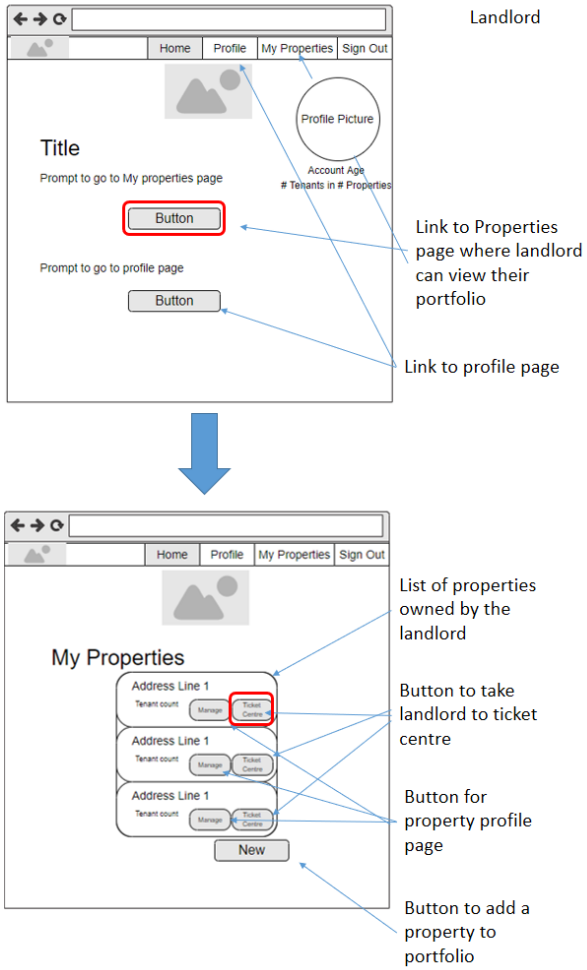
This will bring up the modal box for adding noted to the job

Like tenants, contractors can type in a text description to add. They can also set the status of the job from the dropdown

Upon pressing add, the job page refreshes and shows the new note and status

2.2.5 Epic Story 5

"As a Landlord I want to see a summary of all tenants and maintenance jobs at my properties"



This is the landing page seen when a landlord logs in

This page was already shown in Epic Story 1

In order to see a list of properties and maintenance jobs, the user must navigate to the Properties page by clicking the highlighted button

Once at the properties page, a list of the landlords' properties will be shown

Inside each property card is displayed the number of tenants living at a property as well as the number of active jobs

The user can choose to view a more detailed breakdown for each property by clicking on the highlighted button

Figure 16 Low Fidelity wireframe for epic 5

2.3 Feedback from Low Fidelity Wireframes

After making the low fidelity wireframes, they were shown to a group of stakeholders in the tenant bracket. The general consensus was that the layout is simple and easy to follow. They did not, however, like the colour scheme which was not intended to stay and was already a planned change.

Further feedback stated that the flow between pages was logical for each user group however the navbar needed much improvement. Many of the stakeholders who saw the wireframes said they would prefer a 'sleek' design. This will be taken into serious consideration in an attempt to maximise user experience.

Another helpful piece of feedback was that the tables on the Ticket Centre page were not completely centred and the margins on each side should be equal. This will definitely be implemented and was caused by trying to fit a large table onto a small wireframe.

The final piece of feedback to take away was that a small number of stakeholders did not like the look of the modal log in but still wanted it as a pop-up rather than a separate page. Whilst the majority saw no problem with it, efforts will be made to offer an alternative in the high fidelity wireframes.

2.4 High Fidelity Wireframes

In this section, the low fidelity wireframes have been re-imagined as high-fidelity visions of what the system should look like, taking the tenant feedback from the previous section into account.

2.4.1 NavBar

The navbar is important as it is shown at the top of every single page within the system. There are four different navbar designs, one for each of the three user groups to give the correct menu options, and one for unauthenticated users on the home page.

As per the feedback, efforts were made in style and design choices to create a navbar which is both simplistic and elegant. Opting for a dark background with light text, the contrast gives all pages on the website a bold header which is visually appealing.

Contractor

Landlord

Tenant

No User

Figures 17-20 High Fidelity Navigation Bar Wireframes

2.4.2 Home Page

Shown below is the high fidelity of the home page which unauthenticated users will see before they log in. The image placeholders will link landlords to the about page in an attempt to secure their business. The second card is to direct tenants to the sign up page since it is likely they are there under the direction of their landlord.

The design has been made to take responsivity into account and display nicely on smaller screens for when users are not at a home computer or laptop but need to access the system on the go

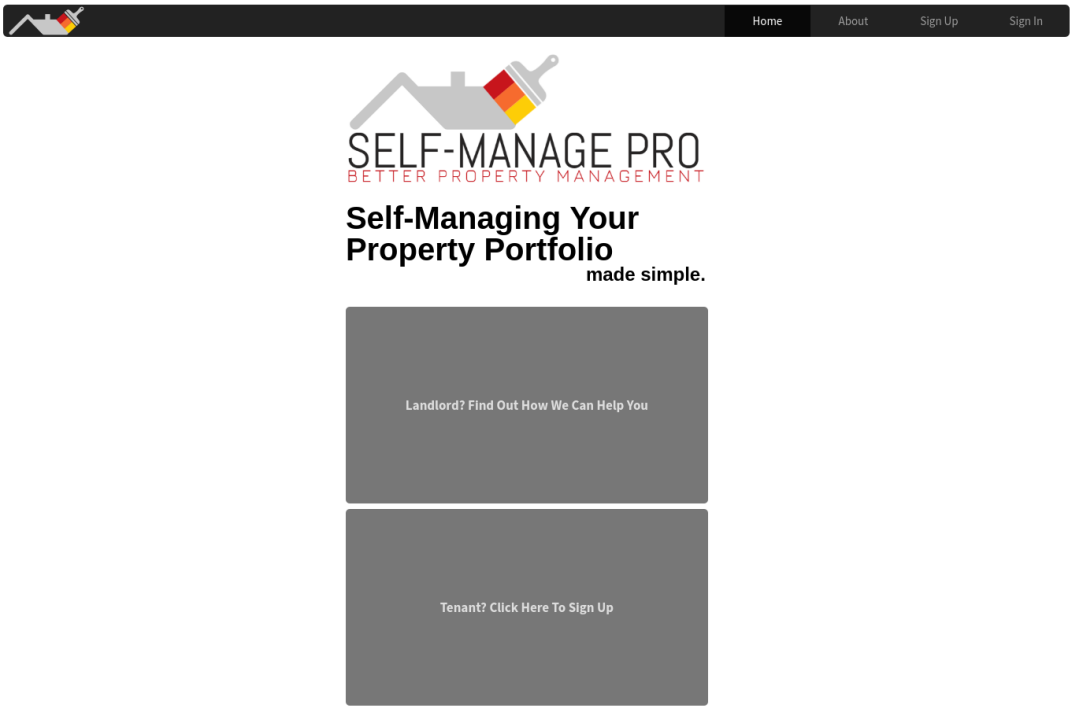


Figure 21 High Fidelity Home Page Wireframe

2.4.3 Home Page Sign In Modal

This is a mock-up of how the sign in modal page, accessed through the home page, will look. It has tabs for the user to select which type of account they have, and text input boxes for their login credentials. User feedback was considered and the layout of the modal has been noticeably changed

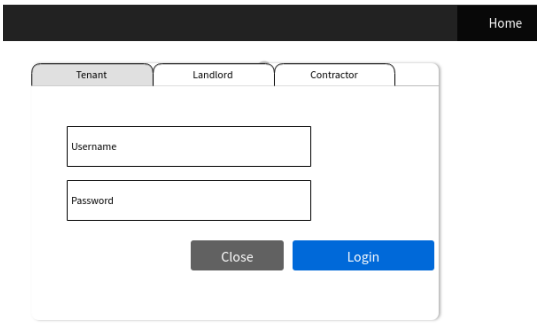


Figure 22 Modal Login HF Wireframe

2.4.4 Welcome Page

Tenant Welcome Page

This is the landing area which greets tenants after they log in, there are three main navigation options to choose from here. Mentioned in the low fidelity wireframe, these are links to the ticket centre, their properties information page, and their profile page. The page also displays how long the user has been registered for, counters for open and closed tickets, and how many tickets they've had completed in total.

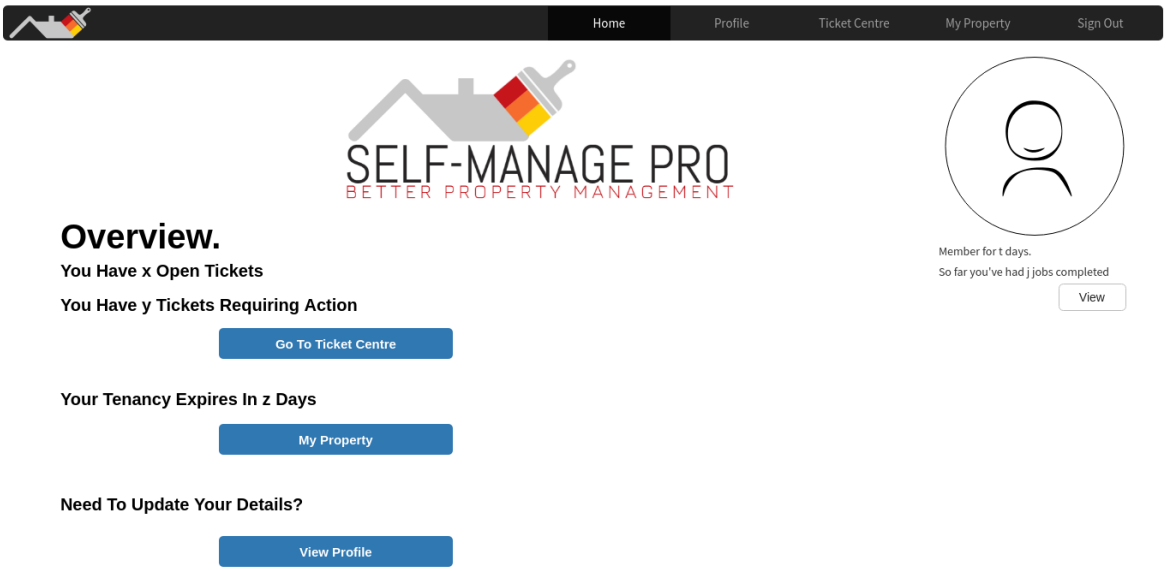


Figure 23 Tenant Welcome Page HF Wireframe

Landlord Welcome Page

This is the landlord version of the previous wireframe, it too displays the time the user has been registered for as well as how many properties they have in total and a count of their tenants. From here, landlords may navigate to their properties page or their profile information.

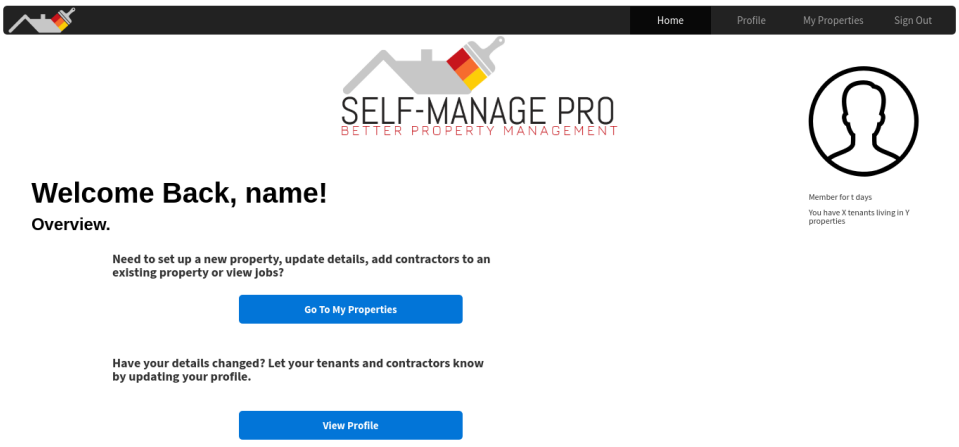


Figure 24 HF Landlord Welcome Page Wireframe

Contractor Welcome Page

This is the contractor counterpart of the previous wireframes, it also displays how many days since the user signed up. As well as this, the total number of jobs they have completed at properties for which they are assigned is shown. In the centre, next to the navigation to ticket centre, there are counters for how many open jobs the contractor is assigned to as well as how many jobs they have which require an update/attention.

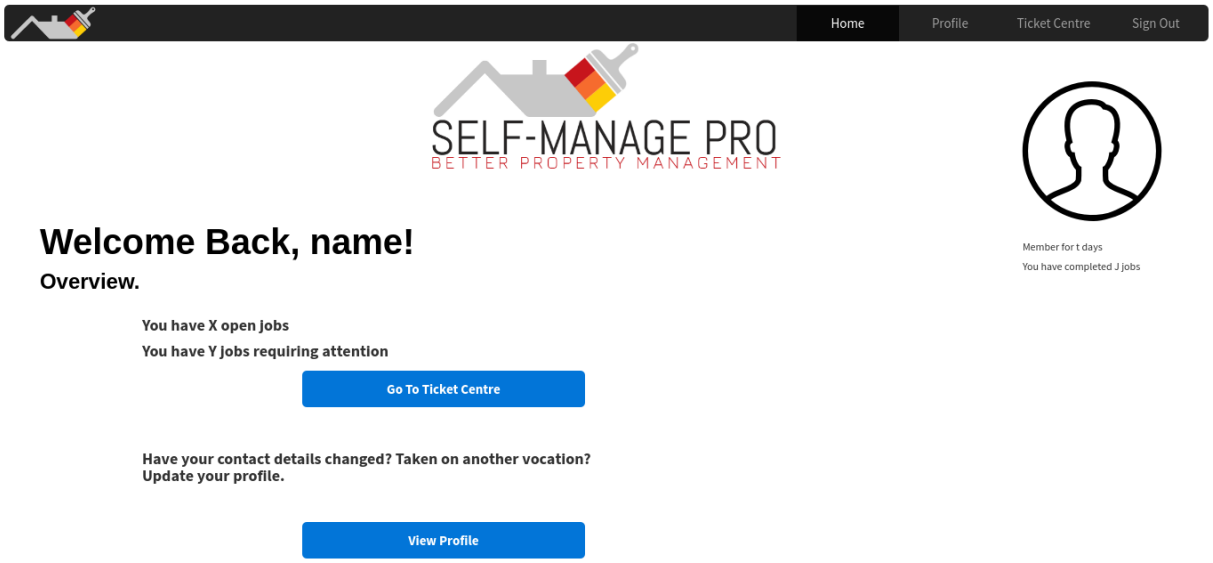


Figure 25 High Fidelity Contractor Welcome Page Wireframe

2.4.5 Ticket Centre

Open Tickets

This is the ticket centre which users will see after navigating to the page either from the landing page as a tenant or contractor, or from the Properties page as a landlord. It has two collapsible sections, one for Open tickets and one for closed tickets. Each collapsible container holds a table with a list of jobs, their descriptions and assigned contractor. As per the stakeholder feedback, changes were made from the low fidelity wireframes to centre the tables, giving them an equal margin on each side and making them more visually appealing.

A new ticket can be created as a tenant or landlord by pressing the New Ticket button

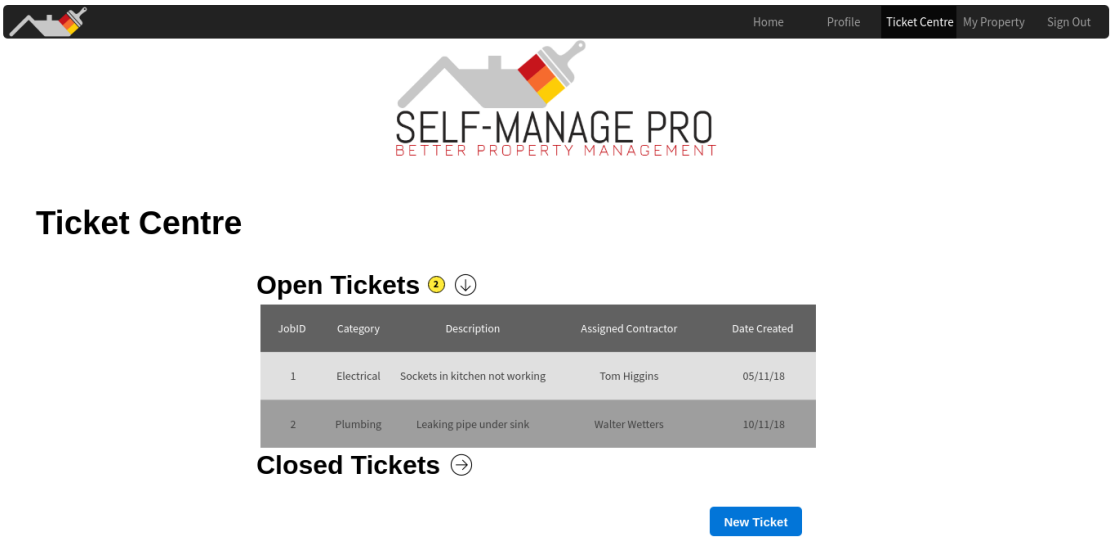


Figure 26 HF Ticket Centre Wireframe – Open tickets

Closed Tickets

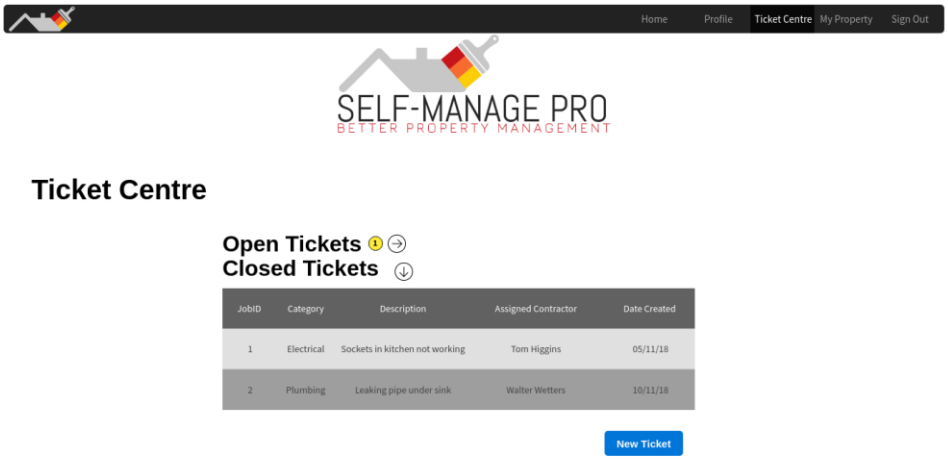


Figure 27 HF Ticket Centre Wireframe - Closed tickets

2.4.6 Ticket Creator

Tenant Ticket Creator

This is the high fidelity tenant version of the ticket creator page from the previous section. Tenants can use this form to create a ticket for the property which they stay in. The tenant selects the trade required from a pre-populated list in the trade dropdown box. They can also add a description to the job which serves to inform the landlord and contractor what needs doing.

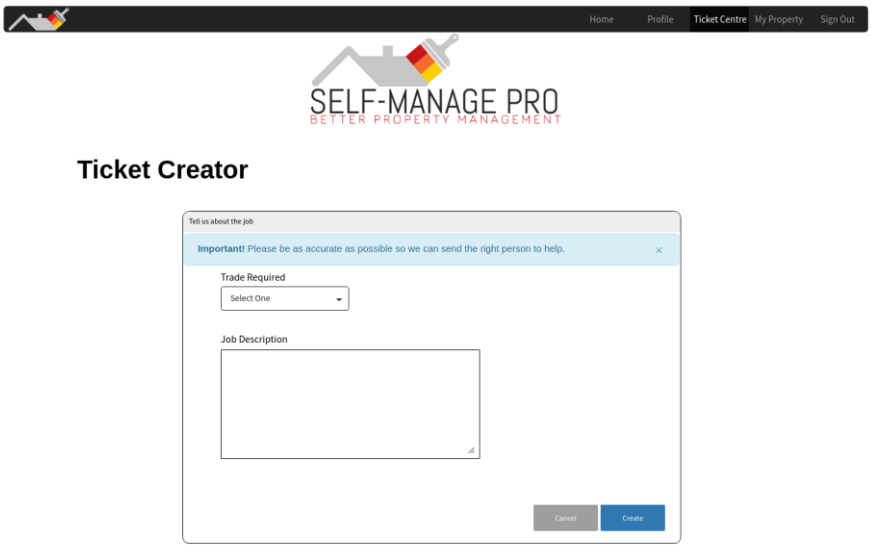


Figure 28 HF Ticket Creator

Landlord Ticket Creator

The landlord ticket creator functions in much the same way as the tenant version, with the key difference being that after a trade has been selected, a dropdown box appears and is populated with a list of the landlords approved contractors for that specific trade. This will be done using AJAX.

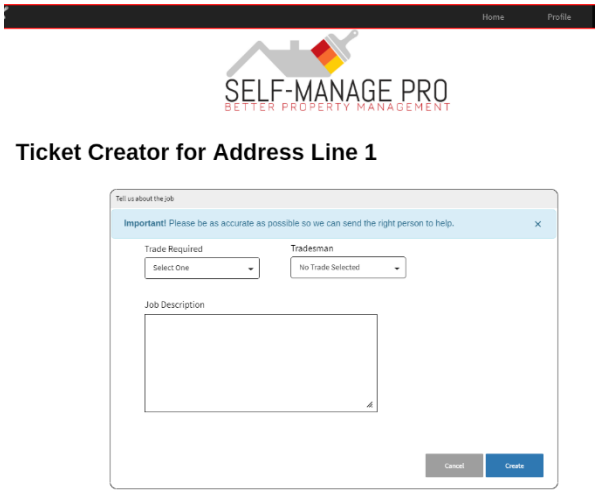


Figure 29 HF Landlord Ticket Creator Wireframe

2.4.7 My Properties page

Zero Properties in Portfolio

This is the page which a landlord will see when they navigate to the My Properties section from the landing page if they do not have any properties saved to their portfolio. The middle contains a card which prompts the user to set up a new property.

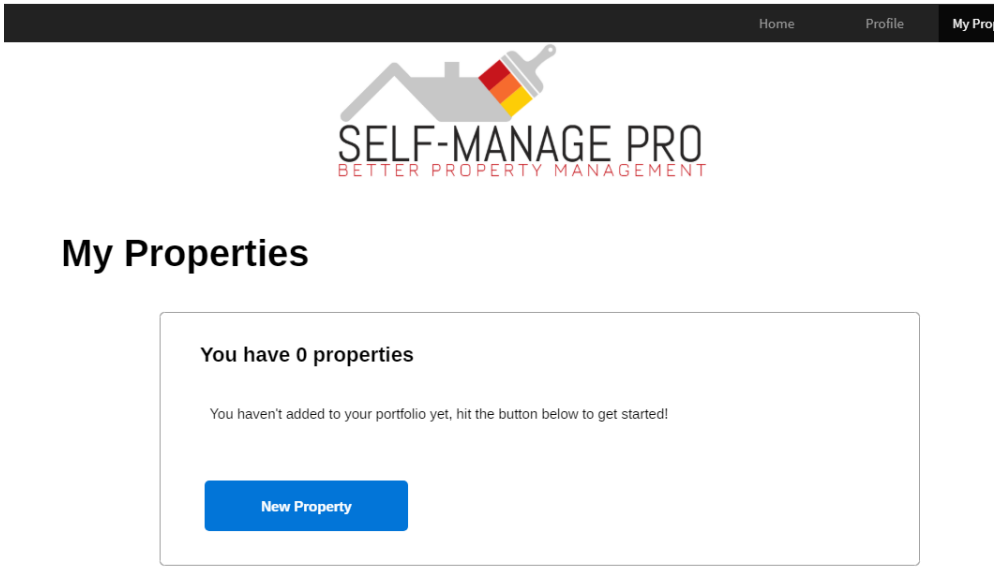


Figure 30 HF My Properties Wireframe

Properties added

This is the view of the My Properties page which a landlord will see if once they have one or more properties added to their portfolio. The cards in the middle are generated on page load and are populated with the first line of address as an identifier, along with the tenant count and how many jobs are currently active. Landlords can navigate to the ticket centre for individual properties by pressing the Ticket Centre button within the card for each property.

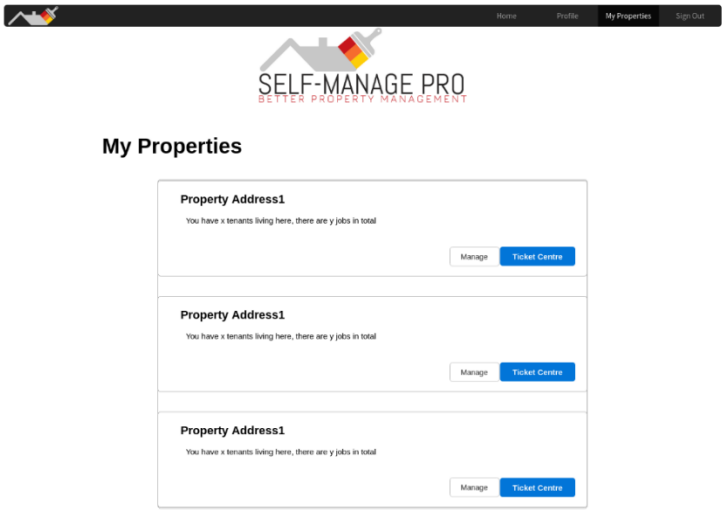


Figure 31 HF My Properties Page

2.4.8 New Property Form

This is the New property form which is accessed as a landlord from the My Properties page. It contains a three part, same-page form which will use javascript and jQuery to navigate through the steps. As stated in the low-fidelity wireframes, the first part takes information about the property type and address; the second part, information about the occupying tenants; the third part, contractors assigned to the property.

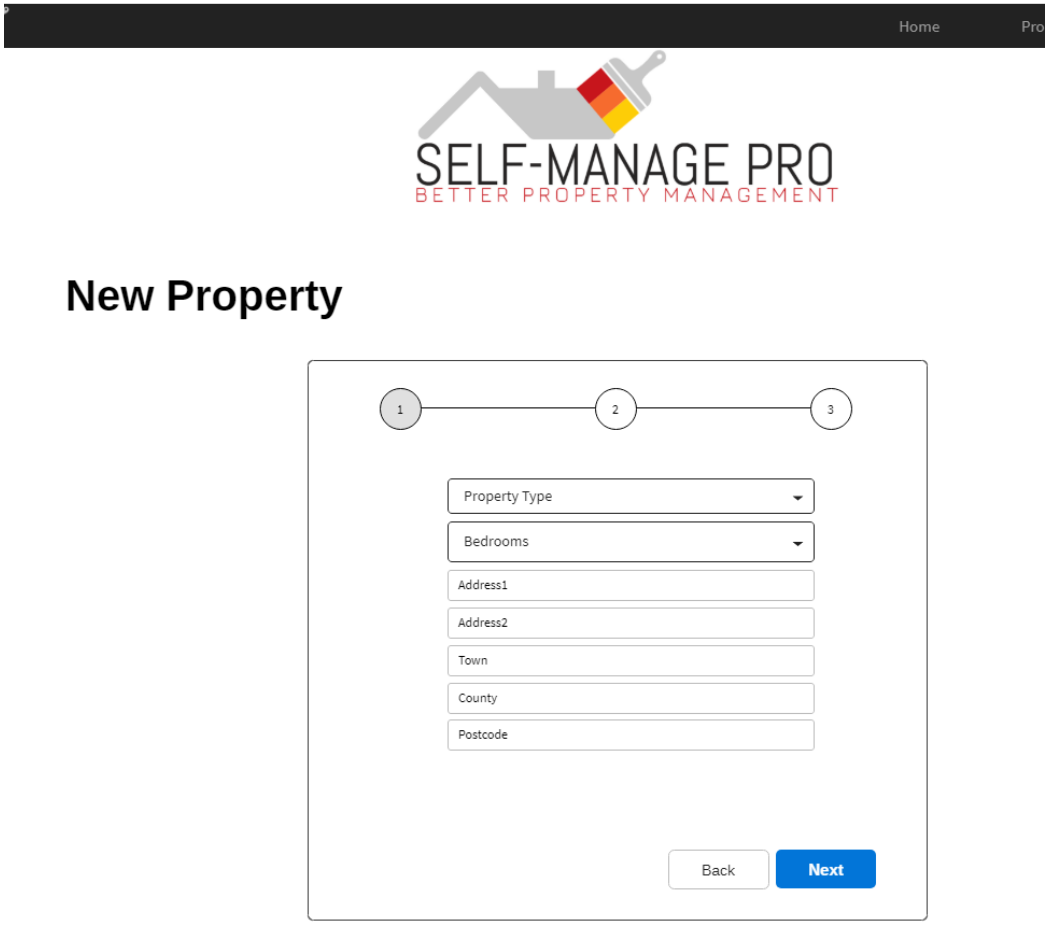


Figure 32 HF New PropertyPage

2.4.9 Job View

Open Ticket

This is the Job View page which all user groups will see when they click on the ID of a job from the ticket centre. It displays the jobs ID, the status of the job, its description, details about the assigned contractor and any noted which have been added. There is a dismissible tooltip which displays when the job was created and when the last update was.

If the logged in user is a contractor, they will see a Tenant Details card in place of the Tradesman card, showing names and telephone numbers for each tenant at the property as well as the property address.

Stakeholders liked the original low fidelity design of this page since it displayed all key information required in an easily readable and understandable format, therefore, its design has remained unchanged.

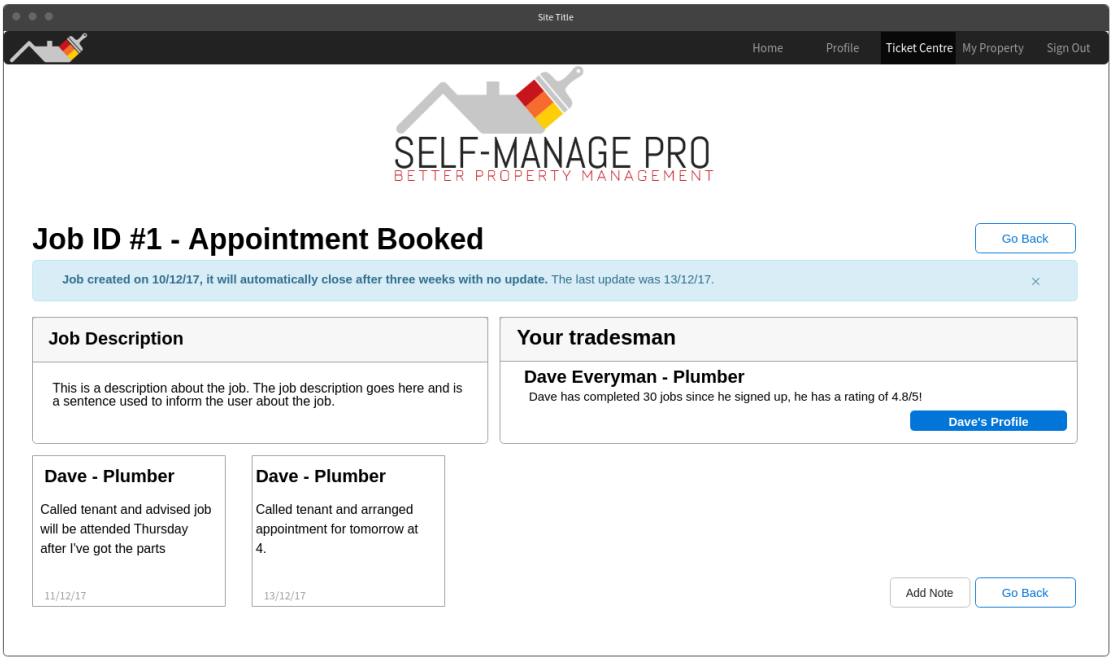
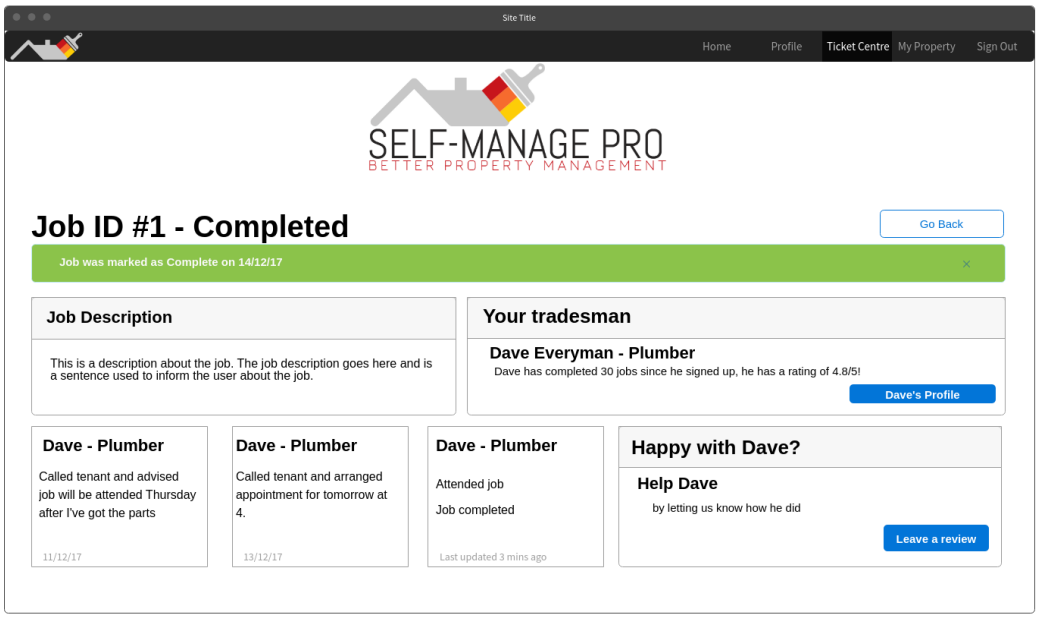


Figure 33 HF Job view Page

Completed Ticket

This is the high fidelity wireframe of what the job view page looks like after a job has been marked as completed. The add note button has been removed since the job is finished, and the colour of the tooltip bar has been changed to reflect the updated job status.

Figure 34 HF Job View Job Completed



2.5 Feedback from High Fidelity Wireframes

Stakeholders really liked the new navigation bar, saying it was a great improvement from the previous wireframes and looks very professional. The minimalist theme is not intrusive to the body of the page, and the spacing of its elements is appropriate.

Another piece of important feedback is that stakeholders thought on some pages, the margin between page content and title was too great, this will be reflected in the implementation stage to ensure every page is as visually appealing as possible.

Having changed the sign in modal at the request of a minority of the stakeholders whilst the others remained indifferent; there was overwhelming support for the original low fidelity mock-up to be implemented as they did not like the design of the high fidelity modal whatsoever.

The new property form was assessed and users said the liked the way the form has three parts, broken up but still all on one page rather than having to navigate through multiple web pages.

2.6 DATA DICTIONARY

The data dictionary is an important step of designing any system. It contains all of the stored business data, including the storage requirements, datatypes, formatting and relations. It is organised into database tables for readability. The data dictionary is used to guide the development process when building the application as well as for designing the Entity relationship diagram.

2.6.1 CONTRACTOR

The Contractor table holds the login credentials for registered contractors. When a user attempts to log in, the password they enter is prepended to the stored salt value, hashed using the SHA-256 algorithm and compared to the stored hash.

Column	Type	Null	Default	Comments
CONTRACTORID	int(4)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
email	varchar(255)	No	-	
PWDHASH	char(64)	No	-	
SALT	char(3)	No	-	

2.6.2 CONTRACTORDETAILS

Contractor details stores all contact information for individual contractors, each record has a foreign key which relates to an entry in the Contractor table.

Column	Type	Null	Default	Comments
CDNUM	int(4)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
CONTRACTORID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
FIRST	varchar(50)	No	-	
LAST	varchar(50)	No	-	
DOB	date	No	-	
TELEPHONE	char(11)	No	-	
ADDRESS1	varchar(255)	No	-	
ADDRESS2	varchar(255)	Yes	*NULL*	
TOWN	varchar(35)	No	-	
COUNTY	varchar(35)	Yes	*NULL*	
POSTCODE	varchar(10)	No	-	
CREATED	datetime	No	CURRENT_TIMESTAMP	

2.6.3 CONTRACTORTRADES

Contractor Trades stores a list of trades which each contractor is qualified for. The trades are stored as tinyint values of either 1 or 0 representing true and false respectively. Each record has a foreign key CONTRACTORID relating to its entry in the Contractor table.

Column	Type	Null	Default	Comments
CTID	int(5)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
CONTRACTORID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
Carpenter	tinyint(1)	No	0	
Cleaner	tinyint(1)	No	0	
Electrician	tinyint(1)	No	0	
Gardener	tinyint(1)	No	0	
GasEngineer	tinyint(1)	No	0	
Gen	tinyint(1)	No	0	
Heating	tinyint(1)	No	0	
Lift	tinyint(1)	No	0	
Locksmith	tinyint(1)	No	0	
Painter	tinyint(1)	No	0	
Plasterer	tinyint(1)	No	0	
Plumber	tinyint(1)	No	0	
Pool	tinyint(1)	No	0	
Roof	tinyint(1)	No	0	

2.6.4 JOB

Job is a table for storing information about maintenance tickets logged through the system. Each record has a primary key with which it can be identified, as well as two foreign keys; the first relating to an entry in the Property table; the second relating to an ID in the contractor table. Completed is a date field which is left blank when an entry is created and then updated when the job is marked as closed using the `CURDATE()` command.

Column	Type	Null	Default	Comments
JOBID	int(6)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
PROPERTYID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
CONTRACTORID	int(4)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TRADE	varchar(30)	No	-	
DESCRIPTION	varchar(250)	No	-	
COST	decimal(6,2)	Yes	*NULL*	
STATUS	varchar(20)	No	-	
CREATED	datetime	No	CURRENT_TIMESTAMP	
COMPLETED	date	Yes	*NULL*	

2.6.5 JOBNOTES

Job Notes is a table which stores information appended to a job by a tenant, landlord or contractor. It has a single foreign key which relates to a Job ID in the Job table. Since a note for a job may be left by any user type, there are two fields which store the type of user leaving the note and their ID instead of having three fields as foreign keys to the user type tables. The two fields are combined programmatically within the system to form a resultant relation.

Column	Type	Null	Default	Comments
JOBNOTEID	int(7)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
JOBID	int(6)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
LEFTBYTYPE	varchar(10)	No	-	
LEFTBYID	varchar(6)	No	-	
NOTE	mediumtext	No	-	
CREATED	datetime	No	CURRENT_TIMESTAMP	

2.6.6 LANDLORD

The landlord table stores account credentials for registered landlords. Authentication of users is handled in the same way as the Contractor table.

Column	Type	Null	Default	Comments
LANDLORDID	int(5)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
email	varchar(255)	No	-	
PWDHASH	char(64)	No	-	
SALT	char(3)	No	-	

2.6.7 LANDLORDDETAILS

Landlord Details stores contact information about landlords and is handled in the same way as Contractor Details. It has a foreign key relating to a single entry in the Landlord table.

Column	Type	Null	Default	Comments
LDNUM	int(5)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
LANDLORDID	int(5)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
FIRST	varchar(50)	No	-	
LAST	varchar(50)	No	-	
DOB	date	No	-	
TELEPHONE	char(11)	No	-	
ADDRESS1	varchar(255)	No	-	
ADDRESS2	varchar(255)	Yes	*NULL*	
TOWN	varchar(35)	No	-	
COUNTY	varchar(35)	Yes	*NULL*	
POSTCODE	varchar(10)	No	-	
CREATED	datetime	No	CURRENT_TIMESTAMP	

2.6.8 PROPERTY

The Property table stores information about a property, its address, the number of bedrooms and the type of property (e.g. flat). It has a foreign key which relates the landlord ID to a single row in the landlord table

Column	Type	Null	Default	Comments
PROPERTYID	int(4)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
LANDLORDID	int(5)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
TYPE	varchar(30)	No	-	
BEDROOMS	int(2)	No	-	
ADDRESS1	varchar(255)	No	-	
ADDRESS2	varchar(255)	Yes	*NULL*	
TOWN	varchar(35)	No	-	
COUNTY	varchar(35)	Yes	*NULL*	
POSTCODE	varchar(10)	No	-	

2.6.9 PROPERTYCONTRACTOR

Property Contractor stores the relationship between contractors and properties they are assigned to. Each entry has two foreign keys relating to properties and individual contractors. Each contractor and property may appear in multiple rows since one contractor may serve many properties, and one property may have multiple different contractors.

Column	Type	Null	Default	Comments
PCID	int(4)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
PROPERTYID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
CONTRACTORID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL

2.6.10 TENANT

The Tenant table stores the login credentials for registered tenants. Authentication is handled in the same way as in the Contractor and Landlord tables.

Column	Type	Null	Default	Comments
TENANTID	int(6)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
email	varchar(255)	No	-	
PWDHASH	char(64)	No	-	
SALT	char(3)	No	-	

2.6.11 TENANTDETAILS

The Tenant Details table stores personal information about each tenant. It has a foreign key relating each entry to a single entry in the Tenant table. Unlike Contractors and Landlords, addresses are not stored in this table since a tenants address can be asserted from the property to which they are assigned.

Column	Type	Null	Default	Comments
TDNUM	int(6)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
TENANTID	int(6)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
FIRST	varchar(30)	No	-	
LAST	varchar(30)	No	-	
DOB	date	No	-	
TELEPHONE	char(11)	No	-	
CREATED	datetime	No	CURRENT_TIMESTAMP	

2.6.12 TENANTGROUP

Tenant group stores groups of tenants residing in a single property. Each group may contain up to twelve tenant IDs, all foreign keys relating to entries in the Tenant table. A Tenant group must contain at least one tenant.

Column	Type	Null	Default	Comments
TGID	int(5)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
TENANT1	int(6)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT2	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT3	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT4	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT5	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT6	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT7	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT8	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT9	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT10	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT11	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL
TENANT12	int(6)	Yes	*NULL*	FOREIGN KEY; UNSIGNED ZEROFILL

2.6.13 TENANTGROUPPROPERTY

Tenant Group Property is a table which relates groups of tenants living together to a property. Each entry has two foreign keys, one linking the row to a group of tenants in the Tenant Group table, and the other to a single row in the Property table.

Column	Type	Null	Default	Comments
TGPID	int(4)	No	-	PRIMARY KEY; UNSIGNED ZEROFILL
TGID	int(5)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL
PROPERTYID	int(4)	No	-	FOREIGN KEY; UNSIGNED ZEROFILL

2.7 Entity Relationship Diagram

The ER diagram is constructed to the specification of the Data Dictionary, it is used to view and cross-reference relationships between tables. It can also be used to design SQL queries as it gives a clear picture of how entries in one table relate to entries in another and their cardinality. Each table representation has two columns, one for the datatype and storage requirement of attributes, and the other for the field names and their status as primary and foreign keys.

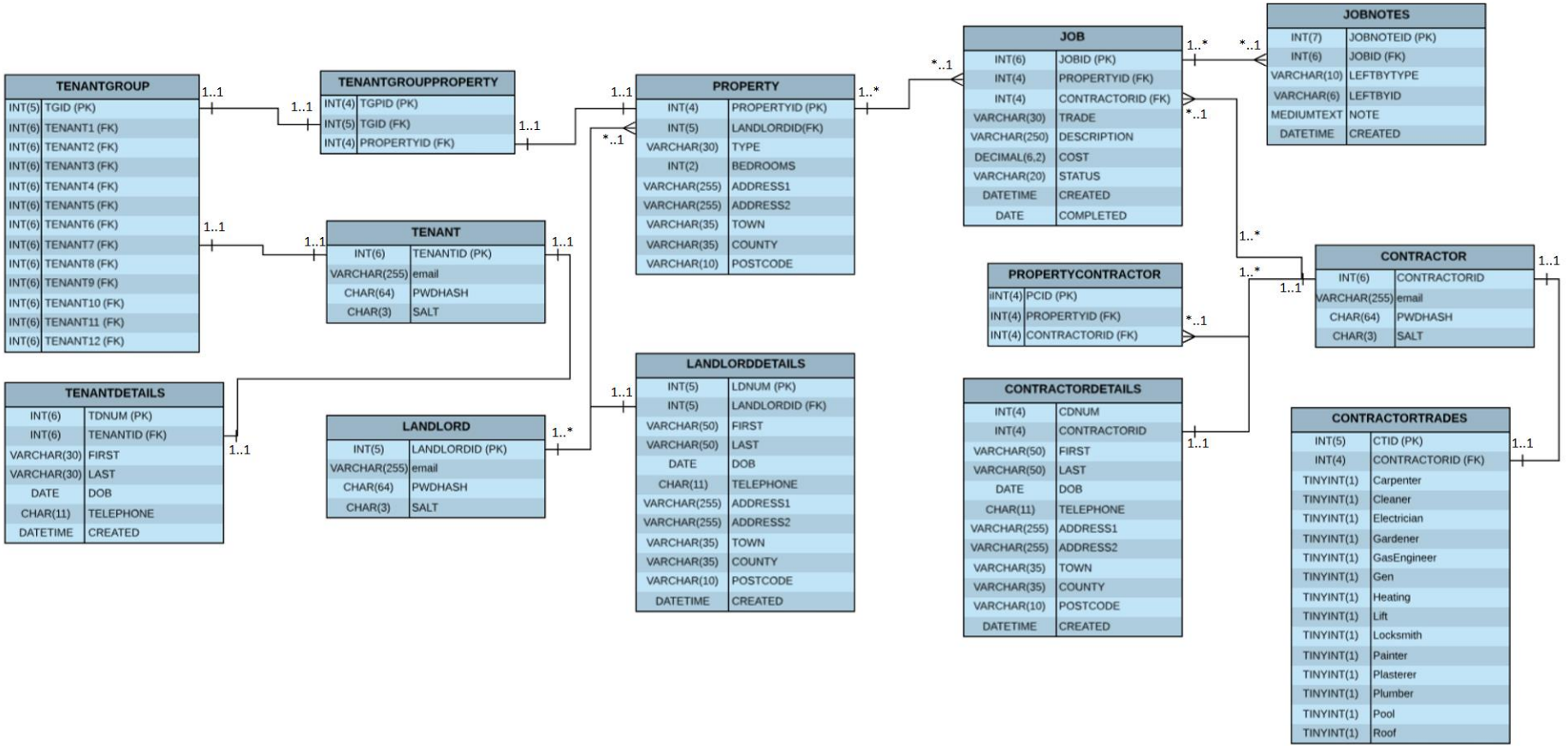


Figure 35 Entity Relationship Diagram

3 IMPLEMENTATION

3.1 USE OF LIBRARIES AND INTERFACES

3.1.1 NetBeans

NetBeans is an interactive development environment which supports multiple programming languages, plugins and frameworks. To shortlist a few, NetBeans provides support for HTML, Java 8, CSS3, Maven, jUnit, Git, Python, pHp and many more. With the ability to load custom plugins and libraries as dependencies for projects, along with my pre-existing experience using this IDE, NetBeans was chosen for developing this project.

3.1.2 phpMyAdmin

phpMyAdmin is an SQL database management tool written in pHp. It offers a useful user interface which makes designing and managing a database and its internal relationships less laborious than hand-writing SQL to accomplish the same tasks. On top of this, phpMyAdmin has an SQL writer which can be used to easily design and test queries for use within the application.

3.1.3 Java Web Enterprise Edition

Java EE is essentially the community driven standard for developing enterprise applications. It supports servlets for delivering web content and have numerous contribution implementations from members throughout the programming community; keeping them up to date with the business needs of small and large scale developers alike.

Having previous knowledge and expertise working with Java, combined with the host of features it offers, it is a suitable choice as the main development language for this system.

3.1.4 jQuery

jQuery is a JavaScript written library which aids manipulation of the Document Object Model (DOM) and DOM tree traversal. It offers a range of CSS animations, Ajax implementations and event handlers as well as plenty of functions to reduce the amount of code required to do most tasks.

jQuery serves as an abstraction layer between complicated JavaScript implementations and developers. It is estimated that around 97% of all websites sue jQuery for its functionality and implementations (W3Techs.com, 2019).

3.1.5 Bootstrap 4

Bootstrap is a component library written in JavaScript which provides HTML, JS and CSS implementations for designing responsive and intuitive web page user interfaces. It uses a class selector system which makes styling a website incredibly easy, and enables developers to write entire projects using virtually no CSS. Bootstrap saves vast amounts of time when building meaning developers can spend more time meeting user requirements and making business logic.

3.1.6 Connector /J

Connector /J is the official Java Database Connectivity (JDBC) driver for SQL. It is a library written in Java which serves as an interface between programmer and JDBC API. It simplifies the process of connecting an application to a database without the use of a framework such as Spring.

3.2 MVC LAYOUT

MVC stands for Model View Controller and is an outline for separating the concerns of the system. Class files which are used for modelling data, such as Tenant, Landlord and Contractor all reside within the Model package of the project. The Controller package contains all of the functional logic e.g. Servlets, and handles requests.

View denotes any files which present data to the user. This system implements the MVC layout throughout. A screenshot of the projects final file list, arranged into MVC layout can be seen below.

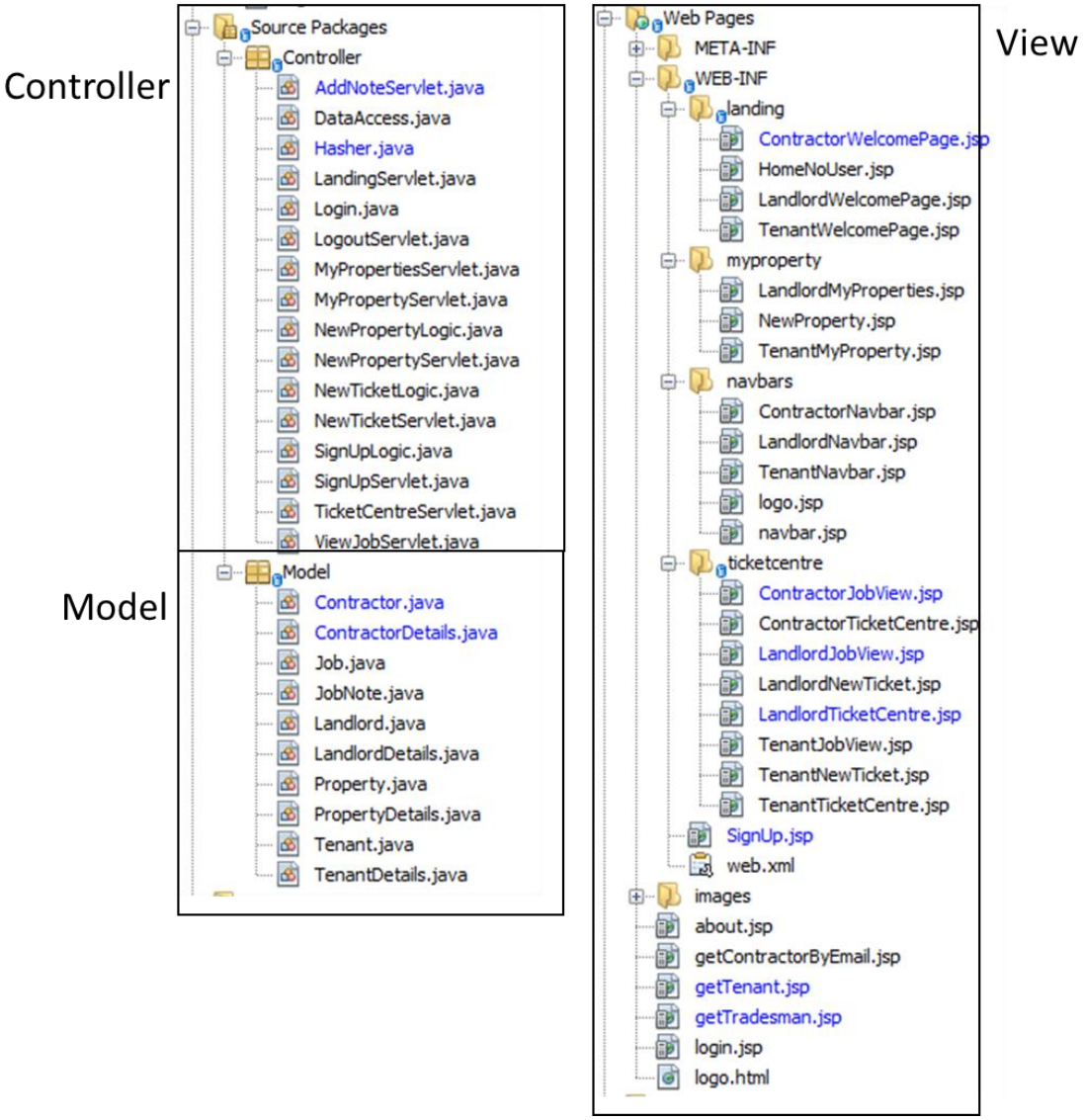


Figure 37 Model View Controller File Structure

3.3 TRELLO LOG

3.3.1 Preliminary work

Throughout this project, a Trello board is used to chart and monitor tasks and progress over time. Before the first sprint, the project vision, must have requirements, user stories and product backlog were arranged onto cards on a Trello board.

The product backlog contains tasks which were deduced from the individual user stories and arranged roughly in the order they will be completed. Arranging tasks in this way is helpful as developers can see how far through the project they are as well as any tasks which are falling behindThe next step was to pick a selection of tasks from the backlog to be completed during the first sprint.

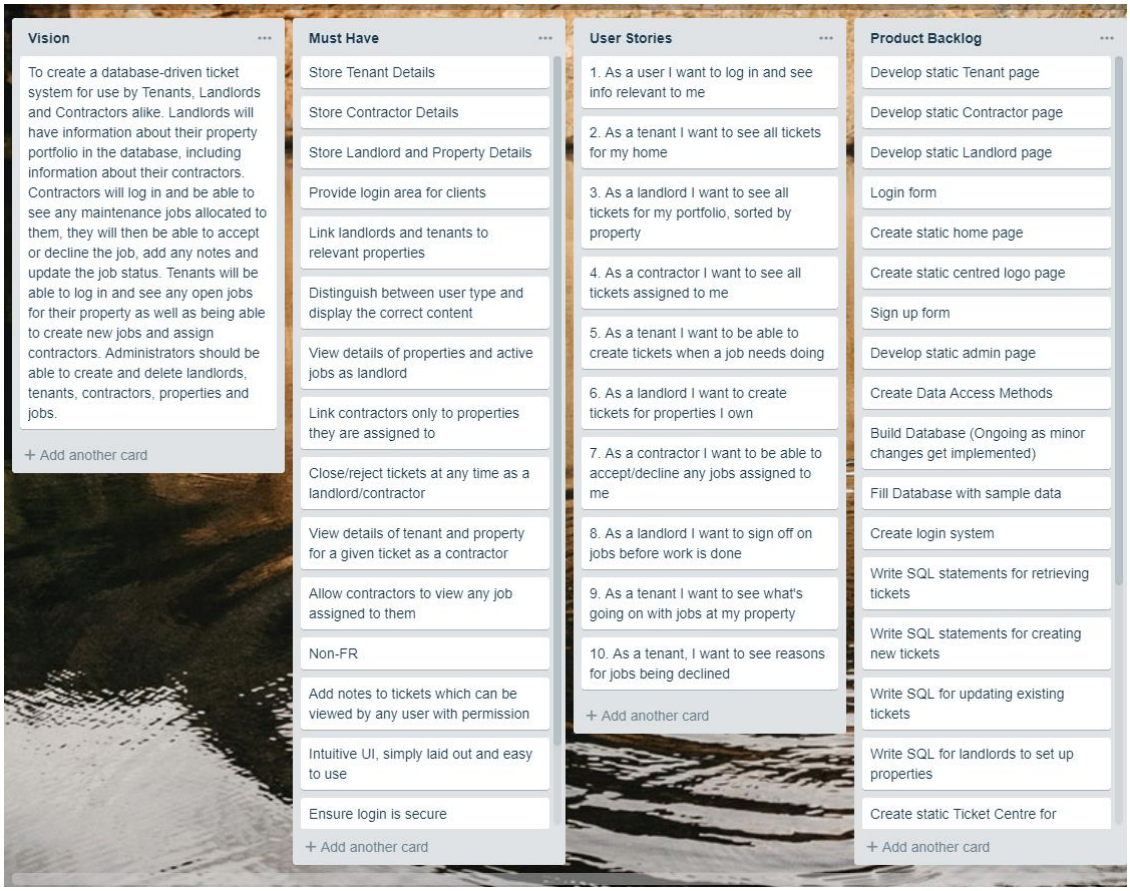
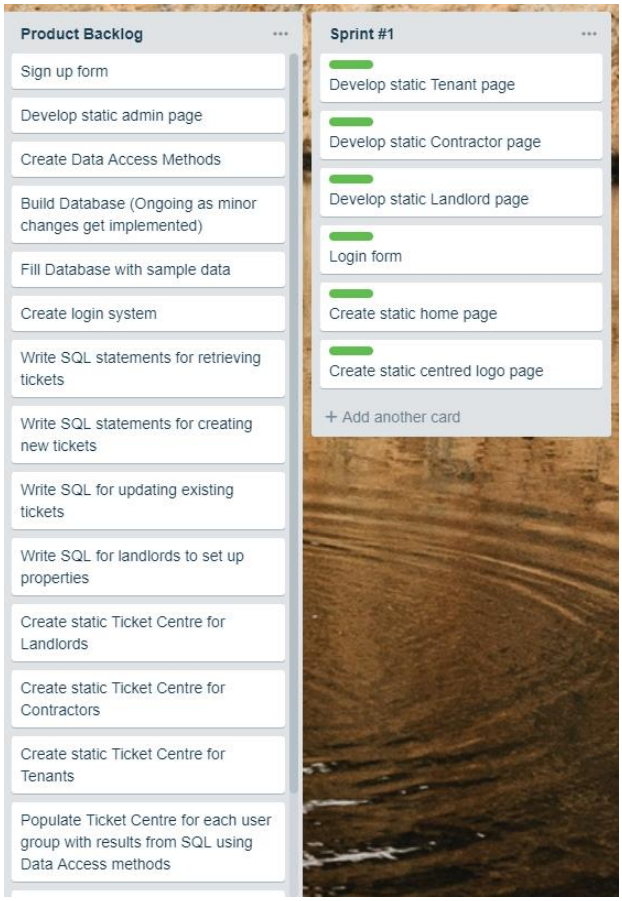


Figure 38 Product Backlog

3.3.2 Sprint 1



The first sprint revolved around making static pages using bootstrap which conformed to the high fidelity wireframes from the design stage.

During this sprint, static pages were developed for the home page, landlord, tenant & contractor landing pages, the login modal and the logo which will appear centred at the top of every page just below the navbar.

Figure 39 Sprint 1 and product backlog

Centred logo page

The logo has its own jsp page by istelf and style rules to match, It contains a single div element, within which is a <img> tag. The image is styled to display in the centre of any page it is a part of and will maintain a minimum width of 250 pixels on small screens.

Figure 40 Centred logo Page

Static Home Page

The home page was made to the specification of the high fidelity wireframes, with one key difference from stakeholder feedback; the image cards in the middle of the DOM are next to each other horizontally rather than vertically, however will stack vertically on resized and smaller devices.

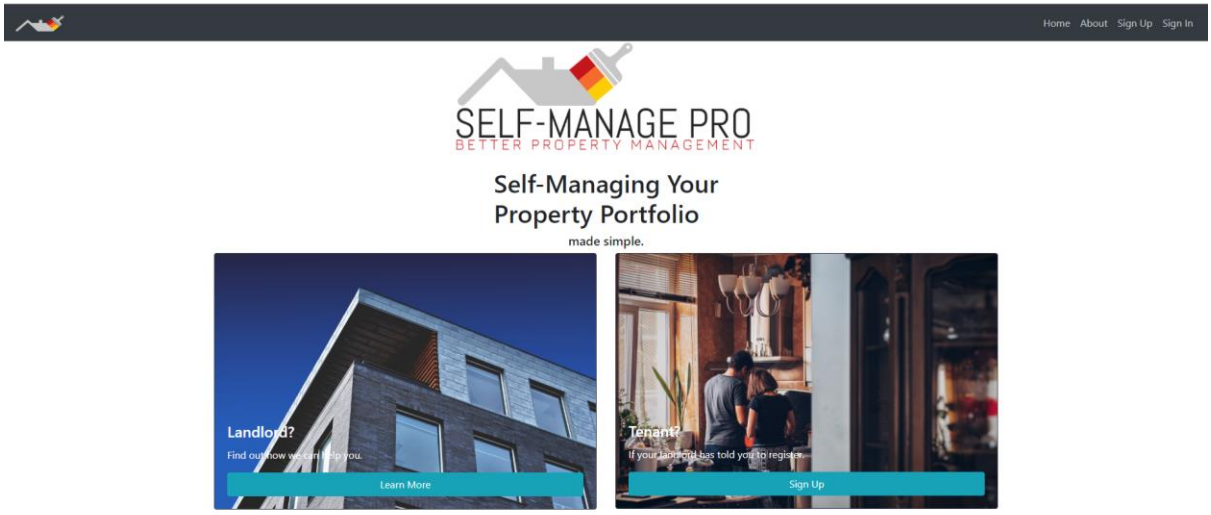


Figure 41 Static Home Page

Modal login form

The modal form was built to the specification of its low fidelity wireframe since it was received better by stakeholders than its high fidelity counterpart. It was made using bootstraps modal class and jQuery to make it appear and disappear.

There are three tabs along the top of the modal, created using a div with the nav-tabs class and using <li> and <a> tags within, these allow the user to select their user type by calling the setuType(type) JavaScript method in the click event handler for each element.

```
<ul class="nav nav-tabs">
  <li class="nav-item">
      <a id="ten" class="nav-link active" onclick="setuType('TENANT');" href="#">Tenant</a>
  </li>
  <li class="nav-item">
      <a id="lan" class="nav-link" href="#" onclick="setuType('LANDLORD');">Landlord</a>
  </li>
  <li class="nav-item">
      <a id="con" class="nav-link" href="#" onclick="setuType('CONTRACTOR');">Contractor</a>
  </li>
</ul>
<form id="logindform" action="<%=path%>/Login" method="POST">
```

Figure 42 JavaScript setuType()

setuType(type) is a function which takes a String, type, as a parameter and sets the value of the hidden form input #utype to that String using a switch statement. This is how the server is able to tell what type of user is trying to log in. The function also serves to add the class active to the clicked nav item and remove it from the from the other nav items. Screenshots of the function and the log in screen are shown below.

```
<script>
    function setuType(type) {
        $('#utype').val(type);
        switch (type) {
            case 'TENANT':
                $('#ten').addClass('active');
                $('#lan').removeClass('active');
                $('#con').removeClass('active');
                break;
            case 'LANDLORD':
                $('#ten').removeClass('active');
                $('#lan').addClass('active');
                $('#con').removeClass('active');
                break;
            case 'CONTRACTOR':
                $('#ten').removeClass('active');
                $('#lan').removeClass('active');
                $('#con').addClass('active');
                break;
        }
    }
</script>
```

Figure 43 Screenshot of setuType() function

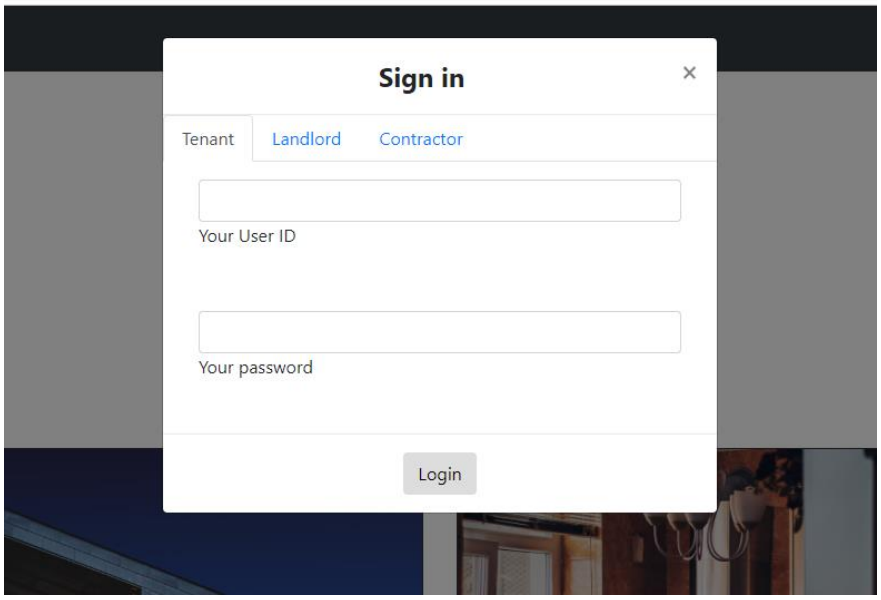


Figure 44 Screenshot of modal login

Landing Servlet

When a user connects to the web application, the server must identify them in order to serve the correct content. This is done within the function processRequest() in the servlet LandingServlet.java with the URL mapping "/" and is a three step process.

First, an HttpSession Object representing the current user session, session is created by calling the function request.getSession(). session Is then checked for a not null value before being checked if it is an old instance by calling if(!session.isNew())

If the session is not new then the servlet checks to see if there is a session attribute "type", and if there is, casts it as a String to the variable type.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    response.setContentType("text/html;charset=UTF-8");
    String type = "none";
    if (session != null) {
        if (!session.isNew()){
            if ((String) session.getAttribute("type") != null) {
                if (!("".equals((String) session.getAttribute("type")))){
                    type = (String) session.getAttribute("type");
                }
            }
        }
    }
```

Figure 45 Determining user type

Next, a PrintWriter object, out is declared, it is used by the servlet to print directly to the DOM. It is with this that the Landing page is constructed. out.print() is used to set up the page by making doctype and head declarations as well as four calls to external javascript libraries; Bootstrap and its three dependencies, one of which is jQuery. An opening Body tag is then printed to the DOM.

```
try (PrintWriter out = response.getWriter()) {
    out.print("<!DOCTYPE html>");
    out.print("<html>");
    out.print("<head>");
    out.print("<link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css\" 
    out.print("<script src=\"https://code.jquery.com/jquery-3.2.1.slim.min.js\" integrity=\"sha384-KJ3o2DKtIkvYIK3UENzm
    out.print("<script src=\"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js\" integrity=\"sh
    out.print("<script src=\"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js\" integrity=\"sha384-J
    out.print("<title>Self-Manage Pro</title>");
    out.print("</head>");
    out.print("<body>");
```

Figure 46 Page Dependencies

After the page and its dependencies have been set up, a switch statement is used to check the value of the session attribute, type. There are five different cases type could be- none; TENANT; LANDLORD; CONTRACTOR; default. Whatever the case, the servlet uses the RequestDispatcher Object, a class which can be used to add the content of a file to the response of the server; to assemble files together into one page. Different users will receive different versions of the navbar as well as their own versions of the landing pages.

The logic created in this servlet is the relevant content delivery system for the entire project. Every page which has multiple versions depending on the user type, has a servlet with this switch

statement and methodology inside to ensure that all users get the correct page version displayed. It is by using this logic that users are also prevented from accessing web pages they shouldn't be by URL.

```
switch (type){
    case "none":{
        request.getRequestDispatcher("WEB-INF/navbars/navbar.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/navbars/logo.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/landing/HomeNoUser.jsp").include(request, response);
        break;}
    case "TENANT":{
        request.getRequestDispatcher("WEB-INF/navbars/TenantNavbar.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/navbars/logo.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/landing/TenantWelcomePage.jsp").include(request,response);
        break;
    }
    case "LANDLORD":{
        request.getRequestDispatcher("WEB-INF/navbars/LandlordNavbar.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/navbars/logo.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/landing/LandlordWelcomePage.jsp").include(request,response);
        break;
    }
    case "CONTRACTOR":{
        request.getRequestDispatcher("WEB-INF/navbars/ContractorNavbar.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/navbars/logo.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/landing/ContractorWelcomePage.jsp").include(request,response);
        break;
    }
    default: {
        request.getRequestDispatcher("WEB-INF/navbars/navbar.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/navbars/logo.jsp").include(request, response);
        request.getRequestDispatcher("WEB-INF/landing/HomeNoUser.jsp").include(request, response);
        break;
    }

}
```

Figure 47 Servlet Relevant Content Delivery Switch Statement

The jsp pages delivered by the RequestDispatcher do not have any doctype or head declarations within since they are assembled into a single file with the declarations all taken care of by the PrintWriter in the previous step. Finally, after the switch statement, closing body and html tags are written using out.print() .

```
out.print("</body>");
out.print("</html>");
```

Navbar

Four different versions of the navbar were made according to the HF wireframes. One for Tenant, Contractor, Landlord and no user. These were made using bootstrap navbar-dark class on a div element and populating with navigation items inside a d-flex flex-row reverse div container which arranges nav elements from right to left on the right hand side of the navigation bar.

Static User Landing Pages

The landing pages were designed to the HF wireframes and are displayed to the user as the combination of the user groups' navbar, logo.jsp and the appropriate WelcomePage.jsp, delivered by the landing servlet. The tenant Landing page is shown below, see Appendix A for the Contractor and Landlord versions of the same page.

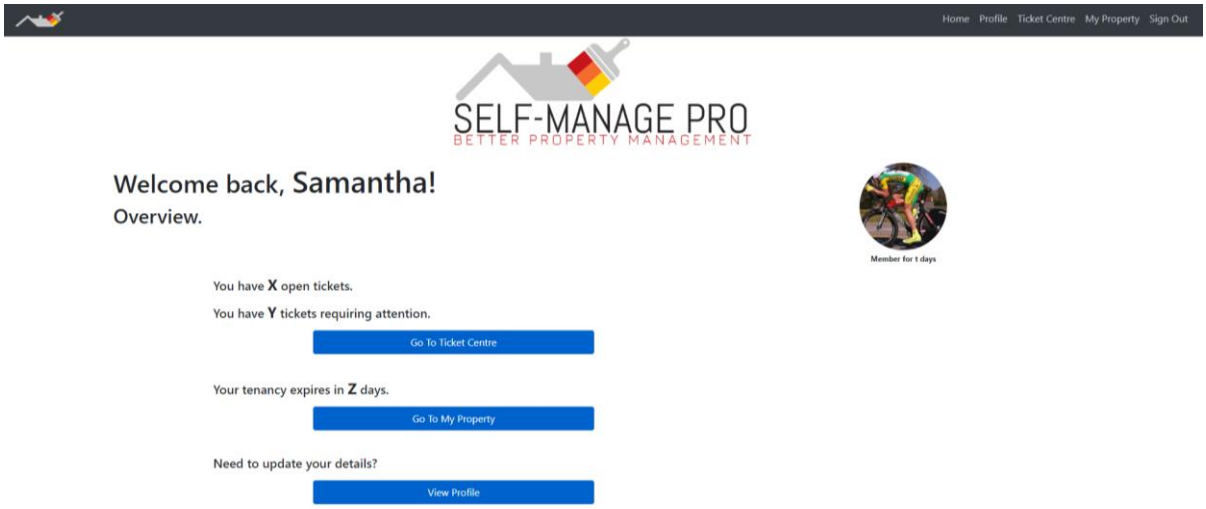


Figure 48 Tenant Landing Page

3.3.3 Sprint 2

Shown below are the tasks accomplished during Sprint 2, all tasks for Sprint 1 were completed so there was nothing to move back to the backlog. In this Sprint, the database was built to the specification of the Data Dictionary, as well as implementations using Connector /J to access and manipulate the data. The login system was then created and a static sign up form to match. A static version of ticket centre was also made.

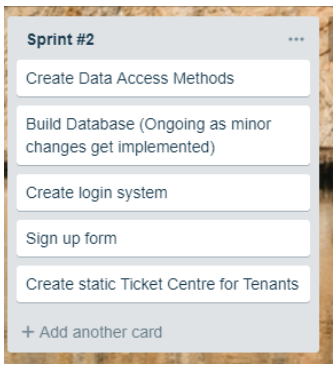


Figure 49 Sprint 2 tasks

Writing Data Access methods

The DataAccess.java controller class provides the JDBC interface for reading and writing data within the database. It contains a private Connection variable with a getter method, and a Prepared Statement with a getter and setter. The class contains the following functions:

1. openConnection()

Gets an instance of the JDBC DriverManager class and assigns the value of its .getConenction() method to the private connection variable using a connection String shown in the screenshot below

```
public void openConnection() {
    try {

        String url = "jdbc:mysql://kunet01.kingston.ac.uk:3306/db_kl423138";
        Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
        conn = DriverManager.getConnection(url, "kl423138", "disdatarent");
        if (conn != null) {
            System.out.println("Connected");
        } else {
            System.out.println("not connected");
        }
    } catch (Exception ex) {
        System.out.println("openConnection not run");
    }
}
```

Figure 50 openConnection()

2. getConn()

Returns the private connection variable

3. getPreparedStatement() & setPreparedStatement()

Getter and setter methods for the private Prepared Statement variable, which is set by calling .getConn().prepareStatement(SQL)

4. executeUpdate() & executeQuery()

Takes a Prepared Statement as a parameter by calling .getPreparedStatement(), a Result Set is returned if calling .executeQuery(), otherwise, void is returned for .executeUpdate()

```
public Boolean executeUpdate(PreparedStatement ps) {
    setPreparedStatement(ps);
    try {
        getPreparedStatement().executeUpdate();
        return true;
    } catch (SQLException ex) {
        Logger.getLogger(DataAccess.class.getName()).log(Level.SEVERE, null, ex);
        return false;
    }

}

public ResultSet executeQuery(PreparedStatement statement) {
    try {
        setResult(getStatement().executeQuery());
    } catch (Exception ex) {
        System.out.println(ex);
    }
    return getResult();
}
```

Figure 51 execute and update query methods

5. closeConnection()

Used to terminate the connection to the database in-between queries since leaving it open when not in use is not advised

6. tearDownAfterQuery()

Used to close and set to null the Connection and Prepared Statement variables

Login Functionality

The login Servlet uses the DataAccess class to communicate with the database, it also uses an instance of the Hasher class in the Model package, which returns an upper case, 64 character hash representation of a String using the SHA-256 algorithm.

The hasher class has two methods

1. generateSalt()

This method generates a three digit random number, converts it to a String and returns it

```
public String generateSalt(){
    String rtn = "";
    Integer i = (int)Math.floor(Math.random() * 999);
    if (i < 100){
        if (i < 10){
            rtn = "00"+i.toString();
        }else{
            rtn = "0"+i.toString();
        }

    }else{
        rtn = i.toString();
    }
    return rtn;
}
```

Figure 52 Generate Salt

2. getSHA(String hashThis)

A Message Digest variable for the SHA-256 algorithm, md is created by calling MessageDigest.getInstance("SHA-256"). hashThis is then converted to a byte array by using md.digest(hashThis.getBytes()). This array is then converted to a Big Integer and then a String. If the String is less than 64 characters long, it is left padded with zeroes and returned as upper case.

```
public String getSHA(String hashThis)
{
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256"); //Get Hashing Algorithm
        byte[] digest = md.digest(hashThis.getBytes()); //Convert string to byte array values
                                                        //using hashing algorithm
        BigInteger byteVal = new BigInteger(1, digest); //Convert byte value hash to hex
        String hash = byteVal.toString(16);             //convert hash to string

        while (hash.length() < 64) {                    //Left Pad with zeros if length < 64
            hash = "0" + hash;
        }
        return hash.toUpperCase();
    }
    catch (NoSuchAlgorithmException e) {
        System.out.println("Incorrect algorithm: " + e);
        return null;
    }
}
```

Figure 53 Hasher getSHA()

Depending on the user type selected, a Tenant, Contractor, or Landlord row is retrieved from the database using either the users email or ID.

```
ResultSet rs = null;

try{
    da.openConnection();
    user = request.getParameter("ID");
    pass = request.getParameter("password");
    type = request.getParameter("type");
    String sql = "SELECT * FROM "+type+" WHERE "+type+"ID = ? OR email = ?";
     ps = (da.getConn().prepareStatement(sql));
    ps.setString(1, user);
    ps.setString(2,user);
    da.setPreparedStatement(ps);
    rs = da.executeQuery(da.getPreparedStatement());
```

Figure 54 Fetch user record

The stored, hashed value for password is checked against the hash of the user entered password + stored salt. If the values are equal, session attributes for the userID and type are set and the user is redirected to the Landing page, else they are redirected to the home page.

```
        pass = pass+salt;
    }
} catch (SQLException ex) {
    Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);
}
pass = hasher.getSHA(pass);

if(pass.equalsIgnoreCase(passInDb)){
    session.setAttribute("ID", user);
    session.setAttribute("type", type);
    response.sendRedirect(request.getContextPath()+"/");
}else{
    response.sendRedirect(request.getContextPath()+"/?loginFailed=true");
}
```

Figure 55 Successful login

If the login is unsuccessful the parameter loginFailed=true is given in the URL which opens the navbar modal and prompts the user to try again.

```
<script type="text/javascript">
    $(document).ready(function () {
        $('#invalid').hide();
        <%if ((String) request.getParameter("loginFailed") != null) {
            if (((String) request.getParameter("loginFailed")).equals("true")) {%>
                $('#modalLoginForm').modal('show');
                $('#invalid').show();
            <%}
        }%>
    });
</script>
```

Figure 56 hide/show login

Sign up form

The sign up page contains three versions of the same form, the user selects their user type from the buttons at the top of the page and the relevant form is displayed. The screenshot below is of the Contractor form (See Appendix B for Tenant and Landlord forms). Tenants are displayed the columns on the left of the screenshot. Landlords and Contractors are displayed the columns on the right, additionally, contractors see the checkboxes at the top where they mark their qualified trades. A simple switch statement is used to display the correct version of the form. **Figure 57** (Left) Display correct sign up form

```
function setType(type) {
    $('.usertype').val(type);
    switch (type) {
        case 'TENANT':
            showTenantForm();
            break;
        case 'LANDLORD':
            showLandlordForm();
            break;
        case 'CONTRACTOR':
            showContractorForm();
            break;
    }
```

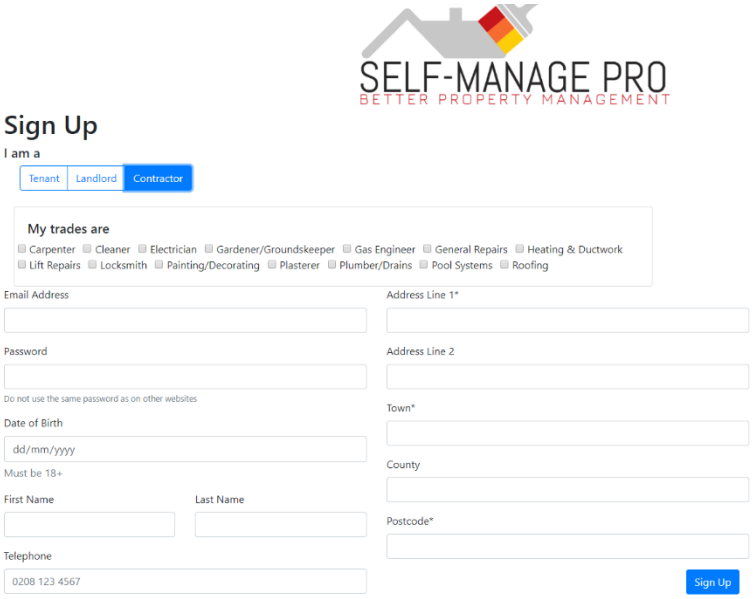


Figure 58 Contractor Sign Up Form

Ticket centre for tenants

The form shown below is where tenants will see tickets for their property and create new tickets. It was created using a Bootstrap container, columns and rows.

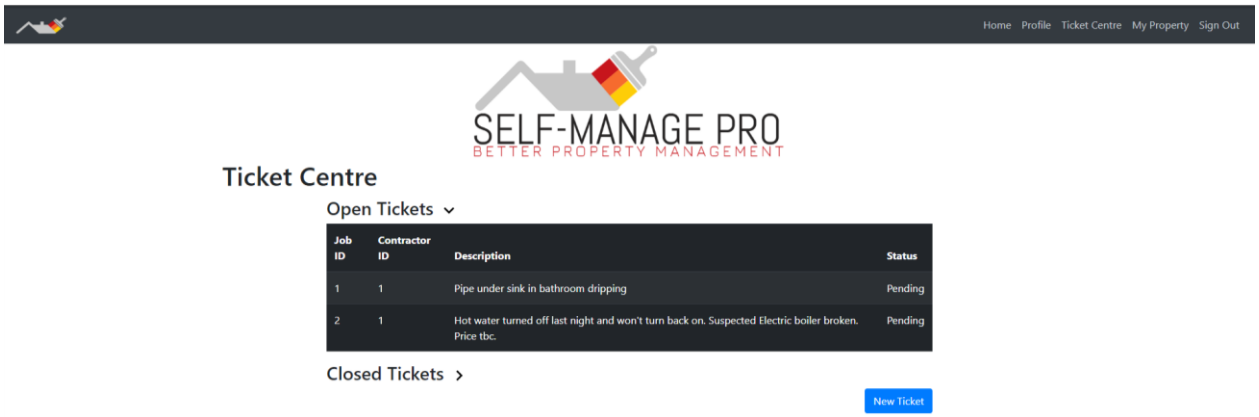


Figure 59 Tenant Ticket Centre

The open and closed ticket tables expand and collapse by calling openRowCollapser() and closedRowCollapser() respectively. Both functions have a Boolean which is true if open and false if collapsed. The .show() and .hide() methods are used to display and hide the div elements.

The Boolean is then set to its inverse.

```
var openRowOpen = true;
var closedRowOpen = false;
function openRowCollapser() {
    if (openRowOpen) {
        $('#openrow').hide(1000);
        $('#openimage').attr("src", collapsed);
    } else {
        $('#openrow').show(1000);
        $('#openimage').attr("src", expanded);
    }
    openRowOpen = !openRowOpen;
}
```

Figure 60 open and collapse rows

Building the Database

The database was built according to the specification of the ER diagram and consists of 13 tables which can be seen below. It was constructed using create table statements, entered into the SQL panel of phpMyAdmin. Below is a list of the created tables.

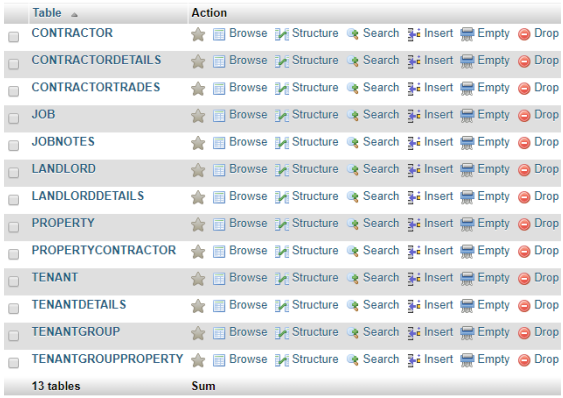


Figure 61 Created Tables

Table showing the Create Table SQL Statements used to build the database

Table	Create Table Statement
CONTRACTOR	CREATE TABLE `CONTRACTOR` (`CONTRACTORID` int(4) unsigned zerofill NOT NULL, `email` varchar(255) NOT NULL, `PWDHASH` char(64) NOT NULL, `SALT` char(3) NOT NULL);
CONTRACTORDETAILS	CREATE TABLE `CONTRACTORDETAILS` (`CDNUM` int(4) unsigned zerofill NOT NULL, `CONTRACTORID` int(4) unsigned zerofill NOT NULL, `FIRST` varchar(50) NOT NULL, `LAST` varchar(50) NOT NULL, `DOB` date NOT NULL, `TELEPHONE` char(11) NOT NULL, `ADDRESS1` varchar(255) NOT NULL,`ADDRESS2` varchar(255) DEFAULT NULL, `TOWN` varchar(35) NOT NULL, `COUNTY` varchar(35) DEFAULT NULL, `POSTCODE` varchar(10) NOT NULL, `CREATED` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP);
CONTRACTORTRADES	CREATE TABLE `CONTRACTORTRADES` (`CTID` int(5) NOT NULL, `CONTRACTORID` int(4) unsigned zerofill NOT NULL, `Carpenter` tinyint(1) NOT NULL DEFAULT '0', `Cleaner` tinyint(1) NOT NULL DEFAULT '0', `Electrician` tinyint(1) NOT NULL DEFAULT '0', `Gardener` tinyint(1) NOT NULL DEFAULT '0', `GasEngineer` tinyint(1) NOT NULL DEFAULT '0', `Gen` tinyint(1) NOT NULL DEFAULT '0', `Heating` tinyint(1) NOT NULL DEFAULT '0', `Lift` tinyint(1) NOT NULL DEFAULT '0', `Locksmith` tinyint(1) NOT NULL DEFAULT '0', `Painter` tinyint(1) NOT NULL DEFAULT '0', `Plasterer` tinyint(1) NOT NULL DEFAULT '0', `Plumber` tinyint(1) NOT NULL DEFAULT '0', `Pool` tinyint(1) NOT NULL DEFAULT '0', `Roof` tinyint(1) NOT NULL DEFAULT '0');
JOB	CREATE TABLE `JOB` (`JOBID` int(6) unsigned zerofill NOT NULL, `PROPERTYID` int(4) unsigned zerofill NOT NULL, `CONTRACTORID` int(4) unsigned zerofill DEFAULT NULL, `TRADE` varchar(30) NOT NULL, `DESCRIPTION` varchar(250) NOT NULL, `COST` decimal(6,2) DEFAULT NULL, `STATUS` varchar(20) NOT NULL, `CREATED` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP, `COMPLETED` date DEFAULT NULL);
JOBNOTES	CREATE TABLE `JOBNOTES` (`JOBNOTEID` int(7) unsigned zerofill NOT NULL, `JOBID` int(6) unsigned zerofill NOT NULL, `LEFTBYTYPE` varchar(10) NOT NULL, `LEFTBYID` varchar(6) NOT NULL, `NOTE` mediumtext NOT NULL, `CREATED` datetime NOT NULL DEFAULT CURRENT TIMESTAMP);
LANDLORD	CREATE TABLE `LANDLORD` (`LANDLORDID` int(5) unsigned zerofill NOT NULL, `email` varchar(255) NOT NULL, `PWDHASH` char(64) NOT NULL, `SALT` char(3) NOT NULL);
LANDLORDDETAILS	CREATE TABLE `LANDLORDDETAILS` (`LDNUM` int(5) unsigned zerofill NOT NULL, `LANDLORDID` int(5) unsigned zerofill NOT NULL, `FIRST` varchar(50) NOT NULL, `LAST` varchar(50) NOT NULL, `DOB` date NOT NULL, `TELEPHONE` char(11) NOT NULL, `ADDRESS1` varchar(255) NOT NULL, `ADDRESS2` varchar(255) DEFAULT NULL, `TOWN` varchar(35) NOT NULL, `COUNTY` varchar(35) DEFAULT NULL, `POSTCODE` varchar(10) NOT NULL, `CREATED` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP);
PROPERTY	CREATE TABLE `PROPERTY` (`PROPERTYID` int(4) unsigned zerofill NOT NULL, `LANDLORDID` int(5) unsigned zerofill NOT NULL, `TYPE` varchar(30) NOT NULL, `BEDROOMS` int(2) NOT NULL, `ADDRESS1` varchar(255) NOT NULL, `ADDRESS2` varchar(255) DEFAULT NULL, `TOWN` varchar(35) NOT NULL, `COUNTY` varchar(35) DEFAULT NULL, `POSTCODE` varchar(10) NOT NULL);
PROPERTYCONTRACTOR	CREATE TABLE `PROPERTYCONTRACTOR` (`PCID` int(4) unsigned zerofill NOT NULL, `PROPERTYID` int(4) unsigned zerofill NOT NULL, `CONTRACTORID` int(4) unsigned zerofill NOT NULL);
TENANT	CREATE TABLE `TENANT` (`TENANTID` int(6) unsigned zerofill NOT NULL, `email` varchar(255) NOT NULL, `PWDHASH` char(64) NOT NULL, `SALT` char(3) NOT NULL);
TENANTDETAILS	CREATE TABLE `TENANTDETAILS` (`TDNUM` int(6) unsigned zerofill NOT NULL, `TENANTID` int(6) unsigned zerofill NOT NULL, `FIRST` varchar(30) NOT NULL, `LAST` varchar(30) NOT NULL, `DOB` date NOT NULL, `TELEPHONE` char(11) NOT NULL, `CREATED` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP);
TENANTGROUP	CREATE TABLE `TENANTGROUP` (`TGID` int(5) unsigned zerofill NOT NULL, `TENANT1` int(6) unsigned zerofill NOT NULL, `TENANT2` int(6) unsigned zerofill DEFAULT NULL, `TENANT3` int(6) unsigned zerofill DEFAULT NULL, `TENANT4` int(6) unsigned zerofill DEFAULT NULL, `TENANT5` int(6) unsigned zerofill DEFAULT NULL, `TENANT6` int(6) unsigned zerofill DEFAULT NULL, `TENANT7` int(6) unsigned zerofill DEFAULT NULL, `TENANT8` int(6) unsigned zerofill DEFAULT NULL, `TENANT9` int(6) unsigned zerofill DEFAULT NULL, `TENANT10` int(6) unsigned zerofill DEFAULT NULL, `TENANT11` int(6) unsigned zerofill DEFAULT NULL, `TENANT12` int(6) unsigned zerofill DEFAULT NULL);
TENANTGROUPPROPERTY	CREATE TABLE `TENANTGROUPPROPERTY` (`TGPID` int(4) unsigned zerofill NOT NULL, `TGID` int(5) unsigned zerofill NOT NULL, `PROPERTYID` int(4) unsigned zerofill NOT NULL);

For all ALTER statements used to set auto increment values, primary and foreign keys, see Appendix C.

After sprint 2, all tasks were complete with the exception of building the database which may still have minor changes made as the project develops. It is treated as completed and does not move back to the backlog.

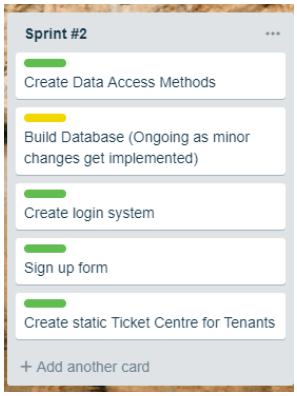


Figure 62 Sprint 2 Complete

3.3.1 Sprint 3

During this sprint, the database was populated with sample data for testing the application with. The Sign Up servlet was created in the Controller package, static job creation and property set-up pages were made and SQL was designed for creating and retrieving tickets from the database.

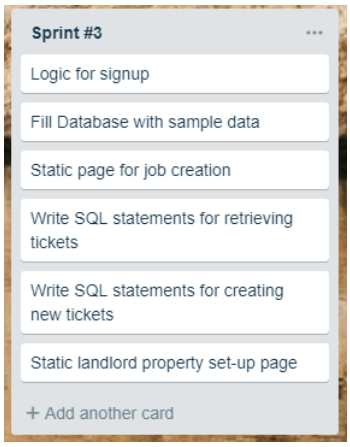


Figure 63 Sprint 3

Populating database

The database was filled with test data, creating numerous credentials and matching details for contractors, landlord and tenants. Records and their associations were created for Contractor Trades, Properties, Tenant Groups. A screenshot of sample data is shown below (More in Appendix D).

TDNUM	TENANTID	FIRST	LAST	DOB	TELEPHONE
000001	000001	Samantha	Clarkson	1996-08-13	07485666214
000002	000002	Jeremiah	Springer	1995-12-18	07268425799
000003	000003	Richard	Banks	1972-09-01	07554689450
000005	000005	Tom	Testah	1994-11-27	07845145555
000006	000006	Lionel	Testy	1998-09-25	07552849625
000007	000007	Jeremy	Scottest	1993-01-22	02086576660
000008	000008	Greg	Testington	1994-09-19	07562425890
000009	000009	Jill	Tesla	1962-02-12	01548652999
000010	000010	Emma	Gency	1979-12-08	01489855475
000011	000011	Larry	Greyson	1991-02-06	07458256943
000012	000012	Nilah	Tset	1954-07-19	07831464648
000013	000013	Emma	Stone	1980-04-02	07888452146
000014	000014	Marc	Rees	1983-10-18	07485662148
000015	000015	Rhys	Farmer	1990-08-21	07062240367
000016	000016	Corey	Singh	1984-04-19	07086861904
000017	000017	Megan	Lamb	1970-08-26	07754679605

Figure 64 Sample Tenant Details Data

Ticket Creator

The static ticket creator page for tenants was made to meet the design in the HF wireframe using Bootstrap classes for styling the document. The form is contained within a card element which is centred in the page and has a dropdown #trade, for selecting the required trade and a text area for the job description, shown below.

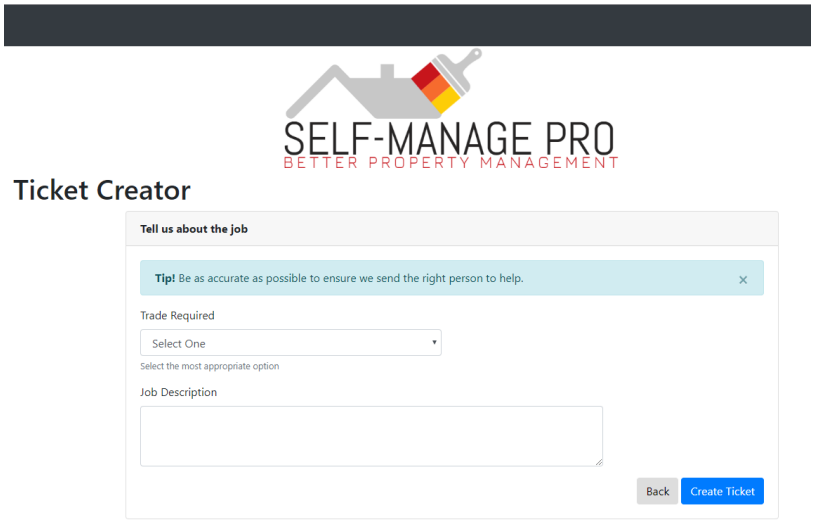


Figure 65 Static Tenant Ticket Creator form

The landlord ticket creation form has an additional field, a dropdown #tradesman, which appears once the user selects a trade from the trade dropdown. If the user selects 'Select One' then #tradesman will be disabled, otherwise, it will eventually be populated with a list of contractors specific to the property and trade in question. Additionally, since a landlord may create tickets for

multiple properties, the first line of address will be displayed by calling <%=property.getAddress1()%> once the Property object is implemented.

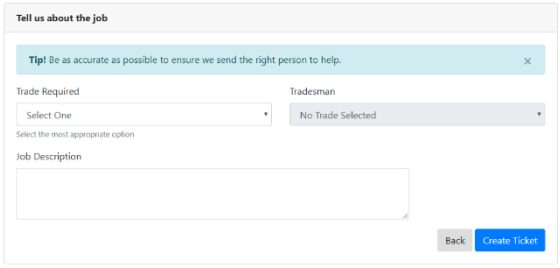


Figure 66 (right) Ticket creation form showing additional disabled dropdown input for landlord

Static page for landlord property creation

The property creation page was implemented as per the specification in the wireframes. Each section of the form has a button linking to the next step within the same page. The button onclick events are bound to JavaScript functions which utilize the jQuery .show() and .hide() methods to display each section of the form.

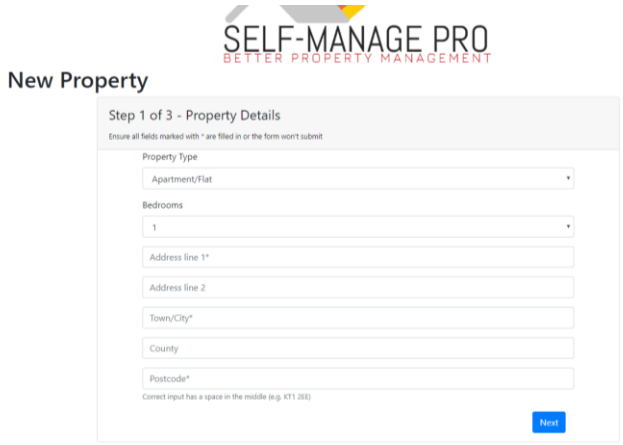


Figure 67 New property form – Step 1

In the second part of the form, the user will enter the email addresses of their tenants who will already have accounts, the form will eventually dynamically lookup tenants and allow the landlord to select and add them to the property

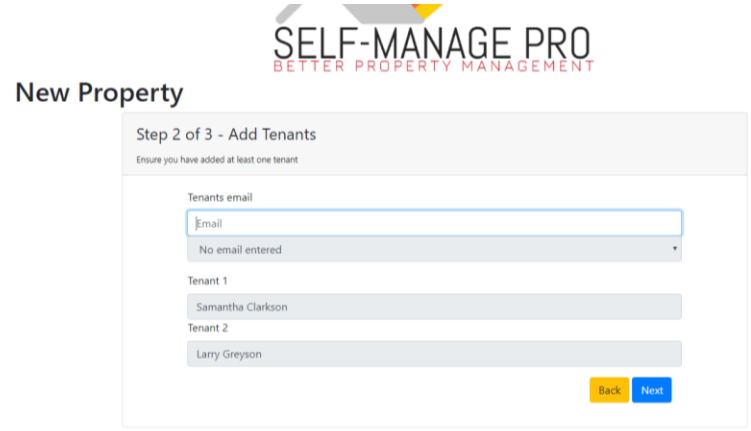


Figure 68 New property form – Step 2

Part three, the user will enter the email addresses of their contractors who will also already have set up accounts. As with the previous step, the form will dynamically load any contractors matching the entry; the landlord will select them from a dropdown and they will be added to the property

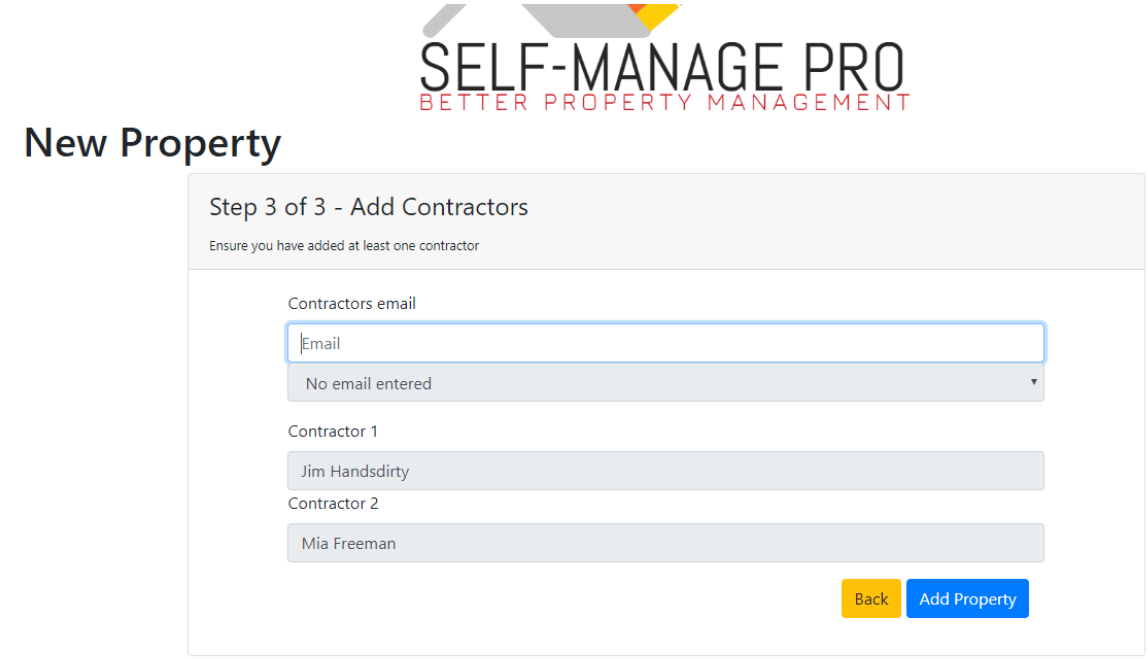


Figure 69 New Property Form - Step 3

Mid-sprint review

Since the database has been filled with sample data for tenants, contractors and landlords however their POJO equivalents have not yet been written, it does not make sense to write SQL for the creation and retrieval of tickets as was originally planned.

Instead, AJAX methods will be written to populate the tenants and contractors field for the landlord property creation form. Tasks cards have been added to sprint 3 to reflect this as well as writing the SQL for property creation; a more logical order of tasks.

Shown right, is the updated sprint 3 card with the new task cards

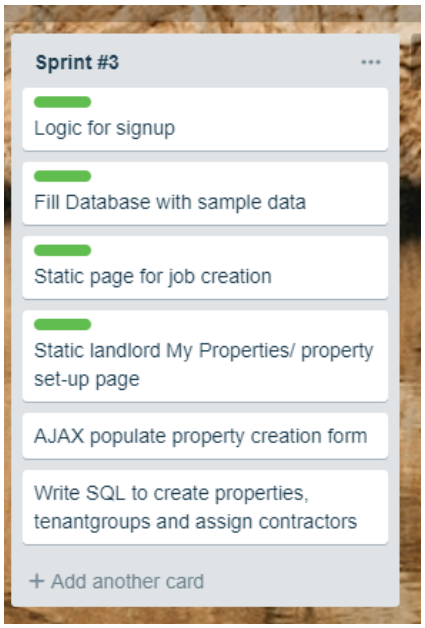


Figure 70 Sprint 3

AJAX tenant lookup

AJAX stands for asynchronous JavaScript and XML, it is a method of updating pages with information from a server request without having to reload the page. This was used for getting tenants and contractors from the database to populate the property creation form.

The onkeyup event handler for the email input box #email is bound to the JavaScript function searchTenants(). It is called every time a key press is registered within the input box.

```
var request = new XMLHttpRequest();
function searchTenants() {
    $('#tens').show();
    var email = $('#email').val();
    if (email === "")
            email = "none";
    var url = "getTenant.jsp?val=" + email;
    try {
        request.onreadystatechange = function () {
            if (request.readyState === 4) {
                var val = request.responseText;
                document.getElementById('tenants').innerHTML = val;
            }
        };
        request.open("GET", url, true);
        request.send();
    } catch (e) {
        alert("Unable to connect to server");
    }
}
```

An XMLHttpRequest object is created which is used to call code within the jsp page getTenant.jsp and output the response from the request to the innerHTML property of the div element #tenants.

#tenants is nested within a hidden div element #tens, which is made visible then the function is first called.

searchTenants() Javascript function

Figure 71 (Above) searchTenants()

The following SQL query was designed to return any tenants details where the email address associated with the account contains the value entered by the user at any point

SELECT TENANT.TENANTID,FIRST,LAST FROM `TENANTDETAILS` INNER JOIN TENANT ON TENANTDETAILS.TENANTID = TENANT.TENANTID AND TENANT.email LIKE '%" + email + "%'.

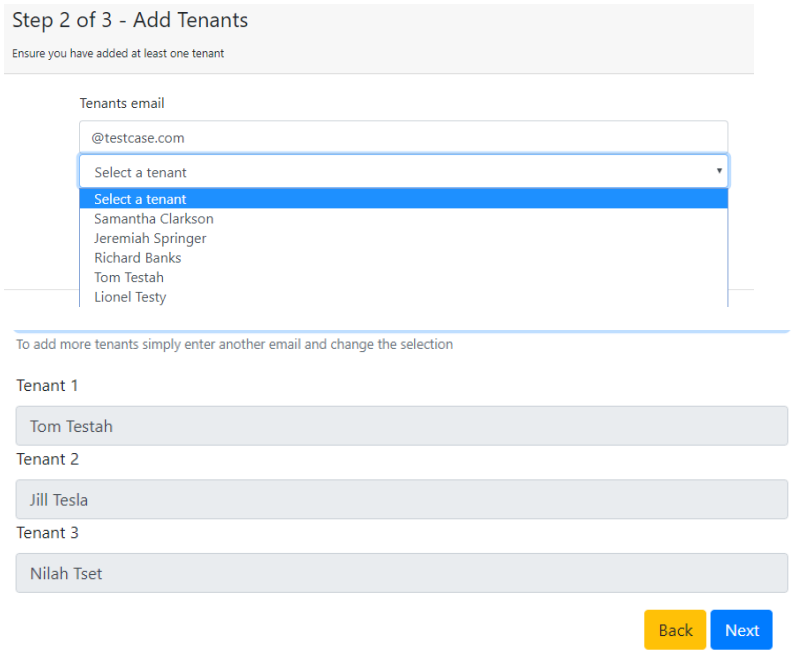
```
String email = request.getParameter("val");
if (!email.equalsIgnoreCase("none")) {
    try {
        DataAccess da = new DataAccess();
        da.openConnection();//query needs changing to the one at the bottom once more properties are set up with contractors
        PreparedStatement ps = da.getConn().prepareStatement("SELECT TENANT.TENANTID,FIRST,LAST FROM `TENANTDETAILS` INNER JOIN "
                + "TENANT ON TENANTDETAILS.TENANTID = TENANT.TENANTID AND TENANT.email LIKE '%" + email + "%';");
        da.setPreparedStatement(ps);
        ResultSet rs = da.executeQuery(ps);

        if (!rs.isBeforeFirst()) {
            out.println("<p>No Record Found!</p>");
        } else {
            out.print("<select onchange=\"addInput('selectedTenants');\" name=\"tenant\" id=\"tenant\" class=\"form-control\">");
            out.print("<option value=\"none\">Select a tenant</option>");
            while (rs.next()) {
                out.print("<option value=\"" + rs.getString(1) + " \">" + rs.getString(2) + " " + rs.getString(3) + "</option>");
            }
            out.print("</select>");
            out.print("<small id=\"emailHelper\" class=\"text-muted\">To add more tenants simply enter another email and change the s
        }
```

Figure 72 getTenant.jsp

The code above shows code within getTenant.jsp and how an instance of the Data Access class is created and a Prepared Statement used to query the server for any close matches to the email

address; passed to the page from the XMLHttpRequest within the searchTenants(), and return the list of possible tenants. A loop is used to populate the content of #tenants.

AJAX methods were then used to dynamically display the results in a select box within #tens, which, when selected, adds the tenant to an array and a hidden input is also created holding the ID of the tenant alongside a disabled input box displaying the tenant's name to illustrate that the landlord has selected the correct tenant.

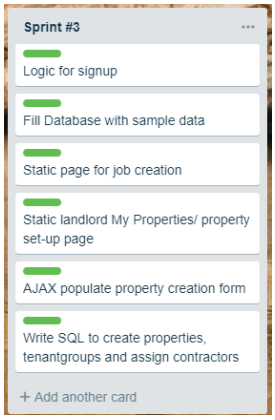


Figures 73 & 74 Populated list of possible tenants & Tenants successfully selected

AJAX Contractor lookup

The system being built is a tool for simplifying the day to day maintenance management of landlords. With this in mind, the landlord should already have contractors which they uses pre-system. The landlord should therefore know the email addresses of their contractors and will have instructed them to create accounts on the system. Using a similar method to the tenant lookup above, the following SQL query was designed to populate the contractor section of the property creation form

```
SELECT CONTRACTOR.CONTRACTORID,FIRST,LAST FROM `CONTRACTORDETAILS` INNER JOIN CONTRACTOR ON CONTRACTORDETAILS.CONTRACTORID = CONTRACTOR.CONTRACTORID AND CONTRACTOR.email LIKE '%"+email+"%'
```



All tasks for Sprint 3 were marked as complete and the planning for Sprint 4 commenced.

Figure 75 Sprint 3 Completed

3.3.1 Sprint 4

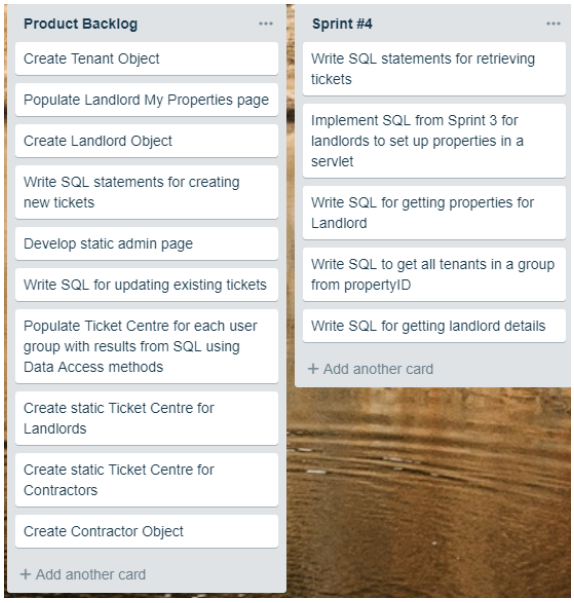


Figure 76 Sprint 4 cards and backlog

During Sprint 4, a lot of the SQL queries were designed that they could be implemented to provide functionality to the website in future sprints.

With the new property form complete, SQL to add the property, its tenants and contractors to the database was constructed alongside a servlet to process the form parameters and execute the queries using the previously written Data Access methods.

The screenshot below shows the first of four queries which was executed.

```
try {
    String propertySQL = "INSERT INTO PROPERTY"
            + "(LANDLORDID, TYPE, BEDROOMS, ADDRESS1, ADDRESS2, TOWN, COUNTY, POSTCODE) VALUES"
            + "(?,?,?,?,?,?,?,?)";
    PreparedStatement ps = null;
    ps = this.getConn().prepareStatement(propertySQL, new String[]{"PROPERTYID"});
    ps.setString(1, landlordID);
    ps.setString(2, propertyType);
    ps.setString(3, bedrooms);
    ps.setString(4, address1);
    ps.setString(5, address2);
    ps.setString(6, town);
    ps.setString(7, county);
    ps.setString(8, postcode);
    this.setPreparedStatement(ps);
    this.executeUpdate(this.getPreparedStatement());
    ResultSet rs = this.getPreparedStatement().getGeneratedKeys();
    if (rs.next()) {
        propertyID = rs.getString(1);
    }
} catch (SQLException ex) {
    Logger.getLogger(DataAccess.class.getName()).log(Level.SEVERE, null, ex);
    System.out.println("Add Property Failure");
    return false;
} finally {
```

Figure 77 New Property SQL

In the call to prepareStatement, a second parameter was given as new String[]{“PROPERTYID”}, this was used to get the automatically generated property ID which would then be used as a foreign key to link the relational entries in the TENANTGROUPPROPERTY and PROPERTYCONTRACTOR tables.

Upon successful addition of data to the PROPERTY table, selected tenants are added to a TENANTGROUP row. Since there can be between 1 and 12 tenants per property, the hidden input parameter tencount was used to store the number of tenants selected by the landlord.

This was then used in a loop to add the parameters tenant + (count <= tencount) to an ArrayList storing all of the selected tenant IDs. A StringBuilder was used to generate the query which would then be executed, shown below.

```
} finally {
    if (propertyAdded) {
        //Generate SQL for adding tenants
        StringBuilder sb = new StringBuilder();
        String tenantSQL = "INSERT INTO TENANTGROUP(";
        sb.append(tenantSQL);
        for (Integer c = 1; c <= tenants.size(); c++) {
            sb.append("TENANT");
            sb.append(c);
            if (c < tenants.size()) {
                sb.append(",");
            }
        }
        sb.append(") VALUES(");
        for (Integer c = 1; c <= tenants.size(); c++) {
            sb.append("?");
            if (c < tenants.size()) {
                sb.append(",");
            }
        }
        sb.append(")");
        tenantSQL = sb.toString();
```

Figure 78 New TenantGroup

After creating the row in TENANTGROUP, a relational entry is added to TENANTGROUPPROPERTY, a table which links groups of tenants to the property being created by the landlord. To do this, the PROPERTYID and Tenant Group ID (TGID) were used. The TGID was acquired in the same way as the PROPERTYID from the from the PROPERTY table, by using getGeneratedKeys() on the PrepareStatement() method.

```
PreparedStatement ps = null;
ps = this.getConn().prepareStatement(tenantSQL, new String[]{"TGID"});
Integer count = 1;
for (String s : tenants) {
    ps.setString(count, s);
    count++;
}
this.setPreparedStatement(ps);
this.executeUpdate(this.getPreparedStatement());
ResultSet rs = this.getPreparedStatement().getGeneratedKeys();
if (rs.next()) {
    tenantGroupID = rs.getString(1);
}
```

Figure 79 Getting TGID

The following query was used to make the addition to the TENANTGROUPPROPERTY table "INSERT INTO TENANTGROUPPROPERTY (TGID, PROPERTYID) VALUES (?,?)". The values were then set by calling the .setString() method on the Prepared Statement which was then executed.

The final step to add the property to the database was to get the contractors selected from the New Property form, this was done in the same way as getting the tenants in the previous step, with a hidden input counter and textboxes displaying the name of the contractor and the value set to the Contractor's ID.

The query, however, was not created using a string builder as the Tenant Group (TG) was. Where the TG stores all tenants associated with a property in one line as a group, Contractors and their property associations are stored as multiple lines in the PROPERTYCONTRACTOR table to make swapping out contractors in the future straightforward.

A for loop was used to loop through each contractorID String stored in contractors and for each iteration, the query "INSERT INTO PROPERTYCONTRACTOR (PROPERTYID, CONTRACTORID) VALUES (?,?)" was used, again using the .setString() method on the Prepared Statement object; setting the PROPERTYD to the previously asserted value and the CONTRACTORID to the value of the current String in the loop.

```
if (tenantGroupPropertyAdded) {
    String propertyContractorSQL = "INSERT INTO PROPERTYCONTRACTOR"
            + "(PROPERTYID, CONTRACTORID) VALUES"
            + "(?,?)";
    for (String con : contractors) {
        try {
            PreparedStatement ps = null;
            ps = this.getConn().prepareStatement(propertyContractorSQL);
            ps.setString(1, propertyID);
            ps.setString(2, con);
            this.setPreparedStatement(ps);
            this.executeUpdate(this.getPreparedStatement());
            propertyContractorAdded = true;
        } catch (SQLException ex) {
            Logger.getLogger(DataAccess.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error adding to PROPERTYCONTRACTOR");
            return false;
        }
```

Figure 80 Inserting Property Contractor

Other SQL statements were designed during this sprint, shown in the table below; their functionality includes getting a landlords details and all properties they own by LandlordID; getting all jobs stored for a property with a specific ID; Getting a tenants TENANTGROUPID; using the Tenants Group ID to get the ID of their property; assigning tenants details by their ID, getting a list of tenants belonging to a property by their propertyID and finally, getting all jobs for a property by its ID.

Query no	Function	Query
1	Gets the details of a landlord by their ID	SELECT * FROM LANDLORDDETAILS WHERE LANDLORDID = ?
2	Select all properties owned by a specific landlord	SELECT * FROM PROPERTY WHERE LANDLORDID = ?
3	Select all jobs for a specific property	SELECT * FROM JOB WHERE PROPERTYID = ?
4	Select the tenant group a user appears in	SELECT * FROM TENANTGROUP WHERE ? IN TENANT1, TENANT2, TENANT3, TENANT4, TENANT5, TENANT6, TENANT7, TENANT8, TENANT9, TENANT10, TENANT11, TENANT12
5	Select the row in TENANTGROUPPROPERTY where the Tenant Group ID is equal to a certain value	SELECT * FROM TENANTGROUPPROPERTY WHERE TGID = ?
6	Gets the details of a specific tenant by their ID	SELECT * FROM TENANTDETAILS WHERE TENANTID = ?
7	Gets the group of tenants associated with a tenant group using the ID of a property and an entry within the table TENANTGROUPPROPERTY	SELECT * FROM TENANTGROUP INNER JOIN TENANTGROUPPROPERTY ON TENANTGROUPPROPERTY.TGID = TENANTGROUP.TGID AND TENANTGROUPPROPERTY.PROPERTYID = ?

All tasks for Sprint 4 were completed successfully, this is shown below.

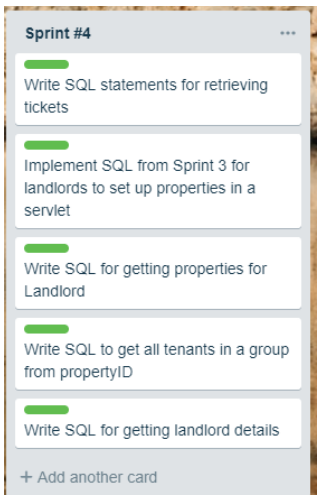


Figure 81 Sprint 4 tasks complete

3.3.2 Sprint 5

During sprint 5, Tenant and Landlord class files were written so that details on the otherwise static pages could start being filled in. The My Properties page was populated with data and the static Ticket Centre for landlords was written. These tasks were added to the Sprint #5 card from the product backlog, shown right.

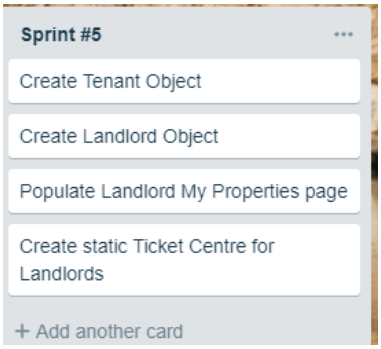


Figure 82 Sprint 5

Tenant Object

The Tenant object is a final Java object representation of information stored about a tenant within the database. It has 7 private attributes, shown on the right. Final means that the values of these attributes are set when they are initialized and then not changed. Each attribute has a getter method to retrieve its value.

```
public final class Tenant{
    private String tenantID;
    private String tenantGroupID;
    private String propertyID;
    private ArrayList<Job> allJobs;
    private ArrayList<Job> closedJobs;
    private ArrayList<Job> openJobs;
    private TenantDetails tenantDetails;
```

Figure 83 Tenant Class Attributes

Figure 84 Tenant Constructor

```
public Tenant(String tenantID){
    this.tenantID = tenantID;
    this.setTenantGroupID();
    this.setPropertyID();
    this.setTenantDetails();
    this.setAllJobs();
    this.setOpenAndClosedJobs();
```

The class has one constructor method, which takes String tenantID as it's only parameter. tenantID is then used to populate the other attributes using the classes setter methods.

Query 4 from Sprint 4 is used when .setTenantGroupID() is called, by utilizing an instance of the Data Access controller and setting the tenantGroupID attribute to the returned value.

tenantGroupID is then used by setPropertyID in conjunction with Query 5 to set the ID of the property to which the tenant belongs. TenantDetails is an Object which holds a Tenants full name, date of birth and telephone number, it has basic getters, setters and constructor methods. The tenants ID is used with query 6 to and .setTenantDetails() to set the Tenant Details attribute.

With the property ID attribute set, the constructor calls the method .setAllJobs()which uses propertyID and query 3 in order to get the list of all jobs associated with property.

The ResultSet returned by query 3 is iterated over using while (rs.next()), the values in each row are read into a Job object, which is then added to a temporary Array List which becomes the assigned value of the allJobs Array List. This is shown below:

Figure 85 Setting allJobs

```
ArrayList<Job> jList = new ArrayList<>();
while (rs.next()){
    Job j = null;
    String jobID = rs.getString("JOBID");
    String propertyID = rs.getString("PROPERTYID");
    String contractorID = rs.getString("CONTRACTORID");
    String trade = rs.getString("TRADE");
    String description = rs.getString("DESCRIPTION");
    Double cost = rs.getDouble("COST");
    String status = rs.getString("STATUS");
    Date created = rs.getDate("CREATED");
    Date completed = rs.getDate("COMPLETED");
    j = new Job(jobID,propertyID,contractorID,trade,description,cost,status,created,completed);
    jList.add(j);
}
this.allJobs = jList;
```

allJobs is then looped over within the .setOpenAndClosedJobs() method which checks the status of each job, and adds it to a temporary Array List, closedTemp or openTemp. These Array lists are then assigned to closedJobs and openJobs respectively.

```
public void setOpenAndClosedJobs(){
    ArrayList<Job> closedTemp = new ArrayList<>();
    ArrayList<Job> openTemp = new ArrayList<>();
    if (this.getAllJobs().size() > 0){
        for (Job j : this.getAllJobs()){
            if(j.getStatus().equalsIgnoreCase("closed")){
                closedTemp.add(j);
            }else{
                openTemp.add(j);
            }
        }
    }
    this.closedJobs = closedTemp;
    this.openJobs = openTemp;
}
```

Figure 86 setOpenAndClosedJobs()

Landlord Object

The landlord class is structured in a very similar way to Tenant. Instead of a Tenant Details object, it has a Landlord Details object, which contains the users address additionally. Queries 1 and 2 from Sprint 4 are used to set the list of properties the landlord owns as well as getting the landlord details. The landlord object is created when at page loads when a landlord is logged in.

```
public final class Landlord{
    private String landlordID;
    private ArrayList<Property> properties;
    private LandlordDetails landlordDetails;

    public Landlord(String ID){...5 lines }
    public String getLandlordID(){...3 lines }

    public LandlordDetails getLandlordDetails(){...3 lines }
    public ArrayList<Property> getProperties(){...3 lines }
    public void setLandlordDetails(){...54 lines }
    public void setLandlordID(String landlordID){...3 lines }
    public void setProperties(){...35 lines }

}
```

Figure 87 Landlord Class

Job Class

The Job class was created to store details about jobs which are stored inside an ArrayList in the Tenant class. The Job class also holds an Array List jobNotes which stores the Job Notes object, a simple object which stores details from the table JOBNOTES. The Job class is shown on the right.

It has two constructor methods, one to initialize the object with a Contractor, and one without.

```
public class Job {

    private String jobID;
    private String propertyID;
    private String contractorID;
    private Contractor contractor;
    private String trade;
    private String description;
    private Double cost;
    private String status;
    private Date created;
    private Date completed;
    private ArrayList<JobNote> jobNotes;

    public Job(String ID) {...4 lines }
    public Job(String jid, String pid, String cid,String trad
    public Job(String jid, String pid, String trade, String d
    public void setJobDetails(String ID) {...29 lines }

    public void setJobNotes() {...27 lines }

    public Contractor getContractor() {...3 lines }

    public String getJobID() {...3 lines }
    public String getPropertyID() {...3 lines }

    public String getContractorID() {...3 lines }
    public String getTrade() {...3 lines }
    public String getDescription() {...3 lines }

    public Double getCost() {...3 lines }

    public String getStatus() {...3 lines }

    public Date getCreated() {...3 lines }

    public Date getCompleted() {
        return completed;
    }
    public ArrayList<JobNote> getJobNotes() {...3 lines }
```

Figure 88 Job Class

Property Class

The property class is used to store information about individual properties. The Landlord class has an ArrayList of Property objects which are displayed on the My Properties page. The class has a constructor which takes propertyID as a parameter which is then used in the query "SELECT TYPE,BEDROOMS,ADDRESS1,ADDRESS2,TOWN,COUNTY,POSTCODE FROM PROPERTY WHERE PROPERTYID = ?" The results of which are used to set the Objects attributes (Shown right).

```
public final class Property {
    private String propertyID;
    private String type;
    private String bedrooms;
    private String address1;
    private String address2;
    private String town;
    private String county;
    private String postcode;
    private ArrayList<Job> jobList;
    ArrayList<Job> closedJobs = new ArrayList<>();
    ArrayList<Job> openJobs = new ArrayList<>();
    private ArrayList<Tenant> tenantList;
```

Figure 89 Property class attributes

tenantList is populated by running the query “SELECT * FROM TENANTGROUP INNER JOIN TENANTGROUPPROPERTY ON TENANTGROUPPROPERTY.TGID = TENANTGROUP.TGID AND TENANTGROUPPROPERTY.PROPERTYID = ?”. The results of which are looped over and to create tenants which are put in the Array List.

```
while (rs.next()){
    Tenant t = null;
    for (Integer i = 2; i <= 13; i++){
        if(rs.getString(i) != null){
            t = new Tenant(rs.getString(i));
            tempTenantList.add(t);
        }
    }
}
this.tenantList = tempTenantList;
```

Figure 90 Populating Tenant List

Populated My Properties page

With the Landlord and Property classes built, the My Properties page was populated. When the page loads, a Landlord Object user, is created using the value of session.getAttribute("ID"), set when the user logs in. An integer, propertyCount, is initialized to the value of user.getProperties().size(). If the size is 0, a card tells the user to create a new property.

```
<%if (propertyCount == 0) {//If num of properties is 0%>
<div class="card">
    <div class="card-body">
        <h4 class="card-title">
            You have 0 properties
        </h4>
        <p class="card-text">
            You haven't added to your portfolio yet, hit the button below to get started!
        </p>
        <a href="<%=path%>/NewProperty"><button class="btn btn-primary">New Property</button></a>
    </div>
```

Figure 91 Zero Properties in Portfolio

Otherwise, the elements returned by user.getProperties() are iterated over, creating a card for each property which displays the tenant and total job count as well as How many jobs require attention, and buttons to manage and go to the Ticket Centre for the property.

```
</div><%} else {//loop through Properties ArrayList if >0 properties%>
<div id="containerCard" class="card">
    <%for (Property p : user.getProperties()){
        Integer attention = 0;%>
    <div class="card">
        <div class="card-body">
            <h4 class="card-title">
                Property <big><%=p.getAddress1()%></big>
            </h4>
            <p class="card-text">
                You have <%=p.getTenantList().size()%> tenants living here, there
                <%if (p.getJobs().size() == 1) {%>
                is
                <%} else {%>
                are
                <%}%>
                <%=p.getJobs().size()%>
                job<%if (p.getJobs().size() != 1) {%>s
                <%}%>
                in total.
            </p>
            <%for (Job j : p.getJobs()){
                if(j.getStatus().equalsIgnoreCase("Pending")){
                    attention++;
                }}
                if (attention > 0){%>
                <p><b><big><%=attention%></big> Job<%if (attention != 1){%>s<%}%> requiring approval</b></p>
                <%}%>

            <div class="col-md-11 text-right">
                <a href="<%=path%>/ManageProperty"><button class="btn btn-outline-primary">Manage</button></a>
                <a href="<%=path%>/TicketCentre?pid=<%=p.getPropertyID()%>"><button class="btn btn-primary">Ticket Centre</button></a>
            </div>
        </div>
    </div>
    <%}%>
```

Figure 92 My Properties Card Creation

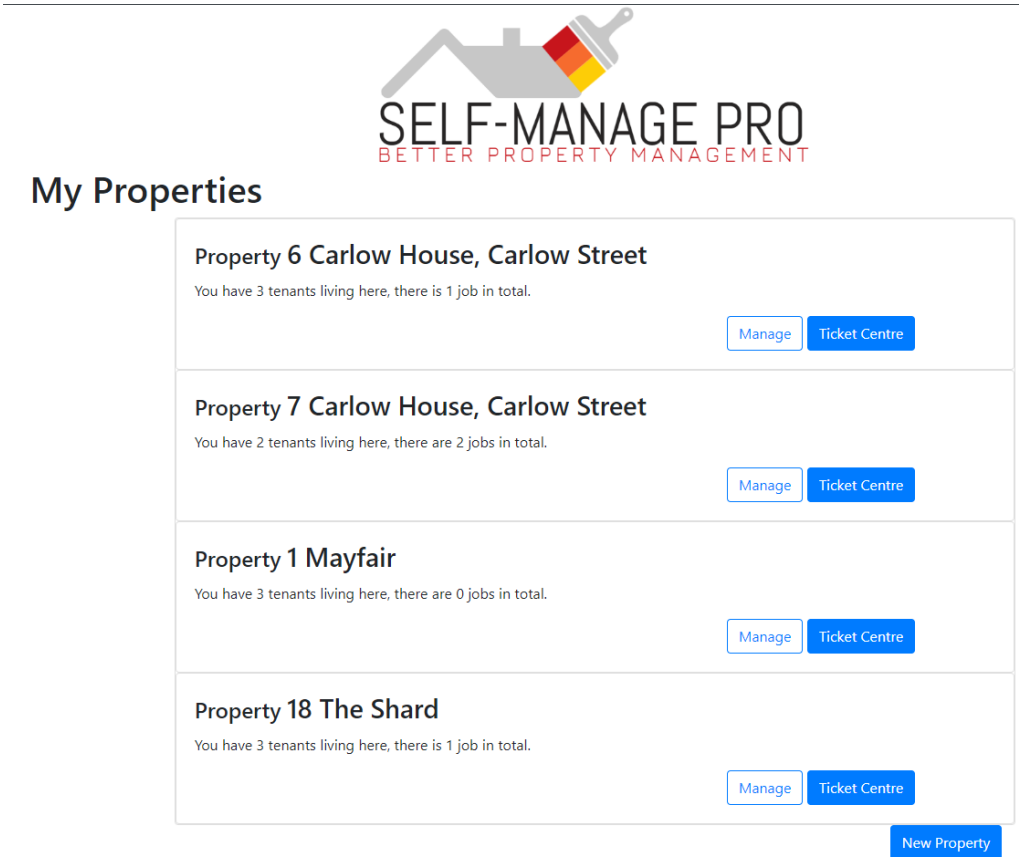


Figure 93 My Properties Cards

Landlord Static Ticket Centre

The static Landlord Ticket Centre was also created this sprint, a screenshot is shown below.

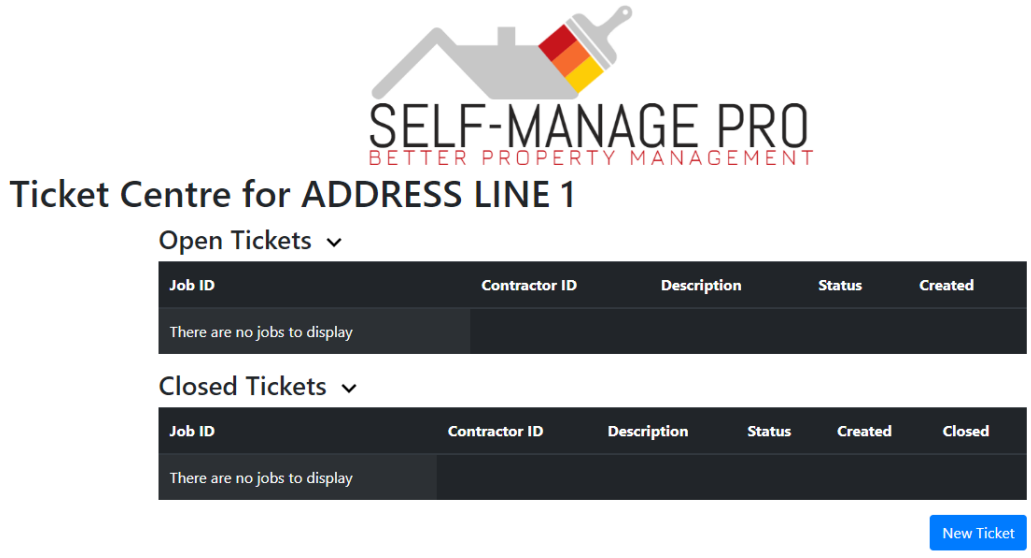


Figure 94 Static Landlord ticket Centre

3.3.3 Sprint 6

All tasks from Sprint 5, as well as extra tasks which did not have task cards, were completed successfully. Sprint 6 tasks are shown in the screenshot on the right. SQL was written for creating tickets, as well as the logic to implement these queries and AJAX methodology to populate the Landlord new Ticket page with approved contractors.

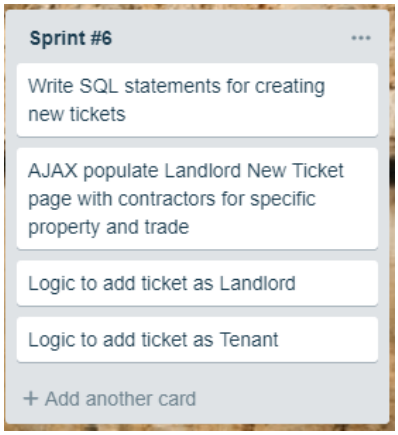


Figure 95 Sprint 6

New Ticket as Tenant

The following SQL was written to add job tickets to the database as a tenant.

INSERT INTO JOB (PROPERTYID,DESCRIPTION,STATUS) VALUES (?,?,?)

Values delimited by '?' are populated with the parameters from the ticket creation form, the SQL statement is prepared in a PreparedStatement Object and the values are set by calling the setString(index, value) method.

Tenant ticket creation takes three parameters since the CREATED column is auto-filled and the COMPLETED column is updated when the job is completed or deleted by the contractor or landlord. CONTRACTORID is not set when the tenant creates a ticket since it is set later by the landlord when they approve the ticket. Status is set to "Pending" since it is awaiting both landlords and Contractor approval.

New Ticket as Landlord

The following SQL is used to insert new tickets if the logged in user type is Landlord.

INSERT INTO JOB (PROPERTYID,CONTRACTORID,DESCRIPTION,STATUS) VALUES (?,?,?,?)

The statement is prepared in the same way as the previous query. This time however, the landlord is creating the ticket and it does not need approval, the CONTRACTORID is set by requesting the parameter "tradesman" from the previous form and the Status is set to "Landlord Approved" ready to be approved by the contractor.

After executing the query and inserting row into the database, the user is redirected to the Job Servlet which will display the Job View. If there is an error inserting the row, the user will be redirected to the NewTicket Servlet so they may try again.

Ajax populating Landlord New Ticket Tradesmen

Tradesman is determined in the New Ticket form using AJAX logic similar to the property creation form except the query takes two parameters, propertyID and trade (determined by user selection). The query designed to achieve this is below

```
SELECT PROPERTYCONTRACTOR.CONTRACTORID, CONTRACTORDETAILS.FIRST,
CONTRACTORDETAILS.LAST

FROM CONTRACTORDETAILS INNER JOIN CONTRACTORTRADES

ON CONTRACTORDETAILS.CONTRACTORID = CONTRACTORTRADES.CONTRACTORID
AND CONTRACTORTRADES."+trade+" = 1 INNER JOIN PROPERTYCONTRACTOR

ON CONTRACTORDETAILS.CONTRACTORID = PROPERTYCONTRACTOR.CONTRACTORID

AND PROPERTYCONTRACTOR.PROPERTYID=”+propertyID+”
```

The SQL statement returns a list of contractor names and IDs where the contractor has been assigned to a specific property and has marked that they are qualified for the specific trade on their profile. The names are then used to dynamically populate the options within the tradesmen Select Input box, the values of which are set to the contractorID.

The Sprint 6 card was updated to show the completed tasks from this sprint, shown right.

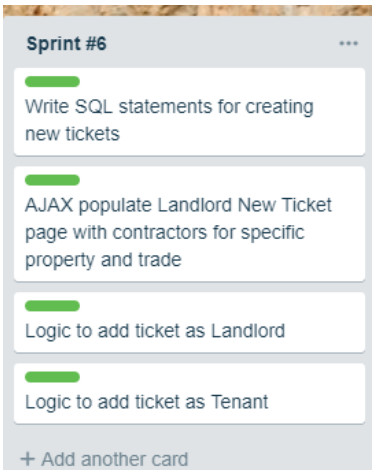
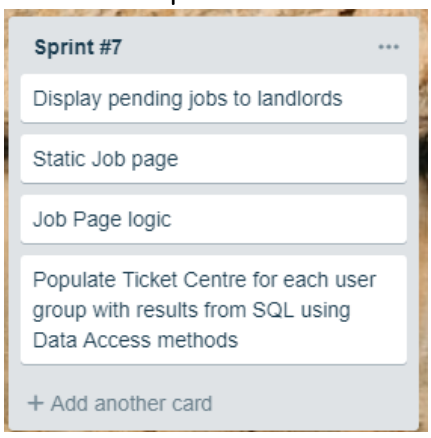


Figure 96 Sprint 6 complete

3.3.4 Sprint 7



Tasks for Sprint 7 are shown on the left, in this sprint the static Job view page was built to the HF wireframe specification and populated for each user type using a combination of the classes previously made.

Figure 97 sprint 7

Alerting landlord to jobs requiring approval on the Landlord Welcome page

```
for (Property p : user.getProperties()){
    landlordTenantCount += p.getTenantList().size();
    for (Job j : p.getJobs()){
        if (j.getStatus().equalsIgnoreCase("Pending")){
            pendingJobs++;
        }
```

Figure 98 Pending Jobs Counter

The loop for counting the number of tenants a landlord has at each property was modified to hold a nested for loop, which loops through the list of jobs for each property, checking if the status of each job is “Pending”. If a match is found then pendingJobs, a counter variable is incremented from its initialisation value of zero. This is then displayed in the body of the page.

```
<%if (pendingJobs > 0){%>
<p>
    <b>There <%if (pendingJobs != 1){%>are<%}else{%>is<%}%>
    <big><%=pendingJobs%></big> job<%if (pendingJobs != 1){%>s<%}%> requiring approval</b>
</p>
```

Figure 99 Display pending jobs, with correct plurality

A similar method was implemented on the My Properties page, this time housing the loop through the job list inside the loop which creates cards to display each property.

Static Job view

Shown below is the Job view page in its static form, illustrating the layout the page will take once it is filled with data. It has been made to conform to the design specification of its wireframe counterpart. There will be minor changes to the information displayed depending on user type, but the cards displaying different information will remain in the same place.

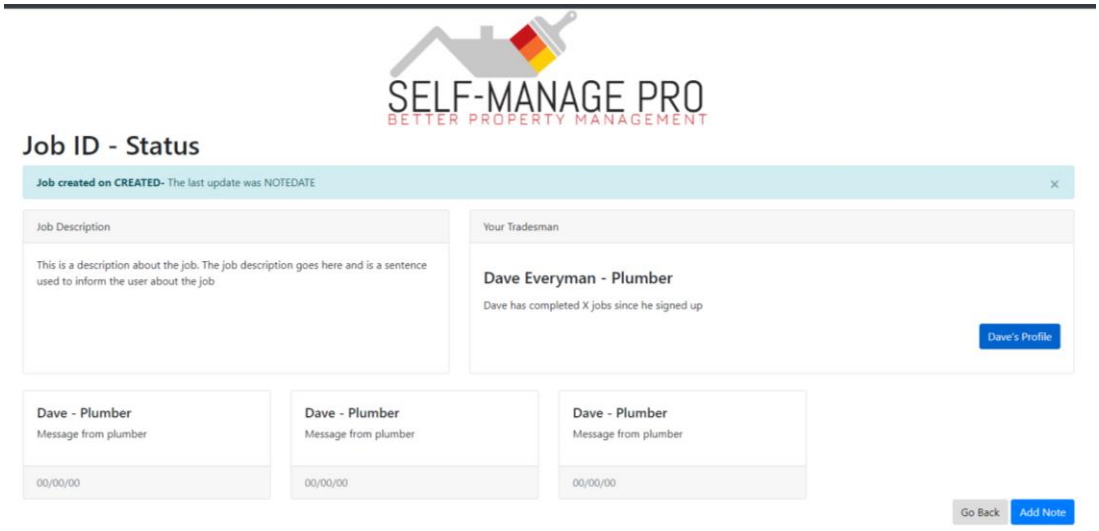
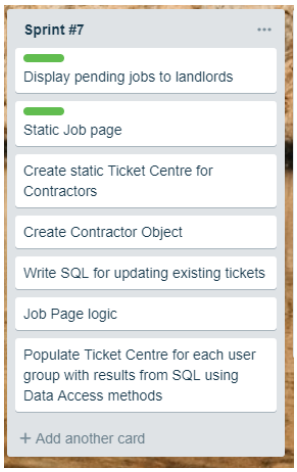


Figure 100 Job View Page

Mid-Sprint review



In order to complete all tasks for sprint 7, specifically, writing the logic for the Job view page, task cards from the product backlog must be moved into sprint 7 and carried out first. These are the creation of the Ticket Centre for Contractors, Contractor Model Object and SQL for updating tickets. The changes are shown left.

Figure 101 Sprint 7 Updated

Static Contractor Ticket Centre

The ticket centre for contractors will be much the same as for the other user types, however, instead of displaying ContractorID, the field is changed to PropertyID. Eventually, the user will be able to click on this to see a 'profile' page for the property, displaying its type, number of bedrooms and full address.

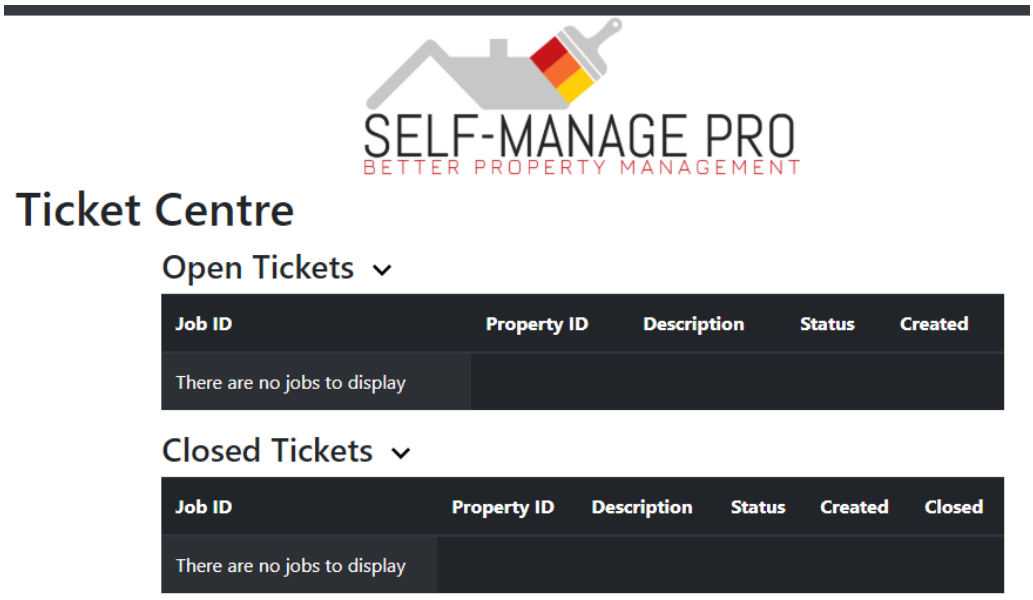


Figure 102 Contractor Ticket Centre

The contractor Object was created along with a ContractorDetails object to store all information relevant to a contractor such as open and closed jobs, their contract information and all trades they selected when signing up. The objects are complete with getters for each attribute they store, setters which use SQL queries and the DataAccess object, and a constructor method which takes only the contractorID as a parameter and uses the setter methods to populate the object.

Contractor Object

```
public final class Contractor{

    private String contractorID;
    private ContractorDetails contractorDetails;
    private ArrayList<String> trades;
    private ArrayList<Job> allJobs;
    private ArrayList<Job> openJobs;
    private ArrayList<Job> closedJobs;

    public Contractor(String id){
        this.contractorID = id;
        this.setContractorDetails();
        this.setTrades();
        this.setAllJobs();
        this.setOpenAndClosedJobs();

    }
    public void setContractorDetails() {...39 lines }
    public void setTrades(){...25 lines }
    public void setAllJobs(){...30 lines }
    public void setOpenAndClosedJobs(){...13 lines }

    public String getContractorID() {...3 lines }
    public ContractorDetails getContractorDetails(){...3 lines }
    public ArrayList<String> getTrades(){...3 lines }
    public ArrayList<Job> getAllJobs(){...3 lines }
    public ArrayList<Job> getOpenJobs() {...3 lines }
    public ArrayList<Job> getClosedJobs() {...3 lines }
```

The Contractor Object is a representation of the information held about each contractor within the database. It has an ArrayList of the trades they are qualified to do as well as lists of all jobs, open, and closed jobs.

The Contractor also has a ContractorDetails Object attribute. The Contractor object is initialized using a contractorID parameter which is then used with three queries to populate its attributes.

Figure 103 Contractor Class

ContractorDetails Object

ContractorDetails is a simple class which holds information about a Contractor. It has getter methods and a constructor.

All of the Strings in this class are final. For this reason, they have no setter methods.

Figure 104 (right) Contractor Details object

```
public class ContractorDetails {
    private final String first;
    private final String last;
    private final Date dob;
    private final String tel;
    private final String address1;
    private final String address2;
    private final String town;
    private final String county;
    private final String postcode;
    private final Date created;

    public ContractorDetails(String first, String last, Date dob, String tel, String ac
    public String getFirst() {...3 lines }
    public String getLast() {...3 lines }
    public Date getDob() {...3 lines }
    public Date getCreated() {...3 lines }
    public String getTel() {...3 lines }
    public String getAddress1() {...3 lines }
    public String getAddress2() {...3 lines }
    public String getTown() {...3 lines }
    public String getCounty() {...3 lines }
    public String getPostcode() {...3 lines }
```

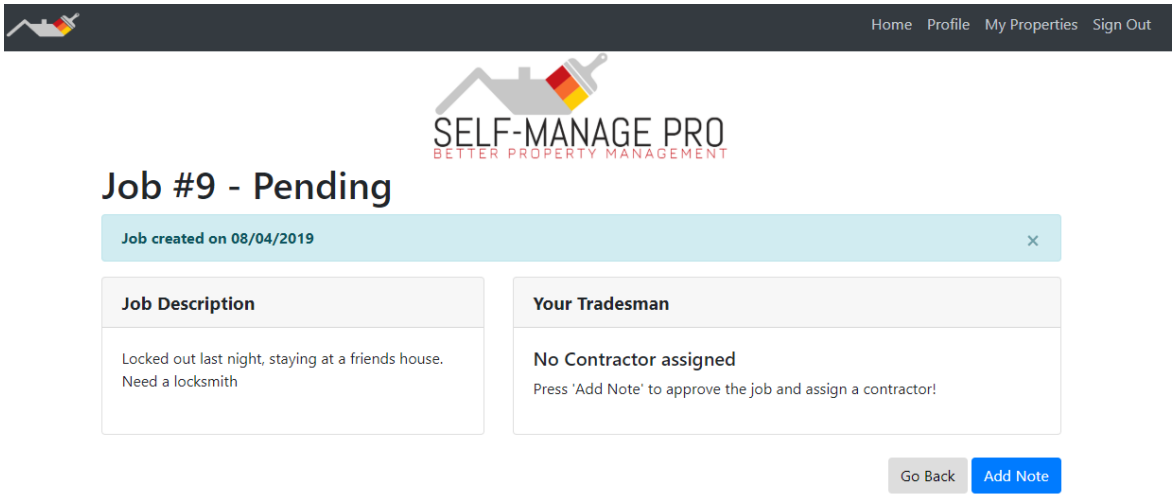
SQL and Servlet for updating tickets

All three user types can add notes to tickets but only landlords and contractors can update the status of the ticket. Additionally, only landlords may set, or change the contractor assigned to a ticket. To do this, five different queries are needed in conjunction with multiple if statements. These are shown below.

Figure 105 Add Note to Ticket / update status and contractor

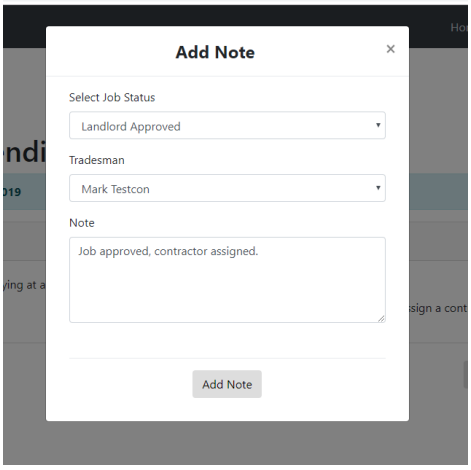
```
if (leftByType.equals("LANDLORD")) {
    status = request.getParameter("status");
    String updateJobStatusSQL = "";
    if (status.equalsIgnoreCase("Landlord Approved")) {
        contractorID = request.getParameter("tradesman");
        updateJobStatusSQL = "UPDATE JOB SET STATUS = '" + status + "',CONTRACTORID = " + contractorID + " WHERE JOBID = ?";

    }else if(status.equalsIgnoreCase("Closed")){
        updateJobStatusSQL = "UPDATE JOB SET STATUS = '" + status + "', COMPLETED = CURDATE() WHERE JOBID = ?";
    }else {
        updateJobStatusSQL = "UPDATE JOB SET STATUS = '" + status + "' WHERE JOBID = ?";
    }
    ps = da.getConn().prepareStatement(updateJobStatusSQL);
    ps.setString(1, jobID);
    da.setPreparedStatement(ps);
    da.executeUpdate(da.getPreparedStatement());
} else if (leftByType.equals("CONTRACTOR")) {
    status = request.getParameter("status");
    if (!status.equalsIgnoreCase("none")) {
        String updateJobStatusSQL = "UPDATE JOB SET STATUS = '" + status + "' WHERE JOBID = ?";
        if (status.equalsIgnoreCase("Closed")) {
            updateJobStatusSQL = "UPDATE JOB SET STATUS = '" + status + "', COMPLETED = CURDATE() WHERE JOBID = ?";
        }
        ps = da.getConn().prepareStatement(updateJobStatusSQL);
        ps.setString(1, jobID);
        da.setPreparedStatement(ps);
        da.executeUpdate(da.getPreparedStatement());
```

Figure 106 Landlord Job view – Job Pending



Assigning a contractor and approving the job as a landlord

From the pending job view, the ‘Add Note’ button can be pressed to change the job status and assign a contractor. When ‘Landlord Approved’ is selected from the dropdown, the tradesman field appears and is dynamically populated using the same AJAX as in the job creation form. This will display only contractors assigned to the specific property and of the specified required trade.



The note can then be added and the page will refresh to display the new job status, assigned contractor, the note which was added and when the last update on the job was. The page also shows how many jobs the selected contractor has completed since they signed up.

Figure 107 AJAX Tradesman field

Contractor Job View

After navigating to the Job view page through Ticket Centre, the assigned contractor can see the job description as well as the properties address and contact information for its tenants. The contractor can either set the job to Open or choose from ‘Contact made’, ‘Appointment booked’ and ‘Closed’ depending on if, when and how the job needs to be carried out. A note will be left with the updated job status and displayed on the Job View.

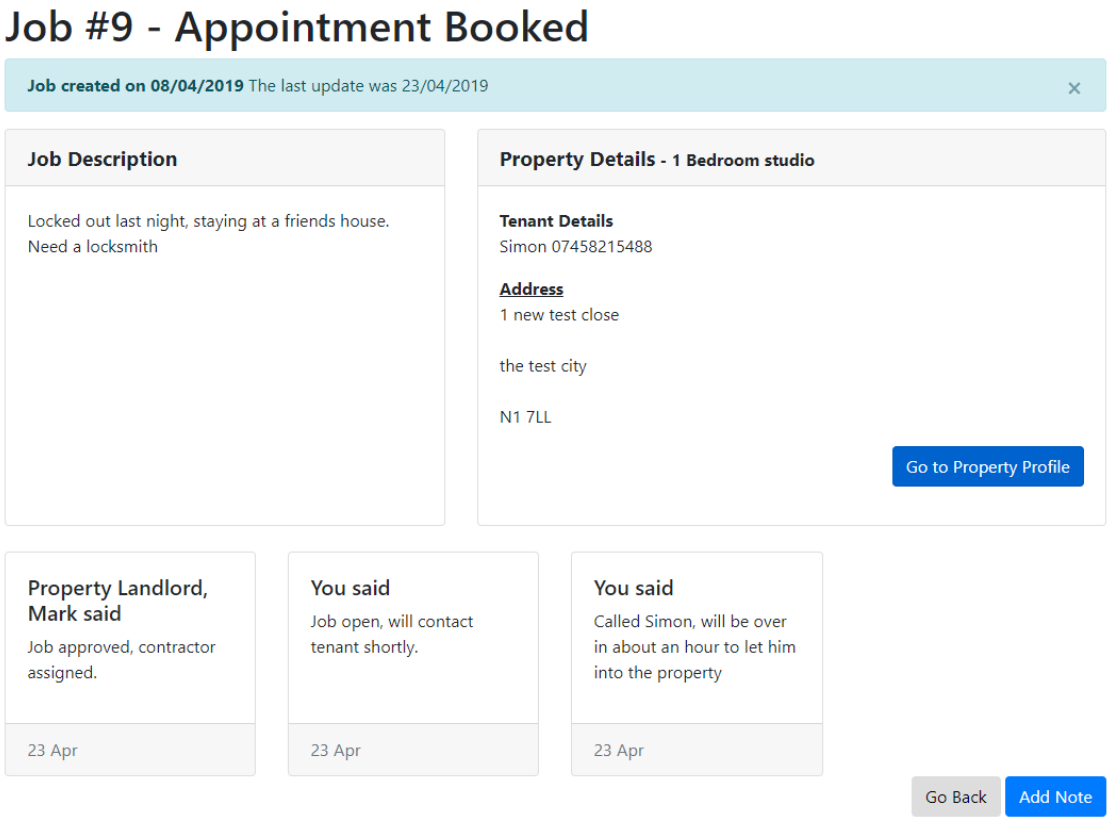


Figure 108 Contractor Job View

Tenant job view

The details of the job are displayed to the tenant in much the same way as they are the landlord, the key difference being that the tenant cannot update the status of the job, they may only display further information to the landlord and contractor. Once the landlord or contractor has closed the ticket, the tenant can see the job as closed and the contractors completed jobs count will be incremented, shown below.

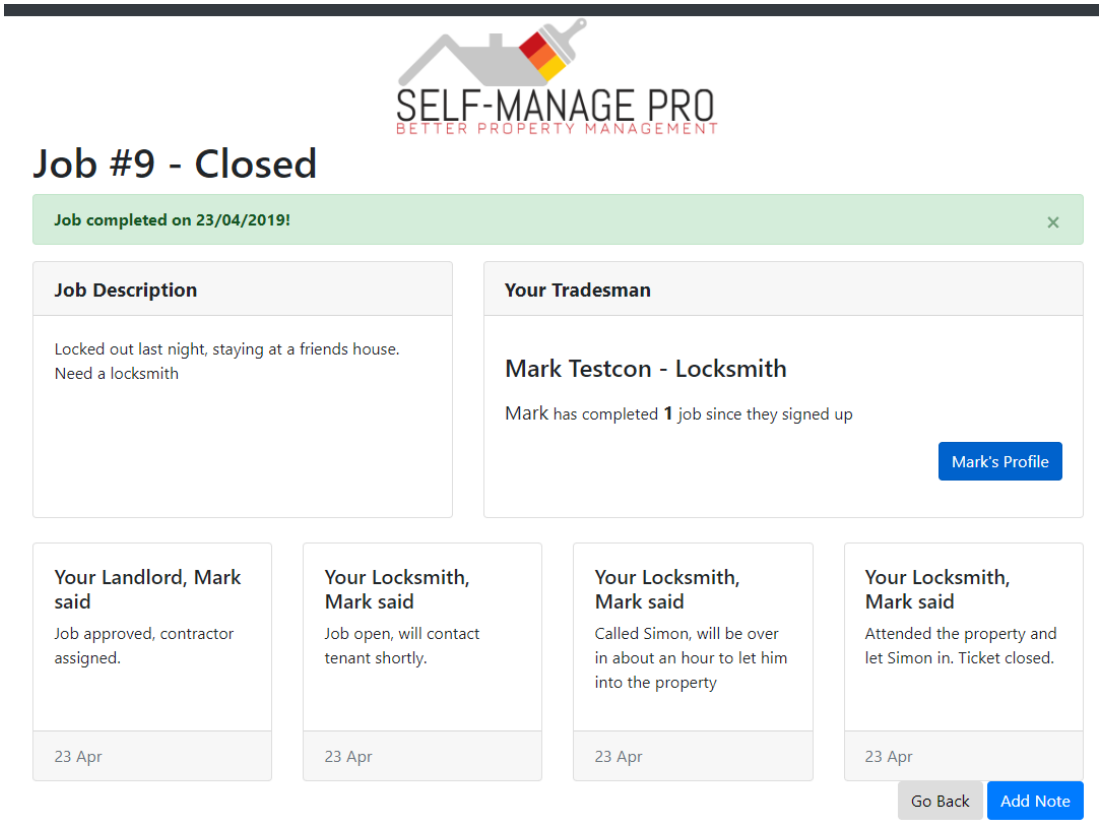


Figure 109 Job View Job Closed

3.3.5 Sprint 8

The final sprint contained the three tasks left over in the product backlog, shown right. Of these, the Admin page was not completed. The contractor Ticket Centre was populated by iterating over the ArrayList of jobs returned by calling user.getOpenJobs().

functions used; application of coding principles; critical discussion of coding issues; sophistication of code; code structure; choice of methodologies; modularity

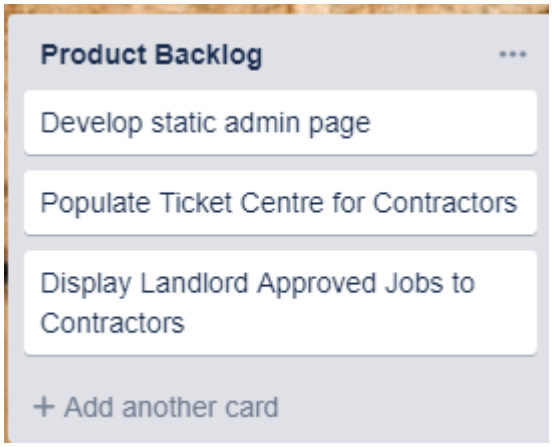


Figure 110 Final Product Backlog

4 VALIDATION

During any project, testing is an important process which provides quality assurance for code, as functionality is tested with a range of criteria to check that the project performs as it should. Projects without any testing have a greater chance of crashing or not working as intended, this can cause businesses to lose money and customer loyalty could be affected so it is imperative that testing is performed.

Validation can be performed within the application, implemented using native HTML5 methods and Regular Expression language standards. This is done within the signup form on the postcode and telephone fields by setting the `pattern` attribute of the `input` elements to the following RegEx (html5pattern.com, 2019)

UK Postcodes: `[A-Za-z]{1,2}[0-9Rr][0-9A-Za-z]? [0-9][ABD-HJLNP-UW-Zabd-hjlnp-uw-z]{2}`

Telephone numbers:
`((\+44\s?\(0\)\s?\d{2,4})|(\+44\s?(01|02|03|07|08)\d{2,3})|(\+44\s?(1|2|3|7|8)\d{2,3})|(\(\+44\)\s?\d{3,4})|(\(\d{5}\))|((01|02|03|07|08)\d{2,3})|(\d{5}))(\s|-|.)(((\d{3,4})(\s|-)(\d{3,4}))|((\d{6,7})))`

Additionally, HTML5 email fields were used to take the email address of the user, these are validated automatically on form submit. Age is also validated to ensure the user is at least 18 years of age by setting the `max` attribute of the Date of Birth input box to the evaluation of:

```
LocalDate.now().minusYears(18).format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
```

A function which subtracts 18 years from the current date and formats it for the input box.

There are two types of testing, Black and White box. In black box testing, the tester does not have access to the code used to build the system and must test a system using its graphical user interface. White box testing is where the user can access the source code of a project and tailor tests to classes and their functions. There are resources and libraries such as jUnit and hamcrest testing frameworks which aid developers in testing by providing functions such as `assertEqual()` and `assertTrue()`.

The next page contains a table with black box tests which were conducted to test that the function elements of the application work as they should. There are also two white box tests included in the table for testing the `getSHA()` and `generateSalt()` methods within the Hasher class.

File To Test	Functionality	Description	Criteria	Outcome
AddNoteServlet.java	JOBNOTE added to database	Test that the servlet file successfully adds a JOBNOTE row with a tenant logged in	Row inserted into JOBNOTE table when file run	Row Added: PASS
AddNoteServlet.java	Update Job Contractor and status as Landlord	Test that a logged in Landlord can set the status of a job "Landlord Approved" and assign a Contractor	Status and Contractor ID Columns updated in Table JOB entry as well as a new row inserted in JOBNOTE	Status and Contractor Set, JOBNOTE added: PASS
AddNoteServlet.java	Update Job Status as Contractor	Test a logged in contractor can update the status of a job and add corresponding JOBNOTE entry	Status column updated in JOB entry & JOBNOTE row created	JOB updated and JOBNOTE created: PASS
DataAccess.java	Test openConnection	Test that the openConnection method successfully connects to the database	Pass if: Connected printed to console, FAIL if: Not Connected Printed to console	Connected printed to console: PASS
DataAccess.java	Test executeQuery	Test that executeQuery successful queries the database and returns a ResultSet	Logging in utilizes executeQuery, therefore if a user can log in then the function works	login Successful: PASS
DataAccess.java	Test executeUpdate	Test that executeUpdate successfully alters and inserts rows	Used in AddNoteServlet, which has been tested as working	Rows inserted and altered: PASS

DataAccess.java	Test Add Property	Log in as Landlord and go to "/MyProperties", click new property and fill out and submit the form	Property will display on My Properties page after submit is pressed	New Property added: PASS
DataAccess.java	Test closeConnection	Test that closeConnection successfully closes an open connection	Pass if Connection Closed printed to console	Connection Closed printed to console: PASS
Hasher.java	Test generateSalt	(White Box Test)Check a string 3 characters long is always generated. Iterate adding the result of generateSalt to an ArrayList 1000 times	Pass if the length of each of the 1000 Strings in the ArrayList is 3	All Strings .length = 3: PASS
Hasher.java	Test getSHA	(White Box Test) Check the hash value of a String against the known hash value of the same String	Pass if known hash = generated hash	Strings equal: PASS
LandingServlet.java	Test No user Home Page	Test the relevant content delivery switch statement delivers the standard home page to a user not logged in	No user home page delivered	PASS
LandingServlet.java	Test Tenant landing page	Test the relevant content delivery switch statement delivers the Tenant landing page to a user logged in as Tenant	Tenant Landing Page delivered	PASS

LandingServlet.java	Test Landlord landing page	Test the relevant content delivery switch statement delivers the Landlord landing page to a user logged in as Landlord	Landlord landing page delivered	PASs
LandingServlet.java	Test Contractor landing page	Test the relevant content delivery switch statement delivers the contractor landing page to a user logged in as Contractor	Contractor landing page delivered	PASS
Login.java	Test Tenant Log In	Use the modal login to log in as Tenant	Tenant area shown after login	PASS
Login.java	Test Landlord Log In	Use the modal login to log in as Landlord	Landlord area shown after login	PASS
Login.java	Test Contractor Log In	Use the modal login to log in as Contractor	Contractor area shown after login	PASS
LogoutServlet.java	Test Log out	Sign in and press log out	User redirected to home page and cannot access features such as Ticket Centre or Properties	URL manipulation could not access account-only areas: PASS
MyPropertiesServlet.java	Test Landlord sees properties	Log in to a landlord account with properties set up, navigate to My Properties	Page should be populated with a list of cards showing information about properties the landlord owns	Landlord has multiple property cards: PASS

MyPropertiesServlet.java	Test Landlord sees no properties	Log in to a landlord account with no properties set up, navigate to My Properties	Page should prompt the user to create a new property	No properties shown, prompt displayed to create new property: PASS
MyPropertiesServlet.java	Test Contractor cannot access page	Log in to contractor account and type in the url mapping "/MyProperties"	Page should redirect contractor to home page	Redirected to home page: PASS
MyPropertiesServlet.java	Test Tenant is redirected	Log in to Tenant account and type in the url mapping "/MyProperties"	Page should redirect tenant to "/Property"	Redirected to "/Property": PASS
NewTicketLogic.java	Test New Ticket Is Created	Log in as Tenant and try to submit a new Ticket	Page should redirect to the Job View for the ticket and show entered details	Job created: PASS
SignUpLogic.java	Test new Tenant Added	Create a new Tenant using the signup form and attempt to login with credentials	User should Log in Successfully	User Logged in: PASS
SignUpLogic.java	Test new Landlord Added	Create a new Landlord using the signup form and attempt to login with credentials	User should Log in Successfully	User Logged in: PASS
SignUpLogic.java	Test new Contractor Added	Create a new Contractor using the signup form and attempt to login with credentials	User should Log in Successfully	User Logged in: PASS

5 CRITICAL REVIEW & CONCLUSION

5.1 CLOSING EXECUTIVE SUMMARY

Overall, the application is a well-engineered product where great effort has been put into both the visual, front end, as well as the functional and business logic of the back end. All of the must have requirements were completed as well as all except one of the Should Have requirements.

The design methodology chosen helped me to plan the application effectively, however, having three user types as well as thirteen relational tables in the database made the project a very long endeavour and some of the original planned functionality with corresponding visual elements were never completed. Examples of this include user profile servlets where users can change their details, with public profiles for contractors, as well as the manage property page for landlords which was linked from the My Properties page but never implemented.

Using a Trello board and splitting the project into Sprints bred objectivity when planning the order of development. At all times, I was able to monitor my progress through the project and have a clear idea of what needed to be done to achieve the epic stories. Whilst this method of planning was effective, some Sprints did need changing mid-way through to best implement certain functionality. Further, it was not always easy to plan how long certain tasks would take and some tasks were left in uncompleted in the backlog, for example the admin page.

Building the majority of the static side of the application before creating any objects to display the stored information enabled the project to not be over-engineered, producing only the functionality which was needed to make the desired features work. It did mean, though, that there was some oversight during planning and therefore, minor design and implementation changes were made during development, however, this was always planned as stated in section 1.6.1.

Using the Bootstrap JavaScript Library proved to be an extremely effective way of making the visual elements conform to the wireframes, especially where CSS is not one of my strengths. jQuery enabled me to perform a lot of DOM manipulation which otherwise would've taken a lot more development time and have a greater complexity.

Choosing the MVC layout was logical since it enabled me to easily separate concerns and build the application with polymorphism in mind. It would be easy to swap implementations of Objects out if use requirements changed in future development. Furthermore, the layout made file management and finding specific files much easier where there were so many files to keep track of.

The user type based content-delivery switch statement developed in Sprint 1 met the criteria for epic 1; "As a User, I only want to see content relevant to my account". This was used as the backbone logic for all servlets which content which is dependent on account type, and also serves to validate page requests and ensure that users are in the correct area. The rest of the epic stories were all fulfilled with a combination of the My Properties, Ticket Centre, Job Creation and Job View pages. Despite this, a major security concern of the application is that any job can be viewed by it's ID if the user figures out the URL pattern. This is because the View Job Servlet checks user type and jobID however, never checks to see if the job the user is trying to view is associated with the user viewing it. This is a fairly major security floor since the users could then add notes, change the status, or if they had a contractor account, see the tenant address and contact details.

Although all must have requirements and epic stories were fulfilled, the legislation problem set out in the introduction was not really addressed by the system. This is largely because the existing systems which were reviewed in the Literature Review section, offered comprehensive solutions for property self-management including compliance with legislation, such as paperwork and scheduling renewal of certificates when they expire. The product I made had to fill a niche which was not covered by the larger applications in order to have a chance of succeeding if brought to market; user experience driven functionality, with visually appealing and minimalist elements which do the job required without creating an information overflow problem. The application relieves the workload for maintenance jobs, giving landlords more time and resources to handle the ins and outs of self-management at their leisure.

5.2 CONCLUSION

During this project, much was learned, such as the intricacies of casting objects from database rows stored within Result Set objects. A concept I was not previously unfamiliar with, however, in previous projects the database connections were left open after object creation so the ResultSet could be used again. This was bad practice and this project taught me the correct way to use the data as needed, and store it so the connection could then be closed, freeing up application resources; widening the scope of the project to be more scalable (i.e. not excluding the maximum number of permissible open connections).

The primary aims of the project were to create a website this is both responsive and intuitive, and to allow landlords to manage their property portfolios maintenance tasks. The system also had to be an improvement on the current method of self-managing landlords; paper and memory. Further still, the application had to “Let contractors easily work with landlords and tenants” and “Keep tenants informed about work being carried out at their property”. It is suffice to say that all of these aims have been met when examined individually, I have built a well-featured Java Enterprise web application with responsive visual elements which react to user interaction with the DOM. The system is most definitely better than remembering appointments or keeping diaries & printouts. It provides an interface between tenants, landlords and contractors, keeping all stakeholders informed at every step, all in one place.

Objectives of the project included having a secure login, linking contractors only to properties they are assigned to, have a ticket centre displaying open and closed tickets and allow users to add notes to jobs, and view them after. All of these objectives, and the others not mentioned, on the initial list; were completed successfully.

The project could’ve been improved by starting the implementation stage during the design phase so each sprint would have been longer and completed to a better standard, with more features added. Additionally, this would have created the possibility of doing entire sprints of testing to provide quality assurance to stakeholders.

If the project were to be developed further, I would implement a local database so the application could be deployed more readily, as well as payment gateway and invoice generation system. I would also develop profile pages where users would be able to edit their own personal information; information about properties; and showcase Contractors trades and job completion to prospective customers.

6 BIBLIOGRAPHY

[1] Kate Faulkner - The Telegraph. 2017. *Do landlords really need letting agencies?*. [ONLINE] Available at: https://www.telegraph.co.uk/property/landlord-guide/do-you-really-need-a-letting-agency/. [Accessed 4 October 2018].

[2] Laura-Jane O'Hare - RentPro. 2017. *Proportion of landlords using letting agents spikes*. [ONLINE] Available at: https://www.rentpro.co.uk/blog/proportion-of-landlords-using-letting-agents-spikes/. [Accessed 4 October 2018].

[3] PrimeLocation. 2014. *Guide to property management agents*. [ONLINE] Available at: https://www.primelocation.com/discover/letting/guide-to-property-management-agents/#V75ua76MgI47QDih.97. [Accessed 5 October 2018].

[4] https://www.thehouseshop.com/property-blog/our-guide-for-private-landlords-going-it-alone/11002/

[5] https://publications.parliament.uk/pa/cm/cmregmem/190318/190318.pdf

[6] Property Investment Investment. 2018. *Do Landlords need a Fully Managed Letting Service? Probably not.*. [ONLINE] Available at: https://www.propertyinvestmentproject.co.uk/blog/landlord-fully-managed-letting-services/?fbclid=IwAR3vckkHFVcs-V28Am0EcFN9ZlOqd4wvT6QNgb-y4iW0_t9pikWzTIGQshQ#experience. [Accessed 28 November 2018].

[7] The Independent. 2018. *No sprinklers in 96% of London high-rise council blocks, research finds | The Independent*. [ONLINE] Available at: https://www.independent.co.uk/news/uk/home-news/sprinklers-council-high-rise-tower-blocks-grenfell-tower-fire-london-labour-research-a8648966.html. [Accessed 28 November 2018].

[8] BBC News. 2018. *Grenfell contractor: Sprinklers would have saved tower - BBC News*. [ONLINE] Available at: https://www.bbc.co.uk/news/uk-england-london-41230521. [Accessed 28 November 2018].

[9] Karunasena, G, Vijerathne, D & Muthmala, H. 2018. Preliminary framework to manage tenant satisfaction in facilities management service encounters. Facilities; Bradford. 36(3/4)

[10] Birks, D & Southan, J. 1992. An evaluation of the rationale of tenant satisfaction surveys. Housing Studies. 7(4)

[11] Sanderson, D., (2018). Determinants of Satisfaction, Loyalty and Landlord Advocacy amongst Private Rented Sector Tenants in the UK. *In ERES Conference*. Reading, 2018. -: ERES Digital Library.[ONLINE] Available at: https://eres.architexturez.net/doc/oai-eres-id-eres2018-81

[12] Rahman, A., & Khan, P. I. (2014). Effect of service convenience on service loyalty: Moderating role of consumer characteristics. South Asian Journal of Management, 21(3), 7-30. Available at: https://search-proquest-com.ezproxy.kingston.ac.uk/docview/1814318272?accountid=14557

[13] re-leased.com. (2015). *Re-Leased | Property Management Software for Property Managers*. [online] Available at: https://re-leased.com/gb/property-managers/

[14] rman.co.uk. (2019). Rentman Software - Overview. [online] Available at: https://www.rman.co.uk/features/

[15] Zendesk. (2019). *Zendesk | Customer Service Software & Support Ticket System*. [online] Available at: https://www.zendesk.co.uk/

[16] ww4.autotask.net. (2018). *Autotask Web Services API*. [online] Available at: https://ww4.autotask.net/help/Content/LinkedDOCUMENTS/WSAPI/T_WebServicesAPIv1_5.pdf

[17] Parsons, N. (2018). *What Is a SWOT Analysis, and How to Do It Right*. [online] LivePlan Blog. Available at: https://www.liveplan.com/blog/what-is-a-swot-analysis-and-how-to-do-it-right-with-examples/

[18] Contributor, P. (2015). *The Importance of SWOT Analysis*. [online] PESTLE Analysis. Available at: https://pestleanalysis.com/importance-of-swot-analysis/

[19] Mindtools.com. (2018). *PEST AnalysisIdentifying Big Picture Opportunities and Threats*. [online] Available at: https://www.mindtools.com/pages/article/newTMC_09.htm

[20] BBC News. (2018). *Brexit: Carney makes property crash warning*. [online] Available at: https://www.bbc.co.uk/news/business-45516678

[21] BBC News. (2019). *How does Brexit affect the pound?*. [online] Available at: https://www.bbc.co.uk/news/business-46862790

[22] Popli, R., Sharma, H. and Chauhan, N. (2014). Prioritising user stories in agile environment. In: *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques*. [online] Ghaziabad, India: IEEE, p.1. Available at: https://ieeexplore-ieee-org.ezproxy.kingston.ac.uk/document/6781336/ [Accessed 11 Nov. 2018].

[23] Eriksson, U. (2012). *Functional Requirements vs Non Functional Requirements*. [online] ReQtest. Available at: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[24] IBM (2018). *2018 Cost of a Data Breach Study*. [online] Ponemon Institute. Available at: https://www.ibm.com/downloads/cas/861MNWN2

[25] Cvedetails.com. (2019). *Phpmyadmin : Security vulnerabilities*. [online] Available at: https://www.cvedetails.com/vulnerability-list/vendor_id-784/Phpmyadmin.html

[26] Ptsecurity.com. (2018). *WEB APPLICATION ATTACK STATISTICS*. [online] Available at: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf

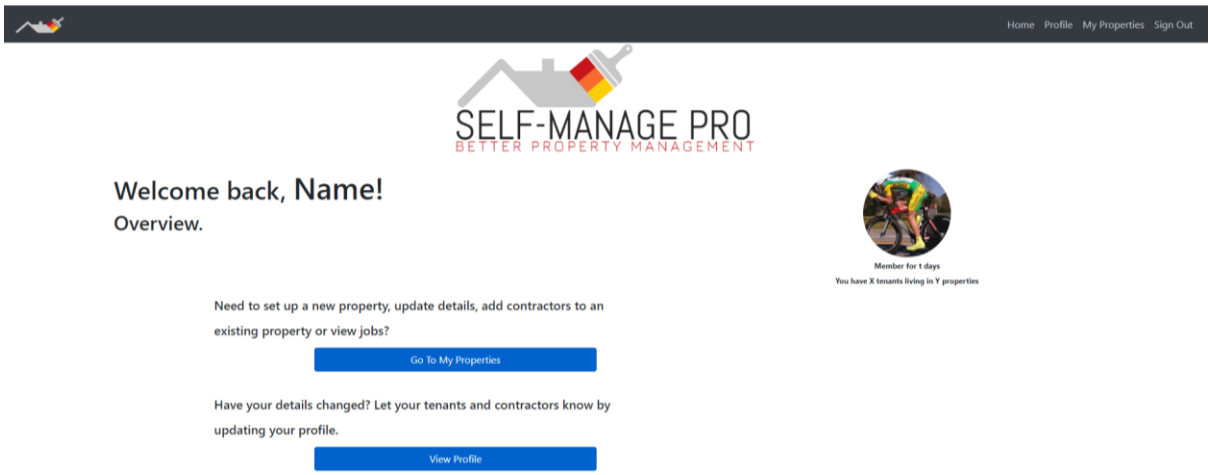
[27] W3techs.com. (2019). *Usage Statistics and Market Share of jQuery for Websites, April 2019*. [online] Available at: https://w3techs.com/technologies/details/js-jquery/all/all

[28] Html5pattern.com. (2019). *HTML5Pattern*. [online] Available at: http://html5pattern.com/Postal_Codes

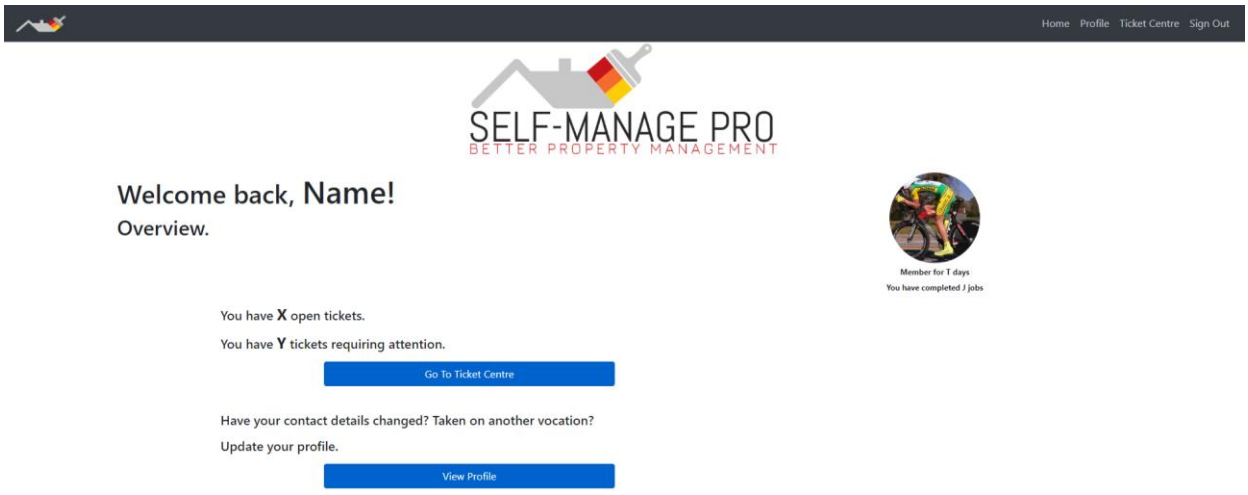
7 APPENDICES

Appendix A

Static landlord page

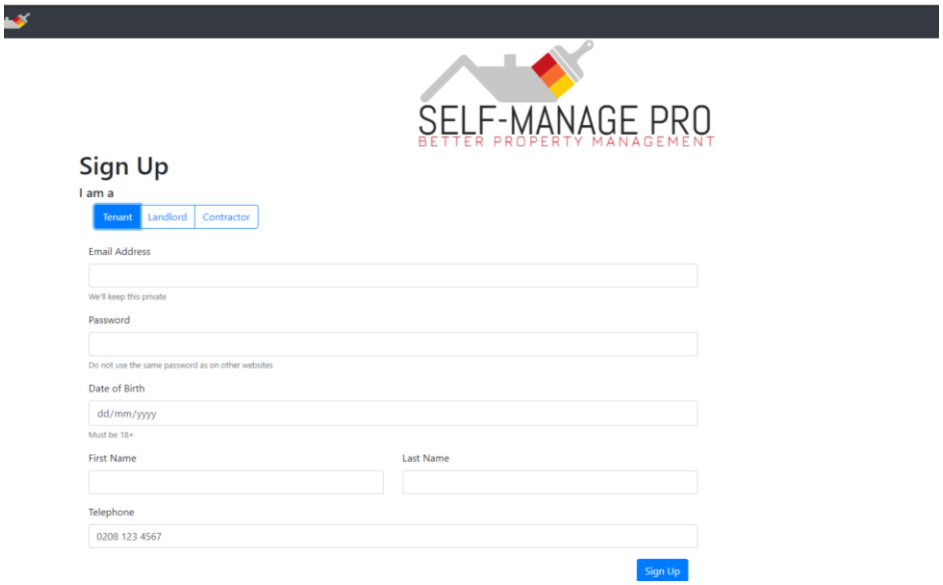


Static contractor page

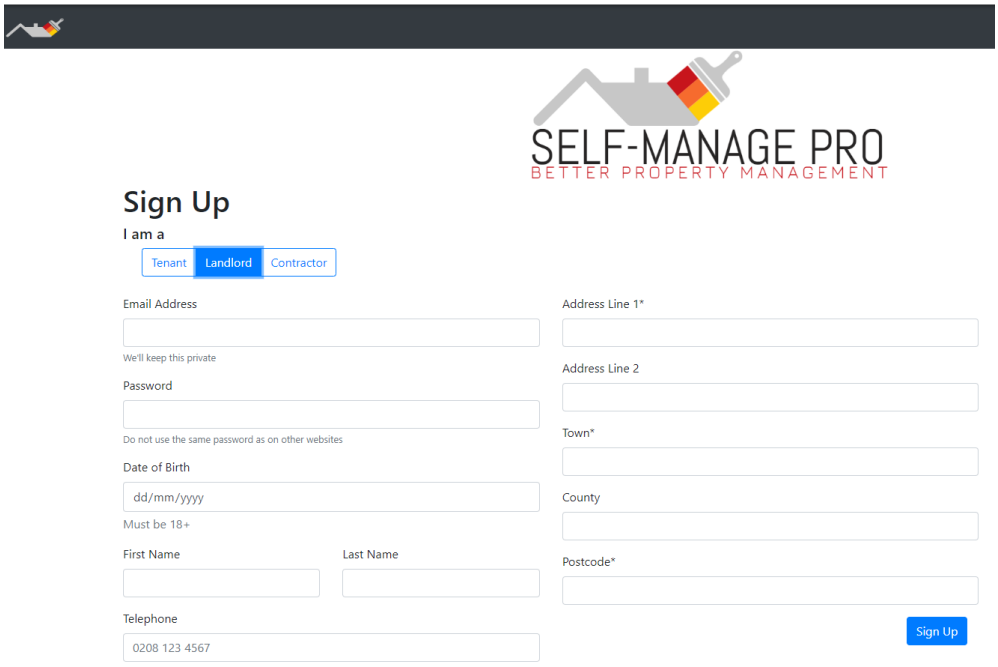


Appendix B

Tenant Sign Up Page



Landlord Sign Up page



Appendix C

Creating Primary Keys with Alter Statements

```
------PRIMARY KEYS------
ALTER TABLE `CONTRACTOR`  ADD PRIMARY KEY (`CONTRACTORID`);
ALTER TABLE `CONTRACTORDETAILS`  ADD PRIMARY KEY (`CDNUM`),  ADD KEY `CONTRACTORID` (`CONTRACTORID`);
ALTER TABLE `CONTRACTORTRADES`  ADD PRIMARY KEY (`CTID`),  ADD KEY `CONTRACTORID` (`CONTRACTORID`),  ADD KEY `CONTRACTORID_2` (`CONTRACTORID`);
ALTER TABLE `JOB`  ADD PRIMARY KEY (`JOBID`),  ADD KEY `PROPERTYID` (`PROPERTYID`),  ADD KEY `CONTRACTORID` (`CONTRACTORID`);
ALTER TABLE `JOBNOTES`  ADD PRIMARY KEY (`JOBNOTEID`),  ADD KEY `JOBID` (`JOBID`);
ALTER TABLE `LANDLORD`  ADD PRIMARY KEY (`LANDLORDID`);
ALTER TABLE `LANDLORDDETAILS`  ADD PRIMARY KEY (`LDNUM`),  ADD KEY `LANDLORDID` (`LANDLORDID`);
ALTER TABLE `PROPERTY`  ADD PRIMARY KEY (`PROPERTYID`),  ADD KEY `LANDLORDID` (`LANDLORDID`);
ALTER TABLE `PROPERTYCONTRACTOR`  ADD PRIMARY KEY (`PCID`),  ADD KEY `PROPERTYID` (`PROPERTYID`),  ADD KEY `CONTRACTORID` (`CONTRACTORID`);
ALTER TABLE `TENANT`  ADD PRIMARY KEY (`TENANTID`),  ADD KEY `email` (`email`);
ALTER TABLE `TENANTDETAILS`  ADD PRIMARY KEY (`TDNUM`),  ADD UNIQUE KEY `TENANTID_2` (`TENANTID`),  ADD KEY `TENANTID` (`TENANTID`);

ALTER TABLE `TENANTGROUP`  ADD PRIMARY KEY (`TGID`),  ADD KEY `TENANT1` (`TENANT1`),  ADD KEY `TENANT2` (`TENANT2`),
ADD KEY `TENANT3` (`TENANT3`),  ADD KEY `TENANT4` (`TENANT4`),  ADD KEY `TENANT5` (`TENANT5`),  ADD KEY `TENANT4_2` (`TENANT4`),
ADD KEY `TENANT6` (`TENANT6`),  ADD KEY `TENANT7` (`TENANT7`),  ADD KEY `TENANT8` (`TENANT8`),  ADD KEY `TENANT9` (`TENANT9`),
ADD KEY `TENANT10` (`TENANT10`),  ADD KEY `TENANT11` (`TENANT11`),  ADD KEY `TENANT12` (`TENANT12`);

ALTER TABLE `TENANTGROUPPROPERTY`  ADD PRIMARY KEY (`TGPID`),  ADD KEY `TGID` (`TGID`),  ADD KEY `PROPERTYID` (`PROPERTYID`);
```

Setting up unsigned zerofill and auto-increment values

```
------Auto increment and unsigned zerofill------
ALTER TABLE `CONTRACTOR`  MODIFY `CONTRACTORID` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `CONTRACTORDETAILS`  MODIFY `CDNUM` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `CONTRACTORTRADES`  MODIFY `CTID` int(5) NOT NULL AUTO_INCREMENT;
ALTER TABLE `JOB`  MODIFY `JOBID` int(6) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `JOBNOTES`  MODIFY `JOBNOTEID` int(7) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `LANDLORD`  MODIFY `LANDLORDID` int(5) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `LANDLORDDETAILS`  MODIFY `LDNUM` int(5) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `PROPERTY`  MODIFY `PROPERTYID` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `PROPERTYCONTRACTOR`  MODIFY `PCID` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `TENANT`  MODIFY `TENANTID` int(6) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `TENANTDETAILS`  MODIFY `TDNUM` int(6) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `TENANTGROUP`  MODIFY `TGID` int(5) unsigned zerofill NOT NULL AUTO_INCREMENT;
ALTER TABLE `TENANTGROUPPROPERTY`  MODIFY `TGPID` int(4) unsigned zerofill NOT NULL AUTO_INCREMENT;
```

Creating foreign key relationships with alter statements

```
----Creating foreign keys------
ALTER TABLE `CONTRACTORDETAILS`  ADD CONSTRAINT `CDFK` FOREIGN KEY (`CONTRACTORID`) REFERENCES `CONTRACTOR` (`CONTRACTORID`);
ALTER TABLE `CONTRACTORTRADES`  ADD CONSTRAINT `CTFK` FOREIGN KEY (`CONTRACTORID`) REFERENCES `CONTRACTOR` (`CONTRACTORID`);

ALTER TABLE `JOB`  ADD CONSTRAINT `JFK1` FOREIGN KEY (`PROPERTYID`) REFERENCES `PROPERTY` (`PROPERTYID`),
ADD CONSTRAINT `JFK2` FOREIGN KEY (`CONTRACTORID`) REFERENCES `CONTRACTOR` (`CONTRACTORID`);

ALTER TABLE `JOBNOTES`  ADD CONSTRAINT `JNFK` FOREIGN KEY (`JOBID`) REFERENCES `JOB` (`JOBID`);
ALTER TABLE `LANDLORDDETAILS`  ADD CONSTRAINT `LDFK` FOREIGN KEY (`LANDLORDID`) REFERENCES `LANDLORD` (`LANDLORDID`);
ALTER TABLE `PROPERTY`  ADD CONSTRAINT `PFK` FOREIGN KEY (`LANDLORDID`) REFERENCES `LANDLORD` (`LANDLORDID`);
ALTER TABLE `PROPERTYCONTRACTOR`  ADD CONSTRAINT `PCCIDFK` FOREIGN KEY (`CONTRACTORID`) REFERENCES `CONTRACTOR` (`CONTRACTORID`),
ADD CONSTRAINT `PCPIDFK` FOREIGN KEY (`PROPERTYID`) REFERENCES `PROPERTY` (`PROPERTYID`);

ALTER TABLE `TENANTDETAILS`  ADD CONSTRAINT `TDFK` FOREIGN KEY (`TENANTID`) REFERENCES `TENANT` (`TENANTID`);

ALTER TABLE `TENANTGROUP`  ADD CONSTRAINT `TGFK1` FOREIGN KEY (`TENANT1`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK10` FOREIGN KEY (`TENANT10`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK11` FOREIGN KEY (`TENANT11`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK12` FOREIGN KEY (`TENANT12`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK2` FOREIGN KEY (`TENANT2`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK3` FOREIGN KEY (`TENANT3`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK4` FOREIGN KEY (`TENANT4`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK5` FOREIGN KEY (`TENANT5`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK6` FOREIGN KEY (`TENANT6`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK7` FOREIGN KEY (`TENANT7`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK8` FOREIGN KEY (`TENANT8`) REFERENCES `TENANT` (`TENANTID`),
ADD CONSTRAINT `TGFK9` FOREIGN KEY (`TENANT9`) REFERENCES `TENANT` (`TENANTID`);
ALTER TABLE `TENANTGROUPPROPERTY`  ADD CONSTRAINT `TGPFK1` FOREIGN KEY (`TGID`) REFERENCES `TENANTGROUP` (`TGID`),
ADD CONSTRAINT `TGPFK2` FOREIGN KEY (`PROPERTYID`) REFERENCES `PROPERTY` (`PROPERTYID`);
```

Appendix D

Sample Contractor Accounts

0011	contractor1@testcase.com	A58C692433619486F3493E25EB9F9A65B5FD1EF83497F3D1B5...	456
0012	contractor2@testcase.com	F1F1D30E16E049D8BE057ABF04EE37EB289B4C3864876A700B...	416
0013	contractor3@testcase.com	0215F298E33593CD6A92D50ED5D89801DB273BD33C15F5D63A...	354
0014	contractor4@testcase.com	CBFC81EDA6C684EE01EF43AC59CCB658162A9401DBA2E70F9F...	126
0015	contractor5@testcase.com	4B13B6209D5572B24C7EECF7D60F37B75B1A50B4C7CD121F00...	919
0016	contractor6@testcase.com	AC6DB175202EB28B87426F00D96E056FBE6519149BF833013C...	398
0017	contractor7@testcase.com	B6475DB1A392760F3E3A063E59F5E9B642FD4C7F2358891FA3...	093
0018	contractor7@testcase.com	F0E220259D569351E06250CD2B8BA0E674E9FC2B769B8398DA...	470
0019	contractor8@testcase.com	957AAC82F06BFBF5D3733408726FB2ADD339AC89571EA7A62C...	397
0020	contractor9@testcase.com	63ED1310B8436A351877A6266154D0B2494EC6BEBC7D4E19B8...	264
0021	contractor10@testcase.com	040A1F2300598EE2D61A7C10A88FF4AC6A813D7AC863A1945F...	356
0022	contractor11@testcase.com	ACD1C0EB2B2B1ABF7DE832713AC9F50F3C845C0A8C4A60B9B1...	072
0023	contractor12@testcase.com	A0F9DB26CFC94823039D104DA8802DF5A3BF493973BCB5F1A0...	756

Sample Contractor Details

CDNUM	CONTRACTORID	FIRST	LAST	DOB	TELEPHONE	ADDRESS1	ADDRESS2	TOWN	COUNTY	POSTCODE
0001	0001	Bob	Waterman	0000-00-00	07800404567	28A Rookwood Rd	*NULL*	London	London	N16 6SS
0008	0011	Lyndon	Halfer	1974-12-27	07485962222	14 Embassy Court, Portsmouth Road		Surbiton		KT6 4HW
0009	0012	Jim	Handsdirty	1993-06-13	07444585435	12 Clarence Street		Kingston		KT1 3EE
0010	0013	Sam	Elevati	1966-12-08	07462845997	90 Long East Walk	Kennington	London		SE11 5GG
0011	0014	Sheilah	Spick	1984-05-26	02084556478	673 Longest Road		Portsmouth		PO18 5TG
0012	0015	Jack	O'falltrade	1960-05-05	07562476289	18 Middle Bridge Street		Romsey		SO51 9IH
0013	0016	Tom	Nombre	1985-07-09	01329888549	Station House, Old Station Rd		Droxford	Hampshire	SO32 1DD
0014	0017	Jodie	Elektra	1988-07-12	07481256999	78 Guild Street		London		NW3 0YU
0015	0018	Scarlett	Holland	1987-08-26	07943140625	45 Folkestone Road		Bournemouth		DT11 2GH
0016	0019	Mia	Freeman	1991-04-22	07921546607	81 Hudson St		Dunbeath		KW6 8BP
0017	0020	Patrick	Conway	1994-07-21	07771587201	118 Sutton Wick Lane		Bridge of Tynet		AB56 2QZ
0018	0021	Adam	Davey	1975-12-22	07015108572	101 Botham Crescent		Ringshall		IP14 6EN
0019	0022	Sebastian	Poole	1987-10-30	07882600264	68 Maidstone Rd		Wellington		TA21 9DZ
0020	0023	Eva	Brookes	1990-01-26	07032405379	55 Bishopgate St		Norbiton		KT1 1UL

Sample Contractor Trades

CTID	CONTRACTORID	Carpenter	Cleaner	Electrician	Gardener	GasEngineer	Gen	Heating	Lift	Locksmith	Painter	Plasterer	Plumber	Pool	Roof
1	0001	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0011	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0012	1	0	1	0	0	1	0	0	0	0	0	1	0	0
5	0013	0	0	0	0	0	1	0	1	0	0	0	0	0	0
6	0014	0	1	0	1	0	1	0	0	0	0	1	0	0	0
7	0015	0	0	1	0	0	1	0	0	0	0	1	1	0	0
8	0016	0	0	0	0	1	0	1	0	0	0	0	1	1	0
9	0017	0	0	1	0	1	1	1	0	0	0	1	1	0	1
10	0018	0	1	0	0	0	0	0	0	0	0	0	0	0	0
11	0019	1	0	0	1	0	1	0	0	0	1	1	1	0	1
12	0020	0	0	1	1	0	0	1	0	0	1	0	0	1	0
13	0021	0	0	0	0	0	1	0	0	1	0	0	0	0	0
14	0022	1	0	1	0	1	0	0	0	0	0	0	1	1	0
15	0023	0	0	0	1	0	1	1	1	1	1	0	0	1	0

Sample Landlord Accounts

00006	landlord1@testcase.com	7DB4D7ACD7019C26A7ADDA7353B81383F110B2AC5E21AAF900...	800
00007	landlord2@testcase.com	03A082BF9735B4B06EC39EC7274F79BA21E5C2883B078DF2B1...	008
00008	landlord3@testcase.com	A5F34C0FF679BA2F3420AD2F0F91CD1EBFF08D481D702DD0E6...	976
00009	landlord4@testcase.com	E63172391FCD68CEE36321C603820430041C0DE067190D3026...	229
00010	landlord5@testcase.com	ABF0532132A5C6A507763A3D82944456D98C4ECED64F726FB1...	097
00011	landlord6@testcase.com	FCA90CCA04449D74FF26AC7D36E1EAC9AB859F8BDC3E301702...	919
00012	landlord7@testcase.com	40B334B675C585AB29583629D272B11070CECD02B4B4395A57...	220

Sample Landlord Data

LDNUM	LANDLORDID	FIRST	LAST	DOB	TELEPHONE	ADDRESS1	ADDRESS2	TOWN	COUNTY	POSTCODE
00001	00001	Timmothy	Richman	1967-03-05	07855346721	64 Grafton Way	Fitzrovia	London	London	W1T 5DN
00004	00006	Michael	Kwadjo Omari Owuo Jr	1993-07-26	07488825633	2201 City West Tower	6 High St	London		E15 2GR
00005	00007	Jane	Higgins	1962-05-09	07845123695	4, The Shard	32 London Bridge St	London	*NULL*	SE1 9SG
00006	00008	Maestar	Chife	1985-06-24	07465215898	18 West Street		London		E3 2AL
00007	00009	Ruby	Amstel	1976-06-16	07845256984	17b High Street	Notting Hill Gate	London		W11 3JE
00008	00010	Mark	Tyson	1988-09-24	07833254692	18 Green Walk		Birmingham		B5 8AL
00009	00011	Gemma	Griffin	1960-04-14	07458215963	18 Battery Hill	Bishops Waltham	Southampton	Hampshire	SO32 1BS
00010	00012	Jacob	Mulhall	1969-04-08	07845321598	18 Buckingham Palace Rd		London		SW1W 0SR

Sample Tenants

TENANTID	email	PWDHASH	SALT
000001	testuser@testcase.com	53747D0477F9D623945897FFCDA47ECA12CFF57E1835F86213...	744
000002	anothertestuser@testcase.com	9050DD26EC688AB282E93E895A837FC4765B020E5AE6505F0E...	946
000003	testthree@testcase.com	872CBABD231A9EEAD773ABFC6D97196A60FA193AA9CF1196F7...	100
000005	tenant1@testcase.com	4D2EDB7CB7CC77FB8A5323131663725083B2F6059FF5E0DBA2...	688
000006	tenant2@testcase.com	86D1DB2E8D2C133E2438627A470D5DCC4C3EED98588758EEEF...	727
000007	tenant3@testcase.com	FB0F89AFCF29954B03454359B20A20EE9D458E16C1C24F25EE...	397
000008	tenant4@testcase.com	29DFD27B849F673E936EE137C6AE52CC654AAC01598E1764C6...	480
000009	tenant5@testcase.com	192A6EAE170591D6CC5615E1C8D83C6C46742CCF75B5C69217...	059
000010	tenant6@testcase.com	7B63123B56DC98E6013210A7CEF95BF0D08B1CD0312672878D...	845
000011	tenant7@testcase.com	532F81EBBA169786E936F8B21C4D8CED532B8FCB58F06FFCF0...	740
000012	tenant7@testcase.com	FEBF1555BB75308A247CB27C84E46CFC9DB28A8F117300AD67...	570
000013	tenant8@testcase.com	268499EE2849813742152F0ACEDE6FE9CBC27A2351E819DBAD...	406
000014	tenant9@testcase.com	15B5321A11B31DDA4E63BF08C00B91630227AE451B33F25391...	085
000015	tenant10@testcase.com	628A10AFA6D44B90E32BB68B1B1D109F8E08CB2F8CE5E36E8A...	990
000016	tenant11@testcase.com	D39F9A1C9E3725EE6C0E31DA0AE8B6351D566120F9A2A57CE8...	534
000017	tenant12@testcase.com	980750584742D496CFCE76893B4E29D08530F8844A81B3458B...	644