B&B Booking System

COMPGC22 Software Engineering Group Report

MSc Computer Science

21st of March, 2018

Team B

Aleksi Anttila Michael Aring Kai Klasen
Christina Kronser Zaid Al Lahham Lorenz Luboldt
Philip Spencer Paul Venhaus

Department of Computer Science

University College London

Executive Summary

This project report covers the processes involved in the design of the web-based software system *Room.io*, serving the Bed and Breakfast (B&B) industry in the UK. It is accompanied by a video showing the system design. The objective has been to design the system up to the point where it could be passed across to a development team for implementation.

The system described in this work allows B&B owners to register properties, to advertise them, and to accept bookings from users. Guests are able to search for properties along different dimensions and complete bookings in the software.

The report commences with the vision and scope of the project. The following chapters cover the design stages starting with the identification of the system requirements and then progressively drilling down into the use cases together with the system analysis class diagram.

Following the analysis stage, the report moves on to considering the object-oriented design phase where key interactions and activities are illustrated through sequence, activity, and state machine diagrams. A design class diagram is presented based on the analysis class diagram. It further includes methods and attributes in the various classes that provide guidance on implementation issues that the implementing developers will need to address. The report also includes component and deployment diagrams that show the intended interaction between hardware and software components.

The role and theory in relation to the various stages and related diagrams have been commented upon in the relevant sections.

The report ends with a conclusion and reflection on the project.

Contents

List of Figures

List of Tables

Introduction

CHAPTER 1

1.1 Background

This report sets out the processes undertaken in the design of a software system serving the Bed and Breakfast (B&B) industry in the UK which will enable prospective guests to search for and make bookings for available accommodation.

It describes the design of *Room.io*, a software system serving the B&B industry in the UK. It will be the first choice platform for owner-operators of B&B properties to receive bookings for their accommodation and for prospective guests to search for and make bookings on available dates.

In designing the system, the group has sought to put into practice the principles covered in the Software Engineering module at UCL, notably the Unified Software Development Process (USDP). The chapters below set out the stages in the design process and through commentary and illustrations/diagrams demonstrate the methodology involved in designing a system up to the point where it can be implemented by software programmers.

Name	Email
Aleksi Anttila	aleksi.anttila.17@ucl.ac.uk
Christina Kronser	christina.kronser.17@ucl.ac.uk
Kai Klasen	kai.klasen.17@ucl.ac.uk
Lorenz Luboldt	lorenz.luboldt.17@ucl.ac.uk
Michael Aring	michael.aring.17@ucl.ac.uk
Paul Venhaus	paul.venhaus.17@ucl.ac.uk
Philip Spencer	philip.spencer.17@ucl.ac.uk
Zaid Al Lahham	zaid.lahham.16@ucl.ac.uk

Table 1.1 – Team Members.

1.2 Problem Statement

B&B's are a popular type of accommodation for tourists around the world. Especially in the countryside, B&B's are highly sought after [1]. They provide a level of intimacy to their location unmatched by competing larger hotels.

Modern travellers require a well-designed platform that allows them to quickly find the properties and rooms that fit their personal criteria. The platform should further allow them to get insight into the experiences of fellow travellers who have frequented a B&B before and left a rating summarising their experience. Since B&B's are much smaller

in size than hotels — offering on average six rooms per location [1] — their operators can benefit immensely from a centralised platform that connects the properties with prospective guests and thus alleviates them from having to rely on advertisements or booking agencies and other middlemen.

1.3 PROJECT SCOPE

This report outlines the design of the modern web application *Room.io* for B&B operators and their prospective customers. It serves as a platform on which B&B operators can display their accommodation and on which potential guests can search for these accommodations according to their specific criteria and subsequently book and pay for a stay within their selected date range.

A potential guest using *Room.io* will be able to search for B&B's using a variety of criteria such as the B&B's location, specific dates or the rating that other users have assigned to the property. Using the information stored within the system, such as room availability based on scheduled bookings, the service is then able to present the user with a selection of the properties that best match their interests. Next, the users are able to select properties, read a longer description and inspect the property's rooms. Here they will also be able to learn more about the specific policies and amenities concerning each room, such as whether smoking is allowed or whether the room is wheelchair accessible. Users are also able to place their most liked B&B's on a favourites list which enables them to quickly access these properties. This way the properties can easily be recalled once it is time to make a booking decision. To facilitate the placement of bookings, *Room.io* stores the payment details of its users so that the payment process can be handled with great user-friendliness. It further sends out notification emails informing both host and guest about the newly created booking.

Hosts that sign up to the system can publish their property and provide a description as well as pictures to give guests a thorough impression of the accommodation. Next, they can add rooms. Here, *Room.io* allows hosts to define the aforementioned policies on a room-by-room basis. Through this, hosts can highlight the different features of each space. They can next set the prices of each room for both single and double occupation. Due to its easy searching and filtering capabilities, *Room.io* allows B&B property owners to be connected to the target audience that best matches their offering and significantly reduces search costs.

1.3.1 *Constraints*

This project was a student project, undertaken as part of the course COMPGC06 Software Engineering at the University College London. Due to the course's structure, the students learned about the different aspects of the USDP as well as the topic of Software Engineering

in general while working on the project. This meant that many iterations and revisions were necessary to improve previously created content based on newly acquired learnings. As per the project definition put forth by the course coordinators, the project did not involve implementation. Therefore, there has not been an opportunity for testing or further requirements iteration beyond the design stage. However, we have endeavoured to contemplate and address potential problem areas as far as possible.

1.4 Approach

From the beginning on the project team placed a high emphasis on taking an efficient and effective systematic approach to software engineering, both out of interest in the topic and because this best reflected the intended outcome of the course.

The project was undertaken from the 25th of January until the 21st of March 2018. The various stages in the design process are shown in the Gantt chart below (Figure 1.4). Whilst these stages have a natural sequential flow as the design process moves toward implementation, there was also a recurrent need to revisit and revise earlier stages. For this reason, the team decided to follow an Agile development process, as opposed to the traditional sequential Waterfall model of software development.

The team met up regularly with their supervisor to ensure that they were applying their learnings in the most productive fashion. Each part of the report was frequently iterated upon, as regular meetings and consultation would bring to light issues that merited modifications of the earlier treatment of relevant areas. This iterative approach was especially evident in the definition of requirements and use cases which went through numerous changes to ensure that they best represented the actual demands of the system.

Apart from meeting with their supervisor, the team also conducted frequent internal Scrum meetings, both face-to-face as well as digitally through services such as Google Hangouts and Slack. These meetings were usually attended by various subgroups of the larger team who were tasked with a certain sprint within the project. Here, each team member could report on their progress and receive feedback and clarifications to ensure a maximum cohesiveness of the overall report.

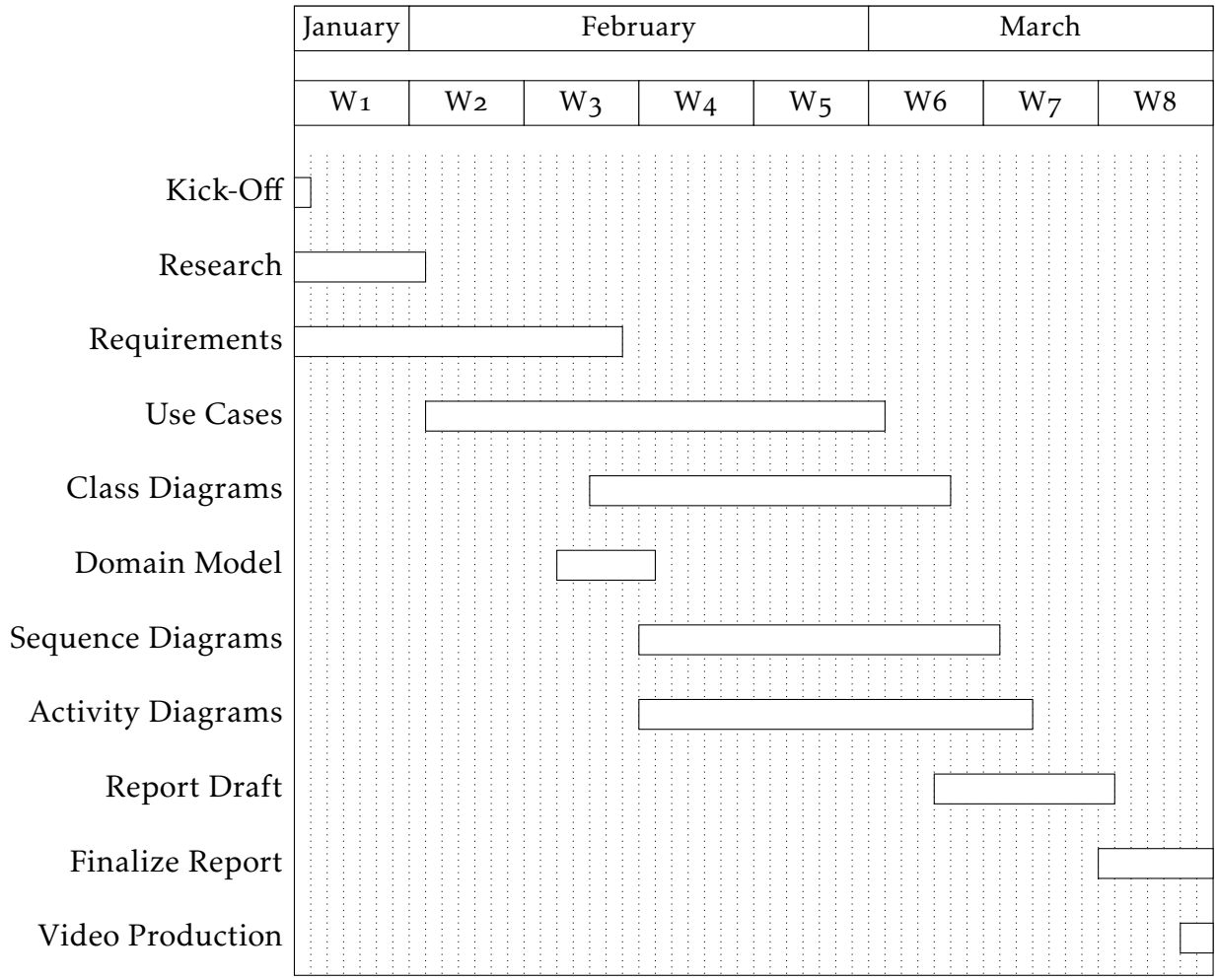

Figure 1.1 – Project Timeline.

1.5 Overview

1.5.1 *Requirements*

The report begins with the listing of both functional and non-functional requirements. Next, a Domain Model is presented which provides an overview of the how the system will operate. In the Domain Model, a high-level view of the system is taken to diagrammatically capture the relationships of the main entities with the separate areas that were identified within the system. This forms the basis for specifying the methods and objects that will be created in the implementation stage.

1.5.2 *Use Cases*

The use case specifications look at the behaviour of each key actor with regard to the system and provide a trail between the requirements and the way the system is being designed to function. From this analysis, it is then possible to further analyse the sequence of activities that will govern the final design of the system.

1.5.3 *Object-Oriented Analysis and Design Models*

The requirements and use cases set out in the previous chapters identify the main attributes of the proposed system to ensure it delivers the intended functionality to the key actors. The next phase is to set out the system so that the concept can be translated into a model which software engineers can program and implement. This is done firstly through an Analysis Class diagram and then progresses to Sequence Diagrams. The stage of the process is referred to as "Object-Orientated Analysis" as the preparation for implementation involves setting out the objects and their methods that will deliver the required functionality.

The Analysis Class Diagram highlights the main classes, objects and behaviours in the process and their general interaction. It can then be represented with greater detail in terms of classes, interaction, objects and methods. This is the Design Class Diagram, which is produced as a further guide to the system implementers. The Sequence Diagrams drill down into component actions to illustrate the interaction between these classes when the action occurs. That action may be, for example, the registration of a property. Next, the Activity Diagrams show the activity-oriented dynamic behaviour that can occur within the system. A Component Diagram captures the component structure by visualising how the different components of the system are connected through their provided interfaces. Lastly, the Deployment diagram outlines the topology of the system in terms of hardware, software and their connections and an exemplary state-machine diagram gives an indication of a discrete behaviour of a part of the designed system through state transitions [2].

1.5.4 *Application Mock-Up*

At the end of the report, an application mock-up and user manual is presented which envisions the front-end of the previously described system.

Requirements

chapter 2

2.1 Overview

The foundation for this project was laid during the requirements gathering and formulation phase. In order to deliver a good system and streamline the planning process, one first has to develop a strong understanding of the tasks a B&B system has to fulfil as well as the challenges that it should overcome. Further, as exemplified by Boehm [3] and Brooks [4], incorrect and incomplete requirements can lead to an exponential increase in cost during later stages of project development. For these reasons, a considerable amount of the early project phase was spent on gathering requirements, as can be seen in figure 1.4.

The requirements were gathered in an iterative process, starting with collective brainstorming during the team's weekly group meetings. Most of the team members had used online accommodation booking solutions before and could thus derive initial requirements from their own prior experiences. Next, the team studied current online accommodation booking services and took note of the functionality they offered. This research resulted in an exhaustive initial requirements list. Most of the essential requirements were so-called CRUD (Create, Read, Update, Delete) operations that each concerned a different entity in the System. These categories were therefore used to structure the requirements list seen below. Through subsequent iteration, especially during the use case definition phase, the initially long requirements list was condensed into a concise set of project requirements. Importance was assigned to the requirements according to the MoSCoW scale, ranking them based on their importance: whether they **M**ust, **S**hould, **C**ould or **W**on't be included.

The MoSCoW ranking facilitated clear communication within the project group and provided a good starting point for the later stages of this project since they indicate the approximate concern with which each issue should be approached.

The requirements lists are split up into multiple tables, sorted first by functional and non-functional requirements and then subdivided by the main entities they are concerned with.

2.2 Functional Requirements

The functional requirements listed below define the application architecture of the B&B system. They are concerned with the services that the system provides and the results that it delivers.

Users			
ID	Description	Priority	Use Case
R-U-C-1	The System shall allow for the registration of new Users.	Must	U-C-1
R-U-C-2	The System shall allow the creation of new Users.	Must	U-C-2
R-U-V-1	The System shall display a User's own profile.	Must	U-E
R-U-V-2	The System shall allow an Admin to view any User profile.	Must	U-E
R-U-L-1	The System shall display a list of User profiles.	Must	None
R-U-L-2	The System shall allow an Admin to filter a list of Users by certain criteria.	Must	None
R-U-E-1	The System shall allow a User to edit their own Account Details.	Must	U-E
R-U-E-2	The System shall allow an Admin to edit any User's Account Details.	Must	U-E
R-U-O-1	The System shall allow an Admin to ban a User from using the System.	Could	U-E

Table 2.1 – User Requirements.

Ratings			
ID	Description	Priority	Use Case
R-R-C-1	The System shall allow Guests to give Ratings to Properties they have visited.	Should	R-C
R-R-L-1	The System shall display a list of Ratings for a given Property.	Should	P-V
R-R-D-1	The System shall allow the deletion of Ratings for a given Property.	Could	None

Table 2.2 – Rating Requirements.

Properties			
ID	Description	Priority	Use Case
R-P-C-1	The System shall allow a Host to register a new Property.	Must	P-C
R-P-V-1	The System shall display information about a Property.	Must	P-V
R-P-L-1	The System shall display a list of Properties.	Must	P-L
R-P-L-2	The System shall allow a User to filter a list of Properties by certain criteria.	Must	P-L
R-P-E-1	The System shall allow a Host to edit their own Property.	Must	P-E
R-P-D-1	The System shall allow a Host to delete their own Property from the System.	Must	P-D
R-P-D-2	The System shall allow an Admin to delete Properties from the System.	Must	P-D
R-P-O-1	The System shall allow a User to share Property Details with their contacts.	Could	None
R-Ro-C-1	The System shall allow the addition of Rooms to a Property.	Must	P-E
R-Ro-E-1	The System shall allow a Host to edit their Property's Rooms.	Must	P-E
R-Ro-D-1	The System shall allow a Host to delete their Property's Rooms.	Must	P-E
R-Ro-O-1	The System shall display all available Rooms in a Property for a given date.	Must	P-V

Table 2.3 – Property Requirements.

Favourites			
ID	Description	Priority	Use Case
R-F-C-1	The System shall allow a Guest to add Properties to a list of Favourites.	Could	F-C
R-F-L-1	The System shall allow a Guest to view a list of their favourite Properties.	Could	F-L
R-F-D-1	The System shall allow a Guest to remove a Property from their list of favourite Properties.	Could	F-D

Table 2.4 – Favourite Requirements.

Bookings			
ID	Description	Priority	Use Case
R-B-C-1	The System shall allow the creation of Bookings for a Property.	Must	B-C
R-B-V-1	The System shall allow a Guest to view their own Booking.	Must	B-L-1
R-B-V-2	The System shall allow a Host to view a Booking on their Property.	Must	B-L-2
R-B-V-3	The System shall allow an Admin to view any Booking.	Must	B-L-1 & B-L-2
R-B-L-1	The System shall display a Guest or Property's Booking history.	Must	B-L-1 & B-L-2
R-B-O-1	The System shall allow the cancellation of a Booking.	Must	B-D
R-B-O-2	The System shall send email confirmations to confirm a new Booking.	Could	B-C

Table 2.5 – Booking Requirements.

Policies			
ID	Description	Priority	Use Case
R-Po-C-1	The System shall allow the creation of Policies.	Should	Po-C
R-Po-L-1	The System shall display a list of existing Policies.	Should	None
R-Po-D-1	The System shall allow the deletion of Policies.	Should	Po-D

Table 2.6 – Policy Requirements.

Out of these functional requirements, all but two were included in the first version outlined in this report. The first of these two requirements is R-R-D-1: The System shall allow the deletion of Ratings for a given Property. It was not included as in the system's current iteration a User can override their prior rating to a property by rating it again. Use Case R-C (Table 3.12) therefore implicitly contains the functionality of R-R-D-1. The second excluded requirement is R-P-O-1: The System shall allow a User to share Property Details with their contacts. While a desirable feature to have on the platform in future iterations, this feature was deemed not essential enough to be included in this first version of the B&B system.

2.3 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements listed below concern the technical architecture of the B&B system. They are concerned with *how* the system performs rather than *what* it delivers.

ID	Description	Priority	Category
R-N-C-1	The System shall be written in a commonly used programming language.	Should	Code
R-N-C-2	The System shall use a relational database system.	Should	Code
R-N-R-1	The System shall be able to backup information.	Could	Reliability
R-N-R-2	The System shall not lose any data.	Must	Reliability
R-N-R-3	The System shall have a high uptime.	Must	Reliability
R-N-Su-1	The System shall be well documented.	Should	Supportability
R-N-Su-2	The System shall provide helpful error messages.	Should	Supportability
R-N-Se-1	The System shall store customer information securely.	Must	Security
R-N-Se-2	The System shall store account details securely.	Must	Security

TABLE 2.7 – Non-functional Requirements.

2.4 DOMAIN MODEL

Based on these requirements, a domain model was created. It describes the relationships between the different entities in the system. This model was instrumental in eliminating possible ambiguity about the different relationships concerning the logical ownership of certain entities. Throughout the subsequent use case definition phase the domain model was used to clarify these relationships while naturally evolving as new relationships became salient and others were deemed out of focus.

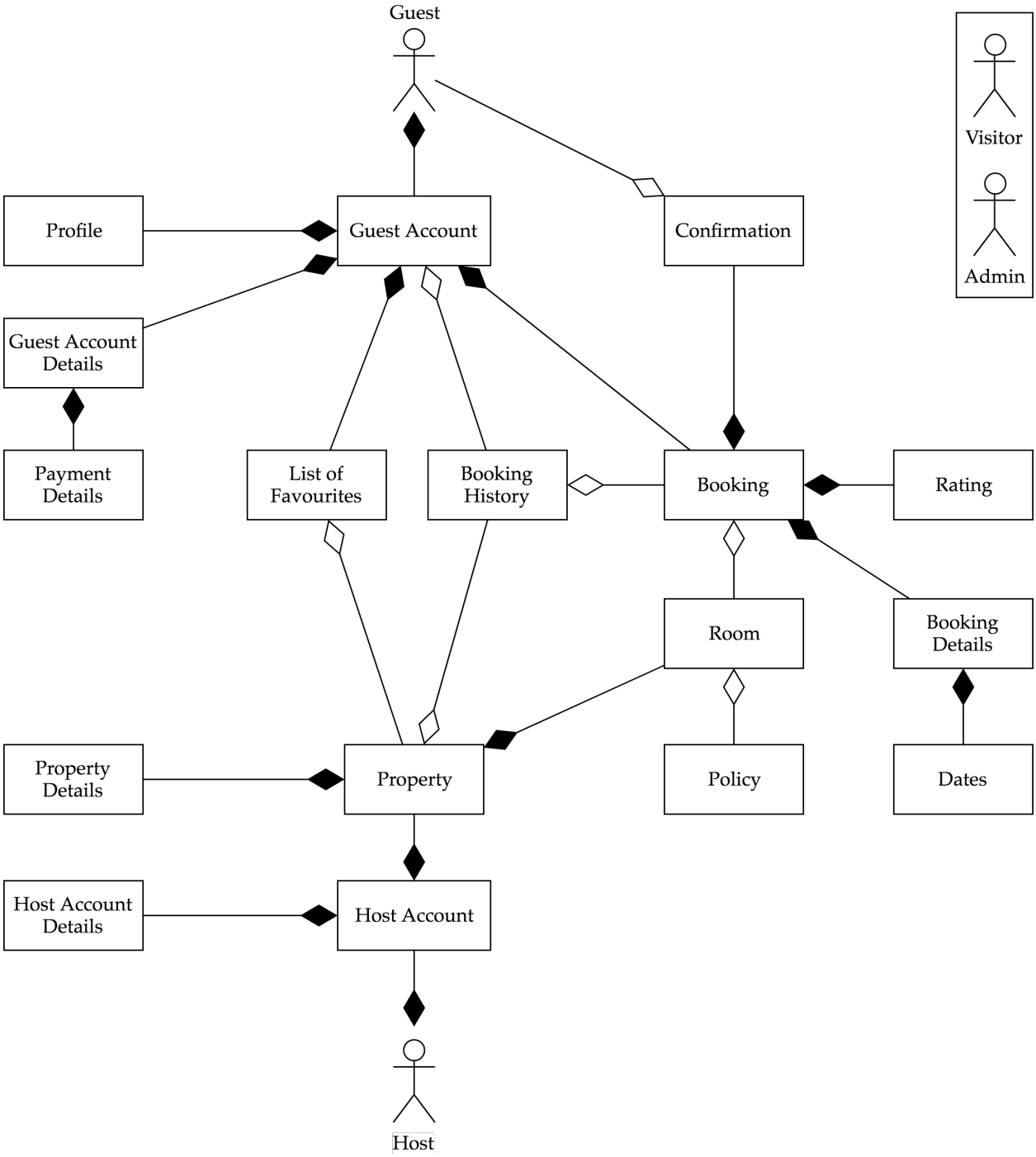

Figure 2.1 – Domain Model.

3.1 Overview

Surveying our functional requirements, we determined that all use cases for the System would involve the manipulation of our main entities in typical CRUD-like fashion: all the required functionality could be achieved by Creating, Viewing/Reading/Listing, Updating/Editing and Deleting these entities. Each functional requirement that did not by itself constitute any such operation (for example, filtering and ordering Properties by their Details) could be construed as forming part of some use case that did (the aforementioned filtering/ordering being subsumed under the Listing of Properties).[1] Table 3.1 shows our final list of use cases, with the cells colour-coded by Primary Actor (see Figure 3.1 for the key to these colours). Each row of the table corresponds to one of our main entities, and the columns represent the different kinds of operations users can perform on these entities.

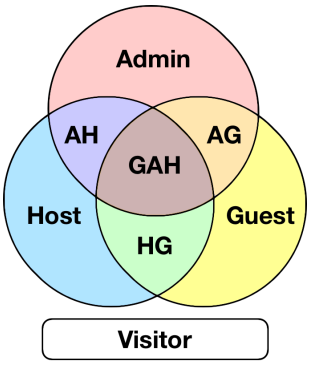

Figure 3.1 – Use Cases Actor Colour Guide.

Target Entity	ID	Use Case (Create)	ID	Use Case (View/List)	ID	Use Case (Edit)	ID	Use Case (Destroy)
User	U-C-1	Register User			U-E	Edit User		
	U-C-2	Create User						
Property	P-C	Register Property	P-V	View Property Details	P-E	Edit Property	P-D	Delete Property
			P-L	View Property List				
Booking	B-C	Make Booking	B-L-1	View User Bookings			B-D	Cancel Booking
			B-L-2	View Property Bookings				
Policy	Po-C	Create Policy					Po-D	Delete Policy
Favourites	F-C	Add to Favourites	F-L	View Favourites			F-D	Remove from Favourites
Rating	R-C	Give Rating						

Table 3.1 – Use Cases.

Figure 3.2 depicts the same information in diagram form; the secondary actors involved in the use cases — Visitor's email service (for registration confirmation) and Payment System (for handling Booking Payments) — are also included.

[1] Refer back to Chapter 2 for more detailed information on which requirements are covered by which use case.

Figure 3.2 – Use Case Diagram.

As can be seen, our set of use cases does not feature any inclusions or extensions. Strictly speaking, some of the cases do flow into each other in a way that could be thought of as extension (Make Booking, for instance, is always preceded by View Property Details). We decided not to use extensions and to include this information in the preconditions for the use cases instead, thus reducing the complexity of this representation of our system.

It can also be seen that some CRUD-like operations on the main entities were not included as full use cases (viewing/listing Users being one example). The System does, in fact, support many of these operations, but we decided to limit our list of use cases to operations that could reasonably be thought of as complete usage scenarios for a BNB Booking System. So, for instance, an Admin does have the ability to view a list of all the users on the System, but would rarely do so without, say, editing some User's details or viewing some User's Bookings afterwards.

We turn now to textual descriptions of the use cases. These have been written in a fair amount of detail; to condense the presentation here, we run through only the most crucial

use cases. The rest are included in Appendix C [2]. The descriptions are grouped according to the main entity being acted upon; each remaining section of this Chapter covers one of these entities. The style of these descriptions is, for the most part, based on the discussion and examples in [5], [6] and [7]. Some more things to note:

- We have included special notation for indicating the steps at which a use case ends/can end:
  - Whenever reaching a specific step means that the use case is definitely finished, the step description is preceded by "(END)".
  - If a use case could end with some specific step, but the primary actor may still optionally perform some actions after the step in question, the step description is preceded by "(POSSIBLE END)".
- Branching is generally indicated by listing the different progression options after different letters (a, b, c...) (This is as opposed to steps that compose sequences of actions; such steps are preceded by numerals — Arabic in the case of Main Flow steps; Roman in the case of substeps to Main Flow steps.)
- The flow steps generally follow the typical dialogue pattern ("The User... The System...") However, when a single actor performs multiple consecutive actions that are different enough in nature, these have been assigned their own steps to increase clarity and readability.
- Many of the finer details of what information is being supplied and checked at each step have been relegated to Appendix A (the Glossary). As noted before the terms that are included in the glossary have been capitalized throughout the report.
- Whenever the Primary Actor supplies and submits some set of details to the System, and the System has the Actor resupply said details for any reason, we implicitly assume that the System saves the submitted details and displays them to the Actor when the Actor has to resupply them (so an incorrectly filled form, say, will be returned to the User with the previously submitted information already filled in).
- We have omitted descriptions of most confirmation messages.
- Some error flows have been included. However, more serious errors that should not occur during the normal operation of the System (database insertion failure, for instance) have not been individually described here for the sake of brevity. Whenever the System detects such an error, the use case being carried out stops, and the System displays an error page.

3.2 USERS

For Users, we have included two creation use cases and one editing use case. The first creation case, U-C-1 (Table 3.2), is special in that it is the only use case with Visitor as the Main Actor. We decided that for simplicity a Visitor must first register as a Guest or

[2] The Appendix also includes some descriptions of how the system operates that did not qualify as full use cases as they are thought of in this report (see the discussion on this above). One example of such an operation is logging in to the System (Table C.1).

a Host before gaining access to any of the real functionality of the System. The second creation case, U-C-2 (Table C.3 in Appendix C), is a special Admin tool, mainly meant to be used for creating more Admins. U-E (Table C.4 in Appendix C), for editing one's User Details, has a very similar flow to U-C-2.

ID	U-C-1
Use case	Register User
Description	A Visitor registers as a Guest or a Host (henceforth, the Guest/Host will be referred to as "User").
Primary Actor	Visitor
Secondary Actor(s)	Visitor's email service
Preconditions	None
Main Flow	1. The use case starts when the Visitor issues a request to register as a new User. 2. The System provides the Visitor with a means to fill in their Core User Details and Role. 3. a. If the Visitor provides their Core User Details and Role: go to 4. b. If the Visitor cancels the operation: go to 6. 4. The Visitor submits the information provided. 5. The System checks the Core User Details. a. If the Core User Details are Correct: i. The System creates a new Unverified User based on the details provided. ii. The System sends a verification email to the email address in the Core User Details. Go to 6. b. If the Core User Details are not Correct: the System displays a notification about the details causing the problem. Return to 2. 6. (POSSIBLE END) The System displays the Landing Page. 7. If 3a. taken: a. If the Visitor confirms their registration via their email service: i. The System changes the Unverified User into a Verified User, unlocking their account. ii. (END) The System confirms the verification to the User and displays the Landing Page. b. (END) If the Visitor never verifies their account, it remains Unverified and hence unusable.
Postconditions	The System displays the Landing Page and: ■ If 3a. and 7a taken: a new Verified User has been created; the User details correspond to those specified during the registration process. The User may begin using Guest/Host services. ■ If 3a. and 7b. taken: a new Unverified User has been created; the User details correspond to those specified during the registration process. The Visitor may not access Guest/Host services until the User is Verified.

Table 3.2 – Use Case U-C-1: Register User.

Other User-related operations: the System does allow listing Users (Admin only; Requirement R-U-L-1) and viewing an individual User's User Details (Admin/Guest/Host; Requirements R-U-V-1 and R-U-V-2). These operations were not considered significant enough to warrant their own use cases. Listing is included, in effect, in all of the Admin's User-oriented use cases (in the form of the Admin's Dashboard). Viewing User Details is

included in U-E. We decided that for auditing purposes, deleting Users from the system entirely would be undesirable; the Admin can instead ban misbehaving Users by editing their Details (thereby rendering them unable to do anything on the System). Logging in and logging out are described in Appendix C.

3.3 PROPERTIES

For Properties, we have a full set of CRUD-like use cases. We decided that viewing an individual Property's Details (P-V; Table 3.5) and the listing of Properties (including searching, filtering and sorting) (P-L; Table 3.4) were both significant enough to have their own Use Cases (a User may simply wish to check what Properties there are in a particular area for future reference, for instance). The use cases for editing and deleting a Property are in Appendix C. (Note that deleting Properties is intended to be a "soft"/"logical" delete: records and details are to be kept on the System.)

It may be worth pointing out that in P-C (Table 3.6) the Core Property Details the Host has to specify include the Rooms the Property has, and all Policies associated with those Rooms. The use case does not go into the specifics of how the Host provides these entity details, but our activity diagram covering a new Host's entire registration process (Figure 4.10 in Chapter 4) includes some more information (see also Figure 4.3 in Chapter 4 for the related sequence diagram.)

ID	P-C
Use case	Register Property
Description	A Host registers a new Property.
Primary Actor	Host
Secondary Actor(s)	None
Preconditions	The Host does not already have a Property registered on their account.
Main Flow	1. The use case starts when the Host issues a request to register a Property. 2. The System provides the Host with a means to fill in the Core Property Details for the new Property. 3. a. If the Host provides the Core Property Details: go to 4. b. (END) If the Host cancels the operation: the System displays the Host Dashboard. 4. The Host submits the Core Property Details. 5. The System checks the Core Property Details. a. If the Core Property Details are Correct: the System creates a new Property based on the details provided. b. If the Core Property Details are not Correct: the System displays a notification about the details causing the problem. Return to 2. 6. (END) The System displays the Property Details for the new Property.
Postconditions	■ If 3a. and 5a. taken: a Property has been created. The Core Property Details correspond to those specified during the creation process. The System displays the Property Details for the new Property. ■ If 3b. taken: the System displays the Host Dashboard.

TABLE 3.3 – Use Case P-C: Register Property.

ID	P-L
Use case	View Property List
Description	A Guest or an Admin (henceforth, "the User") searches for, filters, orders, and views a List of Properties.
Primary Actor	Guest/Admin
Secondary Actor(s)	None
Preconditions	None
Main Flow	1. The use case starts when the User specifies some initial Property Search Conditions and issues a request to view a Property List. 2. (POSSIBLE END) The System fetches a List of Properties conforming to the Property Search Conditions set and ordered according to any Property Sort Criteria set. It also provides the User with a means to apply a new set of Property Search Conditions and to order the List according to some Property Sort Criteria. Additionally: a. If the List is non-empty: the System displays the List and provides the User with a means to access the Property Details for each Property listed. b. If the List is empty: the System displays a notification informing the User that the search has produced no results. 3. If the User applies a new set of Property Search Conditions and Property Sort Criteria and requests to view a new Property List: return to 2.
Postconditions	The System displays a Property List conforming to the set Property Search Conditions, and ordered according to the set Property Sort Criteria.

Table 3.4 – Use Case P-L: View Property List.

ID	P-V
Use case	View Property Details
Description	A Guest, Admin, or a Host (henceforth, "the User") views a Property's Property Details.
Primary Actor	Guest/Admin/Host
Secondary Actor(s)	None
Preconditions	■ Admin/Guest: The Admin/Guest is viewing a Property List. ■ Host: None.
Main Flow	1. The use case starts when the User issues a request to view a Property's Property Details. 2. (END) The System displays these Property Details. Additionally: a. For Guest: the System provides a means to select the Initial Booking Details for a new Booking; these are pre-selected if the Guest already provided these details in some Property Search Conditions in the Precondition. b. For Host: the System displays the Host-Specific Property Details, and provides a means to edit the Core Property Details.
Postconditions	The System displays the selected Property's Property Details and: ■ For Guest: the System provides a means to select the Initial Booking Details for a new Booking; these are pre-selected if the Guest already provided these details in some Property Search Conditions in the Precondition. ■ For Host: the System displays the Host-Specific Property Details, and provides a means to edit the Core Property Details.

Table 3.5 – Use Case P-V: View Property Details.

3.4 BOOKINGS

Make Booking (B-C; Table 3.6) is probably the most crucial use case in the entire System. It in effect extends P-V (which, in turn, extends P-L for the Guest and Admin); the relevant postconditions from P-V are included in the preconditions for convenience.

ID	B-C
Use case	Make Booking
Description	The Guest makes a new Booking.
Primary Actor	Guest
Secondary Actor(s)	Payment System
Preconditions	The Guest is viewing the Property Details for a Property. The System has provided a means to select the Initial Booking Details for a new Booking; these are pre-selected if the Guest has already provided these details in some Property Search Conditions. The System has limited the Initial Booking Details the Guest may select to match the Property's availability data.
Main Flow	1. The use case starts when the Guest begins selecting the Initial Booking Details. a. If the Guest selects the Initial Booking Details, or leaves them unchanged: go to 2. b. (END) If the Guest cancels the operation: the System displays a Property List conforming to any Property Search Conditions and/or Property Sort Criteria set previously. 2. The Guest submits the Initial Booking Details. 3. The System checks the Initial Booking Details. a. If the Initial Booking Details are Possible: the System Locks the Initial Booking Details. Go to 4. b. If the Initial Booking Details are not Possible: the System displays a notification about the details causing the problem. Return to 1. 4. The System provides the Guest with a means to fill in the full Booking Details. 5. a. If the Guest fills in the Booking Details: go to 6. b. (END) If the Guest cancels the operation: the System unLocks the Initial Booking Details and displays the Property Details as in the Preconditions. 6. The Guest submits the Booking Details. 7. The System checks the Booking Details. a. If the Booking Details are Correct: go to 8. b. If the Booking Details are not Correct: the System displays a notification about the details causing the problem. Return to 4. 8. The System sends the Payment Details to the Payment System. 9. The Payment System checks the Payment Details and sends the results back to the System. 10. a. If the Payment System confirmed the Payment: i. The System unLocks the Initial Booking Details. ii. The System creates a Booking based on the details provided. iii. The System sends a confirmation email that includes the Core Booking Details to the Guest and to the Host whose Property the Booking is for. Go to 11. b. If not: the System displays a notification about the failed Payment. Return to 4. 11. (END) The System displays a Booking confirmation and shows the Guest's User Bookings.

TABLE 3.6 – Use Case B-C: Make Booking.

Postconditions	■ If 11 reached: a Booking has been created, and a Payment has been made. The Booking Details correspond to those specified during the creation process. The Guest and the Host whose Property the Booking is for have both been sent a confirmation email that includes the Core Booking Details. The System displays the User Booking List. ■ If 1b. taken: the System displays a Property List conforming to any Property Search Conditions and/or Property Sort Criteria set previously. ■ If 4b. taken: the System displays the Property Details as in the Preconditions.

Table 3.7 – Use Case B-C: Make Booking (continued).

We have two use cases for listing Bookings — View User Bookings (B-L-1; Table 3.8) for Guests and Admins, and View Property Bookings (B-L-2; Table C.7) for Hosts and Admins. These are closely analogous, so the latter has been relegated to Appendix C.

ID	B-L-1
Use case	View User Bookings
Description	A Guest views their own Bookings, or an Admin views any Guest's Bookings. (Henceforth, we will refer to the Primary Actor as "User", and to the User whose bookings are being accessed as "Target User".)
Primary Actor	Guest/Admin
Secondary Actor(s)	None
Preconditions	■ Guest: None ■ Admin: The Admin is viewing their Dashboard.
Main Flow	1. The use case starts when the User issues a request to view the Target User's Bookings. The User Bookings are in Future mode by default. 2. (POSSIBLE END) The System checks whether the User Bookings are in Future mode or Past mode, and: a. If the User Bookings are in Future mode: The System displays all of the Target User's Future Bookings and provides the User a means to switch to Past mode. b. If the User Bookings are in Past mode: The System displays all of the Target User's Past Bookings and provides the User a means to switch to Future mode. 3. If the User switches the User Bookings mode: return to 2.
Postconditions	The System displays either the Target User's Past Bookings or the Target User's Future Bookings. (These might be empty lists.)

Table 3.8 – Use Case B-L-1: View User Bookings.

For the following, note that Bookings are never actually deleted (from the System database) — they are instead flagged as Cancelled Bookings and kept on the System for auditing

purposes. Note also that the Payment System is not a Secondary Actor here: while it does receive the refund request, it never actually performs any actions.

ID	B-D
Use case	Cancel Booking
Description	A Guest cancels one of their own Cancellable Bookings or a Host cancels a Cancellable Booking on their Property. (Henceforth, we will refer to the Primary Actor as "User".)
Primary Actor	Guest/Host
Secondary Actor(s)	None
Preconditions	■ Guest: The Guest is viewing their Future User Bookings. ■ Host: The Host is viewing their Property's Future Property Bookings.
Main Flow	1. The use case starts when the User requests to cancel a Booking. 2. The System checks if the Booking is Cancellable. a. If the Booking is Cancellable: go to 4. b. If the Booking is not Cancellable: the System notifies the User that the Booking may not be cancelled. Go to 8. 3. The System asks the User for confirmation. 4. a. If the User confirms the cancellation: go to 5. b. If the User cancels the operation: go to 8. 5. Using the Payment Details for the Bookings, the System sends a refund request to the Payment System. 6. The System makes the Booking a Cancelled Booking. 7. The System sends a cancellation email that includes the Core Booking Details to the Guest who made the Booking and to the Host whose Property the Booking was for. 8. (END) The System displays the User/Property Bookings as in the Preconditions.
Postconditions	The System displays the User/Property Bookings as in the Preconditions and: ■ If 2a. and 4a. taken: the Booking is now a Cancelled Booking. The Guest who made the Booking and the Host whose Property the Booking was for have been sent a cancellation email that includes the Core Booking Details.

Table 3.9 – Use Case B-D: Cancel Booking.

3.5 POLICIES

Only Admins have access to the Policy-related use cases. Only creating (Po-C; Table 3.10) and deleting (Po-D; Table 3.11) Policies were deemed important enough to qualify for their own use cases. The Admin can view a list of Policies on their Dashboard; editing Policies is not necessary since they are so simple (one would simply delete the existing Policy and create a new one in its place).

ID	Po-C
Use case	Create Policy
Description	An Admin creates a new Policy.
Primary Actor	Admin
Secondary Actor(s)	None
Preconditions	The Admin is viewing their Dashboard.
Main Flow	1. The use case starts when the Admin issues a request to create a Policy. 2. The System provides the Admin with a means to fill in the Name for the new Policy. 3. a. If the Admin provides the Policy Name: go to 4. b. If the Admin cancels the operation: go to 6. 4. The Admin submits the Policy Name. 5. The System checks the Policy Name. a. If the Policy Name is unique: the System creates a new Policy with the Policy Name provided. b. If a Policy with the same Name already exists: the System displays a notification about the potential duplication. Return to 2. 6. (END) The System displays the Admin's Dashboard.
Postconditions	The System displays the Admin's Dashboard and: ■ If 3a. and 5a. taken: a Policy has been created. The Policy has the Policy Name specified during the creation process.

Table 3.10 – Use Case Po-C: Create Policy.

ID	Po-D
Use case	Delete Policy
Description	An Admin deletes a Policy.
Primary Actor	Admin
Secondary Actor(s)	None
Preconditions	The Admin is viewing their Dashboard.
Main Flow	1. The use case starts when the Admin issues a request to delete a Policy. 2. The System asks the Admin for confirmation. 3. a. If the Admin confirms the deletion: go to 4. b. If the Admin cancels the operation: go to 5. 4. The System deletes the Policy and removes it from all affected Rooms. 5. (END) The System displays the Admin's Dashboard.
Postconditions	The System displays the Admin's Dashboard and: ■ If 3a. taken: the Policy has been deleted; the Policy has been removed from all affected Rooms.

Table 3.11 – Use Case Po-D: Delete Policy.

3.6 RATINGS

Everything to do with Ratings is handled via a single use case, R-C (Table 3.12). The Ratings are displayed as part of the Property Details in P-V, and we decided that to be sufficient (in terms of viewing/listing Ratings) for the scope of this project. One straightforward extension of the current setup would enable each Guest to see a list of all Ratings they have given and allow the Admin to see all Ratings System-wide in a single list. No special editing or deleting use cases have been included: Ratings can be replaced by rating the Property in question again.

ID	R-C
Use case	Give Rating
Description	A Guest gives a Rating to a Property for which they have a Past Booking.
Primary Actor	Guest
Secondary Actor(s)	None
Preconditions	The Guest is viewing their Past User Bookings.
Main Flow	1. The use case starts when the Guest selects a Booking from their Past User Bookings and issues a request to give a Rating to the Property associated with that Booking. 2. The System checks if the Guest already has a Rating for this Property. a. If the Guest does not have a Rating for this Property: go to 3. b. If the Guest has a Rating for this Property: the System notifies the Guest that the existing Rating will be overwritten if the operation is completed. Go to 3. 3. The System provides the Guest with a means to fill in the Rating Details for the new Rating. 4. a. If the Guest fills in the Rating Details: go to 5. b. If the Guest cancels the operation: go to 6. 5. The System checks if the Rating Details are Correct. a. If the Rating Details are Correct: i. If 2b. taken: the System deletes the existing Rating. In any case: ii. The System creates a new Rating for this Property based on the details provided, and associates it with this Guest. Go to 6. b. If the Rating Details are not Correct: return to 3. 6. (END) The System displays the Guest's Past User Bookings.
Postconditions	The System displays the Guest's Past User Bookings and: ■ If 2b., 4a. and 5a. taken: the existing Rating for this Property by this Guest has been deleted, and a new Rating has been created. The Rating has the Rating Details specified during the creation process. ■ If 2a., 4a. and 5a. taken: a new Rating for this Property by this Guest has been created. The Rating has the Rating Details specified during the creation process.

TABLE 3.12 – Use Case R-C: Give Rating.

3.7 FAVOURITES

The Favourites are only accessible to Guests, and seeing as the entities are very simple in nature (amounting to nothing more than links between Users and Properties), no editing use case is required. Adding a Property to one's Favourites (F-C; Table 3.13) and viewing one's Favourites (F-L; Table 3.14) are featured here; removing a Property from one's Favourites (F-D; Table C.8) is in Appendix C.

ID	F-C
Use case	Add to Favourites
Description	A Guest adds a Property to their Favourites.
Primary Actor	Guest
Secondary Actor(s)	None
Preconditions	The Guest is viewing the Property Details for Property to be added.
Main Flow	1. The use case starts when the Guest issues a request to add the Property to their Favourites. 2. The System adds the Property to the Guest's Favourites. 3. (END) The System displays the Property Details, with a symbol indicating that the Property is in the Guest's Favourites.
Postconditions	The Property is added to the Guest's Favourites. The System displays the Property Details, with a symbol indicating that the Property is in the Guest's Favourites.

TABLE 3.13 – Use Case F-C: Add to Favourites.

ID	F-L
Use case	View Favourites
Description	A Guest views their Favourites.
Primary Actor	Guest
Secondary Actor(s)	None
Preconditions	None
Main Flow	1. The use case starts when the Guest issues a request to view their Favourites. 2. (END) The System displays the Guest's Favourites and provides a means for the Guest to request to view the Property Details for each Property in the Favourites.
Postconditions	The System displays the Guest's Favourites and provides a means for the Guest to request to view the Property Details for each Property in the Favourites.

TABLE 3.14 – Use Case F-L: View Favourites.

Object-Oriented Analysis (OOA) is a key activity in good software design as it facilitates the difficult transition between the problem domain and the solution domain. During this stage in the development process, the designer switches from a user-centric to an object-oriented point of view. Information from the domain model, use cases and requirements is extracted and systematically translated into a conceptual model. While the analysis is concerned with the "what", the design focuses on the "how" of a system.

The design phase refines the analysis model and describes the objects, their attributes, behaviour, and interactions in detail. Outputs of the design model should, therefore, include all details necessary to implement the product. However, Object-Oriented Analysis and Design activities are best conducted in iterative and incremental steps.

In the following section we will elaborate on the analysis model and on how this output shapes the subsequent design model.

4.1 Analysis Model

The goal of the analysis model is to create a draft version of the system. Although it abstracts the definition of the most important objects, the analysis model should exhaustively capture the system's required entities. Therefore, a data-driven design approach to identifying all classes was taken.

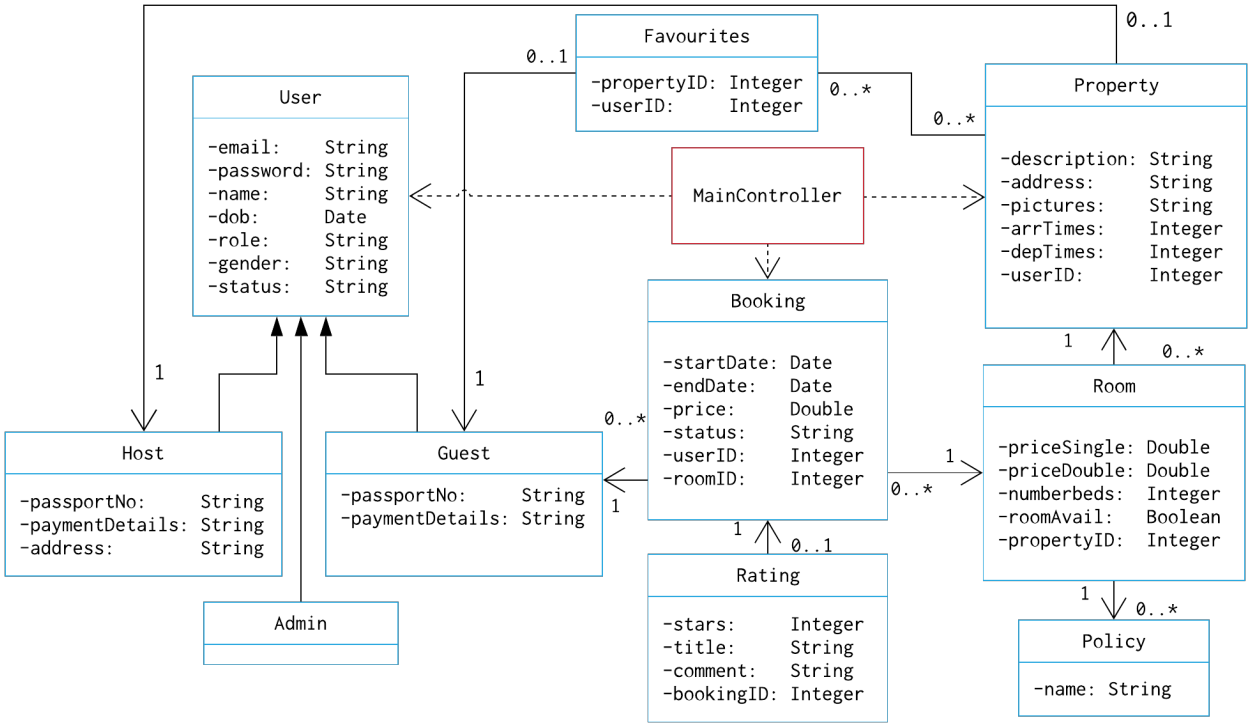

Figure 4.1 – Analysis Class Diagram.

After locking down the requirements and subsequently the use cases, the entity classes in the system were identified, as shown in figure 4.1. This was achieved by extracting the main nouns that were being commonly used, such as: `User`, `Property`, and `Booking` [8].

The Analysis Class Diagram displays the initial classes in the B&B system and indicates the multiplicities and associations between them. The model achieves high cohesion by acknowledging the separation of concerns design principle, therefore assigning only a small and well-defined set of responsibilities to each class.

4.2 DESIGN MODEL

The Design Class diagram shown in figure 4.2, illustrates a more interconnected architecture than that shown in figure 4.1. This diagram's main purpose is to show the classes and their relationships, in addition to their attributes and operations. The system's architecture is made up of a mix of the following types of UML classes: Boundary, Entity, Controller, and Enumeration. It is to be noted that many of the parameters are abbreviated terms from our glossary for brevity.

As shown by the legend in figure 4.2, the classes have been colour-coded according to the class group that they belong to. The system has been designed to comply with the **Model-View-Controller** pattern, which will be discussed in more detail in section 4.6.

- **Models/Entity Classes**: These classes have a connection to the database. By using Object-relational mapping, their objects can be modelled as database records that include attributes and operations that are mainly CRUD based [9]. Any additional business logic will also be placed in these classes. Getter and setter operations such as `User.getName()`, which returns the name attribute of a given user object, have been left out of the diagram due to space constraints. They are implicitly included in the rest of the sequence diagrams in section 4.3.
- **Controller/Control Classes**: These classes provide endpoints to the users in the form of actions. Every HTTP request made by the browser will be connected to an appropriate operation under these classes.
- **View Classes**: After various meetings with the client, the recommendation that was decided on was to add these types of classes which are responsible for generating the HTML to be displayed by the browser. They receive parameters from the appropriate controllers in the form of objects.
- **Boundary Classes**: Within the system, there exists a need to talk to external services such as email service providers and payment gateways. The boundary classes act as the connection between these services and the internal models and controller. The connection is achieved by sending requests and receiving success/failure responses.
- **Enumeration Classes**: These classes represent a common type of data as constants.

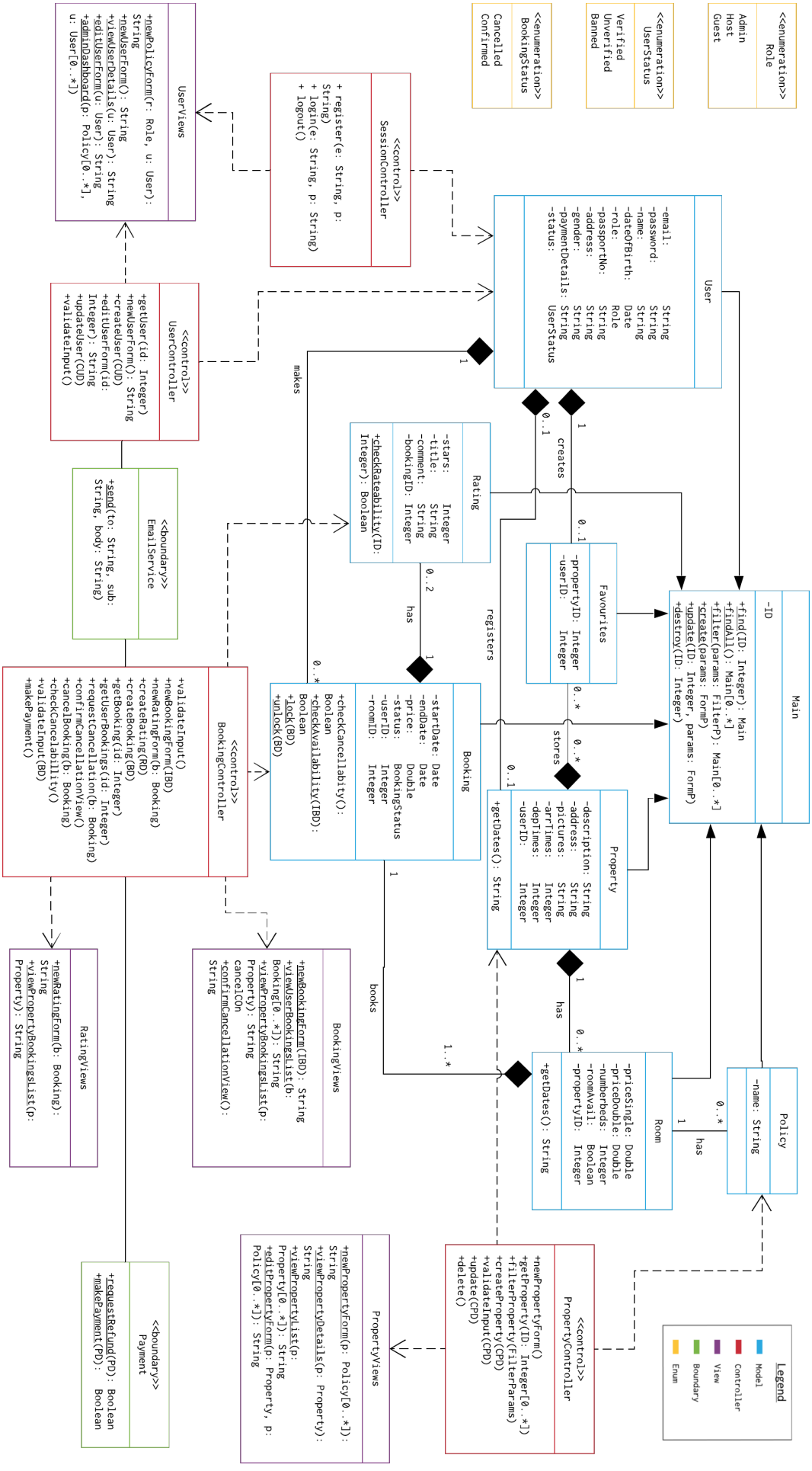

Figure 4.2 – UML Design Class Diagram.

4.3 SEQUENCE DIAGRAMS

Sequence diagrams are valuable tools for specifying how use cases can be implemented by the system in line with the classes, controllers, views, and methods developed in the class diagram. Specifically, they show the explicit sequence in which messages between the interacting parties in a use case are exchanged. The six sequence diagrams presented in this work do not fully exhaust all possible options of a given use case which would constitute a generic form, but rather depict some specific scenario or route through the use case [10].[1]

Following the agile approach, the sequence diagrams have been developed iteratively with the intention of striking a balance between the level of detail for implementation in code and degree of abstraction to emphasize the core concepts (see the more detailed notes on this below).

All the methods and classes listed in the sequence diagrams correspond to the information in the class diagram. Return values that are pure HTML strings, e.g. the details of a property, are denoted as views, i.e. `propertyDetailsView`.

Note the following about the presentation here:

- The class name has been used when referring to the class itself (this usually happens when using the class' static methods), and an object name followed by a colon and a class name when referring to an object (so, for instance `Property` is the Property class, whereas `p : Property` is a Property object.)
- When sending a message to some external actor or system (such as the Payment System in Table 4.6 (Make Booking)), the text associated with the message is not meant to name a method — it is simply a description of the message.
- The parameters of the method calls are not in any strict format — it should be clear from the context what is being sent at each stage. Abbreviations of terms from the glossary are commonly referred to as done in the Design Class Diagram. Note also that some of the methods are overloaded (it should always be clear from the context how this works — `validateInput()` method in `BookingController` can check both Booking Details and Initial Booking Details in the obvious way)
- Each diagram includes a list of the use case Main Flow steps (for example, "P-C.3.a.") corresponding to each of its stages at the very left. Duration lines (the lines with dots at either end) are also attached to the steps to make things even clearer.
- The diagrams have been simplified: due to space and legibility constraints, we have left out many lower-level method calls necessary to carry out the represented scenarios. A typical example of this is the Property constructor call in Figure 4.3

[1] Most scenarios chosen were straightforward success routes, but note that Figure 4.5 (View Property List) portrays two iterations of a Property search loop, with one iteration coming up with an empty result set and hence showing the User a warning notification.

(Register Property). In a more detailed depiction, the constructor would loop over all the Rooms in the Property constructing each of them, and then further loop over all Policies in each of the Rooms, constructing each of those as well. The current diagram is much clearer than one involving nested loops would be, and the intended realization in terms of lower-level method calls is obvious. Another example of simplification: the static database-related method calls made to the Model classes (`filter()`, for instance) are shown as returning objects of the same classes. The group thought it clear that a constructor was being called in such instances, and thought it would be unnecessary to add such details to the diagrams.

- In a similar vein, it is sometimes implicitly assumed that certain pieces of information can be accessed (in cases where it should be clear where this information comes from). This can be illustrated with the User object u in Figure 4.7 (Cancel Booking) and the Payment Details in Figure 4.6 (Make Booking). In the former, it is assumed that u is already constructed, and it is used to get a set of Payment Details. The object corresponds to a`Guest`, the main actor in the diagram, and was obtained when a`Guest` logged in. Its reference is being held by a `SessionController` object. In Figure 4.6, even the method call to the User object is skipped, and it's assumed that the Payment Details are already available. Assumptions like these helped make the diagrams much cleaner. (Another similar assumption in the same diagrams: the Users' email addresses are retrieved without explicitly fetching them at any point.)
- Many of the objects used are portrayed as being destroyed when the method that held their reference finishes executing (and hence there are no explicit destroy messages sent). In practice, when the objects are destroyed would depend on the garbage collection implementation of the finished application, but such details are not considered here.
- Note that in effect all the steps necessary to view a User's Bookings (see B-L-1 (Table 3.8) for the use case) are repeated in Figures 4.6, 4.7 and 4.8. This is due to all the portrayed scenarios finishing with a list of User Bookings. Use case inclusion would have made these diagrams simpler at the cost of potentially making our use cases more difficult to understand.

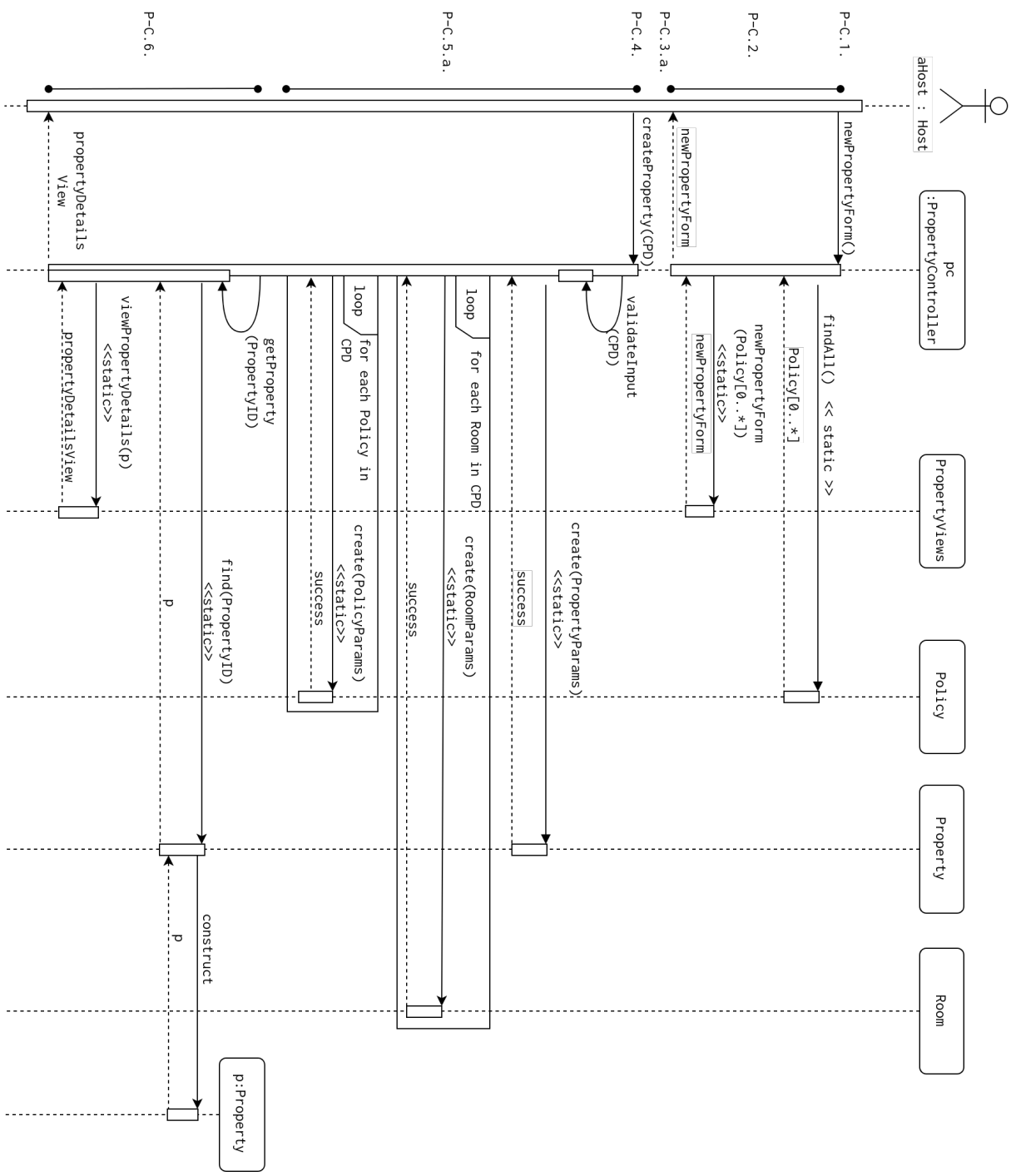

Figure 4.3 – Sequence Diagram: Register Property.

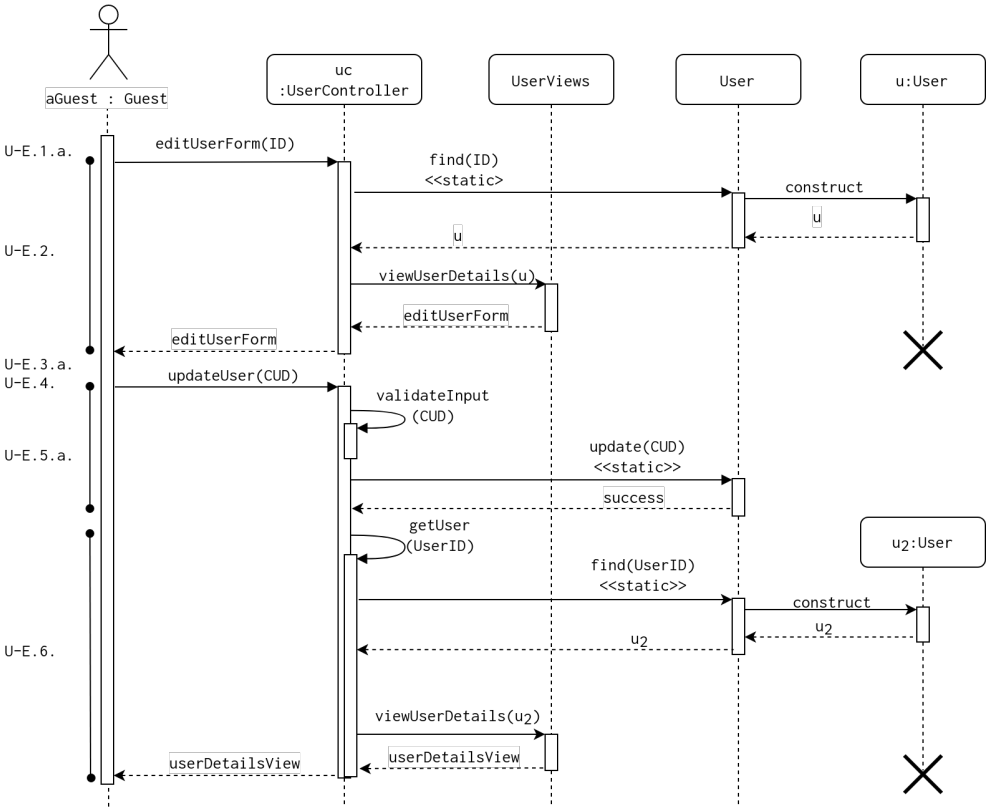

Figure 4.4 – Sequence Diagram: Edit User.

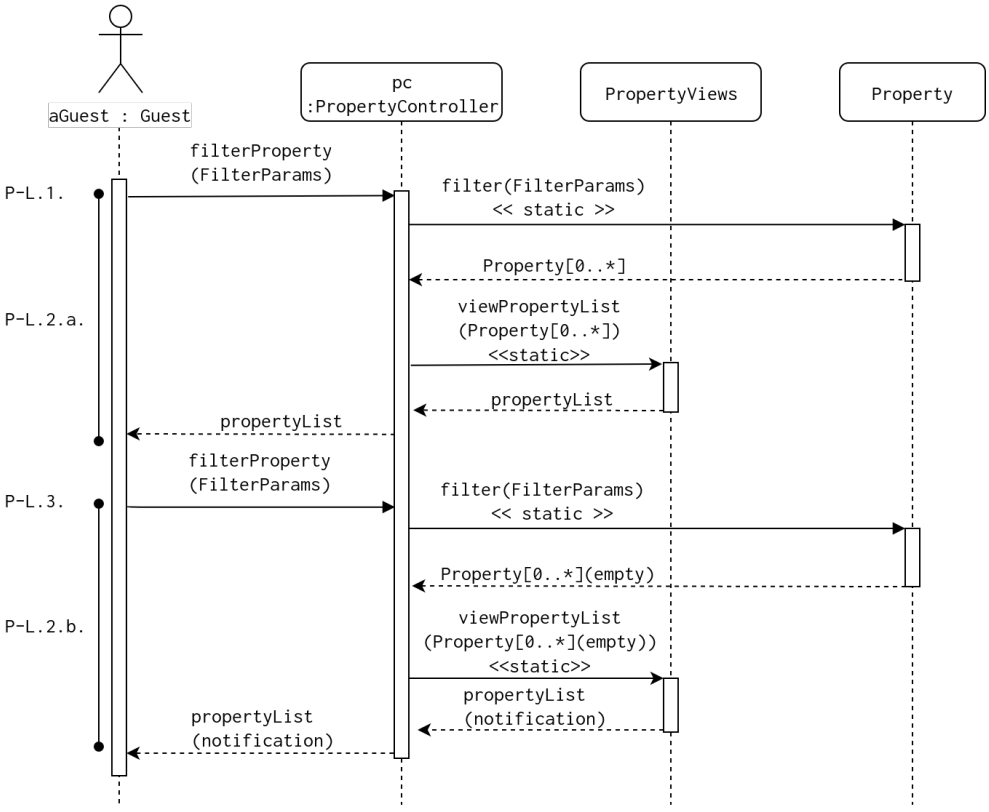

Figure 4.5 – Sequence Diagram: View Property List.

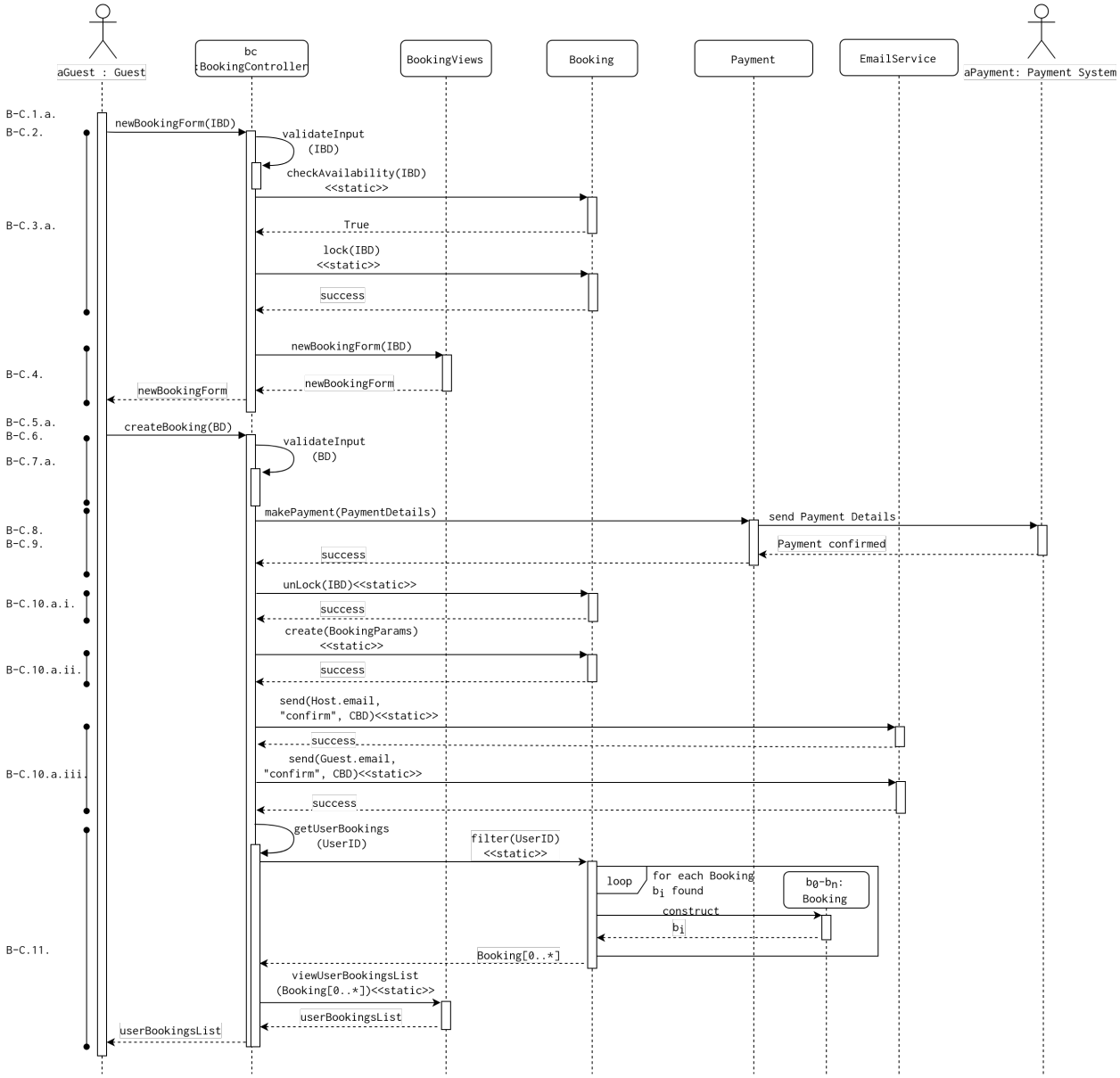

Figure 4.6 – Sequence Diagram: Make Booking.

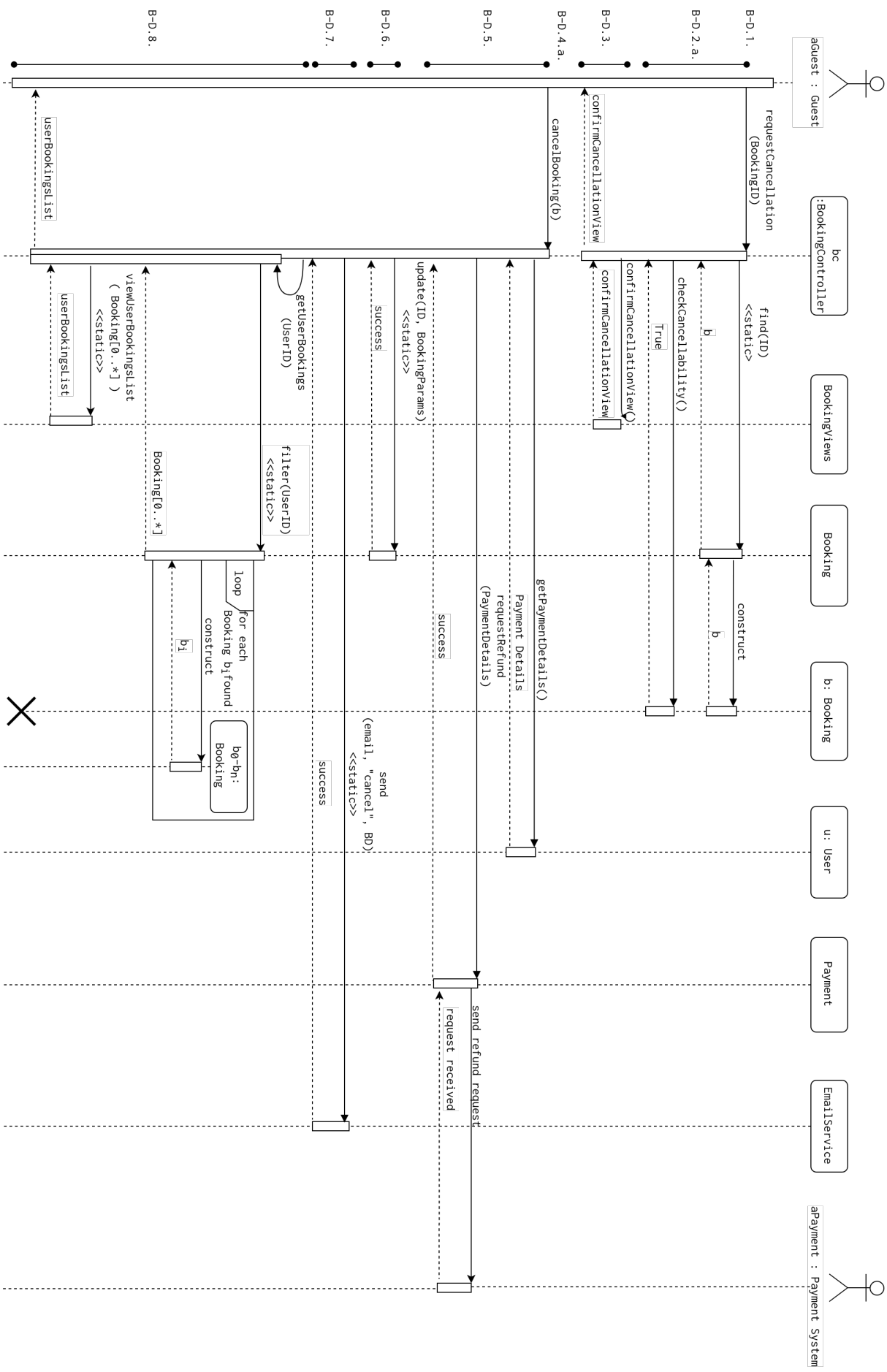

Figure 4.7 – Sequence Diagram: Cancel Booking.

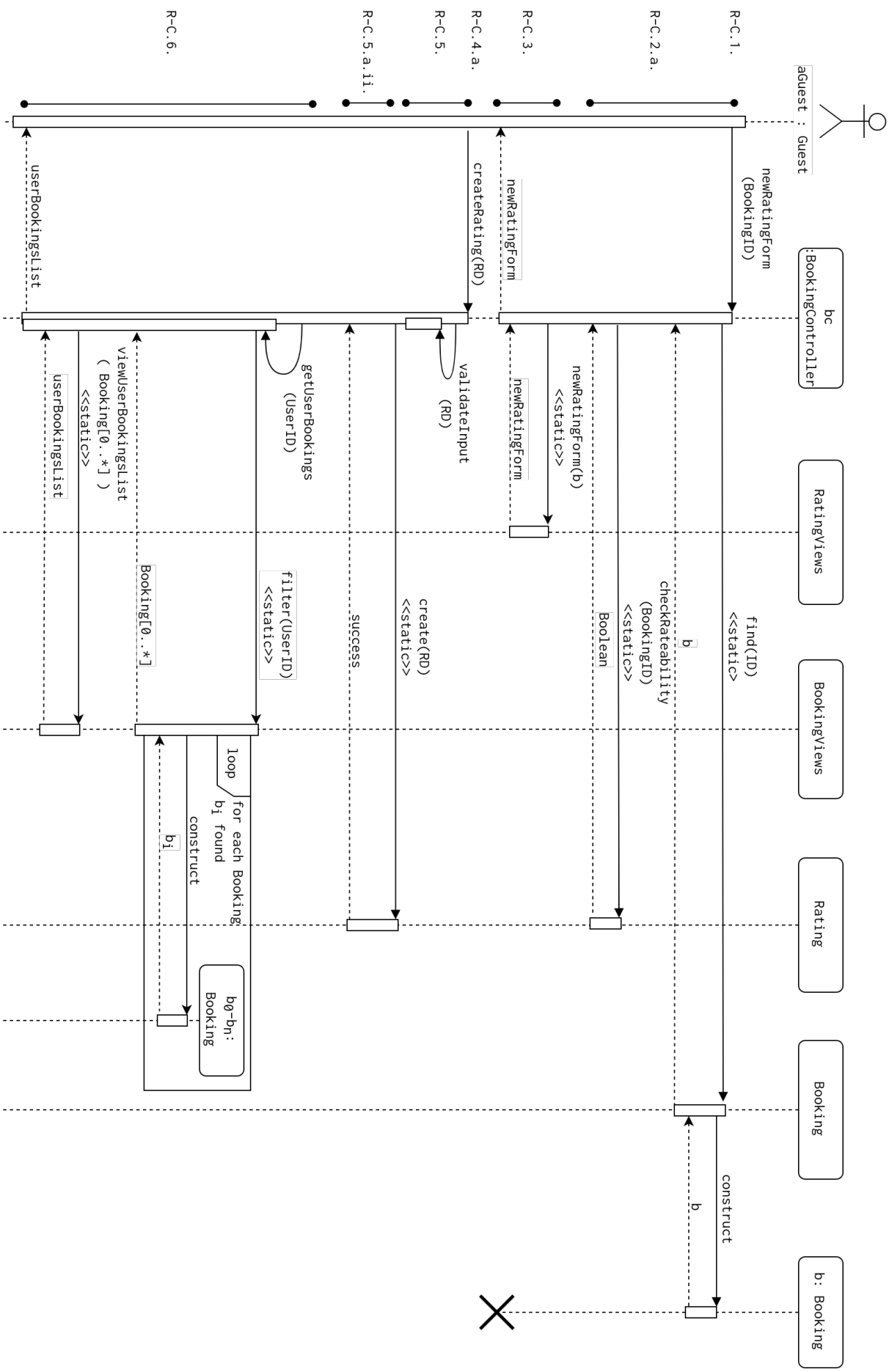

Figure 4.8 – Sequence Diagram: Give Rating.

4.4 State Machine Diagram

A state machine diagram is a type of behavioural diagram that shows transitions between various objects. Based on the input it receives it can change its status and thereby determine which operations an object can perform at a given moment. The case depicted in the following figure shows the process by which a Guest makes a Booking. In the "Check property availability" composite state, the system checks the property for availability in a few different sub-states. If the time is not available on the room, the process will be escaped. If the room shows availability, however, the booking will be added. To transition from available to unavailable, a booking for this property is created. To exit the unavailable state, a booking is either deleted or lies in the past.

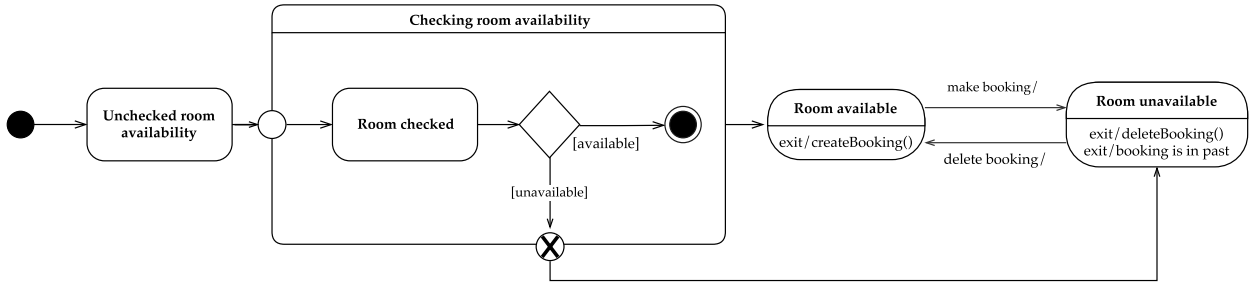

Figure 4.9 – State Machine Diagram: Make booking.

4.5 Activity View

Following the design flow from the static view on the system (cf. Figure 4.2) to the interaction view in the form of sequence diagrams (cf. section 4.3), this section aims to show the control flow in the system in a more abstract way via the activity view. UML activity diagrams present business processes executed by users interacting with the system.

Figures 4.10 – 4.13 describe in more detail how the use cases detailed in chapter 3 translate into a sequence forming main business processes. An overview of which use cases map to a specific activity is given in Figure 4.1[2]. As the final product aims at matching Property hosts with potential Property guests, activities such as the Guest making a Booking, Host registering a Property or cancelling a Booking are selected as primary examples.

Activity diagrams are useful in identifying and specifying the sequential and concurrent nature of dependencies in the system. Whereas the sequence diagrams show the control flow between objects and hence show the technical implementation of a use case depicting operations on classes and message interaction, the activity diagrams below give a high-level overview of the execution behaviour of the system and do not show objects that perform the activities [11]. Note how different types of control nodes specify the sequencing of action nodes and the high-level description of validation processes that are inherited in

[2]The colour code in the 'ID 'column refers to the Actor involved in the activity, rather than the Actor set that is generally eligible for the use case.

the behaviour of decision nodes [2]. Refer to the User Manual in section 5 on how the control described in this section translates into the user interface.

Activity	ID	Use Case
Host & Property Registration	U-C-1	A Visitor registers as a Guest or a Host.
	P-C	A Host registers a new Property.
Guest make Booking	F-L	A Guest views their Favourites.
	P-V	A Guest, Admin, or a Host views a Property's Property Details.
	B-C	The Guest makes a new Booking.
Guest search Properties	P-L	A Guest or an Admin searches for, filters, orders, and views a List of Properties.
	F-C	A Guest adds a Property to their Favourites.
	F-D	A Guest removes a Property from their Favourites.
Host cancel Booking	B-L-2	A Host views their own Property's Bookings, or an Admin views any Property's Bookings.
	B-D	A Guest cancels one of their own Cancellable Bookings, or a Host cancels a Cancellable Booking on their Property.

Table 4.1 – Activity & Use Case Mapping.

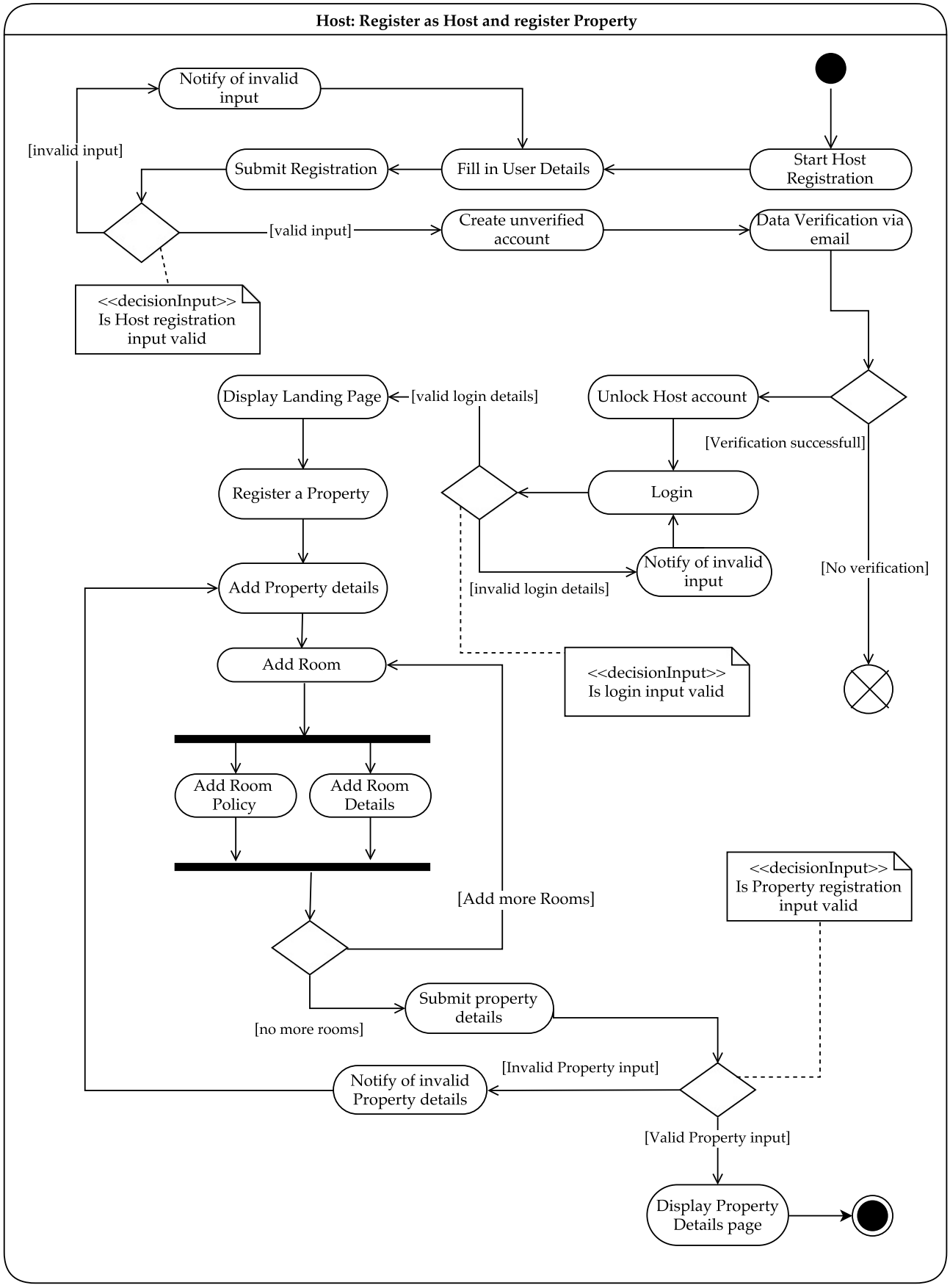

Figure 4.10 – Activity Diagram: Host & Property Registration.

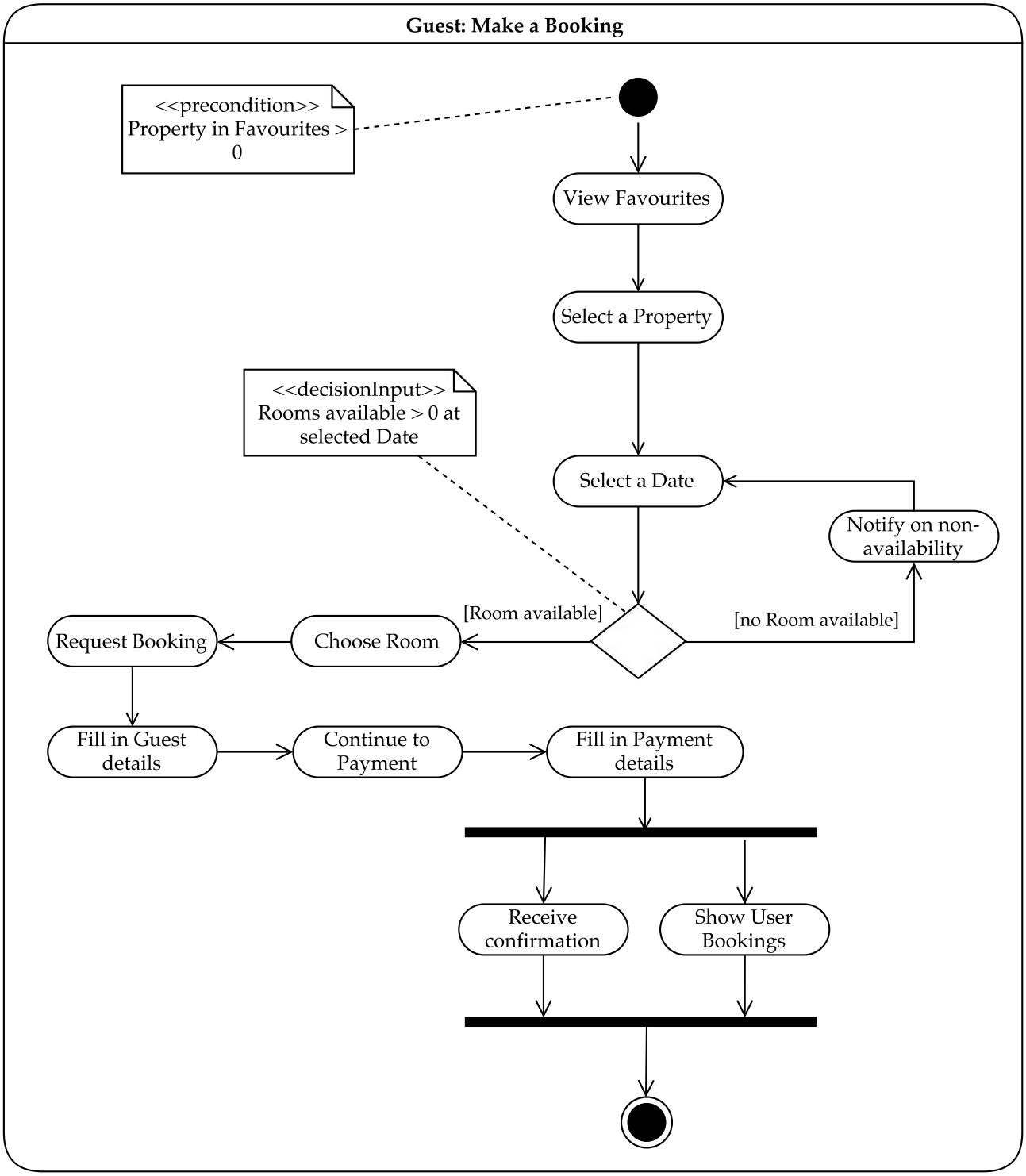

Figure 4.11 – Activity Diagram: Guest Booking.

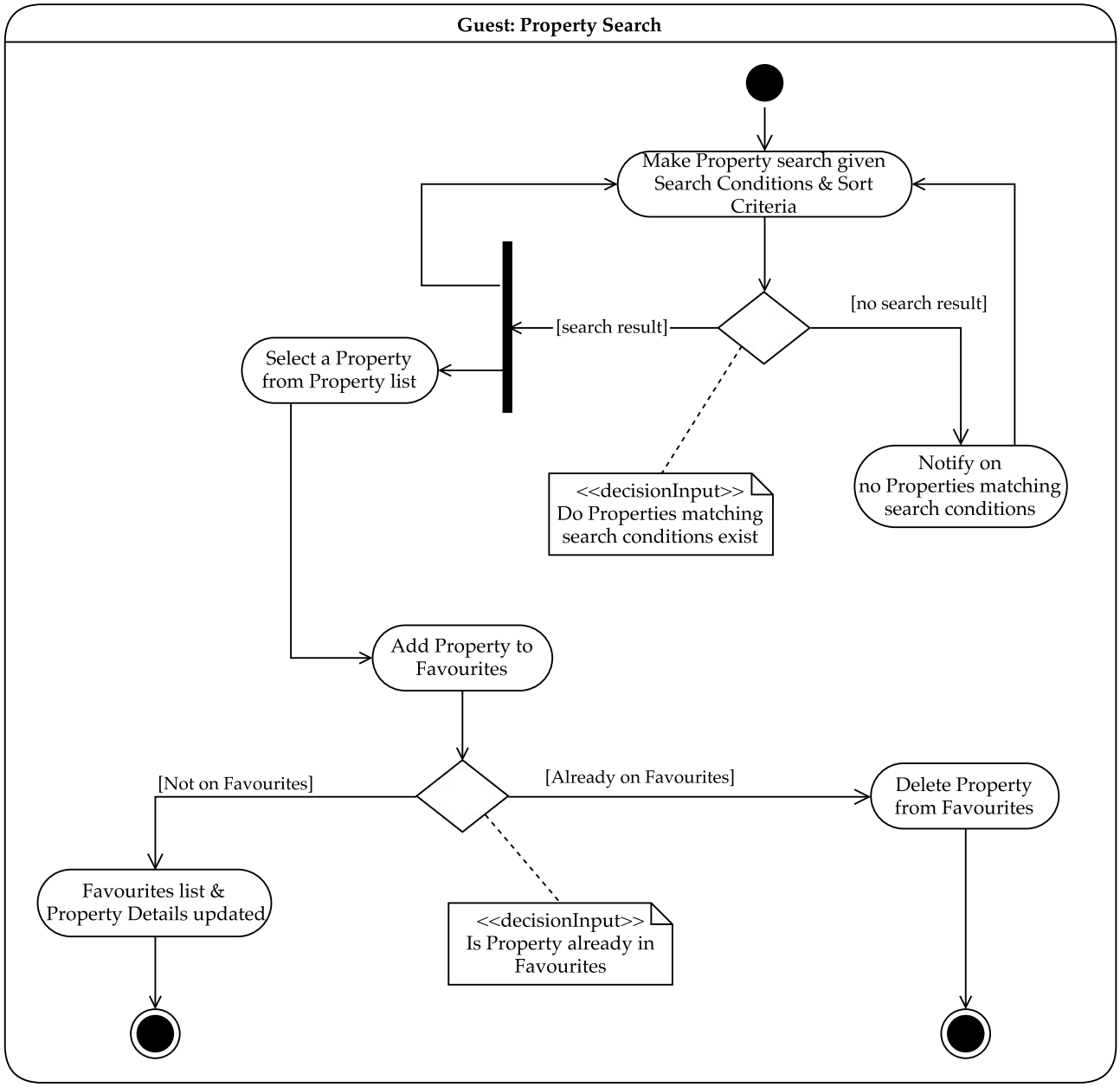

Figure 4.12 – Activity Diagram: Search Properties[3].

[3]The action node for selecting a Property implies the view of the Property Details page

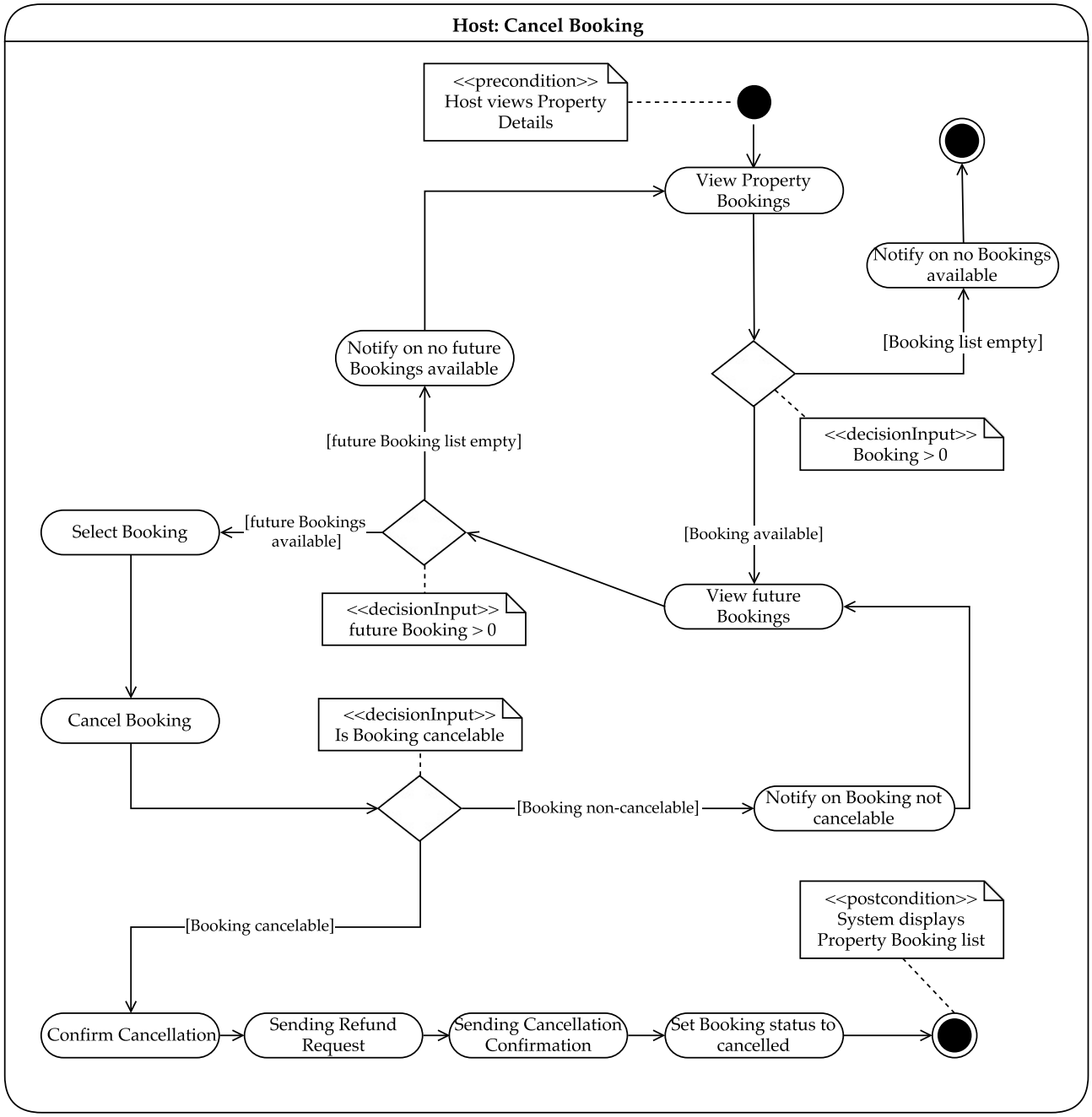

Figure 4.13 – Activity Diagram: Host Cancel Booking.

4.6 component diagram

A generalised architectural overview of the entire system can be visualised using figure 4.14. Since components are parts of the system that are encapsulated, reusable, and replaceable [12], the component diagram shows how the usage of the widely adopted Model-View-Controller architectural pattern, usually referred to as MVC [13], takes full advantage of these best practices.

The aforementioned diagram builds on the Design and Analysis diagrams by combining similar components together. Within the system, there are three main component groups: Views, Controllers, and Models. The classes under each group of components are respon-

sible for a similar set of tasks. The View classes have the sole responsibility of generating strings from the inputs sent to them from the controllers. The strings will then be sent to the client browsers and effectively be rendered as HTML. The Controller classes are responsible for acting as a middle layer that talks to the Models and the Views. Controllers do not contain business logic but rather provide endpoints that can be called from the generated views. The Models group are the classes that contain the business logic and are directly connected to the database.

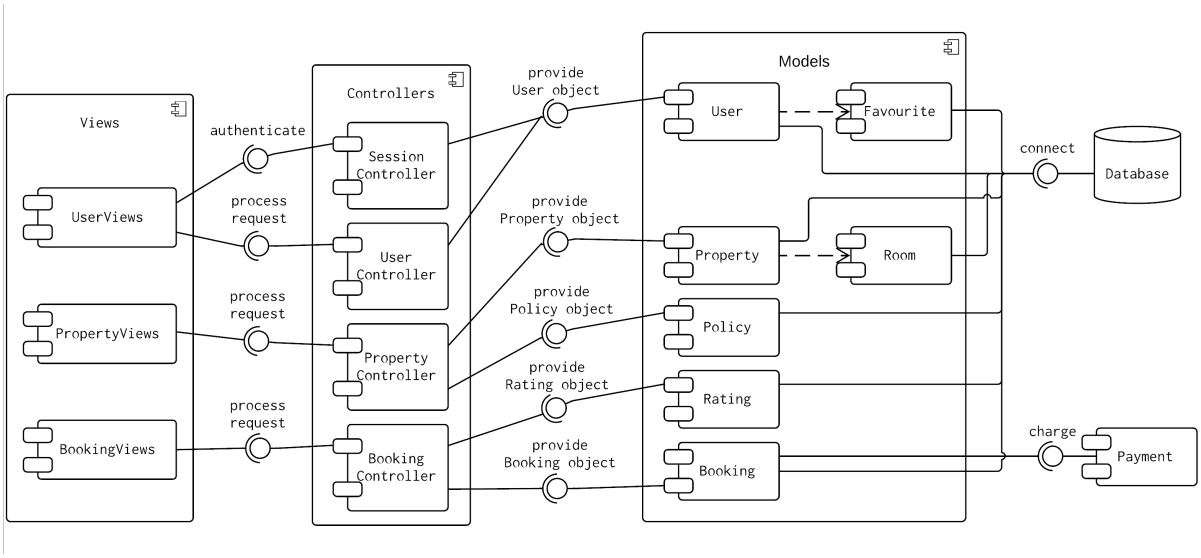

Figure 4.14 – Component Diagram.

4.7 Deployment Diagram

The deployment to hardware environments is described using the following deployment diagram. This diagram shows the physical connections within the system in terms of deploying the software to a physical environment.

The system contains four main nodes:

- *UserClient* - A web browser such as Google Chrome, Safari, or Firefox that renders the HTML assets to the Users.
- *WebServer* - Apache HTTP Server that will be responsible for dealing with HTTP requests and responses to/from clients.
- *ApplicationServer* - A server such as Apache Tomcat which is capable of running Java EE specifications and hosts a suitable execution environment.
- *DatabaseServer* - This server will be responsible for managing and maintaining MySQL Database that is connected to the system.

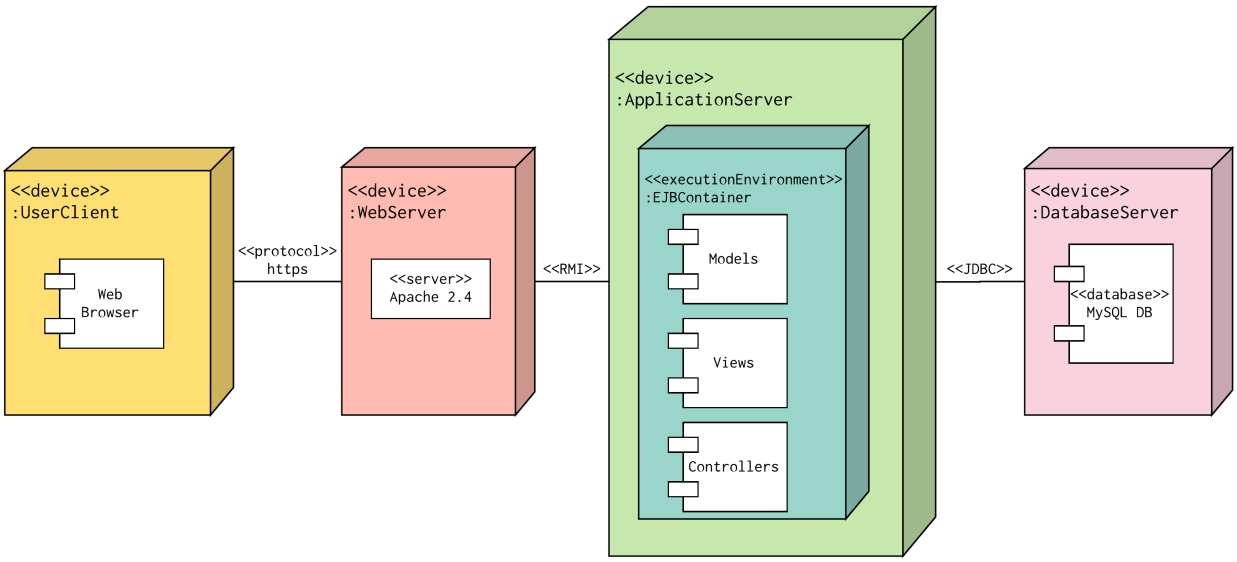

Figure 4.15 – Deployment Diagram.

4.8 SITE MAP

The hierarchical structure of the system is depicted in the following site map. It shows the organisation of the site's content and provides a good overview of the pages of the site.

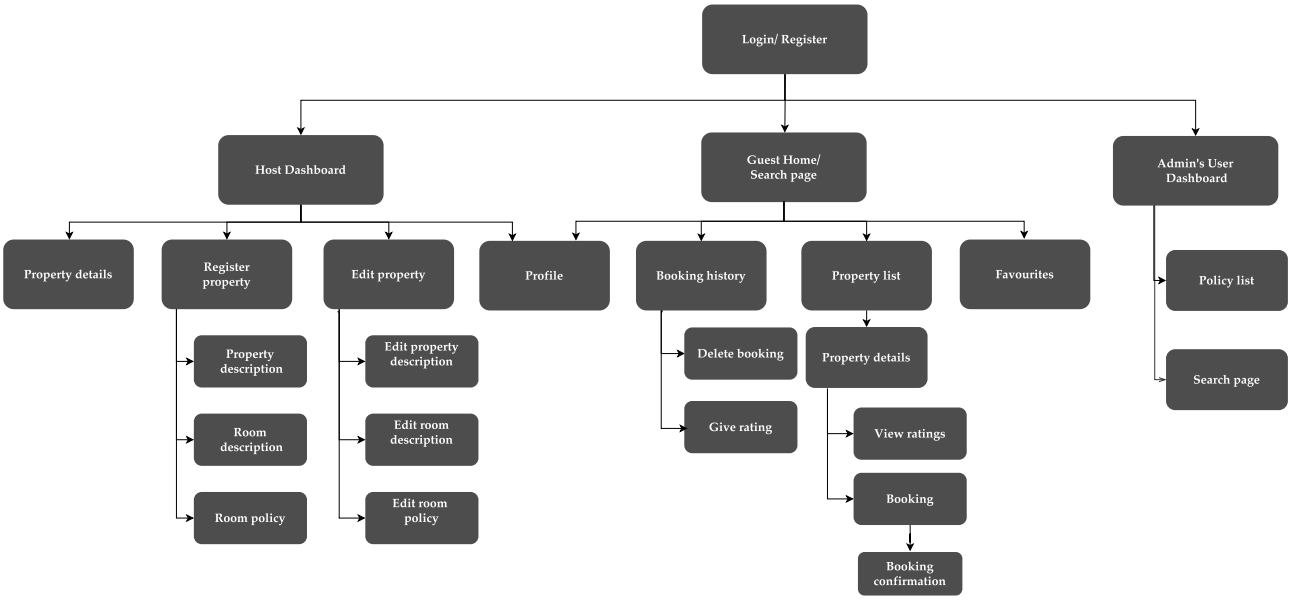

Figure 4.16 – Site Map.

5.1 User Views

5.1.1 *Registration and Login*

To be able to use the services of *Room.io*, a User needs to be logged into their account by entering an email and password. If they have not yet registered on the system, a registration as either a Host or a Guest is necessary. To sign up, they simply need to enter: first and last name, email, password, and their Role. After submitting this information, the User receives an email to confirm the details which they need to respond to in order to become a Verified User. An Admin can only be added by an existing Admin as described in section 5.4.1.

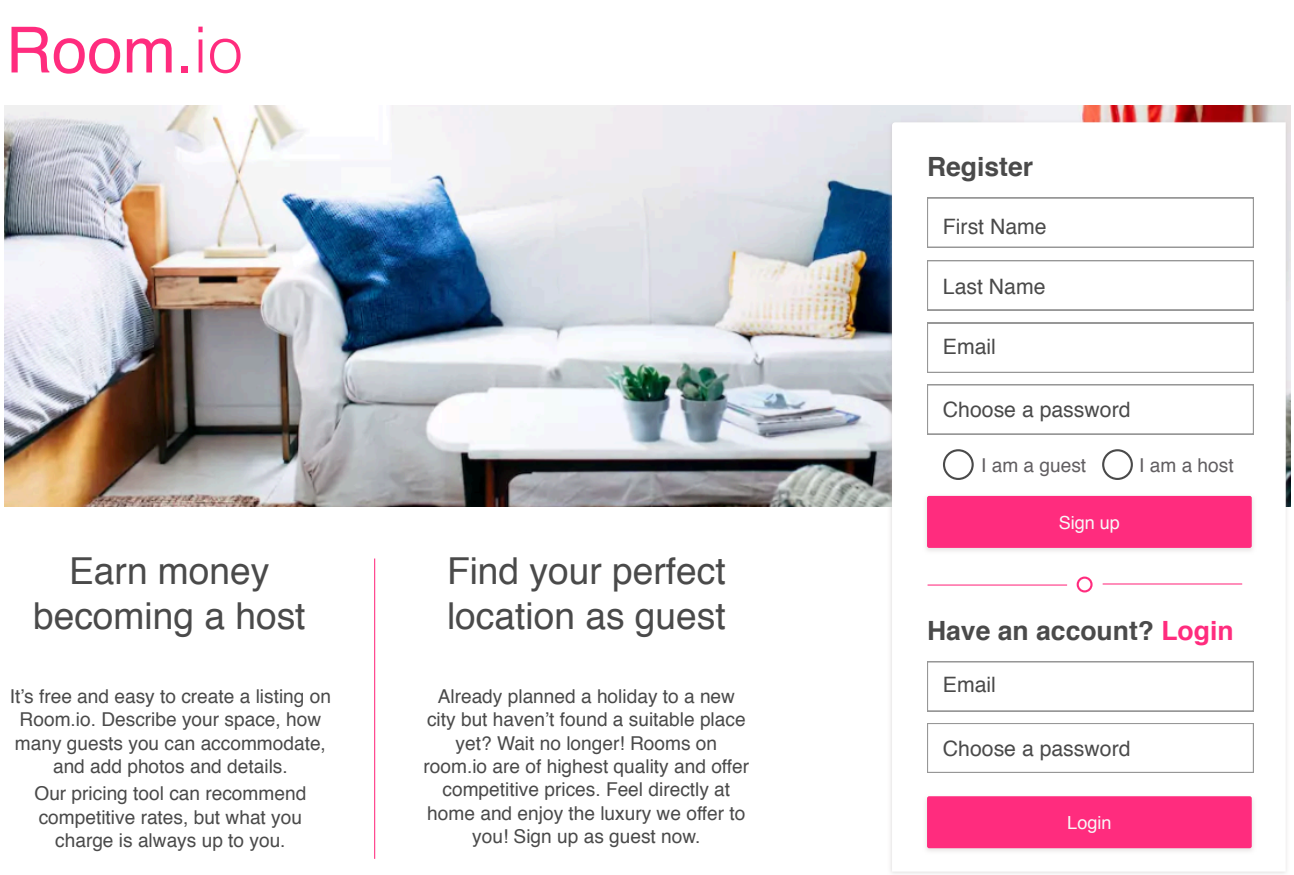

Figure 5.1 – Registration and Login View.

5.1.2 *Profile Page*

To view and edit one's personal details, the Guest/Host can visit their profile page from the search page. Here they can update their information such as first and last name, email, and password.

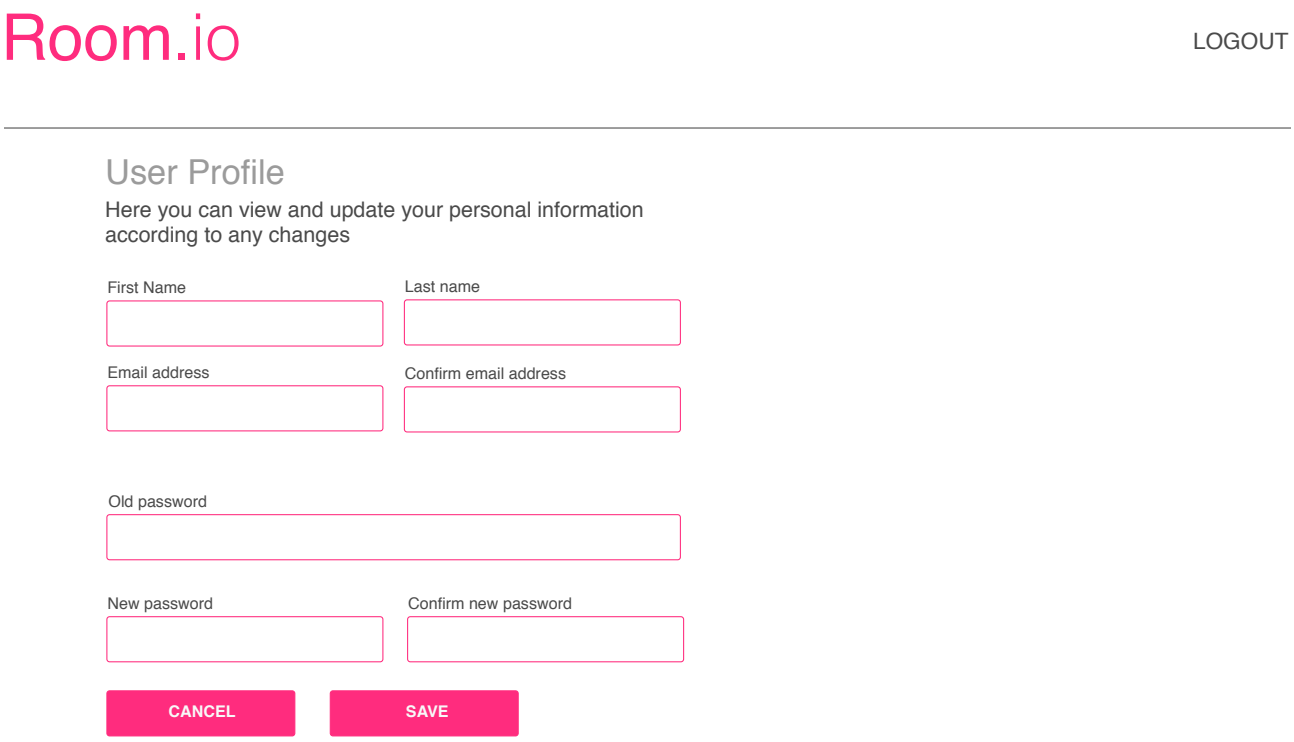

Figure 5.2 – Profile page.

5.2 Guest Views

5.2.1 *Search Request*

This is the page the Guest lands on after logging onto the system. Here they can search for a destination and have the option to indicate a time period for their stay, the number of people travelling as well as the number of rooms needed. Only the choice of a destination is mandatory; the other information can be edited on the detailed property view. The Guest can always return from any other page to this one by clicking on the *Room.io* logo.

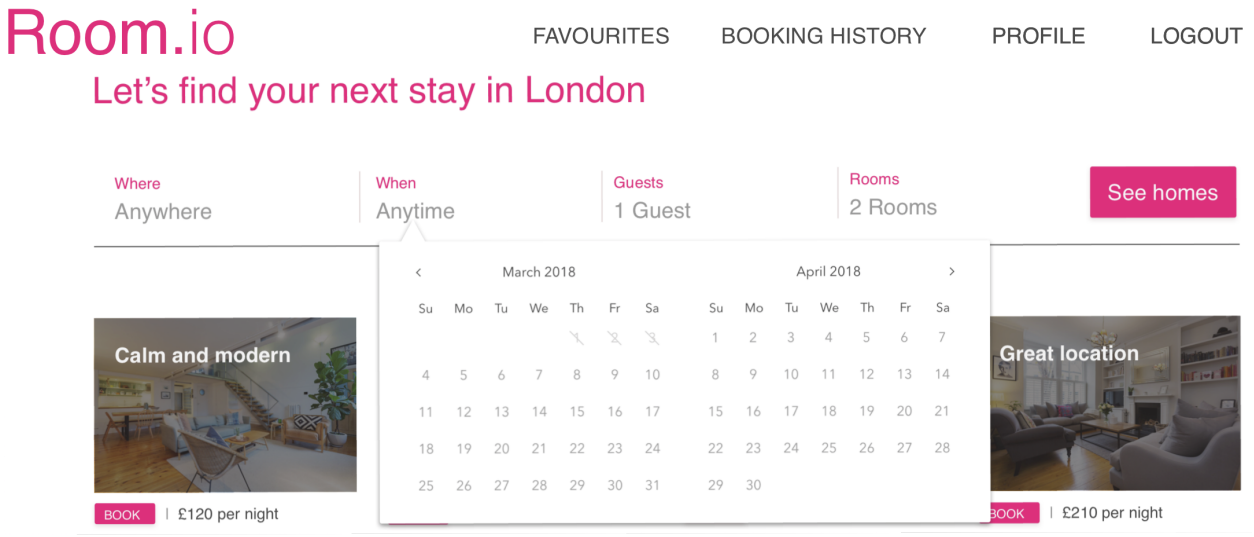

Figure 5.3 – Making a search request.

5.2.2 *Property List View*

After sending a search request, the Guest sees a map of the chosen city with tags showing the price per night at the location of a property adhering to the chosen requirements. Underneath the map, a list of these properties is shown. By hovering over a tag, the respective location is highlighted in the list. The Guest also has the option to filter this list by the number of available rooms, room policies and ratings.

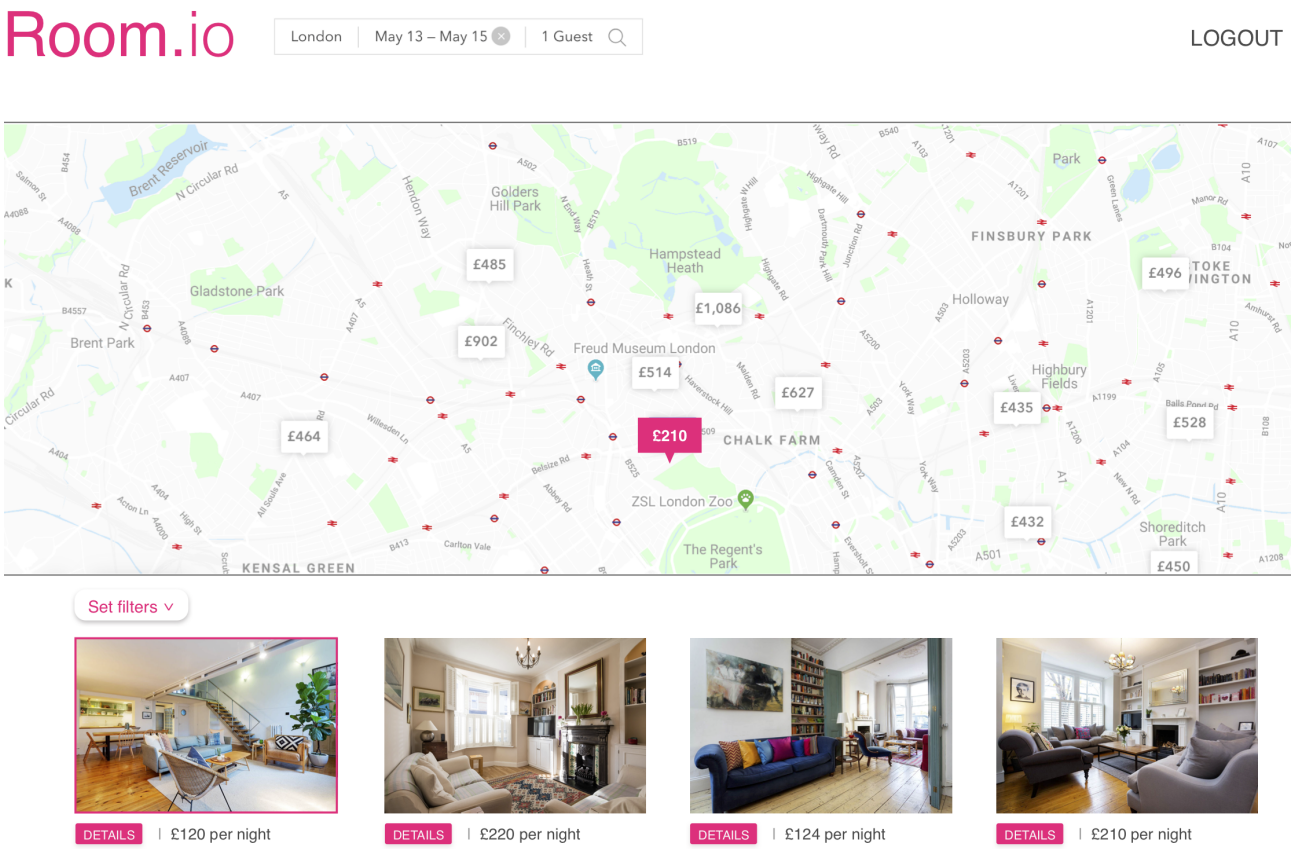

Figure 5.4 – Property list view.

5.2.3 *Detailed Property View*

After selecting a property, the Guest is able to see property pictures in a carousel as well as a detailed property description. Furthermore, the property's available rooms with a short description and policy attributes are depicted. When hovering over the star rating, a list of all given ratings for this room appears. The Guest is able to check the room's availability for a certain date and make a booking for a room. Additionally, they can add the property to their "favourites" list.

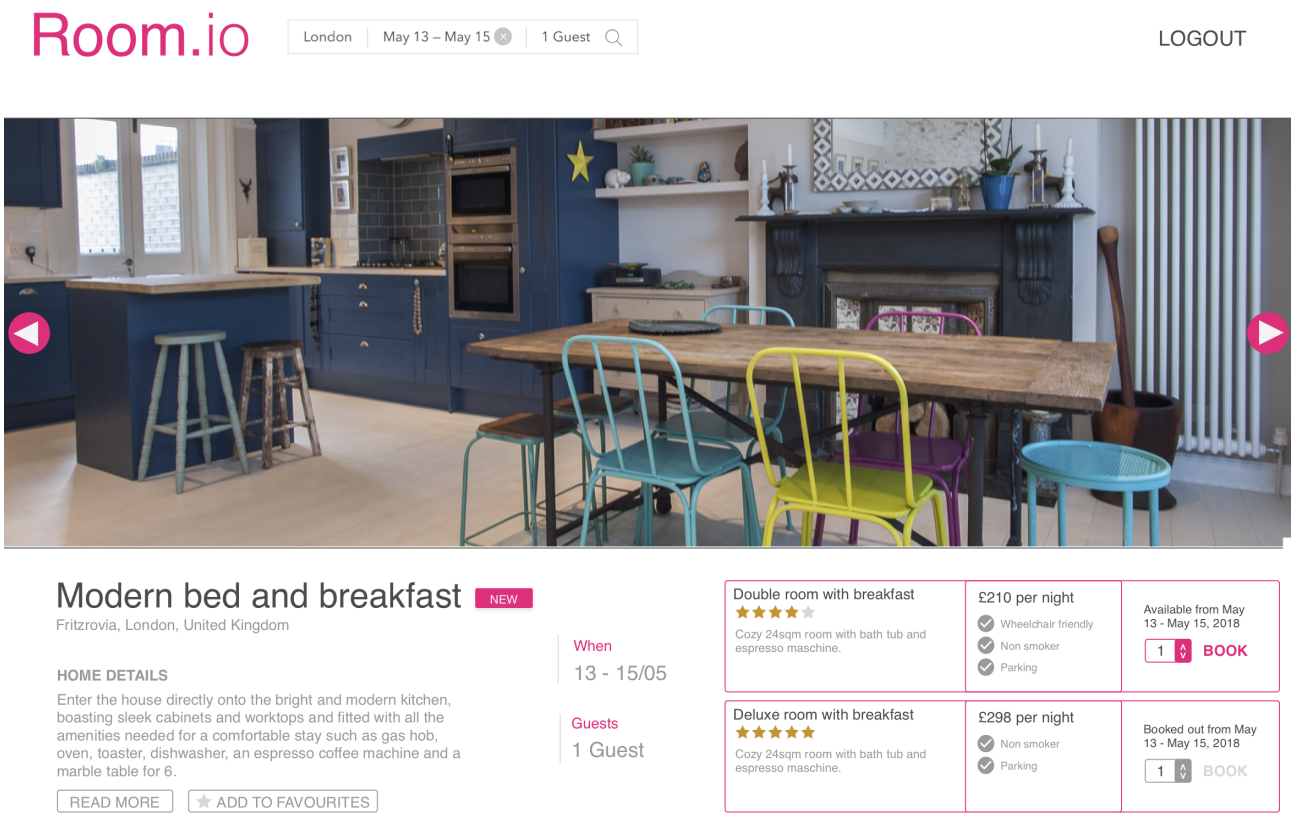

Figure 5.5 – Detailed property view.

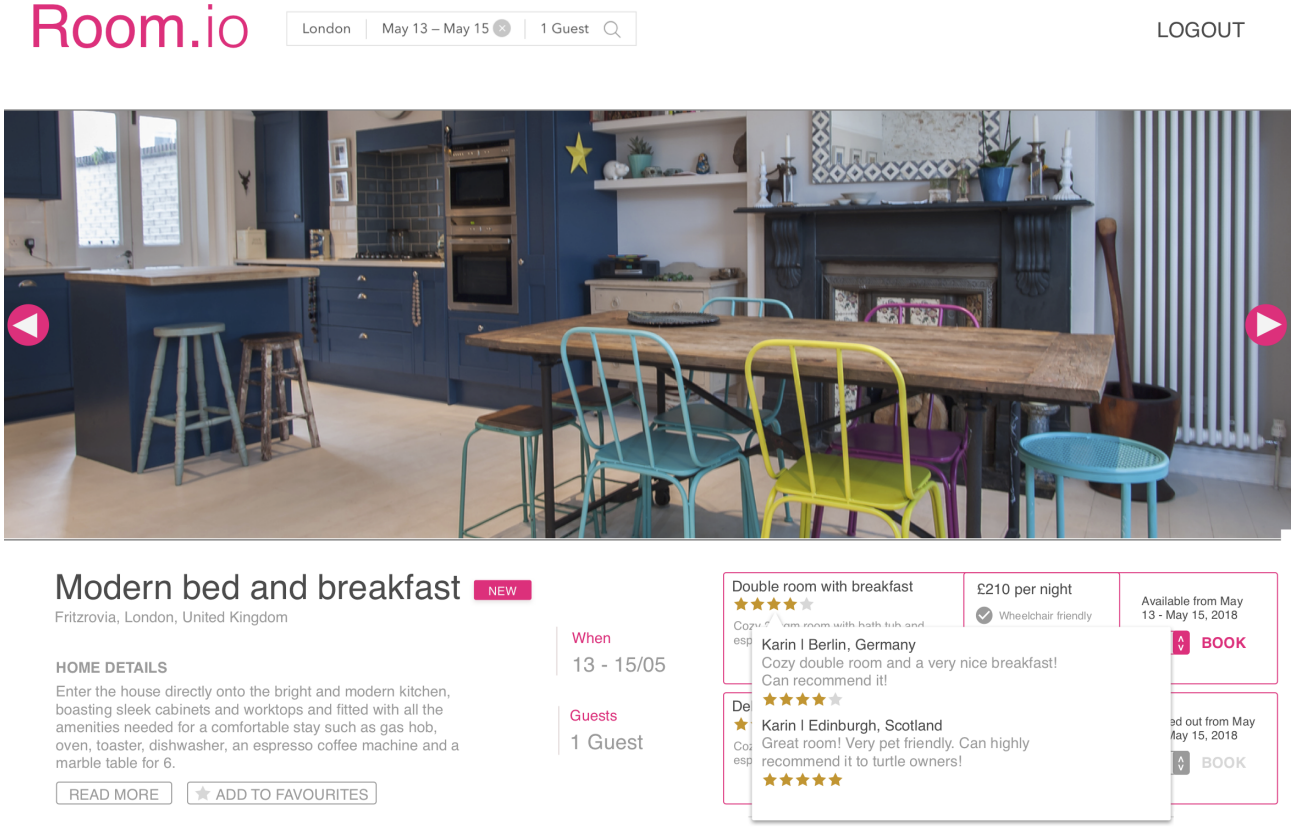

Figure 5.6 – Detailed property rating view.

5.2.4 *Favourites*

On the "favourites" list the Guest is able to save properties. By clicking on one of these properties, they are directed to the detailed property view. In this view, the Guest can also delete properties from the list by clicking on the "Remove from Favourites" button above each property excerpt.

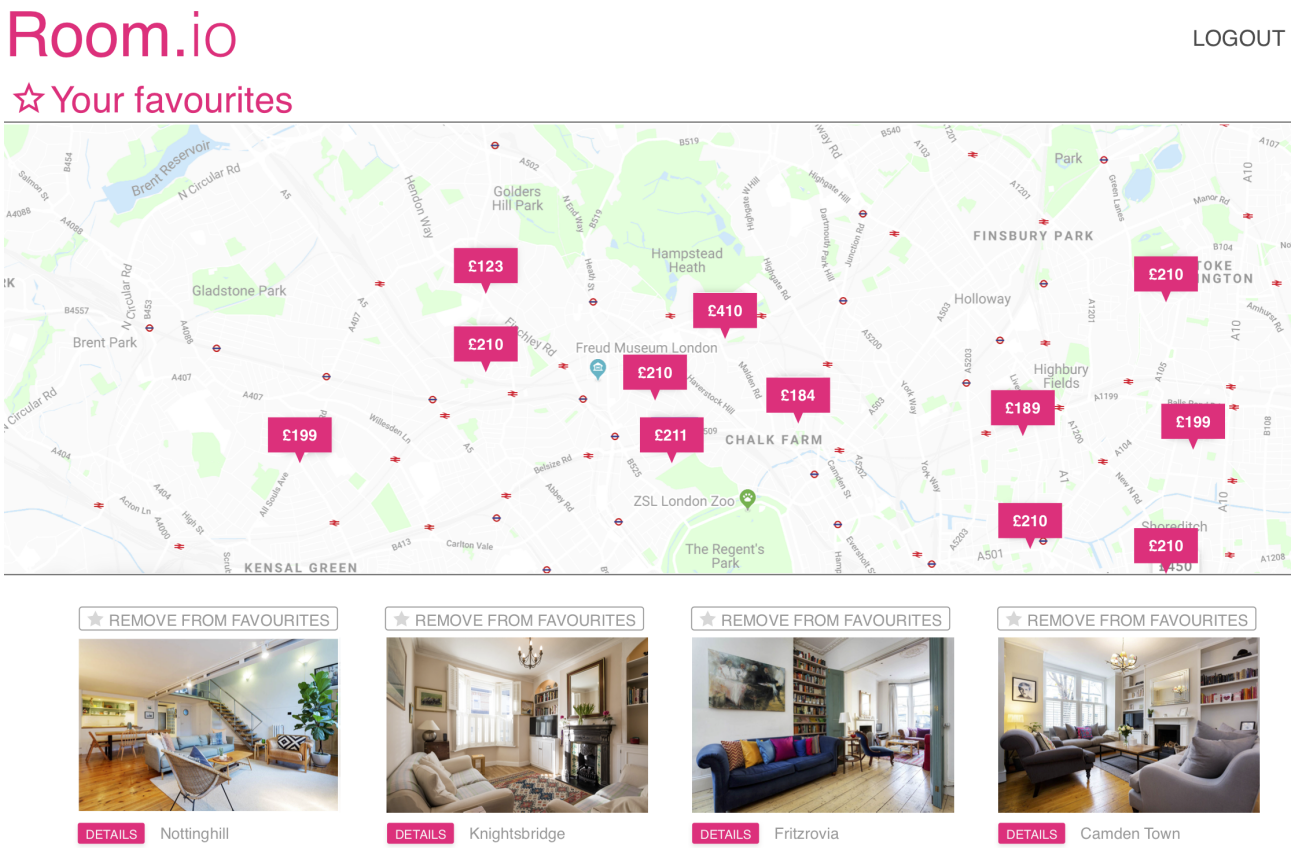

Figure 5.7 – Favourites view.

5.2.5 *Make a Booking*

After confirming the availability for a chosen room on the detailed property view, the Guest can make a booking for it. To do so, they simply need to indicate their personal details and in a second step fill in their credit card details. If their details are correct, they receive a booking confirmation. If they are not valid, a popup informs them about the incorrect details.

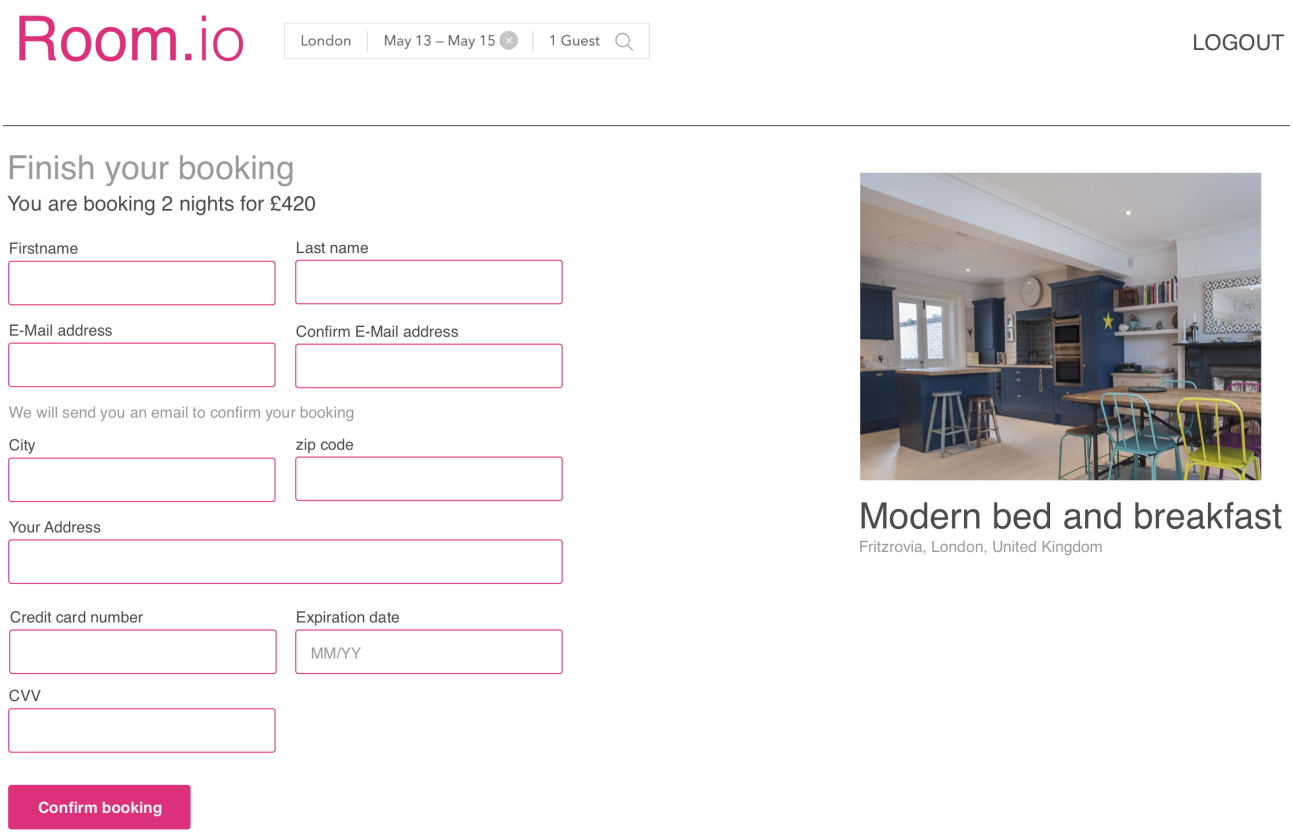

Figure 5.8 – Booking view.

5.2.6 *Booking History*

From the search site, the Guest can also access their booking history. Here they are able to delete future bookings as well as give a rating for the properties they already visited.

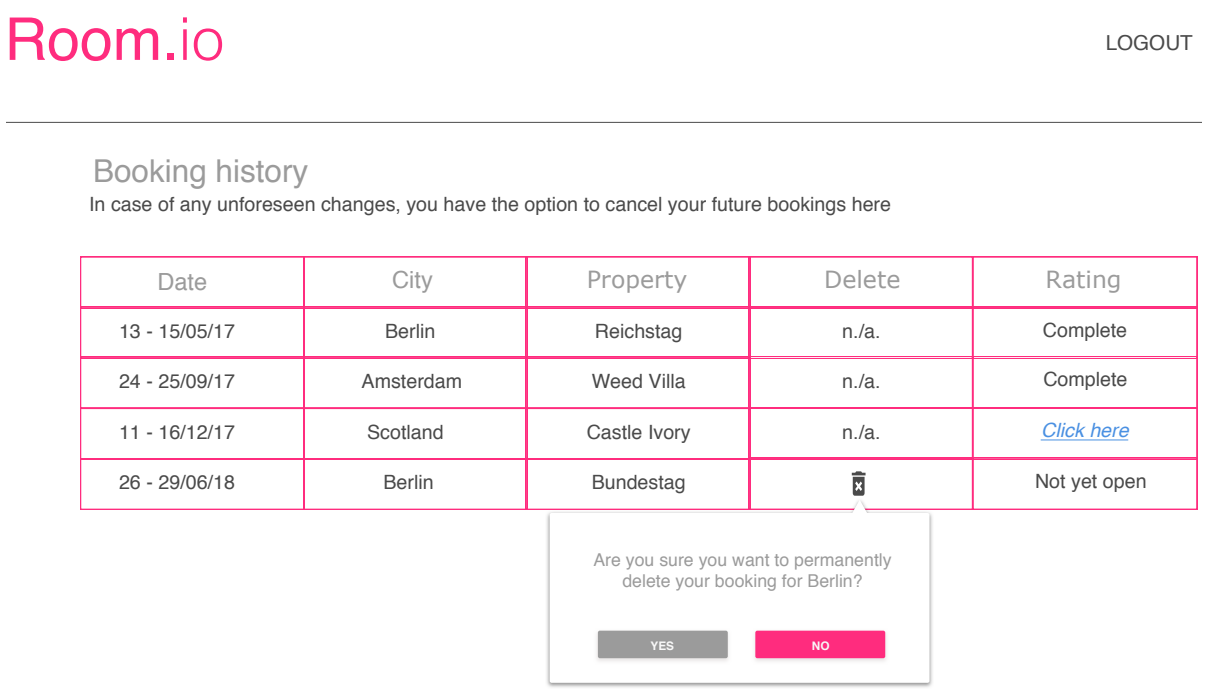

Figure 5.9 – Booking history.

Additionally, they can view their booking history and are able to delete future bookings from here.

5.2.7 *Give a Rating*

After visiting a property, the Guest is invited to leave a rating for the room they stayed at. To do so, they need to rank their stay by assigning stars in the range from 1 to 5 with 5 being the best. Optionally, they can also add a comment with a header to their review. They are later able to change this feedback by reviewing the property again.

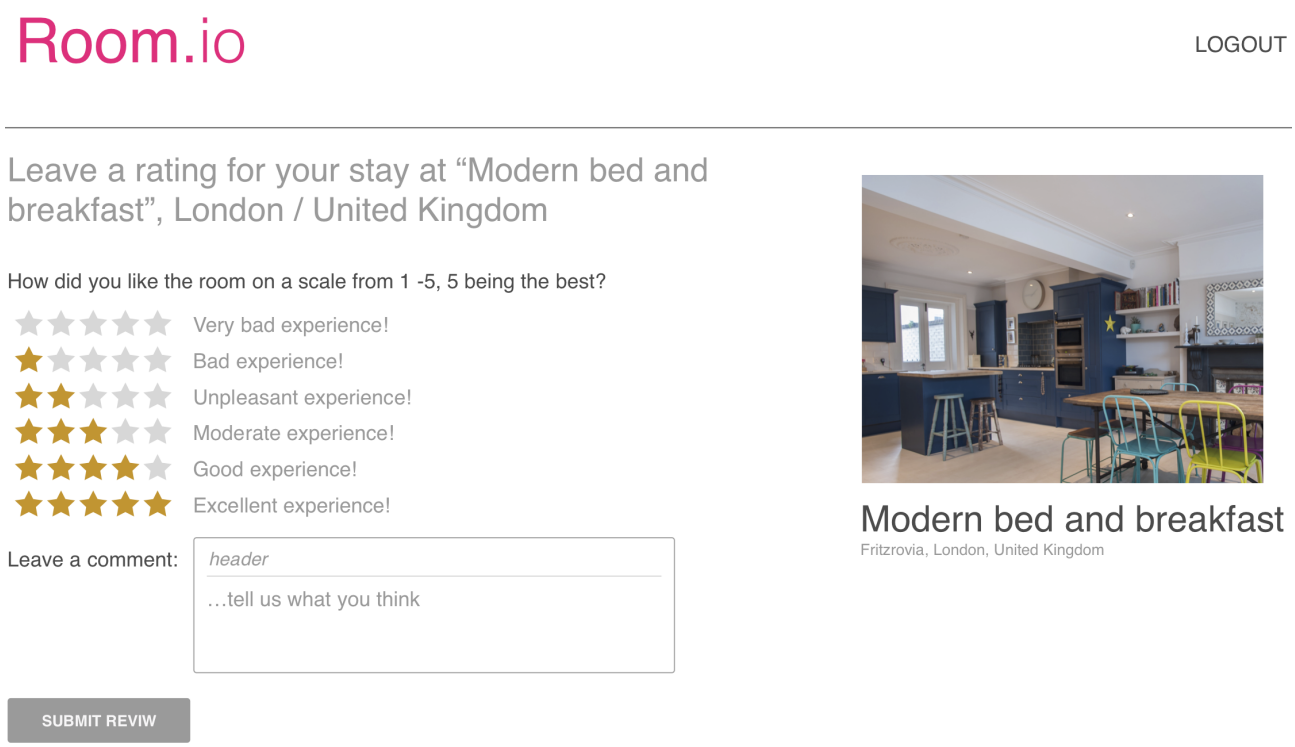

Figure 5.10 – Rating view.

5.3 HOST VIEWS

5.3.1 *Host Dashboard*

If the Host registered a property already, they see an overview of the rooms they currently host in their property together with the future bookings for these. From here they have the option to delete a room, see a preview or edit a room. If no property is registered so far, the Host has the option to add one.

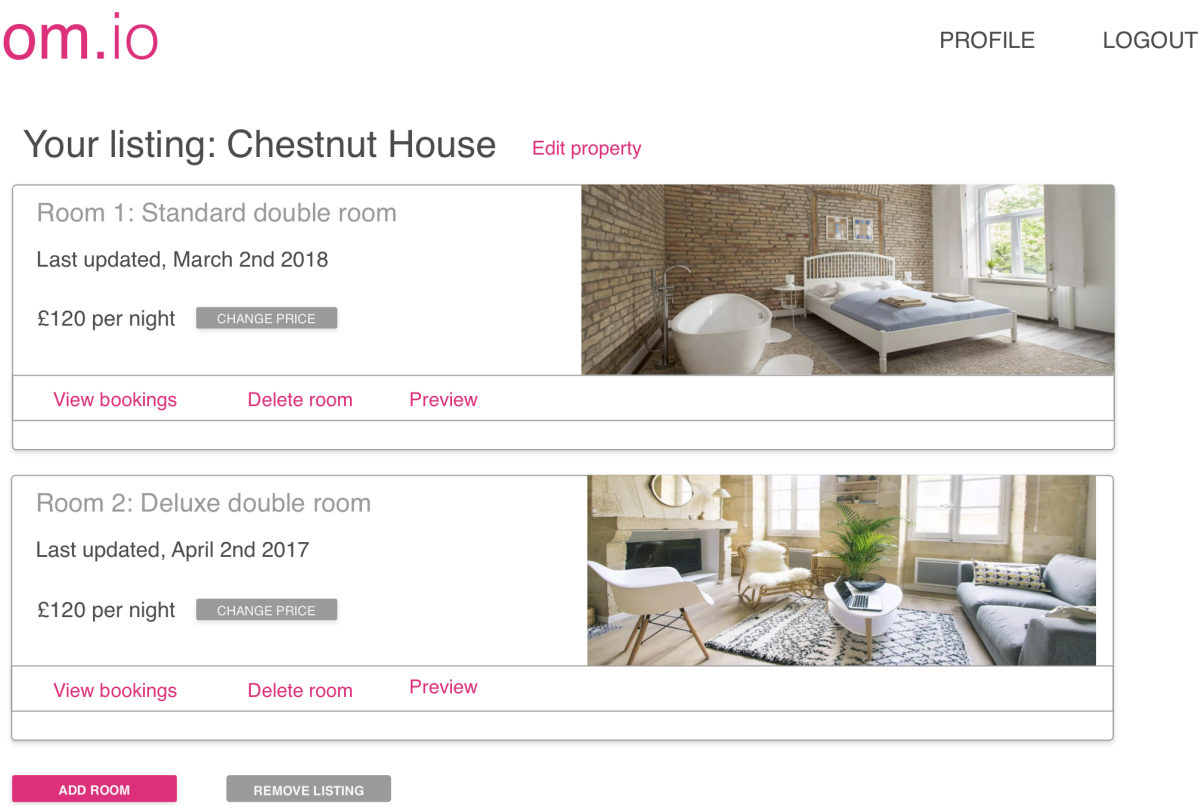

Figure 5.11 – Host dashboard.

5.3.2 *Register a Property*

The Host is only able to register one property. When doing so, they are first asked to provide a property description, an address, earliest check-in, latest check-out, and pictures. Then, they are asked to add a room by inserting a short description, a price for one or two guests, the number of beds the room provides and by selecting policy attributes from a list created by the Admin. Thereafter, they have the option to add the next room. Even though the steps are numbered, the User can return to any step by using the menu on the left side and determine the order of filling in their details by themself. The only requirement for submission, in the end, is that the input complies with all validations.

5.3.3 *Edit a Property*

When editing a property, the Host has the option to edit the property descriptions and details such as prices, upload new pictures or simply edit an existing room or add a new one.

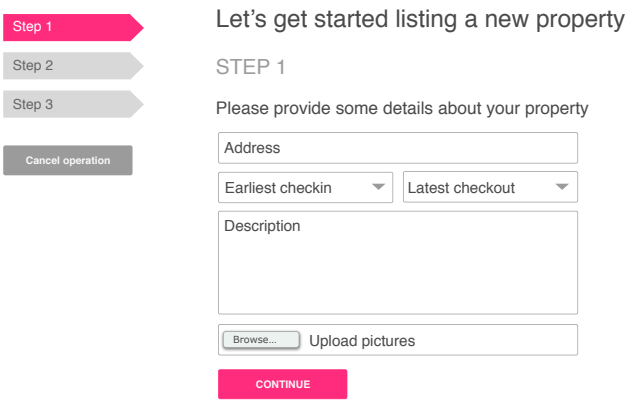

Figure 5.12 – Register a property (1/3).

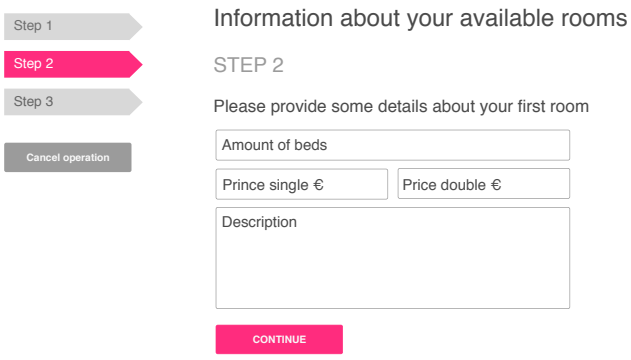

Figure 5.13 – Register a property (2/3).

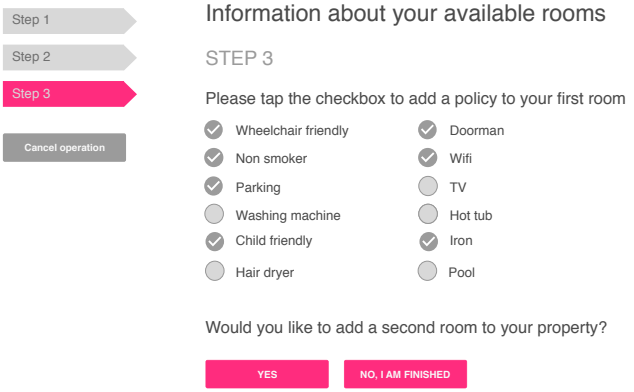

Figure 5.14 – Register a property (3/3).

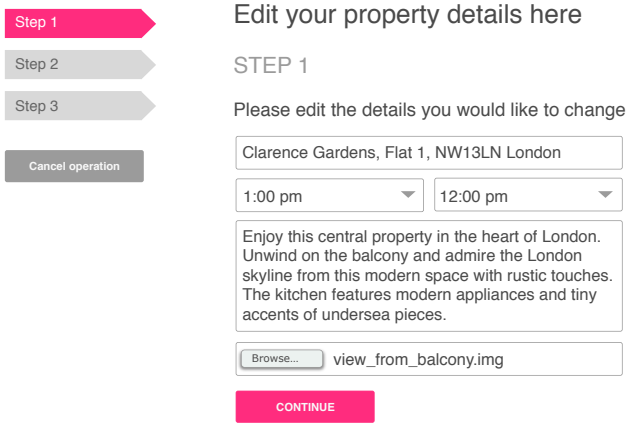

Figure 5.15 – Edit a property (1/3).

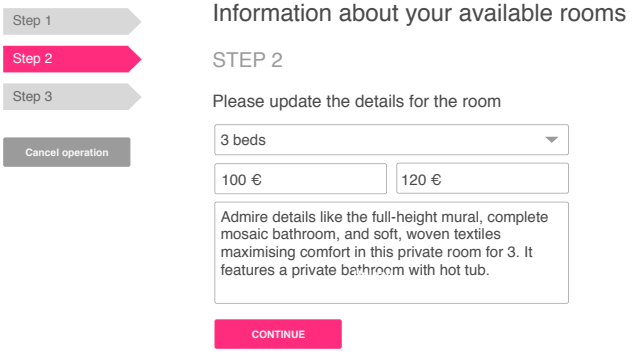

Figure 5.16 – Edit a property (2/3).

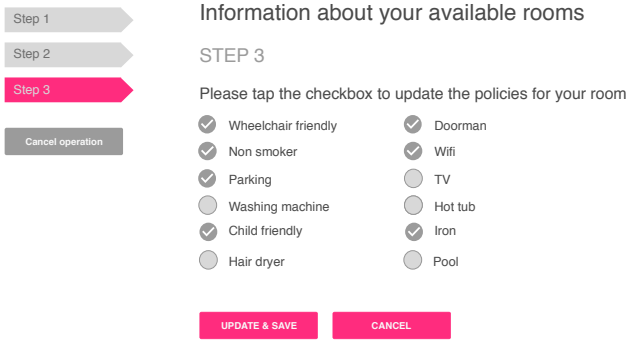

Figure 5.17 – Edit a property (3/3).

5.4 Admin Views

5.4.1 *The Admin's User Dashboard*

The Admin has three main responsibilities: managing the Users of the platform, creating the Policies for the rooms, and overseeing the Properties. After logging in, the Admin lands on a dashboard where a list of all platform Users is depicted. Here, they have the option to edit/add a user. A User is added through a simple popup view that appears when issuing the request to create one. After submitting the required information and

indicating the new User's Role, the new User receives an email to confirm their details.

When requesting to edit a User's personal details on the dashboard, the Admin can view the User's information and edit everything except for the email and password. The password is not visible. The Admin's information also appears on this list, enabling the Admin to edit their own personal details in this view as well.

A menu on the left side leads the Admin to the policy list or to the same search view the Guest sees when clicking on "Manage properties".

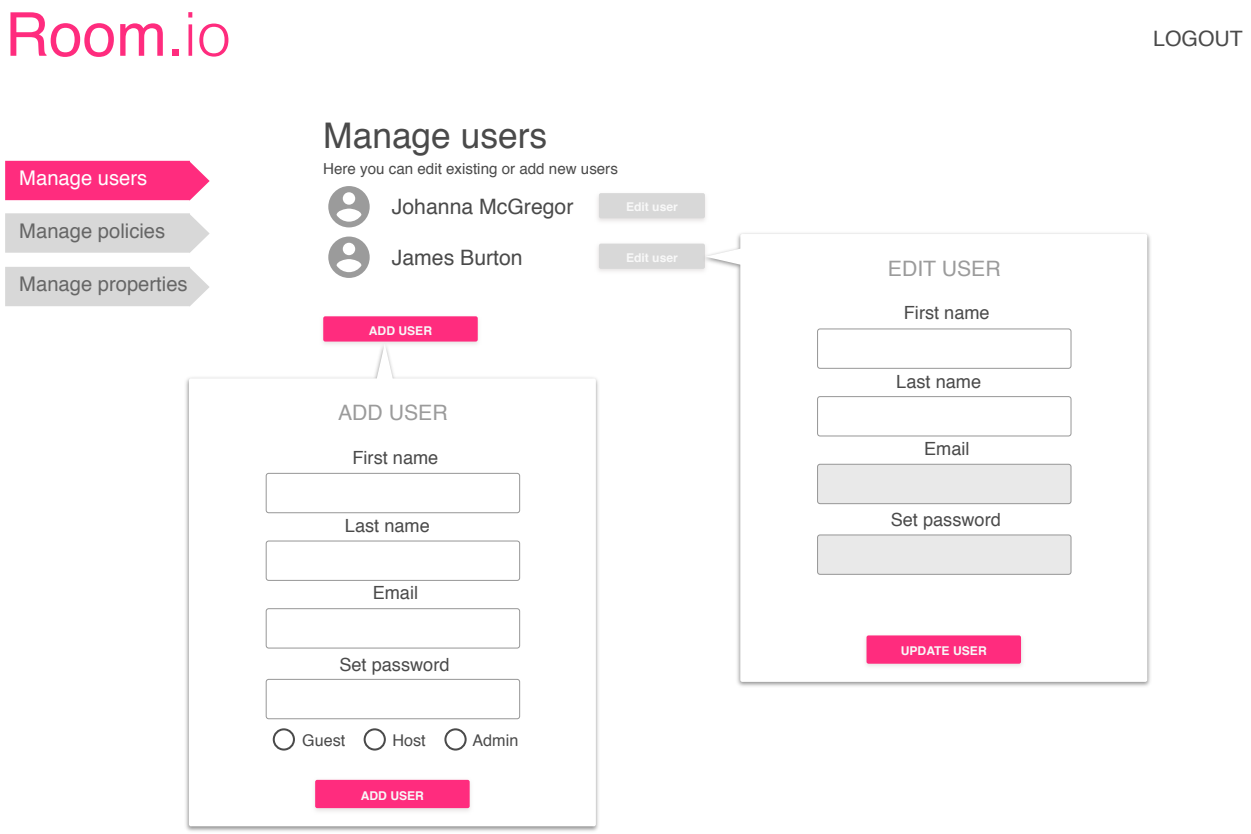

Figure 5.18 – Admin dashboard.

5.4.2 *Policy list*

The Admin is responsible for creating and editing a list of policies for rooms that can be selected by the Hosts when creating a property. Room policies can include house rules such as "no smoking" in addition to amenities and room properties such as "wheelchair friendly" and "concierge". On this page, the Admin has the possibility to delete or add new Policies to the list.

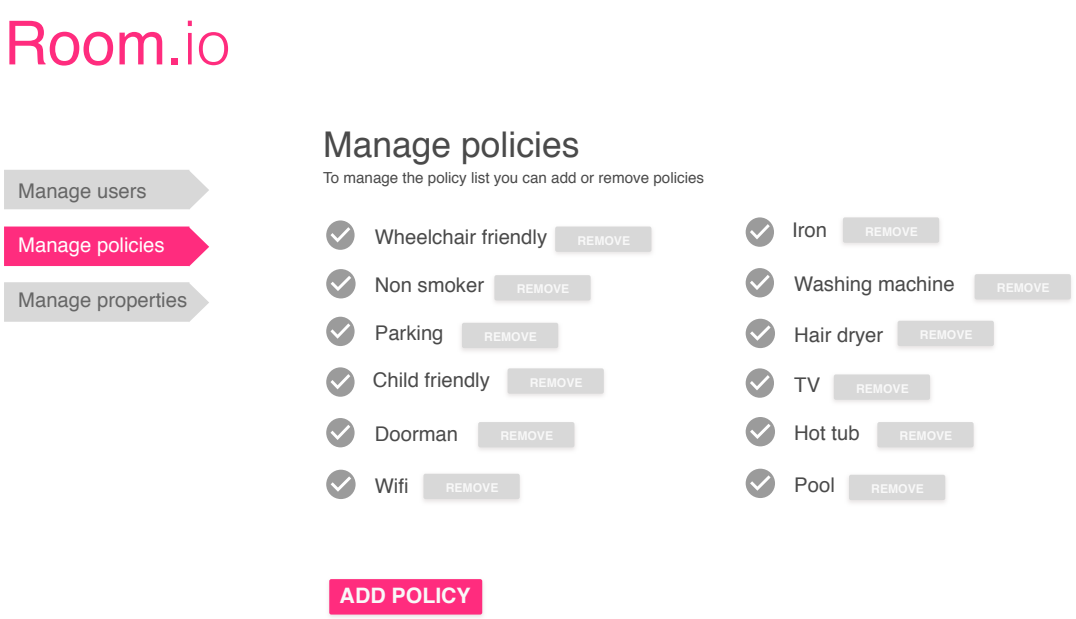

Figure 5.19 – Policy list.

Summary and Conclusions

chapter 6

6.1 summary

This report described the conceptual design of the B&B platform *Room.io*, from gathering initial requirements to defining use cases, specifying necessary sequences and classes up to laying out a plan for implementation and providing mock ups of the final product.

6.2 conclusions

This project neatly drew upon the group's knowledge and skills acquired in other course modules. A prior App Design project had introduced the group to human computer interaction which was applied in identifying the requirements and formulating the use cases for the proposed System. Both App Design and Database modules provided the background to anticipate how the service would be hosted, together with the likely front and back end functionality. The database module in particular provided the necessary insight into the requirements in terms of the likely database schema and how the Users would interact with this. Existing object-orientated programming knowledge helped hone the design in a way that would facilitate the implementation of the system. All of this background was invaluable in putting together the illustrations, tables and diagrams that formed the overall design. A great part of the success of the project, was, however, attributable to the team work involved. The group met as a whole on Friday mornings and Monday afternoons. Monday was the opportunity to divide up tasks for the coming few days and was at times followed by a conversation with Graham Roberts to clarify any points that had arisen over the weekend. On fridays, the group caught up and seized the opportunity to update their supervisor, Rae Harbird. Between these meetings, issues were discussed over Slack channel and in informal sub-team meetings. These sub-teams established the first draft of the requirements and use cases. The group as a whole refined these as they moved further into the analysis and design stages. At the conclusion of the project, a system had been designed that had been considered from many possible angles and that satisfied the requirements identified in the beginning. As a consequence, the group is confident that they laid the foundation for the robust and effective B&B Booking system that they set out to design.

6.3 who did what

Aleksi took on a valuable overview role for much of the project, helping to ensure coherence between different diagrams and correct application of the theory in this context. Other members of the group assisted Aleksi in reviewing output through git hub and Slack postings. Each week there was considerable Slack traffic between all group members with comments on consistency, suggested revisions and updates on tasks. Zaid, with a Software Engineering background, was a useful sounding board when there were queries about the

design features for our system. This was very much a whole team project. As indicated above, the formulation of the requirements, use cases and the diagrams involved all group members, although sub-teams initially focused on specific actors in the process. The final outcome was ultimately down to the fact that each team member was fully committed to the task and to delivering a high quality project. For an overview on the individual responsibilities of each group member, please see Table 6.1.

Aleksi Anttila	Overall cohesiveness and consistency co-ordination; primary responsibility for use cases, sequence diagrams and glossary (setting templates, keeping track of changes and finalising); set up report outline; LaTeXhelper
Michael Aring	Requirements list, use cases, initial sub-group for Host requirements, drafting related sections of report, project management, analysis class diagram, sequence diagram, video production
Kai Klasen	Use cases, analysis class diagram, sequence and activity diagrams initial sub-group for Guest requirements etc, drafting relating sections of report, video narrator
Lorenz Luboldt	Requirements list, use cases, activity diagrams, class diagrams, initial sub-group for Host requirements/use cases, state machine diagram, drafting related sections of report, video production
Christina Kronser	Preparation of meeting agendas, requirements lists, use case tables, class diagrams, sequence and activity diagrams, mockups, initial sub group for Guest requirements/use cases, state machine diagram, video production
Philip Spencer	Requirements list, use cases, activity diagrams, initial sub-group for Host requirements/use cases, drafting scope and vision document, draft of report abstract and conclusion, draft video script, meeting liaison.
Paul Venhaus	Overall monitoring of requirements, use cases, introduction to report including domain model and gantt chart, component and deployment diagrams, initial sub-group focusing on Guest requirements/use cases, video production, and LaTeXsuperuser
Zaid Al Lahham	Analysis and Design class diagram, requirements list, use cases, component and deployment diagrams, initial sub-group to focus on Admin requirements, video production, and LaTeXsuperuser

Table 6.1 – Team Contribution.

Glossary

APPENDIX A

Admin—The administrator of the system is in charge to monitor and ensure the System's correct operation.

Admin's Dashboard—The page that an Admin sees when they log in. It should allow the Admin to gain access to a list of Users, a list of Properties, and a list of Policies, along with related functionality.

Booking—A Guest's explicit confirmation to stay at a Property's room within an agreed-upon time frame (the representation of such on the System).

- **Cancellable Booking**—A Booking that is not already Cancelled and whose check-in date is more than two days in the future.
- **Cancelled Booking**—A Booking cancelled by either the Guest who initially booked it, or by the Host for whose Property the Booking was for. The Booking is still on the System, but it is flagged as cancelled in the User/Property Bookings, and does not factor into the availability of Rooms.
- **Confirmed Booking**—A Booking that is not Cancelled.
- **Future Booking**—A Booking whose check-out date is in the future.
- **Past Booking**—A Booking whose check-out date is in the past.
- **Property Bookings**—The Bookings made for a specific Property.
- **User Bookings**—The Bookings made by a specific Guest.

Booking Details—The details associated with each Booking. These include the check-in date (start date), check-out date (end date), price (amount paid), status (Confirmed or Cancelled), the User that made the Booking, the User's Payment Details, and the Room and Property the Booking is for.

- **Core Booking Details**—A set of Booking Details excluding the Payment Details and status.
- **Correct Core Booking Details**—A set of Core Booking Details that is complete (no details are missing) and correctly specified.
- **Initial Booking Details**—Check-in date, check-out date, Room(s) and Property.
- **Locked Initial Booking Details**—A set of Initial Booking Details such that the System is temporarily treating the Rooms selected as unavailable for the selected check-in date and check-out date.
- **Possible Initial Booking Details**—A set of Initial Booking Details such that no detail is missing and the Rooms selected are still available for the selected check-in date and check-out date.

Guest—A User that is in the process of, or, that has successfully completed a Booking in

the B&B Booking System.

Home/Search Page—The page that a Guest sees when they log in and which allows the Guest to search for Properties.

Host—A User that is in the process of, or has successfully listed a Property and is able to receive Guests via the B&B Booking System.

Host Dashboard—The page that a Host sees when they log in. Should allow the Host to issue requests to register a Property or edit an already Registered Property.

Landing Page—What a user who has not logged into the System should see. Should allow for users logging in and registering.

Payment Details—Associated with a specific User. These include the name on the User's debit/credit card, the card number, its expiration date, and its security number. Only ever displayed to the User associated with them.

Policy—The representation of some B&B Room detail or rule on the System (some examples: "2-person", "non-smoking", "pet-friendly"). Associated with a Room. A Room can have multiple Policies associated with it.

Policy Name—The only detail of a Policy. No two policies may have the same Name.

Property—The representation of a B&B on the System. Associated with a Host. A Host may only have one Property associated with their account. Each Property may have multiple Rooms associated with it.

Property Details—The information associated with each Property that is shown to all Users. Includes the Property description, its address, pictures, arrival times, departure times, ratings, Rooms, Policies associated with those Rooms, and Room availability data for different dates.

- **Core Property Details**—The Property Details that a Host needs to specify when registering a Property. These include the description, the address, pictures, arrival times, departure times, Rooms, and Policies associated with those Rooms.
- **Correct Property Details**—A set of Core Property Details such that is complete (no details are missing) and correctly specified.
- **Host-Specific Property Details**—Includes the usual Property Details, along with statistics about the number of rooms available overall, etc.

Property Search Conditions—The conditions by which Users can filter Properties. These include location, availability for a certain number of Guests for a specific set of dates, average stars, price, arrival times, departure times, and the presence of Rooms with specific Policies.

Property Sort Criteria—The conditions by which Users can order Properties. These include proximity to a location, average stars, and price.

Rating—Evaluation of a Booking that a Guest can undertake upon completion of a stay at a B&B. A Rating is uniquely linked with a Booking, a Property, and a Guest. It consists of a title, a comment and a Rating value.

Rating Details—The title, comment and Rating value (from one star to five) of a Rating.

- **Correct Rating Details** A set of Rating Details that is complete (no details are missing) and correctly specified.

Property—The representation of a B&B room on the System. Associated with a Property. May have multiple Policies associated with it.

System—The website and all its related technical components.

User—A User is an Admin, Host, Guest or Visitor.

- **Unverified User**—A User banned by an Admin.
- **Unverified User**—A User who has not yet confirmed that their provided email address belongs to them cannot log into the System.
- **Verified User**—A User who has confirmed that their provided email address belongs to them is able to log in.

User Details—The details held for each User: email address, password, name, date of birth, Role, passport number, address, gender, status (Verified, Unverified or Banned) and Payment Details. No two users may have the same email address. Details that are irrelevant for a User with a specific Role are represented by empty strings or special characters—an Admin, for instance, need not have a passport number.[1]

- **Core User Details**—The User Details for a User, excluding status and Role.
- **Correct Core User Details**—A set of Core User Details that is complete (no details are missing) and correctly specified (the email is a valid email address, the passport number is a valid passport number, etc., and and the email address is unique.) These constraints are different for Users with different Roles: an Admin, for instance, need not specify a passport number.

Visitor—Someone using the System without having created an account yet.

[1] To explain the rationale for this a little further: the number of Admins on the System will be very low, so we decided that storing a uniform set of data for each User with the occasional missing details represented by special characters would be more efficient than creating a specialized Admin class.

B.1 Functional Requirements

B.1.1 *User*

Description	The System shall allow for the registration of new Users.		
ID	R-U-C-1	Use Case	U-C-1 (Table 3.2)
Type	Functional	Priority	Must Have
Actor(s)	Visitor	Entity	User

Description	The System shall allow the creation of new Users.		
ID	R-U-C-2	Use Case	U-C-2 (Table C.3)
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	User

Description	The System shall display a User's own profile.		
ID	R-U-V-1	Use Case	U-E (Table C.4)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Host	Entity	User

Description	The System shall allow an Admin to view any User profile.		
ID	R-U-V-2	Use Case	U-E (Table C.4)
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	User

Description	The System shall display a list of User profiles.		
ID	R-U-L-1		
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	User

Description	The System shall allow an Admin to filter a list of Users by certain criteria.		
ID	R-U-L-2		
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	User

Description	The System shall allow a User to edit their own Account Details.		
ID	R-U-E-1	Use Case	U-E (Table C.4)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Host	Entity	User

Description	The System shall allow an Admin to edit any User's Account Details.		
ID	R-U-E-2	Use Case	U-E (Table C.4)
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	User

Description	The System shall allow an Admin to ban a User from using the System.		
ID	R-U-O-1	Use Case	U-E (Table C.4)
Type	Functional	Priority	Could Have
Actor(s)	Admin	Entity	User

B.1.2 *Property*

Description	The System shall allow a Host to register a new Property.		
ID	R-P-C-1	Use Case	P-C (Table 3.3)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Property

Description	The System shall display information about a Property.		
ID	R-P-V-1	Use Case	P-V (Table 3.5)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Host, Admin	Entity	Property

Description	The System shall display a list of Properties.		
ID	R-P-L-1	Use Case	P-L (Table 3.4)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Admin	Entity	Property

Description	The System shall allow a User to filter a list of Properties by certain criteria.		
ID	R-P-L-2	Use Case	P-L (Table 3.4)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Admin	Entity	Property

Description	The System shall allow a Host to edit their own Property.		
ID	R-P-E-1	Use Case	P-E (Table C.5)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Property

Description	The System shall allow a Host to delete their own Property from the System.		
ID	R-P-D-1	Use Case	P-D (Table C.6)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Property

Description	The System shall allow an Admin to delete Properties from the System.		
ID	R-P-D-2	Use Case	P-D (Table C.6)
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	Property

Description	The System shall allow a User to share Property Details with their contacts.		
ID	R-P-O-1		
Type	Functional	Priority	Could Have
Actor(s)	Guest	Entity	Property

B.1.3 *Rooms*

Description	The System shall allow the addition of Rooms to a Property.		
ID	R-Ro-C-1	Use Case	P-E (Table C.5)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Room

Description	The System shall allow a Host to edit their Property's Rooms.		
ID	R-Ro-E-1	Use Case	P-E (Table C.5)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Room

Description	The System shall allow a Host to delete their Property's Rooms.		
ID	R-Ro-D-1	Use Case	P-E (Table C.5)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Room

Description	The System shall display all available Rooms in a Property for a given date.		
ID	R-Ro-O-1	Use Case	P-V (Table 3.5)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Admin, Host	Entity	Room

B.1.4 *Bookings*

Description	The System shall allow the creation of Bookings for a Property.		
ID	R-B-C-1	Use Case	B-C (Table 3.6)
Type	Functional	Priority	Must Have
Actor(s)	Guest	Entity	Booking

Description	The System shall allow a Guest to view their own Booking.		
ID	R-B-V-1	Use Case	B-L-1 (Table 3.8)
Type	Functional	Priority	Must Have
Actor(s)	Guest	Entity	Booking

Description	The System shall allow a Host to view a Booking on their Property.		
ID	R-B-V-2	Use Case	B-L-2 (Table C.7)
Type	Functional	Priority	Must Have
Actor(s)	Host	Entity	Booking

Description	The System shall allow an Admin to view any Booking.		
ID	R-B-V-3	Use Case	B-L-1 & B-L-2 (Table 3.8 & C.7)
Type	Functional	Priority	Must Have
Actor(s)	Admin	Entity	Booking

Description	The System shall display a Guest or Property's Booking history.		
ID	R-B-L-1	Use Case	B-L-1 & B-L-2 (Table 3.8 & C.7)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Admin	Entity	Booking

Description	The System shall allow the cancellation of a Booking.		
ID	R-B-O-1	Use Case	B-D (Table 3.9)
Type	Functional	Priority	Must Have
Actor(s)	Guest, Admin, Host	Entity	Booking

Description	The System shall send email confirmations to confirm a new Booking.		
ID	R-B-O-2	Use Case	B-C (Table 3.6)
Type	Functional	Priority	Could Have
Actor(s)	Guest, Host	Entity	Booking

B.1.5 *Policies*

Description	The System shall allow the creation of Policies.		
ID	R-Po-C-1	Use Case	Po-C (Table 3.10)
Type	Functional	Priority	Should Have
Actor(s)	Admin	Entity	Policy

Description	The System shall display a list of existing Policies.		
ID	R-Po-L-1		
Type	Functional	Priority	Should Have
Actor(s)	Admin	Entity	Policy

Description	The System shall allow the deletion of Policies.		
ID	R-Po-D-1	Use Case	Po-D (Table 3.11)
Type	Functional	Priority	Should Have
Actor(s)	Admin	Entity	Policy

B.1.6 *Ratings*

Description	The System shall allow Guests to give Ratings to Properties they have visited.		
ID	R-R-C-1	Use Case	R-C (Table 3.12)
Type	Functional	Priority	Should Have
Actor(s)	Guest	Entity	Rating

Description	The System shall display a list of Ratings for a given Property.		
ID	R-R-L-1	Use Case	P-V (Table 3.5)
Type	Functional	Priority	Should Have
Actor(s)	Guest, Admin, Host	Entity	Rating

Description	The System shall allow the deletion of Ratings for a given Property.		
ID	R-R-D-1		
Type	Functional	Priority	Could Have
Actor(s)	Guest, Admin, Host	Entity	Rating

B.1.7 *Favourites*

Description	The System shall allow a Guest to add Properties to a list of Favourites.		
ID	R-F-C-1	Use Case	F-C (Table 3.13)
Type	Functional	Priority	Could Have
Actor(s)	Guest	Entity	Favourites

Description	The System shall allow a Guest to view a list of their favourite Properties.		
ID	R-F-L-1	Use Case	F-L (Table 3.14)
Type	Functional	Priority	Could Have
Actor(s)	Guest	Entity	Favourites

Description	The System shall allow a Guest to remove a Property from their list of favourite Properties.		
ID	R-F-D-1	Use Case	F-D (Table C.8)
Type	Functional	Priority	Could Have
Actor(s)	Guest	Entity	Favourites

B.2 NON-FUNCTIONAL REQUIREMENTS

Description	The System shall be written in a commonly used programming language.		
ID	R-N-C-1	Category	Code
Type	Non-functional	Priority	Should Have

Description	The System shall use a relational database system.		
ID	R-N-C-2	Category	Code
Type	Non-functional	Priority	Should Have

Description	The System shall be able to backup information.		
ID	R-N-R-1	Category	Reliability
Type	Non-functional	Priority	Could Have

Description	The System shall not lose any data.		
ID	R-N-R-2	Category	Reliability
Type	Non-functional	Priority	Must Have

Description	The System shall have a high uptime.		
ID	R-N-R-3	Category	Reliability
Type	Non-functional	Priority	Must Have

Description	The System shall be well documented.		
ID	R-N-Su-1	Category	Supportability
Type	Non-functional	Priority	Should Have

Description	The System shall provide helpful error messages.		
ID	R-N-Su-2	Category	Supportability
Type	Non-functional	Priority	Should Have

Description	The System shall store customer information securely.		
ID	R-N-Se-1	Category	Security
Type	Non-functional	Priority	Must Have

Description	The System shall store account details securely.		
ID	R-N-Se-2	Category	Security
Type	Non-functional	Priority	Must Have

C.1 General

The following processes are not full use cases as they are thought of in this report (see Chapter 3, but we have described them in a similar fashion to our use cases for completeness.

Use case snippet	Log in
Description	An Admin, Guest or Host logs in to the System.
Primary Actor	Admin/Guest/Host (henceforth, "the User")
Secondary Actor(s)	None
Preconditions	The User is Verified, not Banned, and has not logged in yet.
Main Flow	1. The use case snippet starts when the User specifies their email address and password. 2. The User submits their email address and password. 3. The System checks if the email address and password are part of some User's User Details and whether said User is Verified and not Banned. a. If the email and password are part of some Verified, non-Banned User's User Details: the System allows the User to log in. Go to 4. b. (END)If not: the System displays the Landing Page, along with a notification about the issue encountered. 4. (END) ■ For Guest: the System displays the Guest's Home/Search page. ■ For Admin: the System displays the Admin Dashboard. ■ For Host: the System displays the Host Dashboard.
Postconditions	■ If 3a taken: the User is logged in and: ◇ For Guest: the System displays the Guest's Home/Search page. ◇ For Admin: the System displays the Admin Dashboard. ◇ For Host: the System displays the Host Dashboard. ■ If 3b taken: the System displays the Landing Page, along with a notification about the login issue encountered.

Table C.1 – Use Case Snippet 1: Log in.

Use case snippet	Log out
Description	An Admin, Guest or Host logs out of the System.
Primary Actor	Admin/Guest/Host (henceforth, "the User")
Secondary Actor(s)	None
Preconditions	None (the User is logged in)
Main Flow	1. The use case snippet starts when the User issues a request to log out. 2. (END)The System allows the User to log out, and displays the Landing Page.
Postconditions	The User is logged out and the System displays the Landing Page.

Table C.2 – Use Case Snippet 2: Log out.

C.2 USERS

ID	U-C-2
Use case	Create User
Description	An Admin creates a new User.
Primary Actor	Admin
Secondary Actor(s)	None
Preconditions	The Admin is viewing their Dashboard.
Main Flow	1. The use case starts when the Admin issues a request to create a User. 2. The System provides the Admin with a means to fill in the User Details for the new User. 3. a. If the Admin provides the User Details: go to 4. b. If the Admin cancels the operation: go to 6. 4. The Admin submits the User Details. 5. The System checks the User Details. a. If the User Details are Correct: the System creates a new User based on the details provided. b. If the User Details are not Correct: the System displays a notification about the details causing the problem. Return to 2. 6. (END) The System displays the Admin's Dashboard.
Postconditions	The System displays the Admin's Dashboard and: ■ If 3a. and 5a. taken: a User has been created. The User details correspond to those specified during the creation process.

Table C.3 – Use Case U-C-2: Create User.

Note that the above is mainly for creating new Admin accounts. An existing Admin would first create the account, and the new Admin can then change their password.

ID	U-E
Use case	Edit User
Description	A Guest or Host edits their own Core User Details, or an Admin edits any User's User Details. (Henceforth, we will refer to the Primary Actor as "User", and to the User being acted upon as "Target User".)
Primary Actor	Guest/Admin/Host
Secondary Actor(s)	None
Preconditions	■ Guest/Host: None. ■ Admin: The Admin is viewing their Dashboard.
Main Flow	1. The use case starts when the User issues a request to view and edit the Target User's User Details. 2. The System displays the Target User's User Details, and provides the User with a means to edit these details. 3. a. If the User edits the User Details, or leaves them unchanged: go to 4. b. If the User cancels the operation: go to 6. 4. The User submits the User Details. 5. The System checks the User Details. a. If the User Details are Correct: the System updates the Target User based on the details provided. b. If the User Details are not Correct: the System displays a notification about the details causing the problem. Return to 2. 6. (END) The System displays the Target User's User Details.
Postconditions	The System displays the Target User's User Details and: ■ If 3a. and 5a. taken: the Target User has been updated with the details specified during the editing process.

Table C.4 – Use Case U-E: Edit User.

Note the following caveats to the above: a User's password is never displayed, an Admin cannot edit a User's password, and an Admin cannot view or edit any User's Payment Details.

C.3 PROPERTIES

ID	P-E
Use case	Edit Property
Description	The Host edits their Property's Core Property Details.
Primary Actor	Host
Secondary Actor(s)	None
Preconditions	The Host has a Property registered on their account.
Main Flow	1. The use case starts when the Host issues a request to edit their Property's Core Property Details. 2. The System provides the Host means to edit the Core Property Details 3. a. If the Host edits the Core Property Details, or leaves them unchanged: go to 4. b. If the Host cancels the operation: go to 6. 4. The Host submits the Core Property Details. 5. The System checks the Core Property Details. a. If the Core Property Details are Correct: the System updates the Property based on the details provided. b. If the Core Property Details are not Correct: the System displays a notification about the details causing the problem. Return to 2. 6. (END) The System displays the Property Details for the Property.
Postconditions	The System displays the Property's Property Details and: ■ If 3a. and 5a. taken: the Property's Core Property Details have been updated with the details specified during the editing process. The System displays the Property's Property Details.

TABLE C.5 – Use Case P-E: Edit Property.

ID	P-D
Use case	Delete Property
Description	A Host or an Admin (henceforth, "the User") deletes a Property and all associated entities (Bookings, Ratings and Favourites entries).
Primary Actor	Host/Admin
Secondary Actor(s)	None
Preconditions	■ Host: The Host has a Property registered on their account. ■ Admin: The Adming is viewing the Property Details for the Property to be deleted.
Main Flow	1. The use case starts when the User issues a request to delete the Property. 2. The System asks the User for confirmation. 3. a. If the User confirms the deletion: go to 4. b. (END) If the User cancels the operation: the System displays the Property Details for the Property. 4. The System sends email notifications to all Guests that have a Booking for the Property with a future Check-In Date. 5. The System deletes the Property; all Ratings associated with Bookings for the property; all Bookings for the Property; and removes the Property from all Favourites lists. 6. (END) ■ For Host: the System displays the Host Dashboard. ■ For Admin: the System displays a Property List. If the Admin reached the Precondition by applying some Property Search Conditions or Property Sort Criteria, the List will conform to these.
Postconditions	■ If 3a. taken: The Property and all associated entities (Bookings, Ratings and Favourites entries) have been deleted, and Guests with a Booking for the Property with a future Check-In Date have received email notifications. ◇ For Host: the System displays the Host Dashboard. ◇ For Admin: the System displays a Property List. If the Admin reached the Precondition by applying some Property Search Conditions or Property Sort Criteria, the List will conform to these. ■ If 3b. taken: the System displays the Property Details for the Property, which has not been deleted.

Table C.6 – Use Case P-D: Delete Property.

Note that deleting Properties is intended to be a "soft"/"logical" delete: records and details are to be kept on the System.

C.4 BOOKINGS

ID	B-L-2
Use case	View Property Bookings
Description	A Host views their own Property's Bookings, or an Admin views any Property's Bookings. (Henceforth, we will refer to the Primary Actor as "User".)
Primary Actor	Host/Admin
Secondary Actor(s)	None
Preconditions	The User is viewing the Property's Property Details.
Main Flow	1. The use case starts when the User issues a request to view the Property's Bookings. 2. (POSSIBLE END) The System checks whether the Property Bookings are in Future mode or Past mode, and: a. If the Property Bookings are in Future mode: The System displays all of the Property's Future Bookings and provides the User a means to switch to Past mode. b. If the User Bookings are in Past mode: The System displays all of the Property's Past Bookings and provides the User a means to switch to Future mode. 3. If the User switches the Property Bookings mode: return to 2.
Postconditions	The System displays either the Property's Past Bookings, or the Property's Future Bookings. (These might be empty lists.)

TABLE C.7 – Use Case B-L-2: View Property Bookings.

C.5 FAVOURITES

ID	F-D
Use case	Remove from Favourites
Description	A Guest removes a Property from their Favourites.
Primary Actor	Guest
Secondary Actor(s)	None
Preconditions	A The Guest is viewing their Favourites. B The Guest is viewing the Property Details for the Property which is to be removed from their Favourites.
Main Flow	1. The use case starts when the Guest issues a request to remove the Property from their Favourites. 2. The System asks the Guest for confirmation. 3. a. If the Guest confirms the removal: go to 4. b. If the Guest cancels the operation: go to 5. 4. The System removes the Property from the Guest's Favourites. 5. (END) a. For Precondition A: the System displays the Guest's Favourites. b. For Precondition B: the System displays the Property Details.
Postconditions	■ If 3a taken: The Property has been removed from the Guest's Favourites. ■ In any case: ◇ For Precondition A: the System displays the Guest's Favourites. ◇ For Precondition B: the System displays the Property Details. If 3a taken: the Favourites symbol has been removed.

Table C.8 – Use Case F-D: Remove from Favourites.

Meeting Notes

appendix D

D.1 Monday, 22/01/2018

- General understanding of the project and formulation of hypotheses
- Communicate organisational details (deadlines, general planning)
- Establish tools to work together
- Assign Philip to be in the lead for communication with the supervisor

D.2 Monday, 28/01/2018

- Setting up GitHub, introduction to LaTeX
- Establishing common understanding of use cases
- Setting to dos, everyone on use cases and setting up the tech stack

D.3 Monday, 12/02/18

- Fully reviewed use cases discussed
- Merge pull request of Lorenz
- Reiterate over requirements
- Paul helps everyone with LaTeX

D.4 Monday, 19/02/2018

- Discussion of complete requirement list
- Kai & Aleksi to present use case diagram
- Zaid & Michael to present class diagram

D.5 Monday, 26/02/2018

- Peer-review written out use cases
- Discussions about which ones to include: level of detail versus level of abstraction
- Aleksi to be coordinating the coherence of the diagrams for the rest of the project, increased responsibility

D.6 Friday, 02/03/2018

- Revise and update all diagrams made and ensure correctness and coherence
- Discuss new approach of peer-reviewing the work of others to maintain coherence
- Set new tasks for the week, including a new domain model

D.7 Friday, 09/03/2018

- Show progress on class diagram, activity diagram (draft), sequence diagram (draft)
- Begin work on the glossary to be included in the report
- Schedule new meetings in small groups for team members to work on the class diagram specifically

D.8 Monday, 12/03/2018

- Meeting with Graham Roberts with a focus on:
  - Design Class Diagram / Analysis Class Diagram differences
  - Sequence Diagram, scope
  - Activity diagram, scope

D.9 Friday, 16/03/2018

- Splitting up initial responsibilities for writing up parts of the reports
- Everyone gets a part in main part
- Abstract / Summary to be completed at the end
- New diagrams to be done: component diagram and deployment diagram (Paul & Zaid)

D.10 Saturday, 17/03/2018 - Tuesday, 20/03/2018

- Daily meetings with the team - working together for many hours
- Re-visiting use cases and class diagram
- Increasing the level of depth of sequence diagrams
- Christina takes the lead on wireframes
- All group members show amazing effort

Bibliography

[1] P. A. of Innkeepers International. The b&b industry. [Online]. Available: http://www.innkeeping.org/?The_Industry

[2] O. M. G. Inc., *OMG Unified Modeling Language Specification Version 2.5.1*. OMG, 2017.

[3] B. Boehm, "A spiral model of software development and enhancement," 1988.

[4] F. Brooks, *The Mythical Man-Month*. Addison-Wesley, 1975.

[5] A. Cockburn. (1998) Basic use case template. [Online]. Available: https://http://alistair.cockburn.us/Basic+use+case+template

[6] T. Blain. (2007) How to write use case preconditions and triggers. [Online]. Available: https://tynerblain.com/blog/2007/03/08/use-case-preconditions-and-triggers/

[7] TemplateLab. 40 use case templates & examples. [Online]. Available: https://http://templatelab.com/use-case-templates/

[8] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson, 1994.

[9] R. H. Inc. What is object/relational mapping? [Online]. Available: http://hibernate.org/orm/what-is-an-orm/

[10] J. Valacich and J. George, *Modern Systems Analysis and Design*. Pearson, 2017.

[11] I. J. James Rumbaugh and G. Booch, *The Unified Modeling Language Reference Manual Second Edition*. Addison-Wesley, 2004.

[12] R. Miles and K. Hamilton, *Learning UML 2.0*. OReilly, 2006.

[13] M. Fowler. (2006) Gui architectures. [Online]. Available: https://martinfowler.com/eaaDev/uiArchs.html