# Programming Distributed Components

# COMP1690

Final Report

Joe Schofield

000776975

Due Date: 27/03/2017

Word Count:

# Contents

|  |  |  |
| --- | --- | --- |
| Section 1 | Introduction |  |
| Section 2 | Design Documentation |  |
| Section 3 | Screenshots of Features |  |
| Section 4 | Evaluation |  |
| Section 5f | Algorithms Explanation |  |

# Introduction

This report discusses the design, evolution, and evaluation of the intruder alert system, ‘Safe Home’. The system is composed of three applications, the web forms application, the SOAP API, and the website.

Section 1 contains all the design documentation of the system, including the database ERD and UML diagrams.

Section 2 contains screenshots of the finished product, demonstrating each feature that has been implemented.

Section 3 is a critical evaluation of the evolution of the applications. Including any issues during development, an evaluation of the finished product, and how the implemented system could be improved.

Section 4 is an explanation of how the system checks that layouts are physically feasible (i.e. that no two rooms can reside in the same location).

# Design Documentation

## ERD

Figure 1 shows the Entity Relationship Diagram (ERD) of the Safe Home database. It denotes the tables, column names, keys (i.e. Primary, Foreign), data types, and relationships.

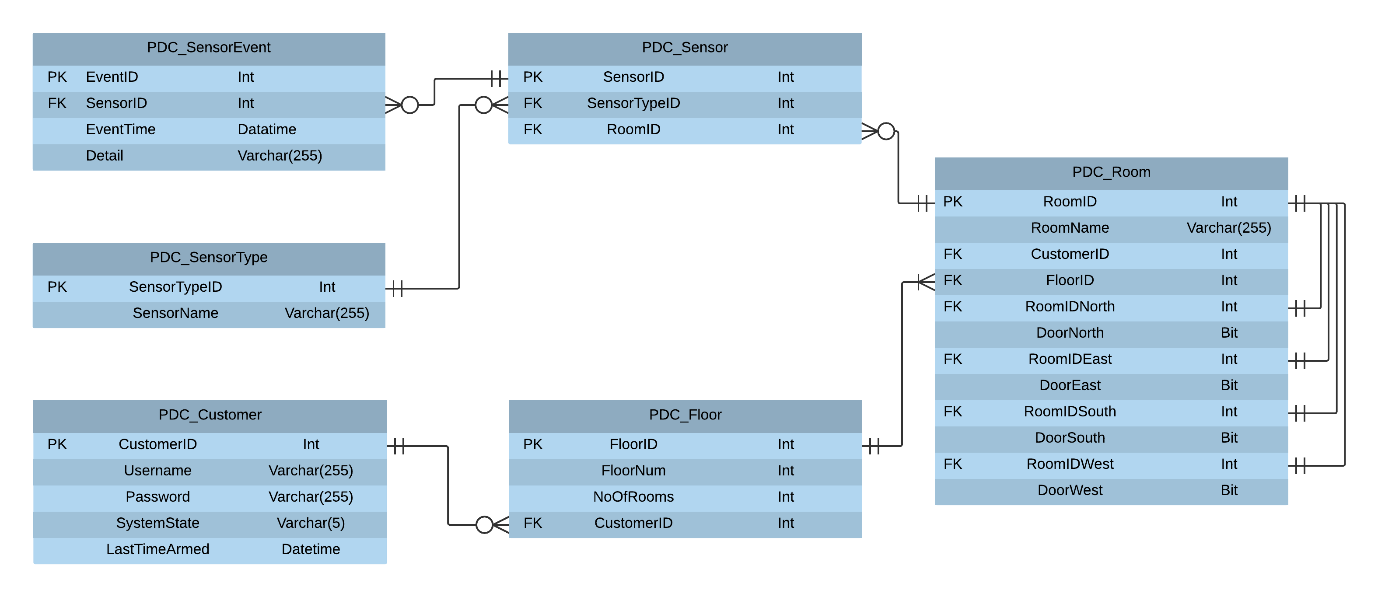


Figure 1 – ERD

## Class Diagram

### Windows Forms Application

Figure 2 displays the Class diagram for the Safe Home Windows forms application (including the Sensor emulator and Floor visualisation pages). The diagram shows the classes, relationships, variables, and methods.

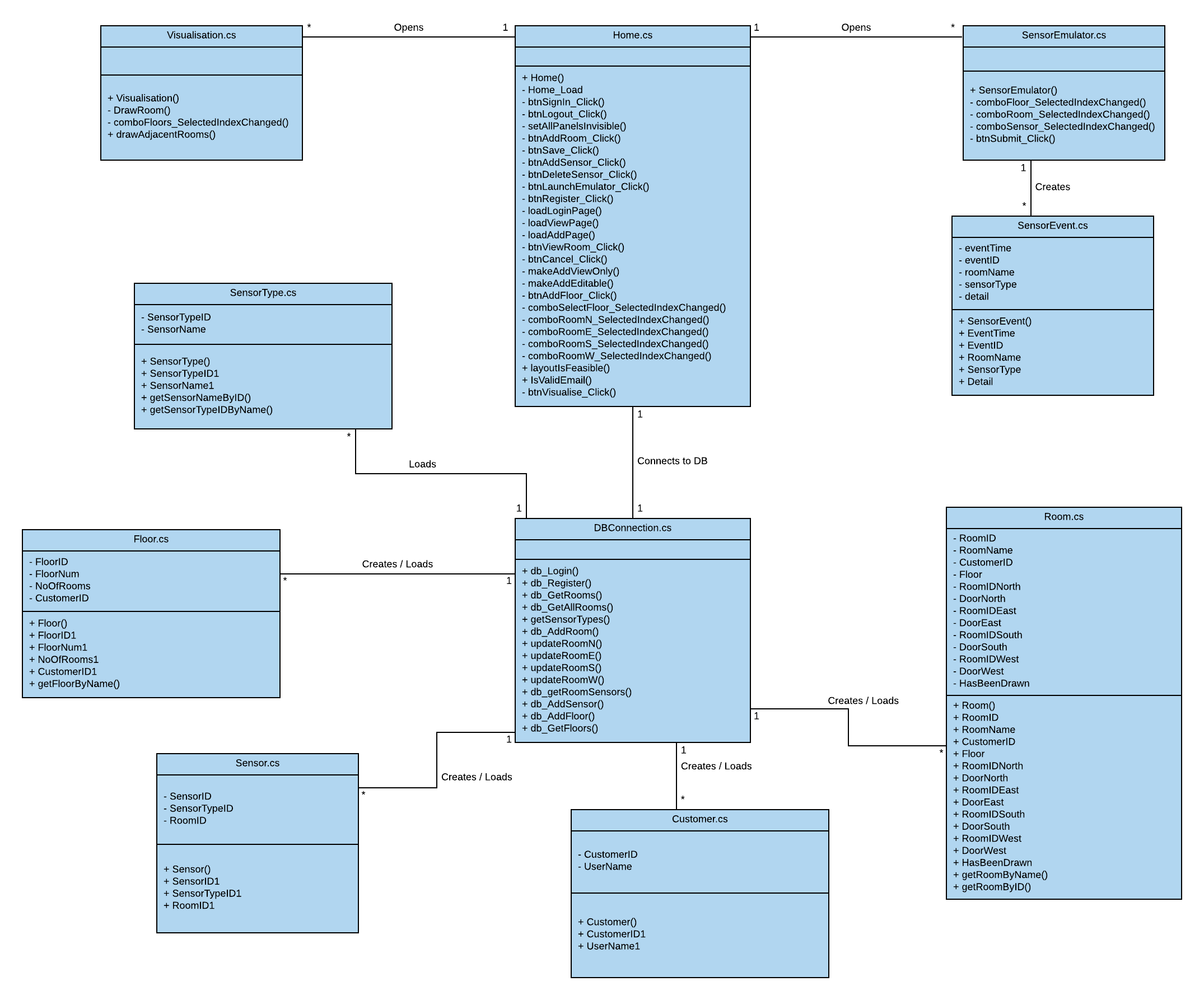


Figure 2 – Windows Forms Class Diagram

### SOAP API

Figure 3 displays the classes, relationships, attributes, and methods used within the SOAP API. There are no visual classes here as the API does not have a visual aspect.

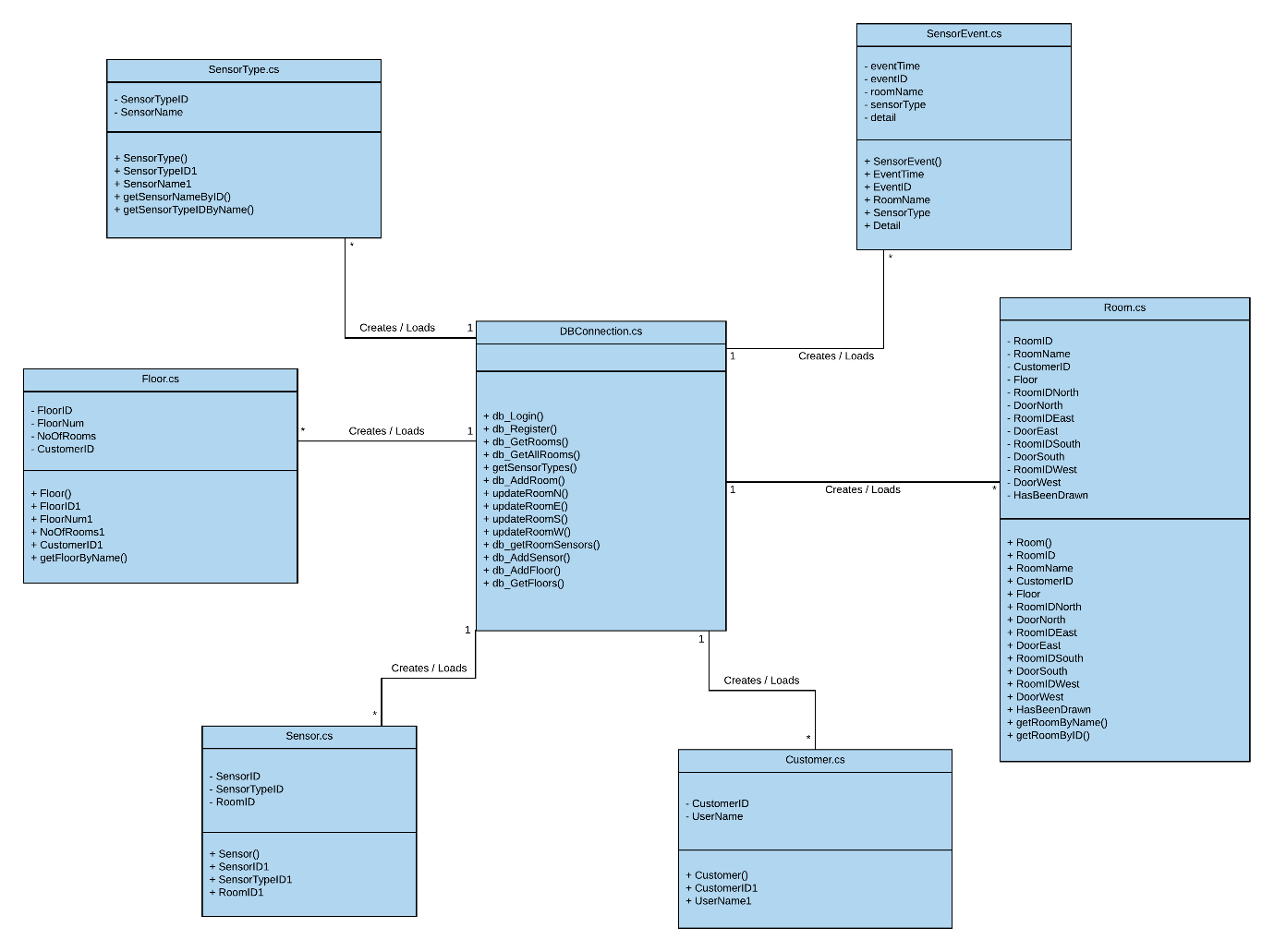


Figure 3 – SOAP API Class Diagram

### Web Forms Application

Figure 4 displays the only two classes used by the Web Forms application. There are no non-visual classes here as the application holds a web reference to the SOAP API in which instances of the classes can be created.

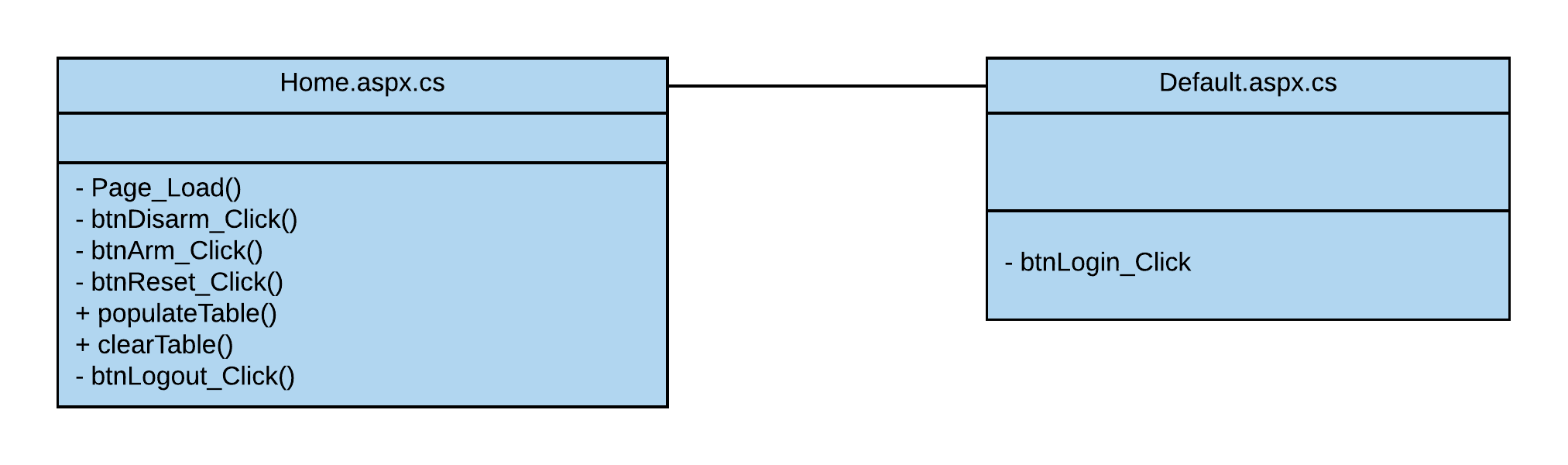


Figure 4 – Web Forms Class Diagram

## Use Case

Figure 5 demonstrates all actors that will use the system, what they will use the system for, and further steps involved in completing these use cases (where appropriate).

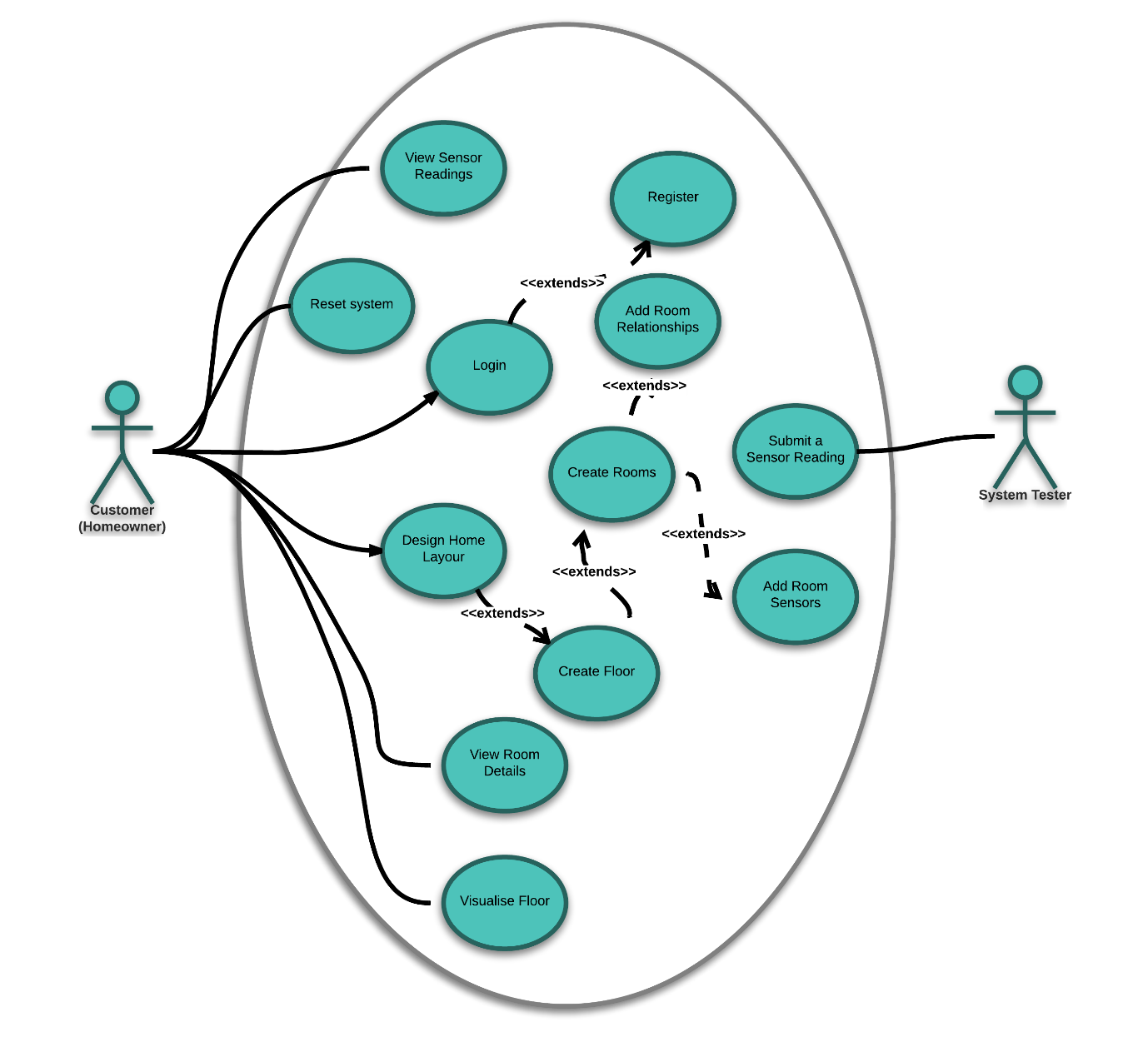


Figure 5 – Use Case

# Screenshots of Features

## Windows Forms Application

### Register / Login Page

Figure 6 shows the SafeHome register and login page. The user simply enters their email address and desired password to register (a message is displayed upon successful registration). Subsequently entering these details into the login form allows the user to sign into their newly created account.



Figure 6 - Register / Login Page

### Home Page

Figure 7 shows what the user is displayed upon first login. The ‘Rooms’ listbox is empty, and the ‘Floor No.’ cannot be selected as they have not been created yet. Likewise, the user cannot view a room, nor add one until a floor is created. The ‘Visualise’ and ‘Launch Emulator’ buttons do function but will have no data to work with. The only function the user can access at this time is the ‘Add a floor’ section. Once a floor has been added, the user can add rooms to that floor.

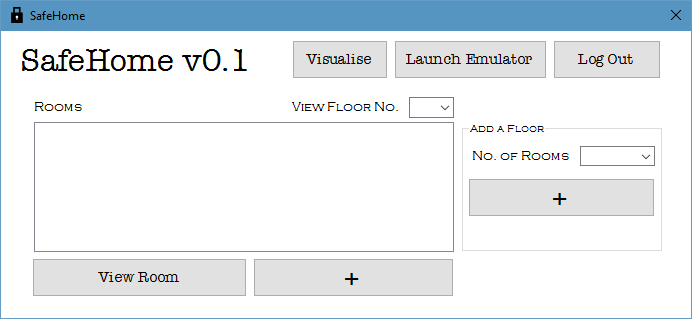


Figure 7 - Empty Home Page

Figure 8 is an example of what the user’s home page could look like after several rooms and floors have been created. The user can select a room from the listbox to view it’s details.

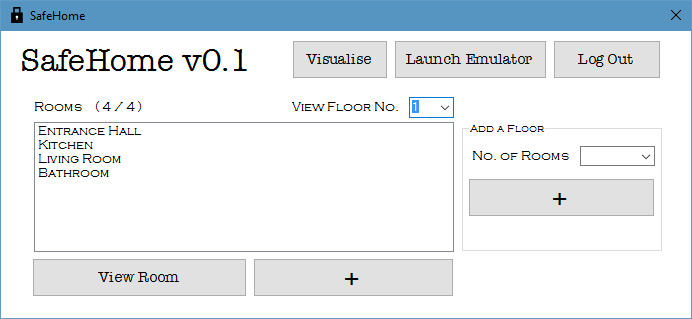


Figure 8 - Example Home Page

### Add Room Page

Figure 9 is an example of how the user can add a room. They can enter a room name, any adjacent rooms, doorways, and sensors. The system ensures that the layout is physically possible so that visualisations can be produced correctly.

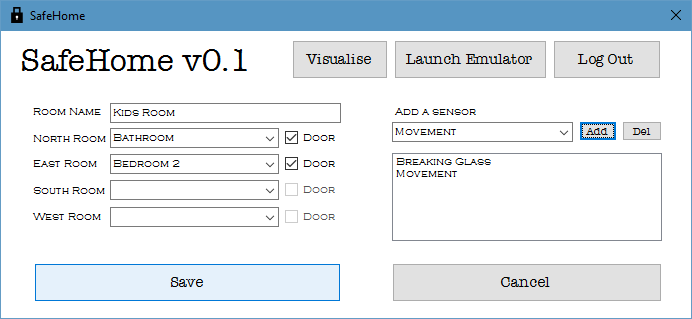


Figure 9 - Adding a Room

### View Room Page

Figure 10 shows the ‘View Room’ page. Once the user has created a room they may view the room’s details. They are shown the room’s name, adjacent rooms, and sensors.

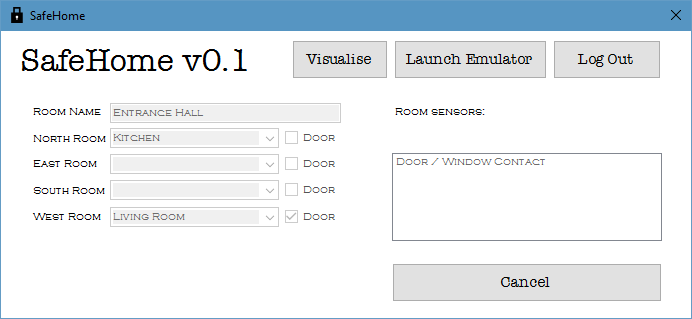


Figure 10 - View Room

### Visualisation

Figure 11 shows the visualisation form within the application. Once the layout has been created the user can select to visualise the floor, this is useful to ensure the house has been created properly. The visualisation tool displays the layout of a floor, the room names, any sensors which are within the room and if the room has a doorway to other rooms.

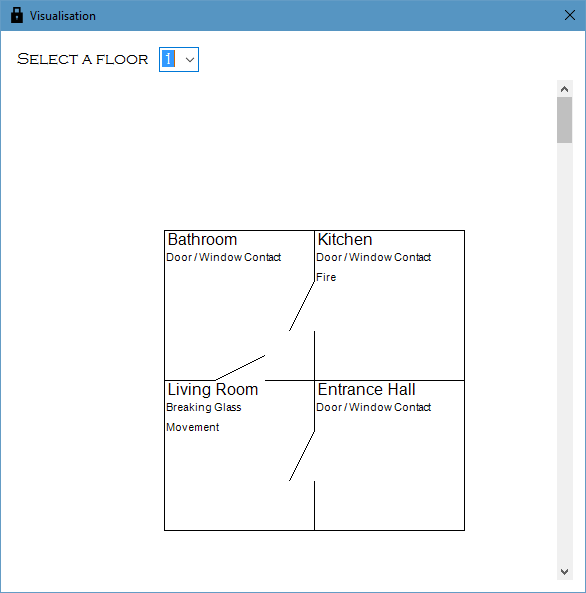


Figure 11 - Visualisation

### Sensor Emulator

Figure 12 displays an example of a user submitting a data reading. The user must first select a floor, then which room on the floor, then the sensor within the room, and finally add any details which are appropriate (such as the temperature for a fire sensor). This then queries the API and, upon a successful request, displays a success message. After a reading has been submitted the system will then be in ‘Alert’ mode, until the user resets the system. It is possible to have multiple instances of the sensor emulator running at any one time.

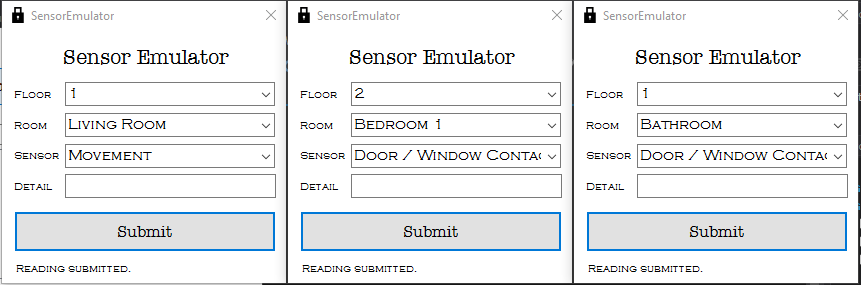


Figure 12 - Sensor Emulators

## Web Forms Application

### Login Page

The website login page has a very simple user interface, simply asking for the user’s email address and password. Once the user has logged in with the correct details they are automatically transferred to the home page.

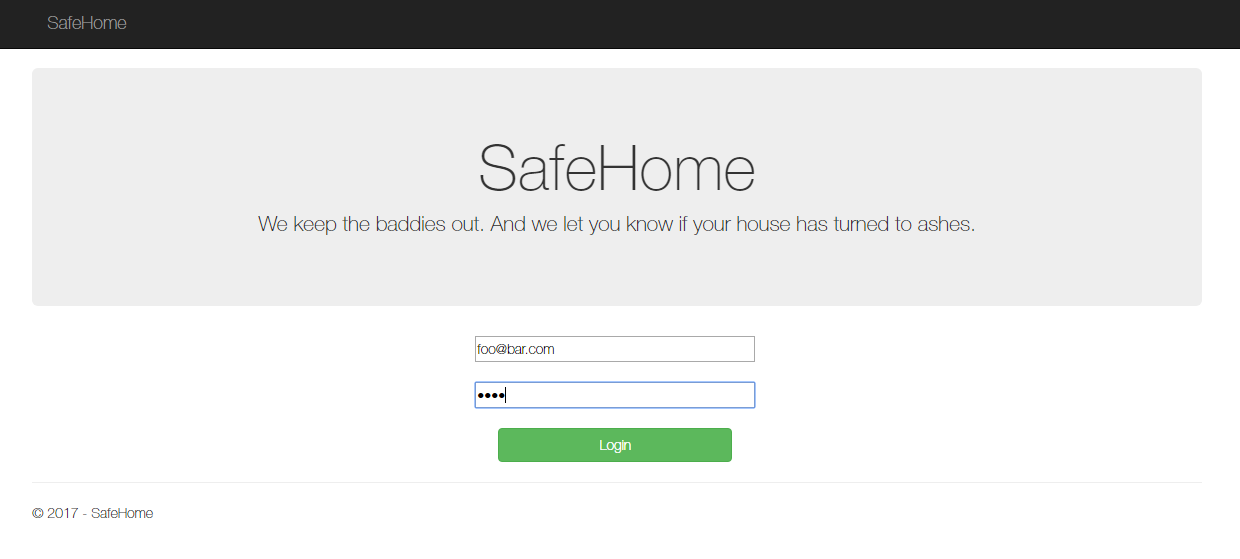


Figure 13 - Login Page

### Home Page

Figure 14 shows the home page of the website. The top of the page shows a large jumbotron div containing the SafeHome name and a welcome message for the user. Beneath this the system status is shown. This is currently ‘Alert’ due to the sensor readings previously submitted. There are 3 buttons below the status, ‘Disarm System’, ‘Arm System’, and ‘Reset System’, which query the SOAP API. Disarm system disables the ability to submit a sensor reading from the emulator, Arm system sets the system to armed mode, in which readings can be submitted, and finally reset does a combination of disarm and arm to clear out all events displayed below.

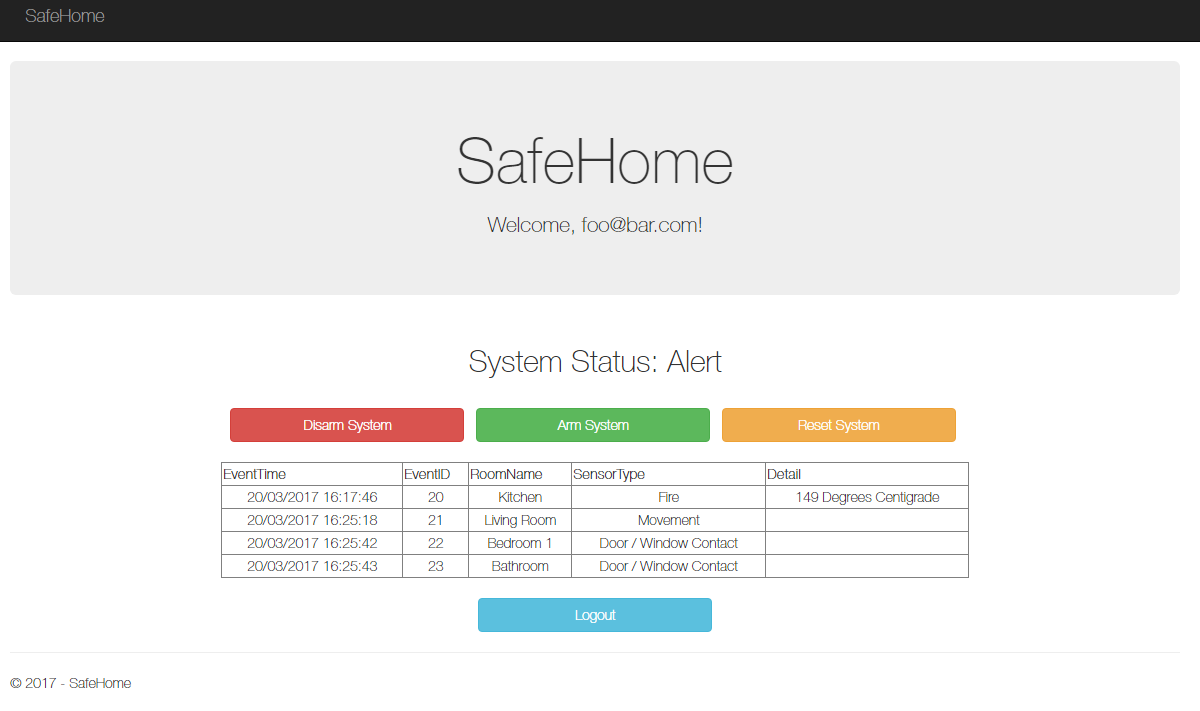


Figure 14 - Home Page on Alert

If the system has been reset, the user is no longer displayed any events (although these are still stored in the database for use as evidence if required), as demonstrated in Figure 15.

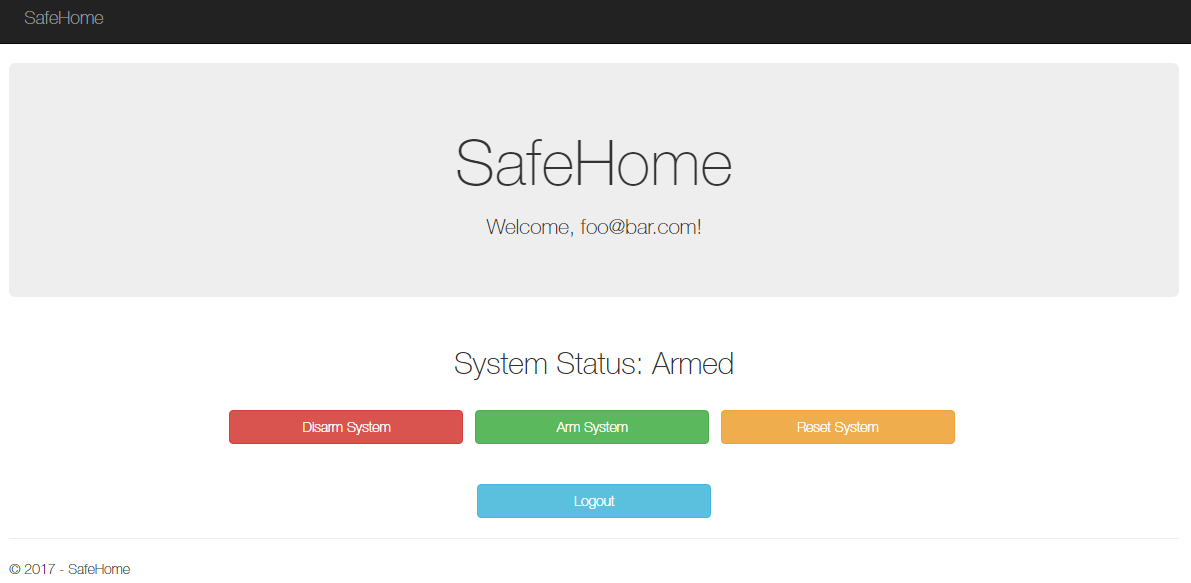


Figure 15 - Armed System

# Evaluation

An evaluation of the evolution of your application. You should discuss any problems you had during implementation. You should be critical (both positive and negative) of your implementation. Be prepared to suggest alternatives. Discuss how your final implementation could be improved.

# Algorithms Explanation

The SafeHome application contains a feature to ensure that floor layouts are physically possible. This ensures that rooms do not overlap when visualising the house. Figure 16 demonstrates an example error message the user might encounter. In this example there are three rooms; ‘Attic Room 1’, ‘Attic Room 2’, and ‘Attic Room 3’. The user has created ‘Attic Room 1’ and ‘Attic Room 2’, setting ‘Attic Room 1’ north of ‘Attic Room 2’. They are now attempting to create ‘Attic Room 3’ in the same location as ‘Attic Room 2’. When they click on ‘Save’ they are displayed the following error message:

“Layout is not feasible. The room ‘Attic room 2’ already exists in this location”

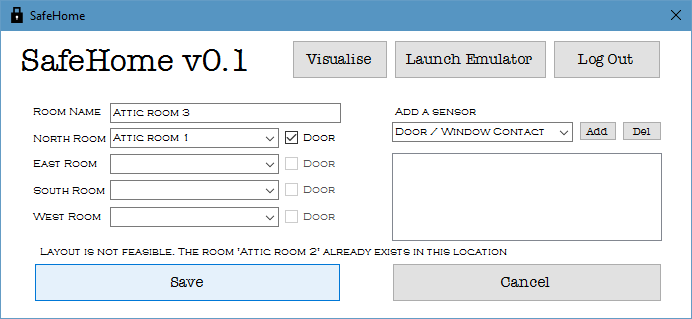


Figure 16 - Layout error

The algorithm works by checking each room currently on that floor (in this case ‘Attic Room 1’ and ‘Attic Room 2’) to see if any adjacent rooms selected (‘Attic Room 1’) are already adjacent in the same position (North) to another room. In the case of the previously stated example, the algorithm discovers that ‘Attic Room 2’ has already stated that ‘Attic Room 1’ is directly North of it, therefore ‘Attic Room 3’ cannot be in this location.

## Pseudo-code

Below is the pseudo-code created to design the algorithm to check that the layout of the floor is feasible.

checkLayoutIsPhysicallyPossible(newRoom, listOfCurrentRoomsOnFloor) {

if (newRoom.hasANorthRoom) {

if (listOfCurrentRoomsOnFloor.RoomHasSameNorthRoom()) {

return false

}

}

if (newRoom.hasAnEastRoom) {

if (listOfCurrentRoomsOnFloor.RoomHasSameEastRoom()) {

return false

}

}

if (newRoom.hasASouthRoom) {

if (listOfCurrentRoomsOnFloor.RoomHasSameSouthRoom()) {

return false

}

}

if (newRoom.hasAWestRoom) {

if (listOfCurrentRoomsOnFloor.RoomHasSameWestRoom()) {

return false

}

}

// If it reaches here then layout is possible

return true

}

A complete implementation of the advanced functionality requires you to devise algorithms for checking layouts for impossible room setups and providing an explanation of why it is impossible. Outline how you implemented these algorithms and wherever possible support your discussion with pseudo code, equations and/or diagrams. If you didn't implement the component or you think your implemented algorithm could be improved, then you can critically discuss how you would have implemented/improved the algorithm.