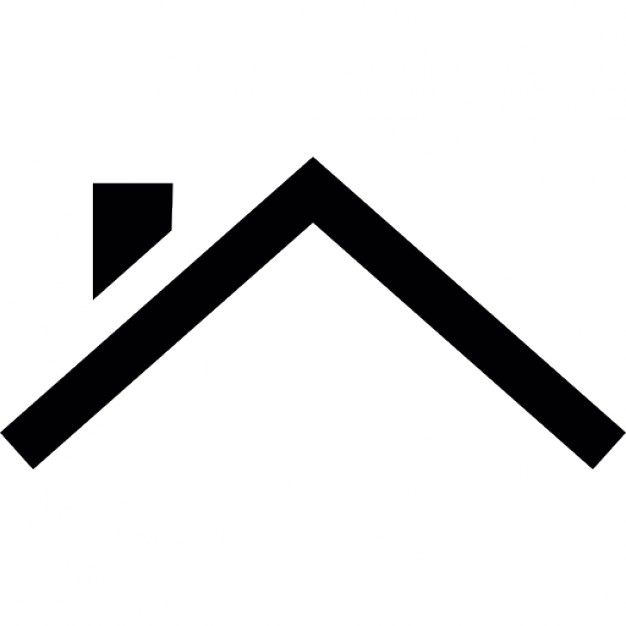
myroom



Jamie Flood

C12492258

Submitted in partial fulfilment of the requirements for the degree of

**Bsc. In Business Computing**

Dublin Institute of Technology

April, 2016

Supervisor: Diana Carvalho E Ferreira

# Declaration

This is an original piece of work. All references and assistance are acknowledged.

Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Acknowledgements

Throughout the design of my final year project and finishing of my studies I have been grateful to have had the patience and help from many people.

I would like to thank Diana, my supervisor, for constantly being helpful and pushing me to make my project better. She has been a great source for creative ideas and has shown great leadership which has aided me throughout my project.

I would also like to thank Audrey, my second reader, for contributing some great ideas during the checkpoint meetings and suggesting some great improvements.

I would also like my classmates who have been a tremendous help over the year as well as all other DIT lecturers for their assistance throughout my four years of college.

# Table of Contents

[Declaration 2](#_Toc448767849)

[Acknowledgements 3](#_Toc448767850)

[Table of Contents 4](#_Toc448767851)

[Introduction 6](#_Toc448767852)

[Project plan 6](#_Toc448767853)

[Project objectives 7](#_Toc448767854)

[Business case 7](#_Toc448767855)

[Technologies used 8](#_Toc448767856)

[Java 8](#_Toc448767857)

[MySQL 8](#_Toc448767858)

[HTML5 8](#_Toc448767859)

[CSS 8](#_Toc448767860)

[Bootstrap 8](#_Toc448767861)

[JavaScript 8](#_Toc448767862)

[Git/GitHub 9](#_Toc448767863)

[Apache Tomcat 9](#_Toc448767864)

[Spring 9](#_Toc448767865)

[Google Maps API 9](#_Toc448767866)

[Requirements analysis 10](#_Toc448767867)

[Functional requirements 10](#_Toc448767868)

[Non-functional requirements 12](#_Toc448767869)

[Usability 13](#_Toc448767870)

[Accessibility 13](#_Toc448767871)

[Responsiveness 13](#_Toc448767872)

[Price 13](#_Toc448767873)

[Security 13](#_Toc448767874)

[Use cases 15](#_Toc448767875)

[Design 16](#_Toc448767876)

[System architecture 16](#_Toc448767877)

[Choices of technologies 17](#_Toc448767878)

[Main programming language 17](#_Toc448767879)

[Main framework 17](#_Toc448767880)

[Database options 17](#_Toc448767881)

[Database connectivity 18](#_Toc448767882)

[Web server 18](#_Toc448767883)

[Front end technologies 18](#_Toc448767884)

[Libraries 18](#_Toc448767885)

[Data model 20](#_Toc448767886)

[House 20](#_Toc448767887)

[User 21](#_Toc448767888)

[Comment 22](#_Toc448767889)

[Task 23](#_Toc448767890)

[Contact 23](#_Toc448767891)

[Roomie 24](#_Toc448767892)

[Message 24](#_Toc448767893)

[Software Implementation 26](#_Toc448767894)

[Overview of implementation 26](#_Toc448767895)

[User registration 26](#_Toc448767896)

[Login & Log out 27](#_Toc448767897)

[Creating a house 28](#_Toc448767898)

[Displaying the map 29](#_Toc448767899)

[Viewing all houses 30](#_Toc448767900)

[Viewing a user’s house 31](#_Toc448767901)

[Searching for houses 32](#_Toc448767902)

[Searching for a user 33](#_Toc448767903)

[Viewing a user’s profile 33](#_Toc448767904)

[Messaging 34](#_Toc448767905)

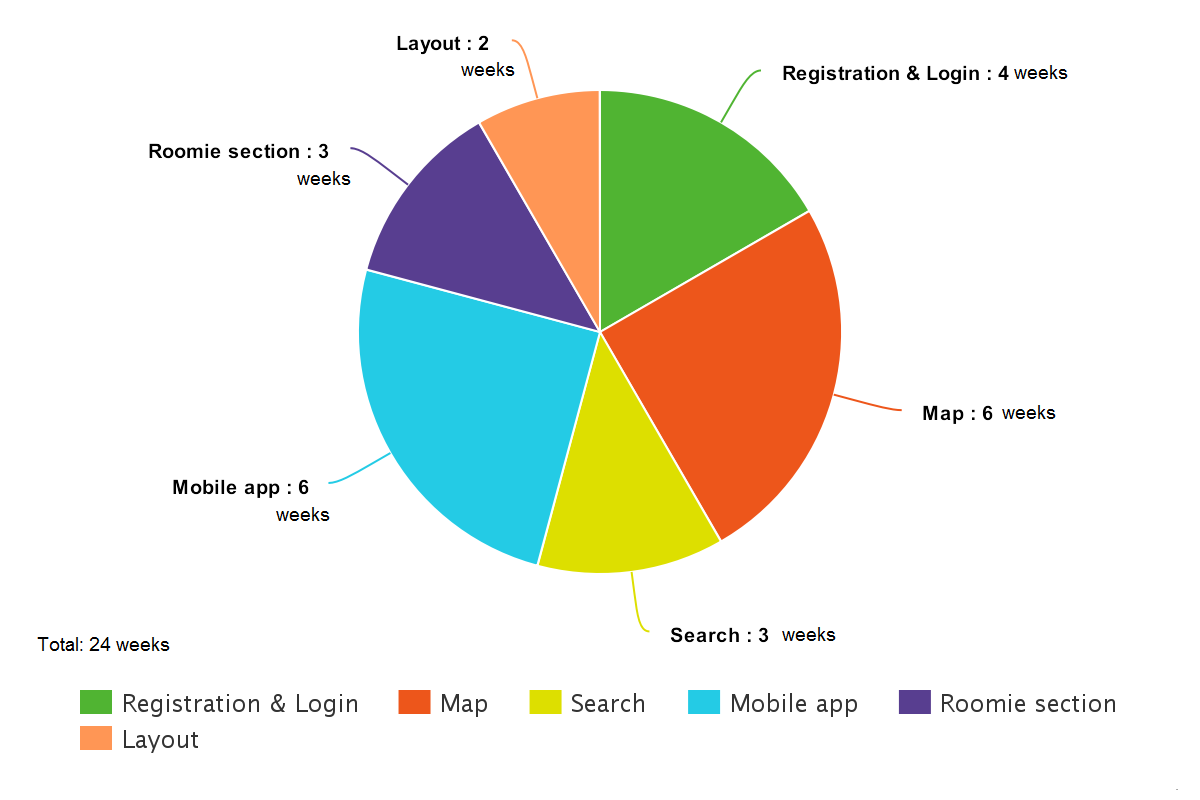
[Roomies 36](#_Toc448767906)

# Introduction

The aim of this project was to build a modern web app for students to find the perfect roommate. I came up with the idea after looking for accommodation in the summer of 2015. I came across many websites which just offered the bare minimum for their users and I was quite surprised. I thought it would be great if there was a website which combined activities like searching for a house, adding a roommate and managing tasks within a house.

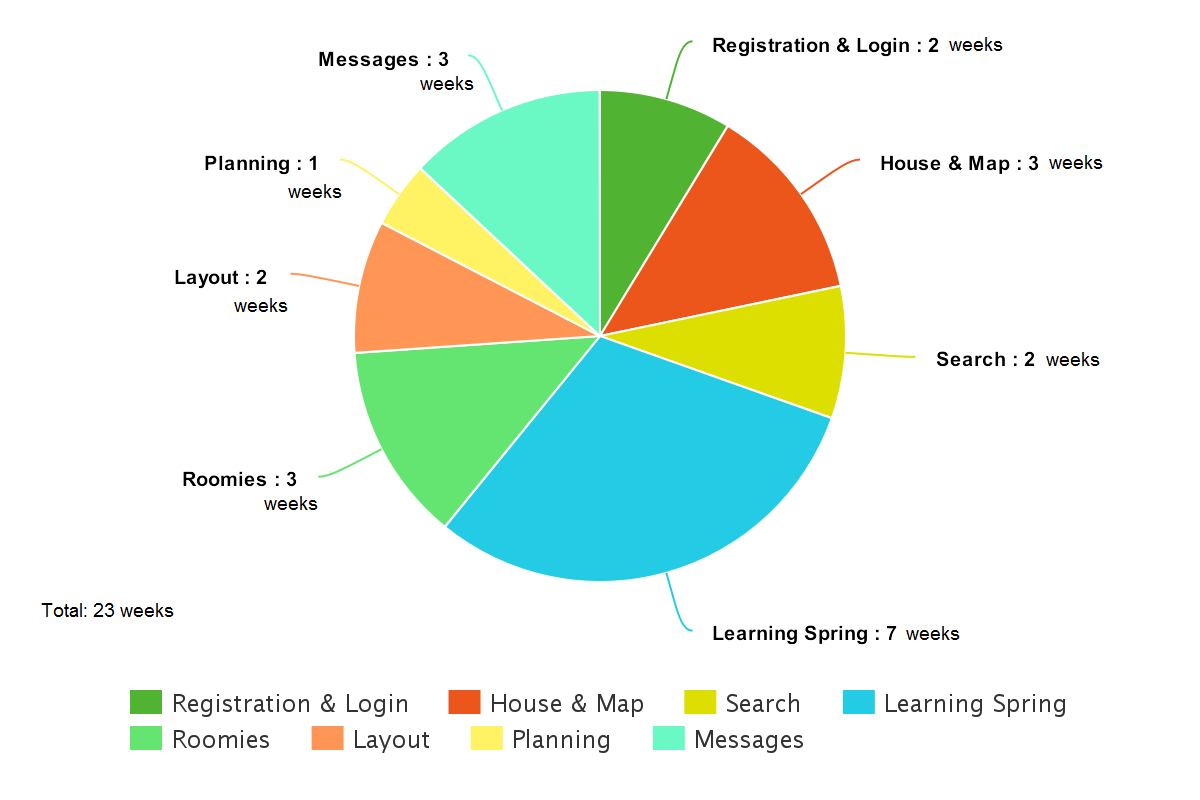
The project was built using technologies which were entirely new to me. This was one of my main aims because I wanted to gain skills in technologies I had not used in DIT. I believe that this project has helped me gain experience with other technologies and it will help me greatly as I embark on a career in software development.

## Project plan



Above is my original project management plan. Initially I had planned for the entire project to take 24 weeks. I ended up finishing it in 23 weeks but some tasks I hadn’t accounted for and some I overestimated how long some would take. I had also planned on developing a mobile app to go along with my web app but it turned out that this would not be realistic so I decided to focus all my attention on the web app.

In my original plan I hadn’t accounted for how long it would take to learn a new technology. Learning how to use Spring took me roughly 6 weeks. Once I had learned how to use Spring most features of my application took less time than I had expected. Below is how long all features of the project actually took.



## Project objectives

The main objective of this project was to provide an all-in-one solution to student accommodation. Users should be able to register and either create a house or join a house. Once they are in a house they can have roomies which allows them to create and manage tasks that have to be completed, create useful contacts within the house and message their roomies. I also thought it would be useful to have an advanced search where users could search for specific features of a house or houses in certain areas. I wanted to make it easier for students to find the perfect house or roommate as this one of the main problems for students in Ireland in modern times.

## Business case

I have mentioned before that my project is aimed at students, but in theory it can be used by anybody who is looking to find accommodation. Ireland currently has a housing crisis and the price of housing has gone up significantly in recent times. My application would allow people to find a single room in a house rather than just a house. I am hoping that my solution could help find people a better, more comfortable place to live.

Early on I decided that the map feature of my application would be vital to its success. Users would like to be able to view houses by their location because this is usually the key element when people are looking for houses.

# Technologies used

Throughout the development of my project I have used a vast amount of technologies. Before I began I was familiar with Java, MySQL, HTML5, CSS, Bootstrap, JavaScript, Git/Github and Apache Tomcat. I had very little experience with Spring, Hibernate or Google Maps API.

## Java

Java was the language I chose to implement myroom. Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented and specifically designed to have as few implementation dependencies as possible. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. It was the language I had the most experience with and I was fully confident I could successfully develop my project in the best way with Java.

## MySQL

MySQL is an open-source relational database management system. I chose MySQL as my database because of its high performance. MySQL can meet the most demanding performance expectations of any system because of its high-speed load utilities and memory caches. Realistically I had to expect to be storing huge amounts of data and relied on speed so I chose the database I felt was best capable of this.

## HTML5

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It was the obvious choice as I had used it throughout my work placement and felt I had gained lots of experience with it.

## CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. CSS allowed my application to look as good as it does. Again, I had used CSS in my work placement so decided to use it when styling my HTML.

## Bootstrap

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. I used Bootstrap along with CSS to style certain elements of my HTML. It was easy to use and documentation was extremely useful.

## JavaScript

JavaScript is a high-level, dynamic, untyped, and interpreted programming language. Alongside HTML and CSS, it is one of the three core technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern Web browsers without plug-ins. I chose JavaScript as it was vital to the successful implementation of the map feature. It was quite difficult to get to grips with because I have never fully enjoyed using JavaScript but I realise its importance in the software development industry and had to study a lot more of it.

## Git/GitHub

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. GitHub is a web-based Git repository hosting service. I used these as it was an easy way of backing up my project and keeping track of any changes that I had made.

## Apache Tomcat

Apache Tomcat, often referred to as Tomcat, is an open-source web server developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment for Java code to run in.

## Spring

The Spring Framework is an open source application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. I decided to learn the Spring framework as there were so many resources available to aid me learning it as well as the fact it is one of the most popular Java frameworks.

## Google Maps API

Google Maps API is an application programming interface (API) developed by Google which allows communication with Google Services and their integration to other services. The Google Maps API is a JavaScript library which had to be included. I chose this as it was the clear choice for displaying information on a map due to its ease of access and popularity. Most users of my application would already have previous experience with Google Maps.

# Requirements analysis

For the requirements analysis stage of my project I had to really think about what exactly a student would want when looking for accommodation. I spoke to a few of my friends who were students and they indicated that the key feature would be searching for specific features of a house or specific hobbies that users have. I tried to use an agile approach to my project, so I really had to put myself in the ‘clients’ shoes to identify the requirements.

## Functional requirements

A functional requirement essentially is something the system should do. The following are the functional requirements that I gathered:

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Title | Requirement | Justification |
| Register | Register an account | Register an account. User must enter username, password, email, name, date of birth, hobbies, college, relationship status | Certain parts of the application are only available to users that are registered. In order to access these a user needs an account. |
| Login | Login using a username and password | All registered users should be able to login to the application | Users must be able to login to view data belonging to them. |
| Logout | Logout | All users should be able to logout of the system once they have logged in | For security reason all users must be able to log out of the system. Once they have logged out the session should be deleted. |
| House | Create a house | Users can create a house. The address, rooms left, rent and description should be entered | This is needed in order for users to view houses. |
| House | Edit a house | Users can choose to edit their existing house | If any changes in real life are made to the house they should be able to be updated |
| House | Delete a house | Users can delete a house | It is the user’s choice when they want to get rid of the house |
| House | Advanced search for a house | Searching is done based on amount of rooms, rent amount and location | Key feature for the application. Users want to be able to find houses based on their interests |
| House | View a house | View a specific house and all its details | If a user views a house they should be able to contact the owner |
| House | View a map of all houses | Display a map with markers on the location of each house | Users want to see exactly where a house is located |
| Admin | Delete a user or house | Delete a user or house from the system | Administrators have complete control of who uses the application and how |
| User | View a user’s page | View a user’s page and a house that they own | In order to connect with a roomie, users will want to view their details first |
| User | Search for a user | Search for a user by name or hobbies | Users would like to be able to find other people whom share similar interests to themselves |
| User | Upload a profile picture | When viewing someone’s page it should display a profile picture | Users may like to see what potential roomies look like before connecting with them |
| User | Add a roomie | When on someone’s page you should be able to add them as a roomie | In order to manage tasks within a house you have to be roomies with the user |
| Task | Create a task | Create a task to be completed by a specific date | Roomies within a house should share tasks within a week |
| Task | View all tasks | View all tasks that have to be completed | Let’s all roomies within a house know what has to be completed for when |
| Task | View a task | View an individual task and all its details | Roomies need to be able to view who created a task, for when and what specifically the task is |
| Contact | Create a contact | Create a contact that roomies can contact when they need to | Roomies can contact these if they need assistance e.g. plumber, electrician, landlord etc. |
| Contact | View all contacts | View all contacts that have been created | All roomies are able to view all contacts created by their roomies |
| Contact | View a contact | View a contact to view its specific details | Roomies need to be able to view a contact in order to contact them |
| Roomies | View all/ delete roomie | View all roomies on a page or delete a roomie if wanted | It is the user’s decision who their roomie is. They have the authority to remove one if they want to |
| Message | Create a message | Users should be able to send other users a message. They don’t have to be roomies with each other | Users might want to message users privately to find out more details about a house or them before adding them as a roomie |
| Message | View all messages | Users should be able to view a list of people they have had conversations with | This is needed to know if a user has a new message |
| Message | View a conversation | View a conversation with another user. Must be able to reply on this page too | This is needed in order to view what you have previously said to the user and what they have said |

## Non-functional requirements

While functional requirements describe what the system does, nun-functional requirements describe how the system works.

### Usability

The application has a very friendly UI as this keeps the users from going elsewhere. I ensured that all necessary pages were accessible from just one or two clicks of a button. Users don’t want to have to spend lots of time getting used to a system. This was a hugely important factor when developing my application.

### Accessibility

I made it a point to make sure my application was supported by most browsers. This was important because some users may not have the latest version of their browsers and it enables more users to have access to my application. All forms are labelled efficiently and any buttons have text describing what the button does.

### Responsiveness

As this application is aimed at students, I figured that it should be easy to view on smart phones as well. This was why I chose to use bootstrap for styling my elements. It is a responsive framework that adapts to the changes in screen size.

### Price

As students don’t tend to have a lot of money, I decided to make my application free to all users.

### Security

Security was a huge factor because I didn’t want people being able to see data they were not authorised to. To enforce security on passwords I used encryption to ensure the information remained private. I added different roles for different users: admin and users. The admin has certain privileges that a user doesn’t such as deleting user accounts or deleting houses. I felt a login page was needed to access certain part of the application such as roomies or creating/editing a house.

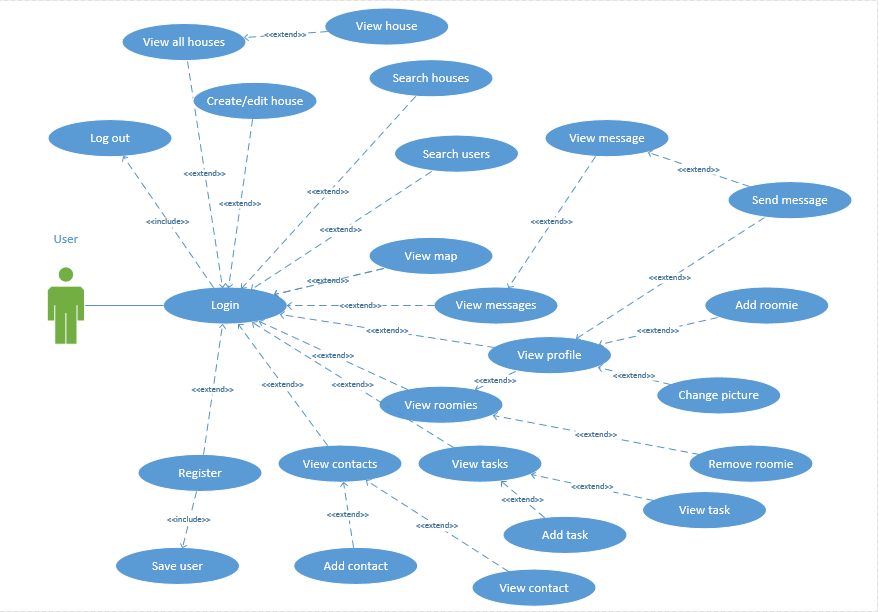
#### Authorization

Spring security provides an easy way to allow the developer to declare who has access to certain URLs in the application. Intercept-url elements specify certain access attributes for a particular set of URLs. Each pattern contains a method in the controller classes which handles the request.

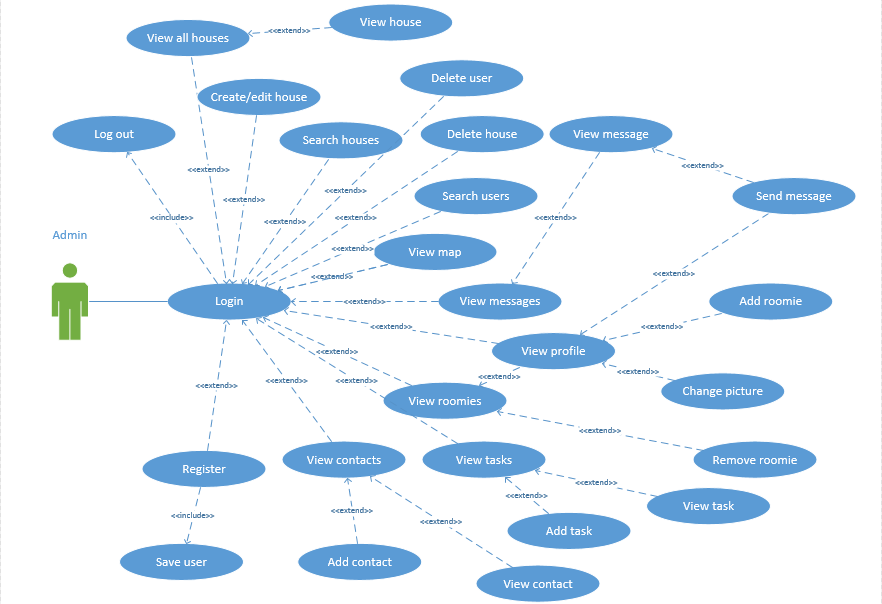
|  |  |  |
| --- | --- | --- |
| Pattern | Access | Role |
| / | permitAll | Anyone |
| /admin | hasRole(‘ROLE\_ADMIN’) | Administrator |
| /addRoomie | isAuthenticated() | User |
| /contactcreated | isAuthenticated() | User |
| /noroomsavailable | isAuthenticated() | User |
| /newcontact | isAuthenticated() | User |
| /createcontact | isAuthenticated() | User |
| /createcomment/\* | isAuthenticated() | User |
| /commentsent | isAuthenticated() | User |
| /deleteRoomie/\* | isAuthenticated() | User |
| /roomieadded | isAuthenticated() | User |
| /showhouseupload | isAuthenticated() | User |
| /houseupload | isAuthenticated() | User |
| /showupload | isAuthenticated() | User |
| /upload | isAuthenticated() | User |
| /newmessage | isAuthenticated() | User |
| /sendmessage | isAuthenticated() | User |
| /sendmessage/\* | isAuthenticated() | User |
| /messagesent | isAuthenticated() | User |
| /messages | isAuthenticated() | User |
| /contacts | isAuthenticated() | User |
| /contacts/\* | isAuthenticated() | User |
| /newtask | isAuthenticated() | User |
| /createtask | isAuthenticated() | User |
| /tasks | isAuthenticated() | User |
| /tasks/\* | isAuthenticated() | User |
| /housenotfound | permitAll | Anyone |
| /roomies | isAuthenticated() | User |
| /roomies/all | isAuthenticated() | User |
| /search | permitAll | Anyone |
| /searchrent | permitAll | Anyone |
| /searchroomie | permitAll | Anyone |
| /searcherror | permitAll | Anyone |
| /user/\* | permitAll | Anyone |
| /message/\* | isAuthenticated() | User |
| /house/\* | permitAll | Anyone |
| /deletehouse/\* | isAuthenticated() | User |
| /deleteuser/\* | isAuthenticated() | User |
| /denied | permitAll | Anyone |
| /createhouse | isAuthenticated() | User |
| /docreate | isAuthenticated() | User |
| /housecreated | isAuthenticated() | User |
| /loggedout | permitAll | Anyone |
| /static/\*\* | permitAll | Anyone |
| /newaccount | permitAll | Anyone |
| /createaccount | permitAll | Anyone |
| /accountcreated | permitAll | Anyone |
| /login | permitAll | Anyone |
| /houses | permitAll | Anyone |
| /map | permitAll | Anyone |
| /\*\* | denyAll | Nobody |

## Use cases

The below diagram describes the functionality for a standard user

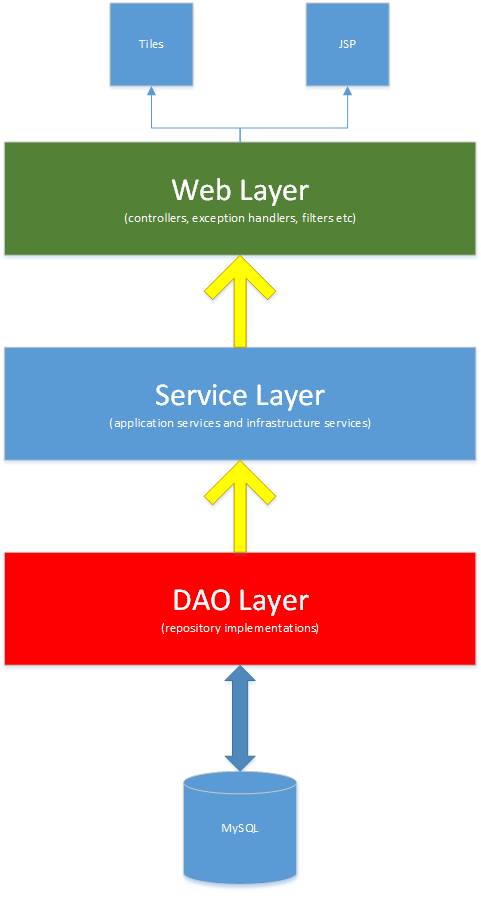


The below diagram describes the functionality for an administrator



# Design

## System architecture



## Choices of technologies

### Main programming language

A hugely important decision that I had to make early on was what programming language I was going to use. I thought of the future and what languages I would like to use when I enter the industry. We had studied many programming languages throughout the course, but there were only two that I felt I would be comfortable choosing to develop an application of this scale.

#### Ruby

At first I considered using Ruby. Ruby is a dynamic, open source programming language with a focus on simplicity and productivity.

#### Java

I chose to use Java to develop my application. This was a no-brainer really because I have studied it for four years and feel extremely comfortable using it. Because I had four years’ experience with Java, it enabled me to focus my attention on learning a framework which I had not previously studied.

### Main framework

#### Ruby on rails

Ruby on rails is a web development framework written in the Ruby language. It is a MVC framework. I had studied this in a class in third year and enjoyed using it but felt I should learn something new that would help me in the future too.

#### Spring

While I was looking for potential frameworks to use I came across Spring. Spring is a Java framework which uses the MVC pattern. One of the main things that attracted me to Spring was how efficient it was for unit testing. I considered this of huge importance at the beginning of my project as I knew it would save be a lot of time if managed successfully.

I also like the way it simplified the complexity when implementing business logic in applications. This was one of the reasons why I didn’t choose EJB because I felt it was quite complex when it shouldn’t have been.

### Database options

#### Microsoft Access

I have used Microsoft Access regularly for a few small projects outside college, mainly just to keep track of tasks. I had considered using it briefly for my application but felt it would handle an application of the scale of mine.

#### MySQL

MySQL is an open-source relational database management system. It is very widely used by companies. I chose MySQL as my database because of its high performance. I had lots of experience with it and I knew it would not take long to set up with my application. Along with using the command line for querying my database, I used the MySQL workbench to reverse engineer my database which allowed me to easily view my database model.

### Database connectivity

#### Hibernate

Hibernate ORM (Hibernate in short) is an object-relational mapping framework for the Java language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions. I had used it before in a module and felt it would be vital to my project. I knew I could use JPA annotations with it and allow hibernate to implement these. Hibernate Query Language (HQL) could also be used to write SQL-like queries against Hibernates data objects. I chose hibernate because of my experience with it and the fact that it had some great documentation which made it very easy to use.

### Web server

#### JBoss

JBoss is an application server used for developing and deploying Java applications and web applications. As it is an eclipse based IDE and I had used it before for other projects, I seriously considered using it for my application.

#### Apache Tomcat

Apache Tomcat, often referred to as Tomcat, is an open-source web server developed by the Apache Software Foundation. Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment for Java code to run in. I chose to deploy my application on a Tomcat server because I felt it met my performance requirements and it could easily handle a project of my size.

### Front end technologies

Both JQuery and JavaScript were used throughout the development of my application. I used a JavaScript library to sort tables in ascending or descending order. I also use it for validation when registering to ensure that the confirmation password is the same as the original password the user has entered. I also use a lot of JavaScript for displaying the house information on the maps. In this I loop through two arrays containing longitude and latitude information and display each location as a marker on the map. It also displays an infowindow when a marker is clicked on.

### Libraries

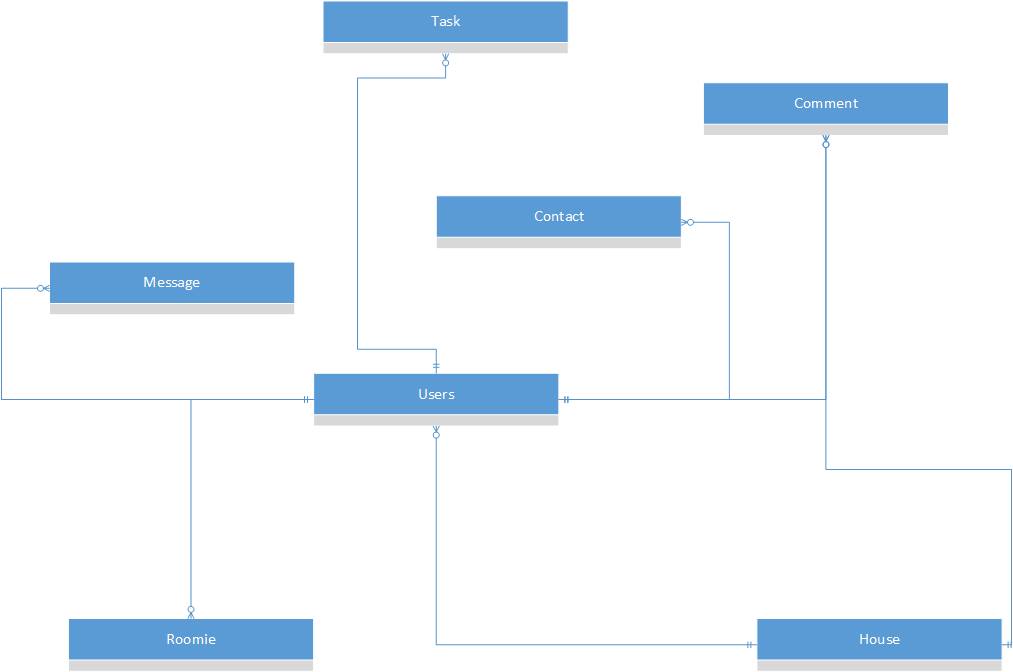
#### Jsoup

Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. Jsoup scrapes and parses HTML from a URL and extracts data using DOM traversal. I used this for extracting the longitude and latitude information for a house when an address is entered.

#### Apache Tiles

Tiles is a free open-sourced templating framework for modern Java applications. Tiles allows authors to define page fragments which can be assembled into a complete pages at runtime. These fragments, or tiles, can be used as simple includes in order to reduce the duplication of common page elements or embedded within other tiles to develop a series of reusable templates.

## Data model



### House

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Id | Int | This is the primary key. It is used throughout the application to reference houses. |
| Address | String | Used to search Google APIs for a location |
| Central\_heating | Boolean | Stores a true or false value if the house has central heating. Default is false. |
| Description | String | Used to describe a house. Stored in the database as LONGTEXT. Cannot be blank. |
| Dishwasher | Boolean | Stores a true or false value if the house has a dishwasher. Default is false. |
| Dryer | Boolean | Stores a true or false value if the house has a dryer. Default is false. |
| Furnished | Boolean | Stores a true or false value if the house is furnished already. Default is false. |
| Garden | Boolean | Stores a true or false value if the house has a garden. Default is false. |
| House\_alarm | Boolean | Stores a true or false value if the house has an alarm. Default is false. |
| Latitude | String | Stores the latitude details of the address. Used to locate the house on a map. |
| Longitude | String | Stores the longitude details of the address. Used to locate the house on a map. |
| Microwave | Boolean | Stores a true or false value if the house has a microwave. Default is false. |
| Parking | Boolean | Stores a true or false value if the house has parking available. Default is false. |
| Rent | Int | Stores the amount to rent the house per month in euros. |
| Rooms | Int | Stores the amount of rooms available to rent in the house. |
| Television | Boolean | Stores a true or false value if the house has a television. Default is false. |
| Username | String | Stores the username of the house owner/landlord. |
| Washing\_machine | Boolean | Stores a true or false value if the house has a washing machine. Default is false. |
| Wifi | Boolean | Stores a true or false value if the house has wifi. Default is false. |

### User

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Username | String | This is the primary key. It is used throughout the application to reference users. Also used to perform validation. Cannot be blank and must be between 8 and 15 characters. |
| Authority | String | Describes what authority the user has. Can be either “ROLE\_USER” or “ROLE\_ADMIN”. |
| College | String | Stores the information for where the user goes to college. |
| Dob | String | Stores the date of birth of a user. |
| Email | String | Stores the email address of the user. Validation is done to check if it is a valid email. |
| Enabled | Boolean | Describes if a user is enabled to access the application. Stores a true or false value if the user is enabled or not. |
| Hobbies | String | Stores the various different hobbies that a user can have. |
| Name | String | Stores the name of the user. Must be at least 8 characters and less than 60. |
| Password | String | Stores the password of the user. After registering it is encrypted in the database. Must be between 8 and 15 characters and must be entered. |
| Phone | String | Stores the phone number of the user. |
| Relationship | String | Stores the relationship status of the user. Can be either “single” or “in a relationship”. |
| Work | String | Stores the work information for the user. |
| House\_id | Int | Stores the id of the house that the user created. This is a foreign key. |

### Comment

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Id | Int | This is the primary key. It is used throughout the application to reference comments. |
| Date | Date | Stores the date that the comment has been created. |
| Text | String | Stores the text that the user has entered. |
| House\_id | Id | Stores the id for the house that the comment belongs to. Used when getting all comments for a particular house. This is a foreign key. |
| User\_username | String | Stores the id for the user that the comment belongs to. Used when getting all comments for the specific user. This is a foreign key. |

### Task

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Id | Int | This is the primary key. It is used throughout the application to reference tasks. |
| Date\_complete | Date | Stores the date that a task is to be completed by. |
| Date\_created | Date | Stores the date that the task was created. |
| Details | String | Stores the details of the task e.g clean the kitchen, wash the dishes etc. Cannot be blank. |
| Name | String | Stores the name of the task. Must be between 7 and 60 characters. |
| Username | String | Stores the username of the user that created the task. This is a foreign key. |

### Contact

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Id | Int | This is the primary key. It is used throughout the application to reference contacts. |
| Email | String | Stores the email address of a contact. Validation is done to check if it is a valid email. |
| Name | String | Stores the full name of the contact. Must be between 8 and 60 characters. |
| Phone | String | Stores the phone number of the contact. Must be between 5 and 20 characters. |
| Role | String | Stores the role of the contact e.g. plumber, electrician etc. Must be between 2 and 60 characters. |
| Username | String | Stores the username of the user that created the contact. This is a foreign key. |

### Roomie

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Id | Int | This is the primary key. It is used throughout the application to reference roomies. |
| Roomie\_username | String | This is the username of the roomie the user has added. |
| Username | String | Stores the username of the user that added the roomie. This is a foreign key. |

### Message

|  |  |  |
| --- | --- | --- |
| Field | Data type | Description |
| Message\_id | Int | This is the primary key. It is used throughout the application to reference messages. |
| Recipient | String | Stores the username of the recipient of the message. |
| Send\_date | Date | Stores the date that the message was sent. |
| Text | String | Stores the details of the text. Cannot be blank. |
| Title | String | Stores the title of the message. |
| Username | String | Stores the username of the user that created the message. This is a foreign key. |

# Software Implementation

## Overview of implementation

Implementing the features of my application was a step-by-step process. Some features were dependent on others so I had to prioritise the most important features and then develop from there. As I was developing the application I had to refactor and remodel quite a lot of data, which often required me to recreate my database.

The first step I made in my application was to build my database the way I originally expected it to turn. This of course wasn’t the finished structure of the database but I used it as a starting point. I made sure that all foreign keys were referenced correctly.

The user interface is incredibly important when developing an application. I ensured that my UI was easy to use. I made sure that my homepage made a lasting impression on users so they would return. Bootstrap helped a lot with my navbar and various buttons and forms across the application. I tried to have similar layouts for pages so that it was easy for users to use.

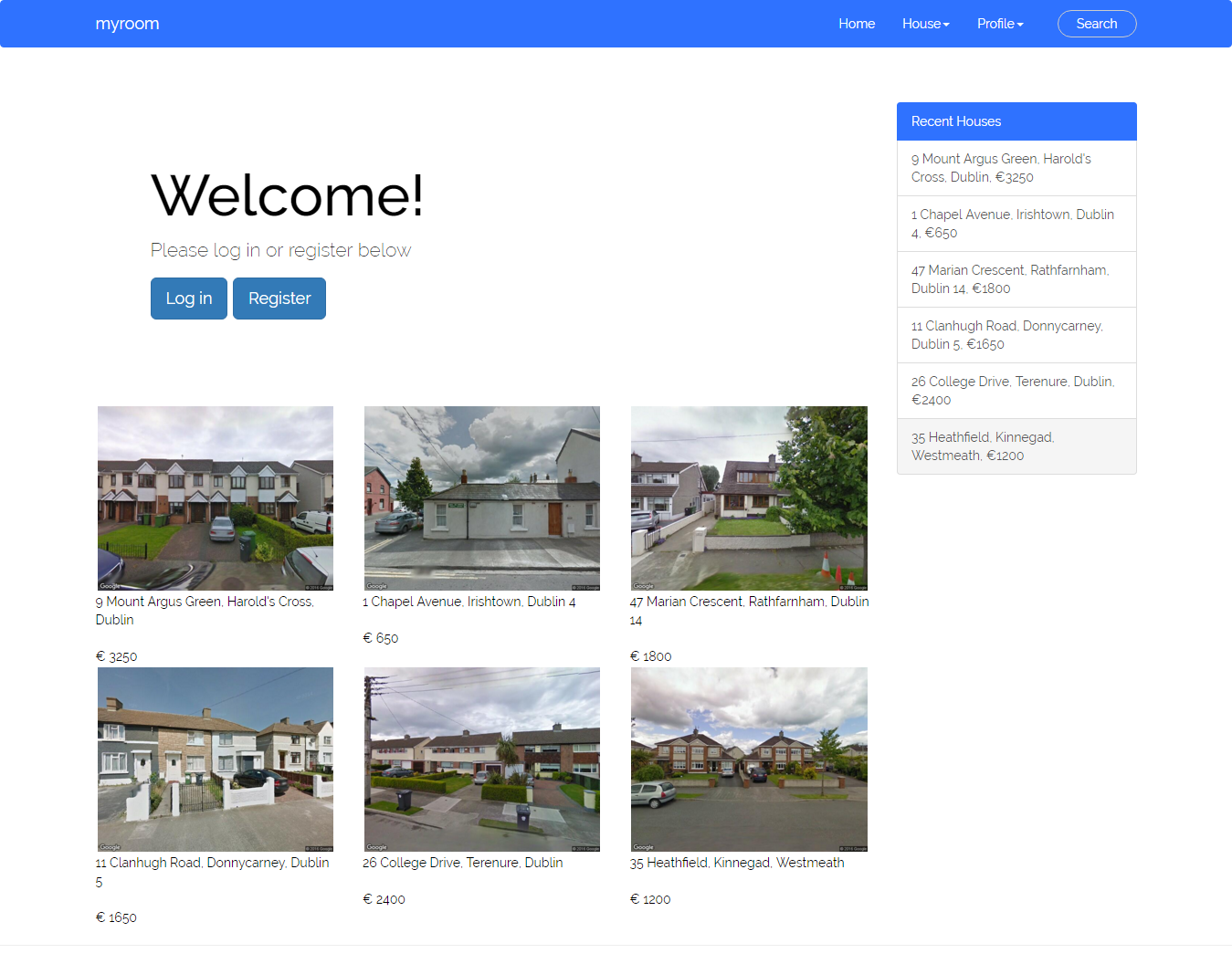


Figure . The initial page that a user is presented with

## User registration

The first step was to develop the ability for a user to register an account with the application. There are two Java methods involved in the registration of an account: showNewAccount() and createAccount(). In showNewAccount(), a new user object is created and added to the model and a registration page is displayed. createAccount() is called once the user submits the form. In createAccount(), validation is performed to ensure that the data that was entered is valid. If any errors are found then the original form is displayed again with the specific errors, as seen in the image below. JavaScript is used to ensure that the two password fields are the same. If they are not then the user cannot submit the form. If the data entered is valid then the user is saved in the database and then notified that the account has been created.

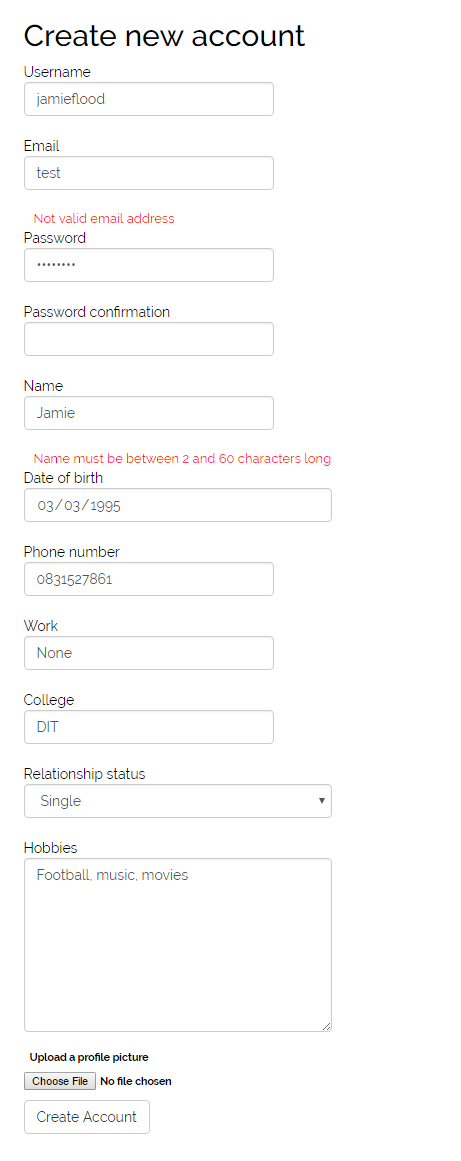


Figure . Errors found when attempting to register an account

## Login & Log out

Spring security has some excellent features which help with logging in and logging out. In my security-context xml file I declared an http element which has logout, login, session and url elements. A login form is displayed and the user is asked to enter their username and password. An authentication provider is used to select users from the database and if the username, password and authority is correct. When the user is logged in a session is created. I have set the session to timeout after 20 minutes as I felt extra security should be added. When the user wishes to log out then a log out success page is displayed.



Figure . Login form

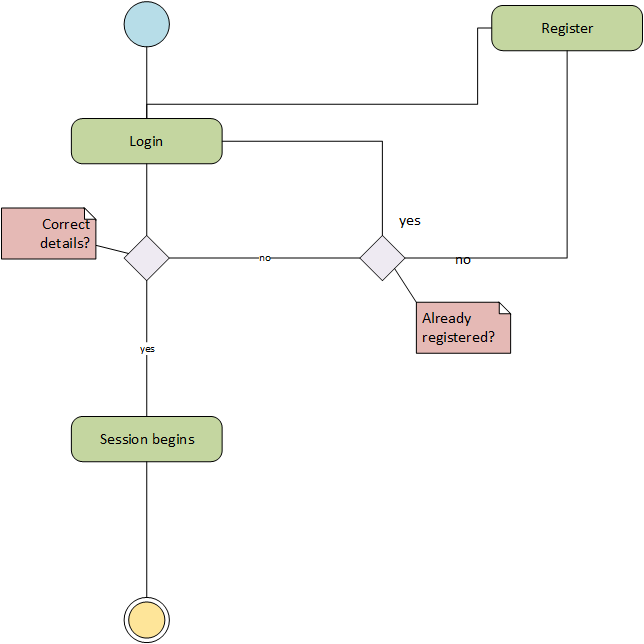


Figure . Login activity diagram

## Creating a house

In order for a user to create a house, they must first be logged in. Once they are logged in then they have permissions to create a house. Like the registration of an account, two methods are used to create a house: createHouse() and doCreate(). The createHouse method's main responsibility is to display the form for users to create or edit their house. A house object is first declared as null. A Principal variable is used to check if the logged in user has a house already created. If it does, the house details are displayed on the form for the user to edit. If not, the form is displayed and the house object is added to the model. The doCreate method's main responsibility is to actually create the house. A house object and the BindingResult is sent from the form. It also checks whether the user wishes to save the house or delete from the jsp. If the BindingResult has errors present then the form is displayed again with the errors.

If the user wishes to save the house, the Google Maps API is used to check if the house address actually exists. Jsoup is used to scrape the latitude and longitude from the result page and the house is added to the database and a "success" page is displayed. Otherwise an error page is displayed. If the user wishes to delete then the id of the house is retrieved from the object and the houseService method for deletion is called.

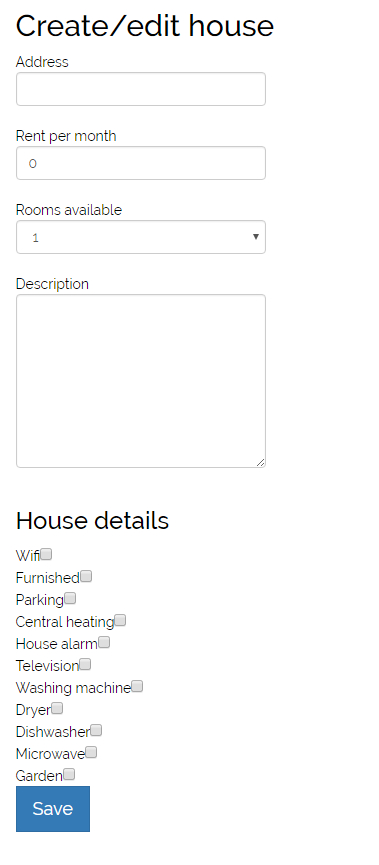


Figure . Creating a house

## Displaying the map

Displaying the map was hugely important to my application. Google Maps allowed me to add each house to a map and display information about the house on an infowindow. All houses are retrieved from the database and added to the model so they can be accessed in the JSP. In the map.jsp, JavaScript is used to display the map. Arrays are created for latitude, longitude, room, rent and id information for each house. These arrays are then looped through to add the marker to a specific location on the map.

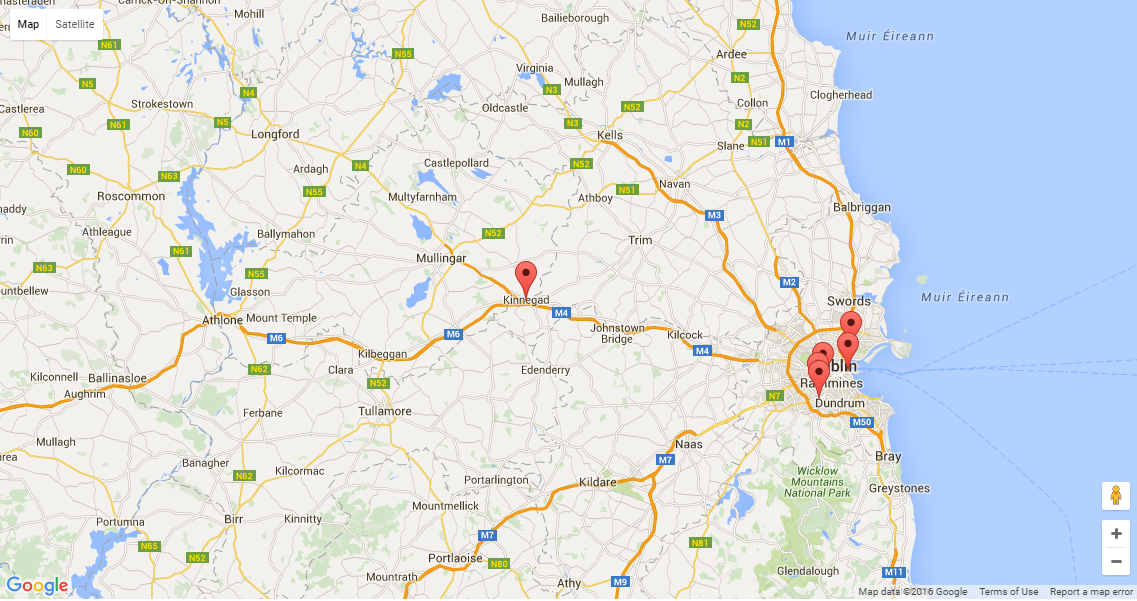


Figure . Houses are displayed on the map

A listener is added to this marker so when it is clicked on an infowindow is displayed with information about the house.

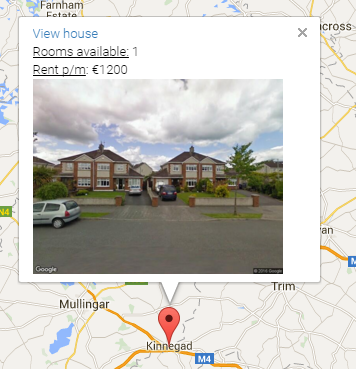


Figure . A marker is clicked on

## Viewing all houses

Displaying all houses that are in the system allows users to choose which matches their needs most. This was vital to the application as it brings the user one step closer to adding a roomie. The user can then click on the house address and view more information about the house. There is also the option to add/edit a house or view the houses on the map. Code wise all it involved was retrieving a list of houses from the database and looping through this list in the JSP page to add each house to a table.

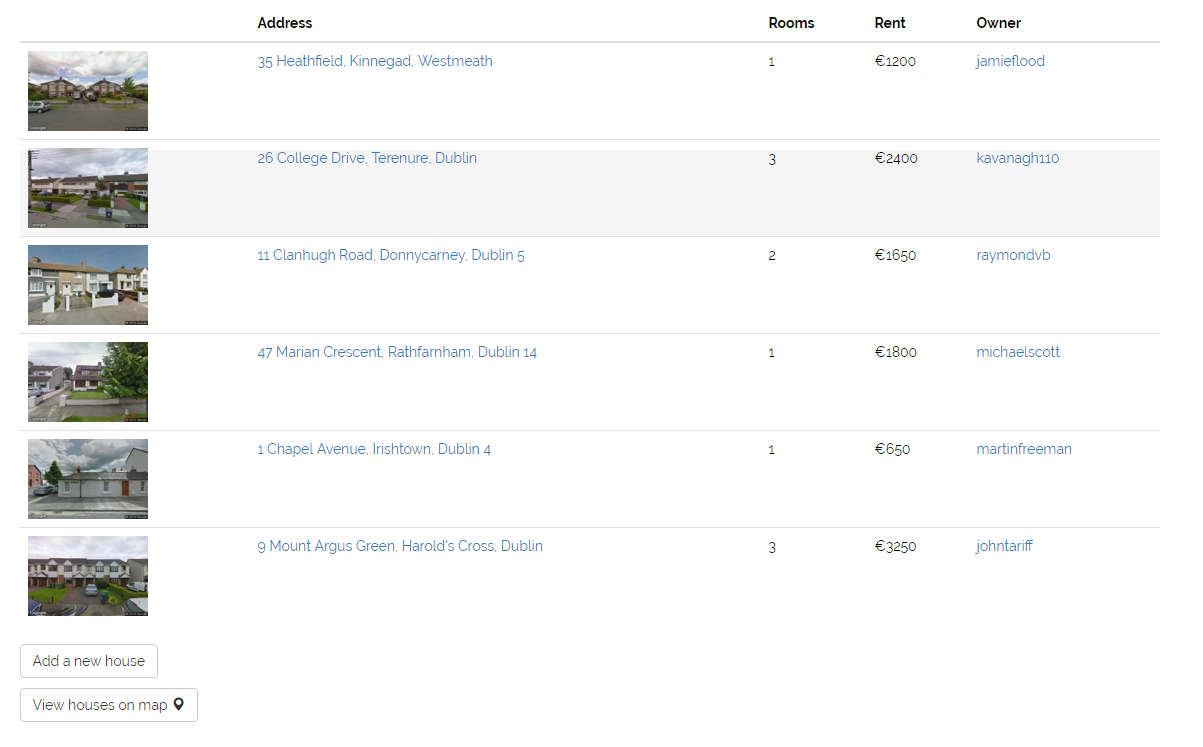


Figure . Displaying all houses that exist

## Viewing a user’s house

Viewing a house can allow the user to do many things: view the owners profile, view the facilities of the house, view the picture the owner has uploaded, message the owner and post a comment. I tried my best to make this as realistic as possible so I designed it like the house pages on daft.ie, but with extra functionality.

In terms of code this only required one method in my controller: showHouse().The showHouse method's main responsibility is to view a particular house. It also takes the house id as a parameter and uses the houseService to retrieve the house information. Message and comment obects are instantiated and added to the model so a user can message the owner or post a comment. A list of all comments on that house is also retrieved from the database and displayed.

House pages look visually different depending on who is viewing them. If the owner is viewing it, then they have certain permissions like editing the house or uploading a picture whereas if it is just a standard user then they have the basic permissions.

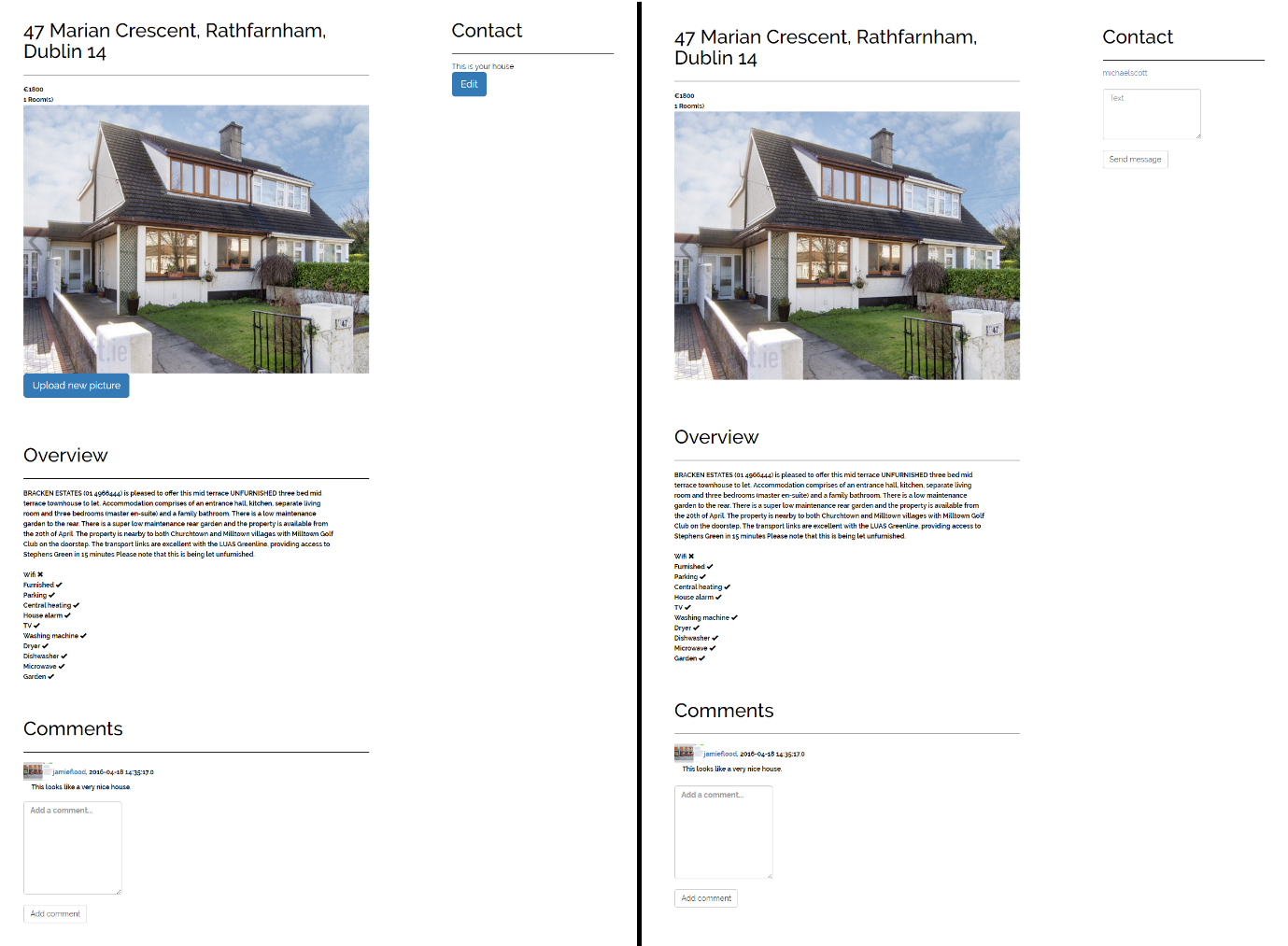


Figure . Viewing the house as the owner vs. viewing the house as a user

## Searching for houses

A user can search for a house on the homepage. They enter a town/county, minimum rooms, maximum rooms, minimum rent and maximum rent and press the search button. A list of results is returned and these can be sorted in ascending or descending order.

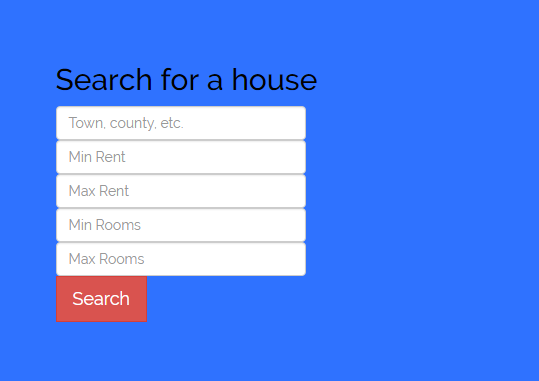


Figure . Search form to search for houses



Figure . Search results sorted by amount of rooms remaining

## Searching for a user

Users can also search for other users from any page as it is in the header. They enter the name of the user and then press the search button. The result is then displayed on a page where the user can then click on the searched users’ profile.



Figure . Search form to search for a user



Figure . Results of the user search

## Viewing a user’s profile

Viewing a user’s profile can allow the user to do many things: view the users personal details, view the picture of the house the user has uploaded and their house, message the user and add them as a roomie.

In terms of code this only required one method in my controller: showUser().The showUser method's main responsibility is to view a particular users profile. It also takes the username as a parameter and uses the houseService and userService service classes to retrieve the information about the houses and users. Message and roomie objects are instantiated and added to the model so a user can message the user or add them as a roomie. Profile pictures and house pictures are also checked to see if they exist using Booleans and added to the model.

User pages also look visually different depending on who is viewing them. If the logged in user is viewing it, then they obviously can’t add themselves as a roomie or message themselves whereas if it is a different user then they can do this.

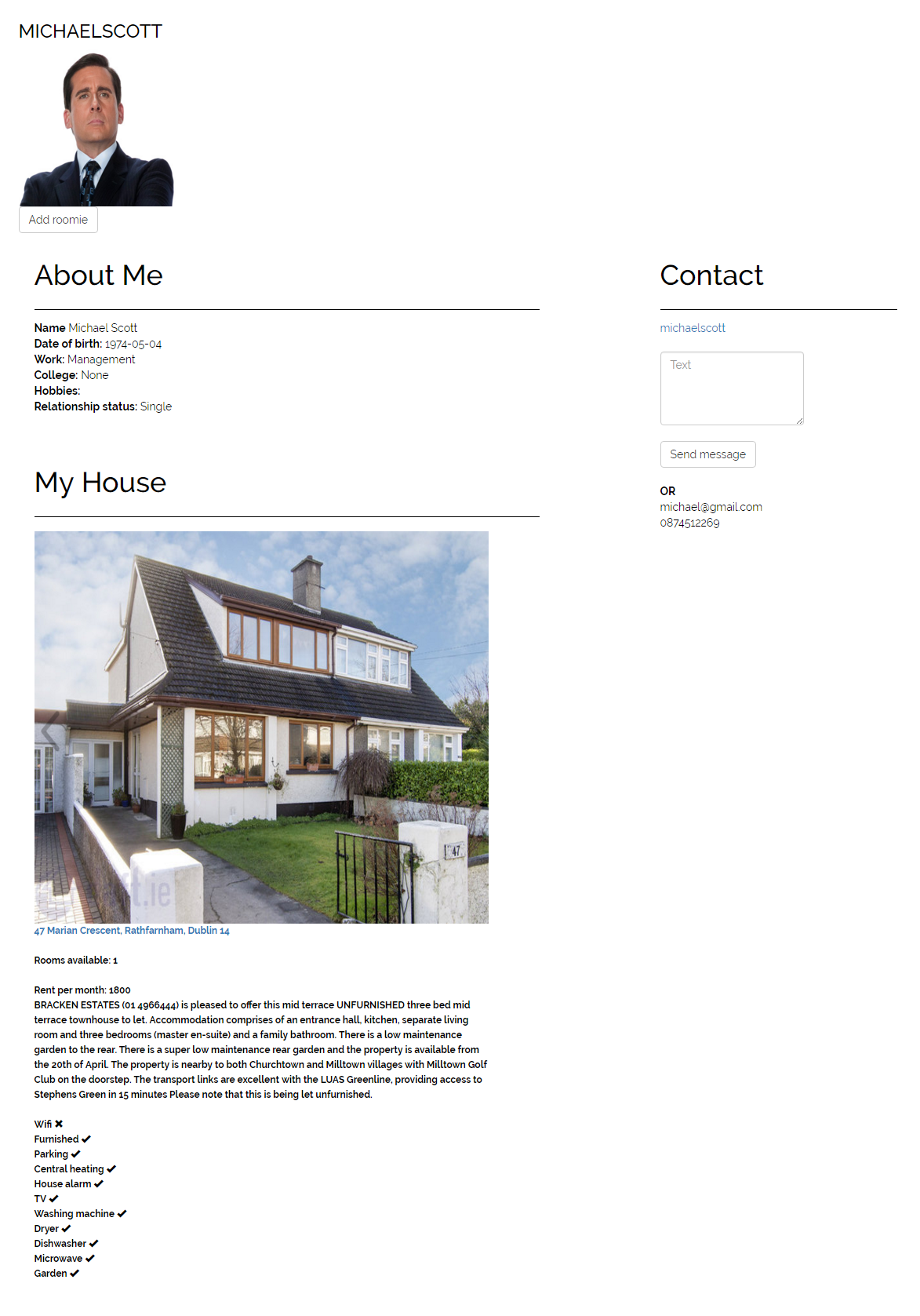


Figure . Viewing a user’s profile

## Messaging

All messages done in this application are done internally. I did not want to use email or Facebook for this as I felt it was important to keep it within the system. When a message is sent it is inserted into the database. A user can view all the conversations they have with other users.

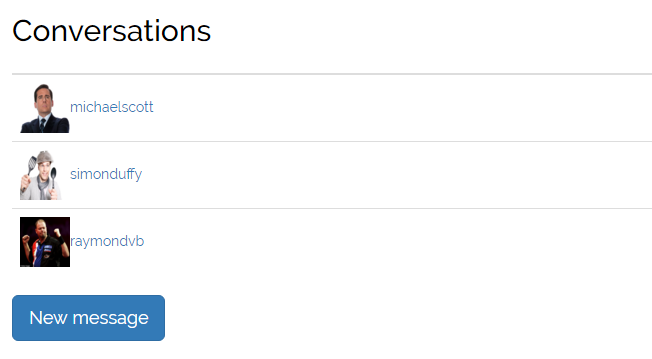


Figure . All conversations for a user

They can then click on one of these users where they are brought to the messaging page. This enables both users to have a live conversation with each other. jQuery is used to automatically refresh the section every 3 seconds.

Stylistically I tried to keep the layout as much like Facebook Messenger as I could as I knew that users would be familiar with this layout and it would be easy to use. Messages that you sent appear on the right of the page with a blue background whereas messages you received appear on the left with a grey background.

When the user sends a new message the message is sent and the page is reloaded.

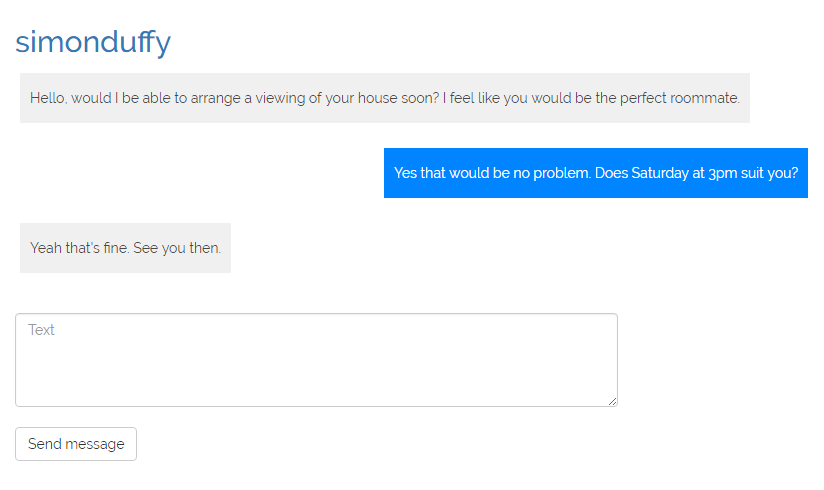


Figure . Having a conversation with a specific user

## Roomies

The roomies homepage enables users to view information about all roomies living in the house. They can choose to create tasks, create contacts or view all roomies where they can then remove roomies. The layout of this page depends on how much information has been created and how many roomies a user has.

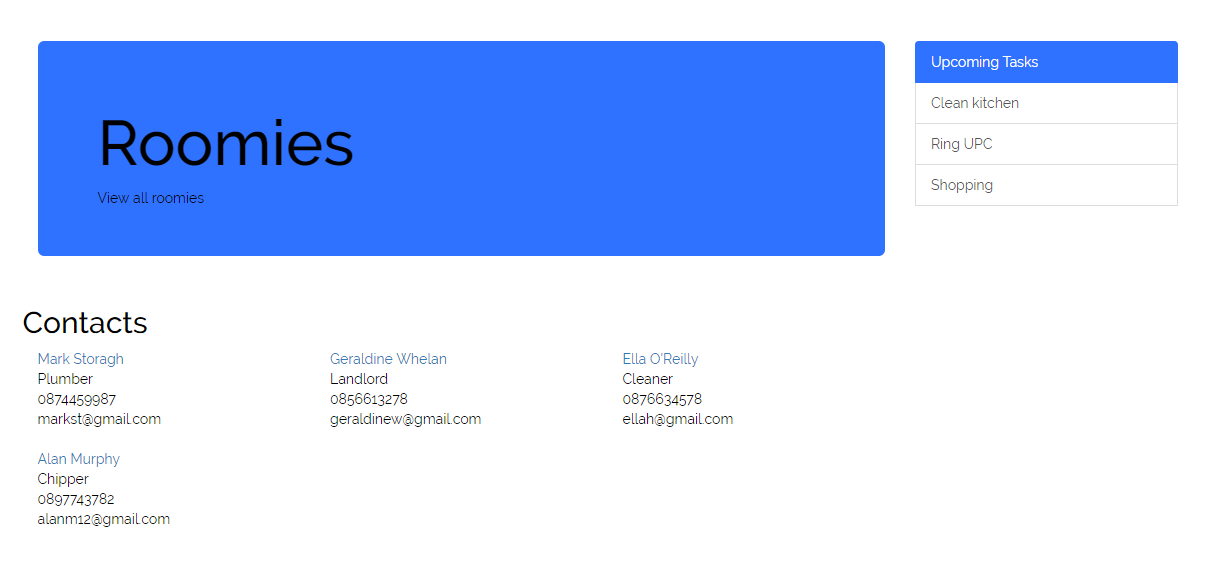


Figure . Roomies main page

All roomies can create contacts and tasks which will be helpful for other roomies. It helps manage the day-to-day tasks within the house.

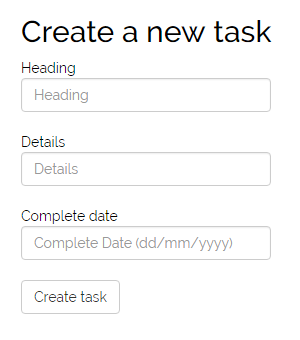


Figure . Creating a new task

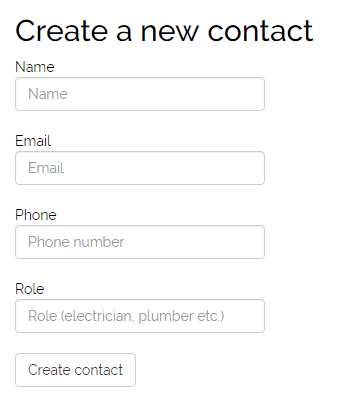


Figure . Creating a new contact

## Administrator

The administrator has extra features to a normal user. They are able to delete a house or user with just the click of a button as well as all the features a user has.

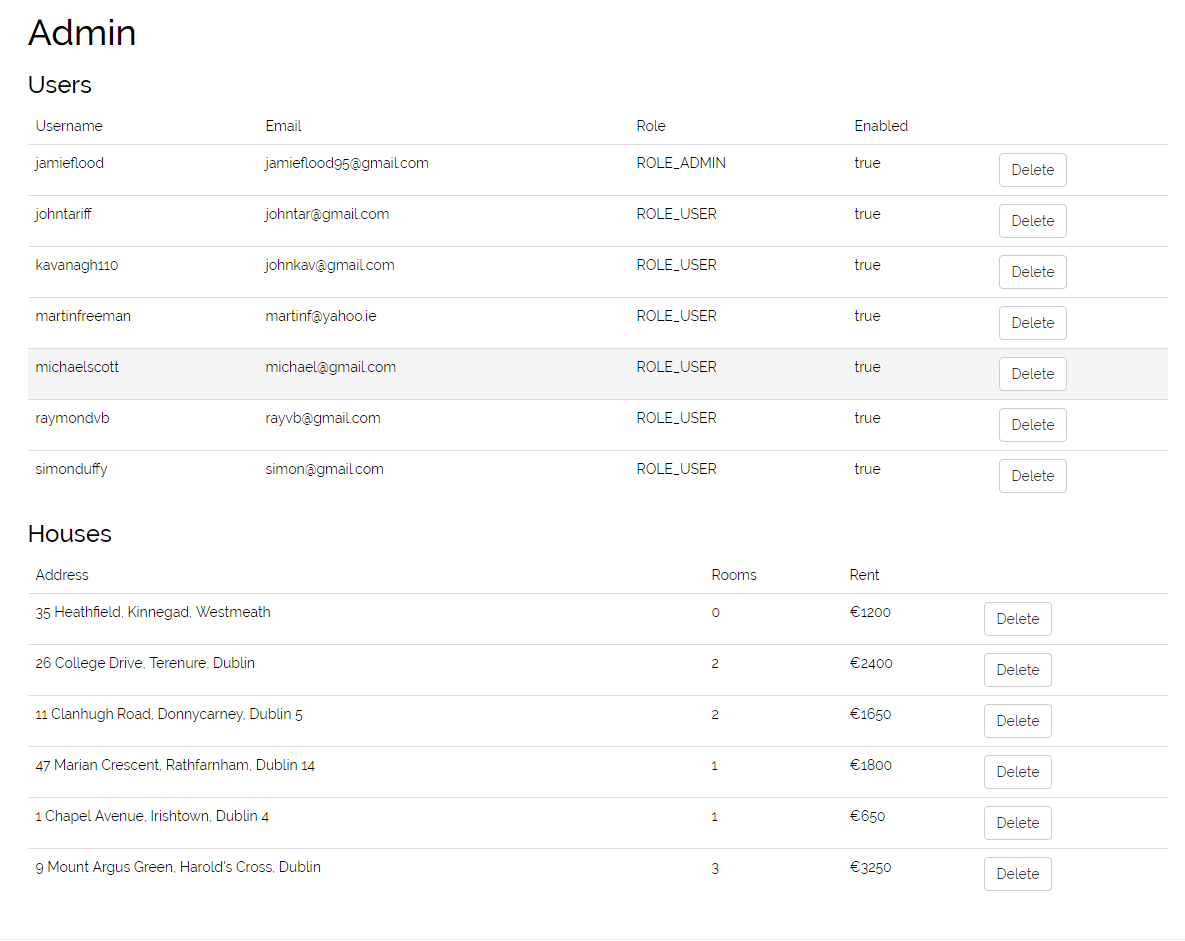


Figure . An administrator can delete a house or user

# Test plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Description | Expected result | Actual result | Test result |
| 1 | Create house | House saved in database, house retrieved from database | As expected | Pass |
| 2 | Delete house by id | House saved in database, house retrieved from database, delete house from database with house id, confirm deletion | As expected | Pass |
| 3 | Delete house by username | House saved in database, house retrieved from database, delete house from database with username, confirm deletion | As expected | Pass |
| 4 | Retrieve house by its id | House saved in database, house retrieved from database, id’s are the same | As expected | Pass |
| 5 | Retrieve a list of houses | Save multiple houses in database, retrieve houses in a list, confirm the size | As expected | Pass |
| 6 | Update a house | House saved in database, update the house address, retrieve house object to make sure addresses are the same | As expected | Pass |
| 7 | Search for houses | Return a list of houses based on the search parameters | As expected | Pass |
| 8 | House exists | Return a true value if a house exists in the database | As expected | Pass |
| 9 | Register an account | User and house saved in database, user retrieved from database | As expected | Pass |
| 10 | User exists | Return a true value if a user exists in the database | As expected | Pass |
| 11 | Retrieve a list of users | Save multiple houses and users in database, retrieve users in a list, confirm the size | As expected | Pass |
| 12 | Retrieve user by its username | User saved in database, user retrieved from database, usernames checked to see if they are the same | As expected | Pass |
| 13 | Delete user | User saved in database, user retrieved from database, delete user from database with username, confirm deletion | As expected | Pass |
| 14 | Search for a user | User saved in database, list retrieved of users with that name, check every user in the list has the same name | As expected | Pass |