



**Samuel Laycock**

23253444

Development of a Java application for a room booking system using the Model-View-Controller (MVC) pattern

MSc Research & Development Project Report

MComp Computing

Department of Computing

Supervisor: Peter Matthew

**2019/20**

Contents

[Acknowledgements 4](#_Toc41254355)

[Abstract 5](#_Toc41254356)

[Chapter 1 – Introduction 6](#_Toc41254357)

[1.1 – Scope 6](#_Toc41254358)

[1.2 – Limitations 6](#_Toc41254359)

[1.3 – Aims 6](#_Toc41254360)

[1.4 – Objectives 6](#_Toc41254361)

[Chapter 2 – Literature Review 7](#_Toc41254362)

[2.1 – Application Development with Java 7](#_Toc41254363)

[2.1.1 – Object-Oriented Language 7](#_Toc41254364)

[2.1.2 – Portability 8](#_Toc41254365)

[2.1.3 – Robustness and Security 8](#_Toc41254366)

[2.1.4 – Criticisms of Java for Development 8](#_Toc41254367)

[2.2 – Model-View-Controller (MVC) Pattern 9](#_Toc41254368)

[2.2.1 – Model 9](#_Toc41254369)

[2.2.2 – View 9](#_Toc41254370)

[2.2.3 – Controller 10](#_Toc41254371)

[2.2.4 – MVC for Application Development with Java 10](#_Toc41254372)

[2.3 – Development of the Solution 11](#_Toc41254373)

[Chapter 3 – Methodology 12](#_Toc41254374)

[3.1 – Software Development Methodologies 12](#_Toc41254375)

[3.1.1 – Waterfall Model 12](#_Toc41254376)

[3.1.2 – Rapid Application Development (RAD) Model 12](#_Toc41254377)

[3.1.3 – Agile Model 13](#_Toc41254378)

[3.2 – Methodology Justification 14](#_Toc41254379)

[3.3 – Social, Legal, and Ethical Considerations 15](#_Toc41254380)

[Chapter 4 – Requirements Specification 16](#_Toc41254381)

[4.1 – Functional Requirements 16](#_Toc41254382)

[4.1.1 – Requirements Table 16](#_Toc41254383)

[4.2 – Non-Functional Requirements 17](#_Toc41254384)

[4.3 – Use Case Diagram 17](#_Toc41254385)

[Chapter 5 – System and Software Design 19](#_Toc41254386)

[5.1 – Database Design 19](#_Toc41254387)

[5.1.1 – Booking 20](#_Toc41254388)

[5.1.2 – Customer 20](#_Toc41254389)

[5.1.3 – Room 20](#_Toc41254390)

[5.1.4 – RoomBooking 20](#_Toc41254391)

[5.2 – Class Diagram 21](#_Toc41254392)

[Chapter 6 – Implementation 22](#_Toc41254393)

[6.1 – Tools Used 22](#_Toc41254394)

[6.2 – View 22](#_Toc41254395)

[6.3 – Model 23](#_Toc41254396)

[6.4 – Controller 23](#_Toc41254397)

[Chapter 7 – Testing and Evaluation 24](#_Toc41254398)

[7.1 – Testing Plan 24](#_Toc41254399)

[7.2 – Critical Evaluation 24](#_Toc41254400)

[7.2.1 – Project Objectives 24](#_Toc41254401)

[7.2.2 – Deliverables 25](#_Toc41254402)

[7.2.3 – Reflection 26](#_Toc41254403)

[Chapter 8 – Conclusion 26](#_Toc41254404)

[Chapter 9 – Bibliography 27](#_Toc41254405)

[Chapter 10 – Appendices 31](#_Toc41254406)

[Appendix A – Poster 31](#_Toc41254407)

[Appendix B – Questionnaire 32](#_Toc41254408)

[Appendix C – Use Cases 34](#_Toc41254409)

[Appendix D – Testing Plan 37](#_Toc41254410)

# Acknowledgements

I would first like to thank my supervisor Dr. Peter Matthew who has been an extremely helpful lecturer and one of the best I’ve had during my time at Edge Hill University. I would also like to thank Dr. Yonghuai Liu for their support and guidance with the dissertation process.

Throughout my time at Edge Hill University, I would like to thank my friends, namely Josh, Andy, Alisha, Josh, Marcus, Will, Stephen, Jonathan, Maurice and the UKNRP community, and my family for their unparalleled support and encouragement.

Most importantly, I would like to acknowledge my cat Eric for keeping me company and saving my sanity during the bulk of this project.

# Abstract

The purpose of this project was to identify the best practices for application development with Java, utilising the Model-View-Controller (MVC) design pattern. Both areas were researched and explored to highlight functional similarities between the two technologies which were to be used together in the implementation of a room booking system as elicited from the interview of the system stakeholder. A review of software methodologies was undertaken to determine a suitable development approach which coincided with the design objectives of MVC and Java, the most important being encapsulation and code reuse, to be used in the development of a proposed solution. The Rapid Application Development (RAD) was adopted to analyse the requirements, develop design documentation, and implement the results of such tasks into a functional application, tested end-to-end. The conclusion of this project was to determine the suitability for MVC as a design pattern with Java throughout the software development life cycle (SDLC).

# Chapter 1 – Introduction

## 1.1 – Scope

The scope of this project is to explore and analyse the usage and suitability of the Model-View-Controller (MVC) design pattern in a Java application, set to be developed utilising the NetBeans integrated development environment (IDE). The project will involve a preliminary literature review, detailing an investigation into software development practices with Java, the usage of the MVC architecture for application development, and the suitability of the MVC pattern in Java applications including the proposed solution.

## 1.2 – Limitations

The major limitation of the project is its time constraints. The project will not review other programming languages and their usage of the MVC pattern or the application of alternative software models in for Java development in-depth. The MVC design pattern can be applied through most major programming languages, and to examine each thoroughly would extend the scope of the project broadly beyond the limitations of the project specification. The project will focus on the application of the MVC paradigm for Java development, both through literary research and the development of a proposed solution for the room booking system.

## 1.3 – Aims

The focus of this research project is the production of a Java-based application for a room booking system, for a hotel or another short-term lodging establishment, utilising the Model-View-Controller (MVC) software design pattern. The research project intends to explore the best practices for software application development and review the effectiveness and suitability of the MVC pattern for software development methodologies and modern Java applications.

## 1.4 – Objectives

* To determine and evaluate the best practices for software application development in Java to use in the implementation of the proposed system.
* To identify the most suitable software development methodology for the design and development of the system using the MVC paradigm.
* To elicit the functional and non-functional requirements of the room booking system through stakeholder identification and use case diagrams.
* To design and define the software architecture and data structure of the proposed solution using MVC design pattern.
* To implement the software design as a runnable Java application which meets the identified requirements through testing.
* To assess the suitability of the usage of the MVC design pattern throughout the software engineering process.

# Chapter 2 – Literature Review

## 2.1 – Application Development with Java

### 2.1.1 – Object-Oriented Language

Object-oriented programming (OOP) is a software programming paradigm which utilize encapsulated objects, containing self-related data through fields and the functions which manipulate the object, its variables, and its interaction with other objects (Kindler & Krivy, 2011).

Java is a poly-paradigm programming language which supports OOP concepts through class-based approach where objects are defined as independent instances of a class, which defines its methods and fields (Aris & Nazeer, 2011). This provides a layer of abstraction where objects are self-contained blocks of code, applying the principle of encapsulation to allow manipulation of the object through the sole manipulation of calling its methods rather than allowing direct access to the internal data of said object (Hourani, et al., 2019).

Java extends this capacity further by allowing specific access restrictions for methods, allowing for methods to be designated as private (to only be internally called within the method), public (accessible from outside the object), or some medium in between using protected methods which provides greater security and prevents any unintended interaction with data (Schildt, 2018).

Java applies the extensibility of objects in OOP through the use of inheritance of classes. This allows for the creation of subclasses which inherit the fields and functions of its parent class, promoting code reuse whilst allowing for augmentation of the class through additional methods and variables or overridden functions (Sciore, 2019). This grants developers a higher degree of accuracy in their development as they do not need to repeat code or validation, reducing the development time and increasing the efficacy of the project (Kallen & Wrigstrad, 2019).

Objects in Java are considered polymorphic; their methods are able to do execute different code depending on the class of the object, as well as its implemented and extended classes and interfaces (Ogihara, 2018). This can be achieved upon run-time through method overriding through functions with the same name as functions in their superclass or on compilation through overloading methods with different quantities of passed parameters which allows for further reuse of code and classes (Sulír, 2020).

A downside, however, of the utilization of OOP is that whilst the programming paradigm promotes the maintainability and reusability of the source code and its classes, as the code becomes significantly more complex through parallel programming or multithreading, the control flow of the program becomes increasingly opaque and difficult to follow, generating issues for Java application development with complex programming (Harper, 2011). However, Java applications do benefit from the benefits of multithreading as it allows for efficient use of system resources and greater levels of performance (Akram, et al., 2006).

### 2.1.2 – Portability

“Write once, run everywhere” was the advertising slogan used by Sun Microsystems to express the portability of the Java language across any device capable of running a Java virtual machine (JVM) (Horntvedt & Åkesson, 2019). For developers, Java’s irrelevance to platform is the language’s major advantage when compared to similar programming languages, such as C and C++, made possible by translating Java source to byte code for execution by a JVM (Lyon, 2012).

When developing an application, this feature of Java allows for the end user to utilize the program on any JVM-compatible device desired, which has enabled the broad adoption of Java as a language for the development of a menagerie of different systems such as enterprise-level software to mobile applications (Diao, 2018).

### 2.1.3 – Robustness and Security

Java is a mature language which has had a ubiquitous presence as a dominant object-oriented programming language since its creation to the present-day (Chan, 2018). Additionally, one of the key objectives in the development of Java was the development of a robust and secure language, achieved through exception handling and error detection during compilation (Alankus, et al., 2019).

Java provides a secure sandbox platform which allows users to execute unverified bytecode in an isolated environment to protect the application, system, and user from harmful code as well as implement access limitations to prevent malicious software doing any meaningful damage which facilitates the creation of large secure business systems in the language (Meng, et al., 2018).

Additionally, Java is protected from issues which plague C and similar languages, such as unspecified behaviors discarded memory failures, and debugging of the language is possible without the specific hardware used in the final implementation due to the portability of the language (Moliavko, et al., 2019). Java achieves this through lack of explicit representation of security-circumventing pointers, preferring to use objects, arrays, and similar data structures (Parlante, 2000).

### 2.1.4 – Criticisms of Java for Development

Poor performance is the major limitation faced by Java, compared to similarly used faster languages such as C++ and C#, which (Lewis & Neumann, 2016). The significant impact to performance in Java is a result of applications running Java bytecode on the virtual machine, whereas other languages execute machine code on the processor (Su, et al., 2019).

As of January 2019, Oracle released changes to the Java Standard Edition (SE) license which discontinued free access to Java SE updates for commercial purposes, instead replacing Java SE’s update system with a subscription model (Oracle, 2019). Because of this, modern Java developers who wish to take advantage of Java SE commercial features, SE updates, support from Oracle are required to adhere to the subscription service or resort to alternatives such as OpenJDK (Diaz, 2018).

## 2.2 – Model-View-Controller (MVC) Pattern

*Figure 1. MVC Diagram* (Nor, et al., 2018)*.*

The Model-View-Controller design pattern, often abbreviated to MVC, is a proposed software architecture model which separates the system logic of applications into three inter-connected components; namely the model, view, and controller (Singh & Iyer, 2016). Through this, MVC addresses the separation of concerns (SoC) principle by detaching the business logic and data manipulation from its presentation to the user through the user interface (Gupta, et al., 2012). The concept of interchangeable and self-contained elements of software adheres to the notion of encapsulation used in object-oriented programming, suggesting a fundamental compatibility and resonance between the design pattern and OOP languages (Thakur & Pandey, 2019).

### 2.2.1 – Model

The model component of the paradigm embodies the core application logic of the system as well containing the system’s data structures (Thakur & Pandey, 2019) When applying MVC design principles to OOP languages, the model consists of objects, or some structure consisting of objects, which store and manipulate data. (Thakur & Pandey, 2019). In most applications, the user does not directly interact with the model; user inputs are handled by the controller, which stores or manipulates the data of the model which then updates the view to express the change of state (Akbar, et al., 2017).

Due to the modular nature of the application provided by adhering to the design pattern, a single model can be displayed through multiple views without additional programming therefore promoting code reuse (Cheon, 2019).

### 2.2.2 – View

The view component embodies the graphical user interface (GUI) of the application which exists to present the data state of the model to the end user (Gupta, et al., 2012). In an MVC implementation, the view separated from the model which facilitates the ability of multiple views to express the state of the model data in different ways for alternative purposes or on varying platforms such as desktop, web, or mobile (Komara, et al., 2016).

### 2.2.3 – Controller

The controller component acts as a medium between the user and the application, facilitating user input through the user interface by manipulating or adding changes to the data objects of the model (Singh & Iyer, 2016). Because of this, the View and controller are intrinsically linked through the controller’s nature of handling the requests of the user, whereas the model should be “functional” without any external requests to the view or controller (Durdik, 2016).

### 2.2.4 – MVC for Application Development with Java

Object-oriented software design shares many advantages with the MVC design pattern, providing a cohesive architectural logic when implementing software solutions (Thakur & Pandey, 2019).

The concept of a model in MVC design directly correlates with the principle of encapsulation in OOP, allowing for objects to exist as independent data structures which contain their own fields and functionality (Hourani, et al., 2019) . The interaction between objects in OOP provides a layer of abstraction, similarly MVC provides a layer of abstraction through separating the internal code of the components from the external therefore enabling easier troubleshooting the modular architecture (Thakur & Pandey, 2019).

The autonomous design of the components designated in the MVC paradigm grant easier simultaneous development, as the separation between the data and interface allow multiple developers to work on different parts of the application concurrently (Kumar, 2019). The modularity of Java further reinforces this benefit, as the self-contained nature of classes allows for developers to work on separate sections of the application without close interference (Mak & Bakker, 2017). This impacts the potential design methodologies for the creation of applications utilizing both MVC and OOP as both designs ameliorate with iterative methodologies, compared to traditional sequential methods (Grogono & Shearing, 2008).

Java promotes the concept of code reuse through the inheritance of its classes and objects, allowing them to obtain the fields and functions of their superclass without requiring the repeated implementation of the functionally identical methods (Tempero, et al., 2013). MVC promotes the reuse of code as models can exist with multiple views and controllers, disabling the requirement of repeated functionality which exists in the model (Cheon, 2019).

Furthermore, the combination of Java and the MVC paradigm allow for a high degree of testability for a developed application. This is resultant of the modularity of MVC allowing for isolated and independent testing of the models, views, and controllers whilst also promoting code reuse which reduces the amount of potential module tests required (Cheon, 2019). Java promotes loosely coupled structures through the object-oriented design architecture due to reduced dependencies between interacting objects (Bajeh, et al., 2020).

## 2.3 – Development of the Solution

The solution will implement both of these technologies in tandem. By applying the MVC design pattern to the solution’s software architecture, it provides an extensible and highly maintainable system as the modular nature of the logically separated components allows for the replacement and customization of separate elements without impacting the overall functionality of the application (Akbar & Handriani, 2018). This would grant the proposed solution devised by this project further maintainability, which is key for an enterprise application if they rely on the software for their business (Kumar, et al., 2017).

Additionally, the logical separation would produce a highly testable solution, which enables highly efficient test-driven development due to the ease of testing the isolated components and rapid prototyping of views to test the functionality of the model which allows quick development and adaptation to the software (Yusop, et al., 2019).

Java, as a robust and mature language, confers many benefits to potential business solutions developed using it (Chin, et al., 2019). The security of the Java virtual machine and the lack of pointers in the language grant applications developed in the language a higher level of security unparalleled by similarly used languages, such as C/C++ (Chen & Tian, 2017). In this era of technology, data is king and the software which utilizes and supports such data is required to comply with industry standards of security, which Java would help enable (Khosrow-Pour & Ramachandran, 2017).

Java provides a multi-purpose ‘jack of all trades’ quality to software, due to being utilized and co-opted by a massive audience of businesses and developers, which grant the language the ability to develop applications quickly and cheaply whilst producing a secure and portable product (Chin, et al., 2019).

Due to the shared concepts and principles of Java and the MVC framework, the solution which combined both of these technologies would establish a highly modular application where the graphical user interface is logically separate from the required business logic which heavily reinforces the re-use of code and manageability of development (Sarker & Apu, 2014).

# Chapter 3 – Methodology

## 3.1 – Software Development Methodologies

### 3.1.1 – Waterfall Model

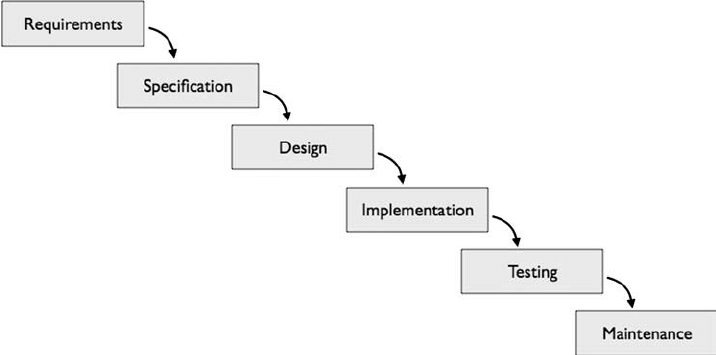


Figure 2. Waterfall diagram (Cohn, et al., 2009)

The waterfall methodology provides a sequential approach to the development of software in which the requirements of the specification suffer little-to-no change during the development process (Sommerville, 2016). This means that the development approach does not inherently support adaptation to the requirements of the system during later stages of the methodology, such as unforeseen changes in circumstances or clarifications in the client’s desires (Balaji & Murugaiyan, 2012).

Because of this, the waterfall model is best suited for less complex and static applications which have clarified requirements and therefore would not require any additional modifications during the software development life cycle (Mahalakshmi & Sundararajan, 2013). However, if such conditions are met, the waterfall methodology provides a highly structured and well-documented approach to producing a software solution, which would involve the translation of UML diagrams into architecture documentation into source code (Light, 2009).

### 3.1.2 – Rapid Application Development (RAD) Model

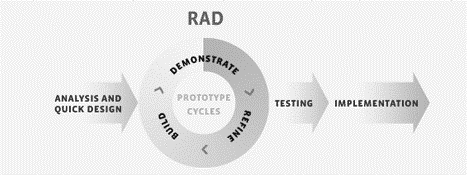


Figure 3. Rapid Application Development (RAD) diagram (Biesemans, et al., 2010).

Rapid Application Development is an umbrella term, assigned to a number of flexible and efficient development methodologies, including Martin’s model for rapid application construction (Daud, et al., 2010). This approach to the development of software reinforces emphasis on the rapid production of software without being restricted by the time constraints of planning and the required development documentation (Shaffi & Al-Obaidy, 2013). The RAD model was developed in reaction to the excessive use of the traditional waterfall methodology, during development the requirements, scope, and system design could change during later stages of the development process (Sommerville, 2016).

RAD approaches development with the understanding that software, by design, is highly mutable and therefore requires an adaptable approach for its development rather than a static methodology, such as the waterfall model (Shaffi & Al-Obaidy, 2013). The aim of the RAD model is to deliver a high-quality product in a short time-period through the cyclical affirmation of requirements and prototyping of the solution for the adaptation of the application during development to better meet the required functionality (Mahapatra & Goswami, 2015).

### 3.1.3 – Agile Model

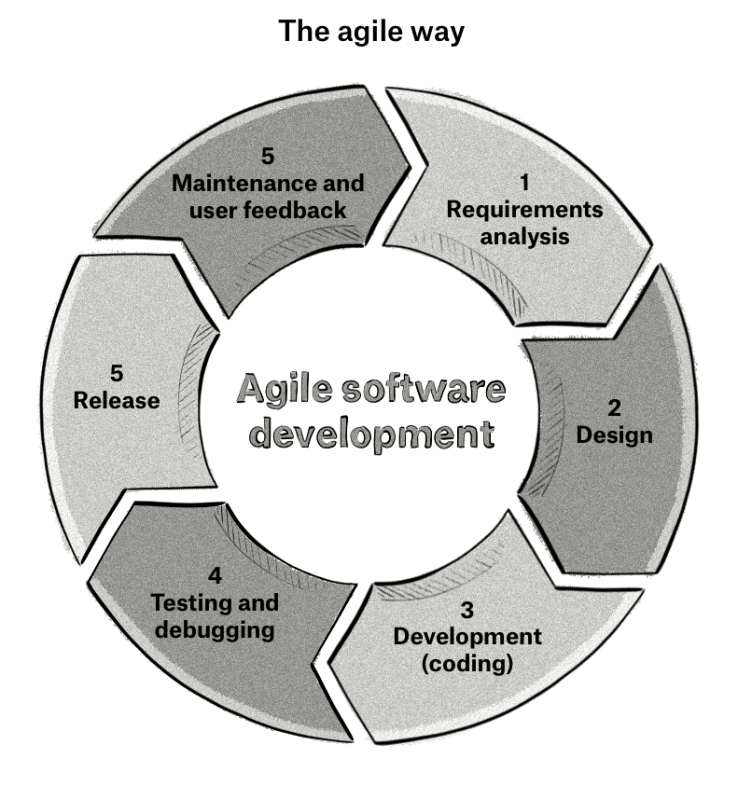


Figure 4. Agile diagram (Sweeney, 2014).

Similarly to RAD, agile development methodologies offer an iterative and cyclical approach to software development, refining the requirements specification and programmed solution through development cycles where the satisfaction of the end user can be gauged based on the current state of development (Sommerville, 2016).

The most popular framework which utilises the principles of agile development is called Scrum. Scrum utilises sprint cycles in which a designated time period is given to a quantity of work within which it is set to be implemented, tested, and completed (Kuhrmann, 2017). With regular meetings and discussions during and after scrum sprint cycles, the implementation should be planned, refined, distilled, implemented, and reviewed (Fowler, 2019). The daily discussion and definition of clear goals to be achieved enables scrum to deliver high-quality products in a short period of time whilst enabling groups of developers to work on the same application through modular division of tasks to be completed (Mahalakshmi & Sundararajan, 2013).

However, agile does have some downsides. The inherent looseness, informality, and mutability of the agile processes are incompatible with the bureaucratic and administrative approaches of the legal and clerical divisions of large companies which may cause development disruptions (Hoda, et al., 2010). Additionally, as a development-driven approach, there is a consequential lack of documentation and the cyclical nature requires constant customer/user involvement which may cause scheduling clashes or the disruption of current business processes (Mergel, et al., 2018).

## 3.2 – Methodology Justification

This project intends to employ the rapid application development (RAD) approach to the software development lifecycle to quickly produce a high-quality solution which meets the requirements of the end user. As previously discussed, the MVC design paradigm allows for logical separation of the graphical user interface and the core business logic of the application, which means that the model can be refined, and the GUI can be amended and modified through cycles of prototyping. This, similarly, to the agile approach to development, will enable the refinement and distillation of a quality software product which meets the determined specification. RAD was chosen over the agile methodology due to the scale of the project; this project will only involve one developer who will contribute to the entire software development lifecycle whereas agile approaches, namely Scrum, utilize the ability to break projects down into manageable bytes and divide them among groups and team members, which isn’t possible in this case. The Scrum agile methodology also has meetings and regular discussion among the developers on the topic of the software and its implementation which isn’t possible in the scope of this project due to having only one contributor.

The lifecycle of the software solution will start with the elicitation of the requirements from the interested user, in this case an individual who wishes to establish room lodging business. The requirements will be derived from interviews with the interested party, which will form the embryo from which the requirements specification for the application will be developed and refined. Techniques such as use case diagrams will be utilized to identify and analyze the system’s key stakeholders from the interview and the tasks that they will undertake which will help visualize and understand the functional requirements of the proposed solution. This process will result in the requirement specification being drafted, as well as the various diagrams and documentation created by the requirement analysis techniques employed. These requirements will then drive the cyclical prototyping of the system as the completion allows for the design and implementation of the system to begin.

With the requirements elicited and analyzed, conceptual models of the proposed system’s architecture will be created, defining the structure of the system as well as the underlying business logic of how the application will operate. Object-oriented class diagrams will be created to establish to logical hierarchical structure of the application. The data structures of the potential system will also be identified and documented, with the database schema and business rules of the system forming an inter-connected blueprint for the implementation of the proposed design. These designs will adhere to the concepts of MVC and object-oriented programming in Java as identified in the research portion of this project.

The requirement specification and design documentation will be utilized to draw the key progress indicators of development to allow the developer to create a system to the end user’s requirements. The functionalities identified will be implemented as designated in the design section of the project, allowing the solution to facilitate the functional requirements as determined earlier in the development lifecycle. Upon the implementation of all the requirements, the testing phase of the project will begin, to ensure that the functional requirements of the system are met. A testing plan will be devised to thoroughly test the entire functionality of the application as well as its ability to meet the requirements determined in the specification, therefore acting as the progression indicators of successful implementation. The testing phase will also help identify any bugs or errors in the program which weren’t ultimately obvious, allowing for the systematic procurement of a high-quality system.

Upon the completion of the implementation and testing phases which highlighted the meeting of the functional requirements, an evaluation into the implementation will be initiated, critically analyzing the implementation in regard to the literature review, the aims, and the objectives of the project. This will help found the conclusion of the study, which will discuss the outcome of the project, how well it went, limitations or hinderances discovered, and areas of the project which would be illuminated with further research or study, such as a more efficient implementation or a wider scope of the project.

## 3.3 – Social, Legal, and Ethical Considerations

For areas of this project, the social, legal, and ethical implications of the work must be considered. The proposition of interviews or questionnaires with the key users or system stakeholders will require their consent and, more importantly, the understanding in regard to their role and involvement in the project. This will be conveyed to them through a participation form, which will detail to the participants the aims and objectives of the project, as well as the reason behind their involvement with such and how the data will be processed. The document will go on to detail the participant’s rights and the legal legislation which protects their personal information.

Due to the project only requiring the feedback and answers, all personal details will be anonymized if the documentation is required to be included in the project report. Personal details of these individuals will be kept confidential and will be stored in a secure place for only as long as they are needed, to comply with GDPR regulations. Participants will be informed of their rights of retraction and how to remove or distance themselves from the project if they wish to.

All of the data of the participants will follow regulations as enforced by the Data Protection act and recent GDPR regulation protections. Additionally, the software utilized as part of this project will be covered by a valid license (if required) and will only be used in compliance with the usage documentation. The data stored in the application will also be stored in concordance with all data protection legislation, to ensure the security and safety of the project’s solution.

# Chapter 4 – Requirements Specification

## 4.1 – Functional Requirements

The functional requirements of a proposed solution are a collection of granular statements which describe a specific process of the system which the software must provide through its functionality (Sommerville, 2016). Functional requirements may involve the manipulation and processing of data from the database and user input or any calculations executed by the application (Eckhardt, et al., 2016).

These requirements have been derived from a questionnaire with the system stakeholder which can be found in the appendices under “Appendix B”. An in-person interview was not possible at the time, so a text-based interview was taken instead due to the social limitations set in place to protect from the coronavirus pandemic of 2020.

### 4.1.1 – Requirements Table

|  |  |
| --- | --- |
| **No.** | **Requirement Description** |
| FR1 | A user must be able to add a room to the system database. |
| FR2 | A user must be able to update a room in the system database. |
| FR3 | A user must be able to delete a room from the system database. |
| FR4 | A user must be able to view all the rooms from the database in a list. |
| FR5 | A user must be able to create a booking in the system database. |
| FR6 | A user must be able to add a customer to a booking in the database. |
| FR7 | A user must be able to fetch the details of a previous customer from the database. |
| FR8 | A user must be able to add a room to a booking. |
| FR9 | A user must be able to book multiple rooms as part of a singular booking. |
| FR10 | A user must be able to delete rooms from a booking. |
| FR11 | A user must be able to update a booking in the system database. |
| FR12 | A user must be able to add a room to a previously created booking. |
| FR13 | A user must be able to delete a room from a previously created booking. |
| FR14 | A user must be able to delete a booking from the system database. |
| FR15 | A user must be able to view all existing bookings in the database. |

## 4.2 – Non-Functional Requirements

The non-functional requirements of a proposed solution are a collection of statements which describe the operational attributes required of the system, rather than a specific function of the application (Sommerville, 2016). The non-functional requirements can typically be separated into the following categories; these are executional characteristics, which can be observed during the operation of the application, and evolutionary characteristics, which qualities can be assessed and acknowledged through the structure of the system (Wiegers & Beatty, 2013).

**Usability –** The system’s ease-of-use should be high. This will be achieved through the “view” component of the system design which is made up of the graphical user interfaces for the product.

**Portability –** The portability of the product is intrinsically entwined with the chosen programming language for the solution, Java. As documented through the literary research, Java as a language possesses a high degree of portability with its applications since Java applications can run on any system bearing the Java Virtual Machine (JVM).

**Maintainability –** The maintainability of the system is dependant on the architectural structure, implementation, and documentation of the system. The MVC pattern will utilise abstraction to decrease the logical distance between the functionality of the code and the problem which needs to be solved.

Java and MVC promote code re-use, which decreases the amount of code which needs to be tested to achieve functionality. The separation between the GUI and the business logic of the system allows future developers to diagnose and understand the existing code of the system and how it meets the application’s functionality.

## 4.3 – Use Case Diagram

In the Unified Modeling Language (UML), use-case diagrams are used as a high-level graphical representation of the system and its use cases. The figure below depicts a use-case diagram which illustrates the system user’s actions developed using the requirements derived from questioning the stakeholder.

In the proposed system, there is only one system stakeholder which has been identified from the requirements, which can be seen through the diagram succeeding. The use cases which they utilize are expressed in the ellipses within the system boundaries.

The individual use case descriptions can be read in the appendices section “Appendix C – Use Cases”, placed there to maintain the readability of this document without too much disruption.

A close up of a piece of paper

Description automatically generated

Figure 5. Use Case Diagram.

# Chapter 5 – System and Software Design

## 5.1 – Database Design

To store, manipulate, and manage the data of the proposed solution, a relational database management system (RDBMS) must be implemented. Because Java is being utilised as the main programming language of this project, the Apache Derby database management system will be used. Apache Derby, rebranded to Java DB by Oracle in 2015, is an open-sourced relational database system programmed entirely in Java which supports Java Database Connectivity (JDBC) and Structured Query Language (SQL) for handling relational data (Apache Derby, 2020).

Documented below are the data structures which express the schema of the database and will be implemented into the database as tables. Four key tables were identified from the requirements. These are:

* **Booking –** This table will store the information about a singular booking, made by a specific customer.
* **Customer –** This table will store the information about a customer, who can make bookings.
* **Room –** This table will store the information about a room in the hotel.
* **RoomBooking –** This table will store the information of a room booking, which consists of the booking of a singular room and belongs to an overarching booking. This facilitates the booking of multiple rooms through one booking.

### 5.1.1 – Booking

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Description** | **Data Type** | **Length** |
| BOOKINGNUMBER | Primary Key. Generated ID for booking record. | INTEGER | N/A |
| MOBILENUMBER | Foreign Key. Mobile number of customers. | VARCHAR | 11 |
| ARRIVALDATE | Arrival date of customers to hotel. | VARCHAR | 10 |
| DEPARTUREDATE | Departure date of customers from hotel. | VARCHAR | 10 |

### 5.1.2 – Customer

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Description** | **Data Type** | **Length** |
| MOBILENUMBER | Primary Key. Mobile number of customers. | VARCHAR | 11 |
| FIRSTNAME | First name of customers. | VARCHAR | 50 |
| LASTNAME | Last name of customers. | VARCHAR | 50 |
| EMAIL | Email address of customers. | VARCHAR | 350 |

### 5.1.3 – Room

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Description** | **Data Type** | **Length** |
| ROOMNUMBER | Primary Key. Unique room identifier. | VARCHAR | 50 |
| FLOORNUMBER | Floor designation of the room. | VARCHAR | 50 |
| BEDNUMBER | Number of beds in the room. | INTEGER | N/A |
| ROOMDESCRIPTION | Brief overview of the room. | VARCHAR | 500 |
| ROOMPRICE | Price of the room. | DOUBLE | N/A |
| ENSUITE | If the room has an en suite facility. | BOOLEAN | N/A |

### 5.1.4 – RoomBooking

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Description** | **Data Type** | **Length** |
| BOOKINGNUMBER | Composite Primary Key. Generated ID for booking record. | INTEGER | N/A |
| ROOMNUMBER | Composite Primary Key. Unique room identifier. | VARCHAR | 50 |
| GUESTNUMBER | Number of guests in the room booking. | INTEGER | N/A |
| CATERED | If room booking is to be catered or not. | BOOLEAN | N/A |
| CONFIRMED | If room booking is confirmed or not. | BOOLEAN | N/A |

## 5.2 – Class Diagram

In the Unified Modeling Language (UML), class diagrams are used to illustrate the main classes of the system, including their fields and functions, for the purposes of modelling an object-oriented application (Sommerville, 2016). The arrows between the classes demonstrate the inter-class relationships, such as inheritance or association, as well as the multiplicities through the arrow heads (Goodwin, 2012). These diagrams can be used as a medium to translate the requirements and structure of the system into programmed code.

In the diagram below, the classes which belong to the View and Controller components, which encompass the graphical user interface and handling of user input, have been omitted to further express with clarity the Model component of the system, which holds the core functionality and implemented data manipulation.

A screenshot of a cell phone

Description automatically generated

Figure 6. UML Class Diagram.

# Chapter 6 – Implementation

## 6.1 – Tools Used

Apache NetBeans 8.2 was selected as the Integrated Development Environment (IDE) for the implementation of the Java-based application. NetBeans utilises integrated component modules which provide additional functionality. For this project, NetBeans’ GUI builder simplifies and streamlines the development of Java Swing applications and allows developers to quickly design user interfaces for a system by positioning ‘drag-and-drop’ GUI components (Apache, 2020).

Apache Derby (also known as Java DB) was selected as the Relational Database Management System (RDBMS) for the implementation of a database for a Java-based application. The RDBMS supports the execution of SQL commands to manipulate the database and can be easily embedded into any Java program.

Javadoc is used to generate API documentation in a HTML format from in-line comments in the source code of an application. NetBeans facilitates the automatic generation of the Javadoc HTML and assists with the documentation of methods and their parameters within the IDE. This improves the maintainability of the system as it allows future developers to understand the source code of the system as the implementer understands and describes it.

## 6.2 – View

The View component of the design pattern encompasses the GUI of the application. NetBeans facilitates the rapid prototyping of Swing user interfaces through its GUI builder. The majority of the View component was implemented before any other implementation, using the requirements, use case diagrams, and data structures to understand what UI elements are required to facilitate the functionality of the system.

The View consists of the following JFrame Form classes:

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| AddBooking | Interface which facilitates the adding of booking, adding of customers, and adding of rooms to bookings. |
| AddRoom | Interface which facilitates the addition of a room to the system. |
| AddRoomBooking | Interface which facilitates the booking of a specific room for a booking. |
| BookingsList | Interface which lists all of the bookings in the system so that they can be viewed and managed. |
| ManageBooking | Interface which facilitates the update and deletion of bookings. |
| MenuForm | Interface which serves as the menu for the application, allowing the user to access the rest of the functionality. |
| RoomList | Interface which lists all of the rooms in the system so that they can be viewed and managed. |

## 6.3 – Model

The Model component of the application embodies the core functionality of the system and is logically separate and loosely coupled from the Controller and View. Once the GUI was prototyped, the development of the functionality expressed in the requirements specification were implemented. The classes which make up the Model are the only classes which manipulate the data of the system and are also the only classes which interface with the database in which the applications data is stored.

The Model consists of the following Java classes:

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| Booking  (extends DatabaseConnection) | The Booking class liaises with the Derby database to create, update, and delete booking records and load records from the database. |
| Customer  (extends DatabaseConnection) | The Customer class liaises with the Derby database to add Customer records and load existing Customers from the database. |
| DatabaseConnection | The DatabaseConnection class connects with the Derby database and facilitates the execution of SQL statements to manipulate system data. |
| Room  (extends DatabaseConnection) | The Booking class liaises with the Derby database to create, update, and delete room records and load records from the database. |
| RoomBooking  (extends DatabaseConnection) | The RoomBooking class liaises with the Derby database to create and delete Room records, load records from the database, confirm room bookings, and delete unconfirmed bookings. |

## 6.4 – Controller

The Controller component of the application serves to facilitate the inputs of the system user through the View, the of the program GUI, and retrieve the information of the Model for the View to display to the end user. The Controller components were implemented last as they connect the user-facing interface to the core functionality of the system contained in the Model.

The Controller consists of the following Java classes:

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| BookingController | The BookingController passes user inputs from the GUI to the Booking Model class to create, manipulate, and retrieve Booking records. |
| CustomerController | The CustomerController passes user inputs from the GUI to the Customer Model class to create, manipulate, and retrieve Customer records. |
| RoomBookingController | The RoomBookingController passes user inputs from the GUI to the RoomBooking Model class to create, manipulate, and retrieve RoomBooking records. |
| RoomController | The RoomController passes user inputs from the GUI to the Room Model class to create, manipulate, and retrieve Room records. |

# Chapter 7 – Testing and Evaluation

## 7.1 – Testing Plan

Testing serves to prove that the implemented application in previous stages of development meet the requirements as defined in the requirements specification and to uncover any potential defects in the program before it is fully deployed (Jan, et al., 2016).

The testing plan acts as the structure for the testing process, connecting the requirements specification to the actual usage of the application by a potential system user, therefore proving that the program meets the desires and requirements of the future stakeholders who would use this system. Each of the tests serve to verify and validate implementation, testing whether the right product has been created and implemented in the desired fashion (Sommerville, 2016).

Full end-to-end system testing was conducted upon apparent completion of the implementation of the system, with the interactions between the GUI and core functionality of the Model through the Controller classes had been finalised. The testing plan served to demonstrate that the implemented solution met the requirements specified in the earlier portions of the software development life cycle (SDLC).

The full testing plan can be found in the appendices of the document, under “Appendix D – Testing Plan”.

## 7.2 – Critical Evaluation

The following sections form the summative evaluation of the project, forming solid judgements founded from the analysis of the project methodology, the generated artefacts and documentation, and the research undertaken.

### 7.2.1 – Project Objectives

The research undertaken within the literature review portion of the project served to highlight the best practices and concepts of object-oriented design, extending to specifically software development using the Java programming language. This was then used in the justification for the application and served to influence the direction of the implementation, including the system designs and the IDE and RDMS tools used for the creation of the application.

The analysis of the software development methodologies served to identify features of the development approaches which coincided with the features and objectives of Java as a programming language and the Model-View-Controller (MVC) design pattern. It was found that the utilisation of the Rapid Application Development (RAD) complemented MVC and Java, in the circumstances of the project problem, the best when compared to other software development methodologies of note, such as traditional waterfall and agile approaches.

The requirements of the system were derived from a text-based interview/

questionnaire with the system stakeholders due to the unforeseen circumstances imposed by COVID-19 limiting in-person contact. However, the requirements were still able to be derived and were analysed to produce structural UML diagrams and data dictionaries to serve as the blueprints for the implementation of the product as a Java application using the MVC design pattern. These requirements were shown to have been achieved through the completed test plan which demonstrated the functionality of the system and how it met the determined requirements.

The suitability of MVC for Java Development was assessed throughout the entire project. The literature review sought to draw similarities between the concepts which Java and MVC both aimed to achieve, which enabled an innate resonance between the two technologies and proved that they served well together in the implementation and design of the software.

### 7.2.2 – Deliverables

The literature review served as the core deliverable upon which the rest of the project was ultimately derived. The purpose of the literature review was to research the best practices for OOP Java development and the core goals and objectives which Java served to achieve as a language compared to other similarly used languages. such as C# and C++. The research also served to highlight the objectives and purpose behind the MVC design pattern and how that correlated with Java as a language, which uncovered similarities between the two which would be explored further throughout the project in the design and development phases of the methodology.

The selection of a suitable methodology for the design and development of the project solution was as important as any other section of the project. The selection of a methodology which complemented the researched technologies was paramount to the success of this project and it was important to pick a viable methodology which would also work well with the requirements of the user.

The system design and implementation served to breakdown the requirements of the user into meaningful documentation which could be used to break down the problem, analyses what needs to be done to solve it, and produce a blueprint for a program which would serve as a solution to the problem. The usage of UML diagrams served to break down the problem into comprehensible tasks which assisted with the rapid implementation of a solution.

The test plan serves as a method of proving the functionality of the system design and implementation, by demonstrating the usage of the system the user can understand that the developed application can actually achieve the requirements they had specified at the start of the project. The tests fully encompassed the requirements which were derived, so therefore substantiated the claim of success of the solution for the problem.

### 7.2.3 – Reflection

The project served to give an insight on various areas of further fields, such as heavily researching into topics and connecting ideas and concepts found among them to help found claims and derive concepts which could be expanded upon through further scholarship whilst also providing an outlook into software engineering as a whole through the utilization of the reviewed literature in the software development life cycle to determine the languages and design patterns to use to achieve specific goals.

# Chapter 8 – Conclusion

In conclusion, the Java programming language and MVC design pattern share many similarities in terms of the objectives they are meant to achieve and work well in-tandem in the rapid design and development of a programmed problem solution, in this case to provide a room booking system for a hotel.

Research has shown that both Java and MVC heavily promote code reuse and modular software design innately through their design; Java is an object-oriented language which promotes modular design and code reuse through inheritance and encapsulation and MVC supports code reuse by allowing GUIs to utilize the same by logically separating the core components of the product into the View, handling the GUI, the Controller, which handles user input and feeds back into the GUI, and the Model, which manipulates the data and the core functionality of the application.

The produced product artefact, the problem solution, isn’t a perfect implementation by any means – it was meant as a rapidly developed application which illustrated the MVC design paradigm through the structure of the project whilst utilizing the concepts of inheritance and encapsulation to further the effective code reuse in the program and reduce redundancies in the application. Further development could go into more depth in the development of the project, implementing functionality for customers to be able to organize their own bookings or allowing for the ability to search the list of rooms for a room with the selected desired features.

Future research could explore the relationship with Java and the MVC paradigm, contrasting development between those technologies with other software common design patterns and a review of developers could be undertaken to identify industry trends in terms of design patterns and programing languages, as well as the development methodologies they use to structure and implement them.

Overall, the research project has uncovered innate similarities in the motivations behind the objectives of the Java Language and the problems the MVC design pattern intends to overcome which was observed and expressed through the research of the topics and the implementation of an application using the core functionalities held by Java as a language and structurally through the MVC architecture used to build the application. This could be investigated further by corroborating such findings with more in-depth analysis of the technologies or by identifying similar trends with other languages and paradigms.

# Chapter 9 – Bibliography

Akbar, F., Rochimah, S. & Akbar, R., 2017. *Investigation of SQL Clone on MVC-based Application.* Surabaya, Indonesia, IPTEK: d International Seminar on Science and Technology.

Akbar, M. & Handriani, I., 2018. *Study and Implementation Information System of Zakat using MVC Architecture.* Jakarta, Indonesia, IOP Publishing: Materials Science and Engineering.

Akram, S., Sartor, J. & Eeckhout, L., 2006. *DVFS performance prediction for managed multithreaded applications.* Uppsala, Sweden, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 12-23.

Alankus, G. et al., 2019. *Java Fundamentals.* 1st ed. Birmingham, UK: Packt Publishing.

Apache Derby, 2020. *Apache Derby.* [Online]   
Available at: https://db.apache.org/derby/  
[Accessed 1 May 2020].

Apache, 2020. *NetBeans IDE - Swing GUI Builder (Matisse) Features.* [Online]   
Available at: https://netbeans.org/features/java/swing.html  
[Accessed 01 May 2020].

Aris, T. & Nazeer, S., 2011. *Object-oriented programming semantics education based on intelligent agents.* Johor Bahru, Malaysia, IEEE Malaysian Conference in Software Engineering.

Bajeh, A. et al., 2020. OBJECT-ORIENTED MEASURES AS TESTABILITY INDICATORS: AN EMPIRICAL STUDY. *Journal of Engineering Science and Technology,* 15(2), pp. 1092-1108.

Balaji, S. & Murugaiyan, M., 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management,* 2(1), pp. 26-30.

Biesemans, J., Horsten, W. & Deroose, T., 2010. *Processing and Archiving Facility for Airborne Remote Sensing: Overview of Hardware, Software System and Operations,* Mol, Belgium: Vito.

Chan, R., 2018. *The 10 most popular programming languages, according to the ‚Facebook for programmers‘.* [Online]   
Available at: https://www.businessinsider.de/international/the-10-most-popular-programming-languages-according-to-github-2018-10/?op=1  
[Accessed 1 May 2020].

Chen, X. & Tian, G., 2017. *Research and implementation of search engine based on Java Technology.* Dalian, China, Atlantis Press: International Conference on Materials Science, Machinery and Energy Engineering (MSMEE 2017).

Cheon, Y., 2019. *Multiplatform application development for Android and Java.* Honolulu, HI, USA, IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA).

Chin, S., Vos, J. & Weaver, J., 2019. *The Definitive Guide to Modern Java Clients with JavaFX.* 1st ed. Berkeley, CA: Apress.

Cohn, M., Sim, S. & Lee, C., 2009. What Counts as Software Process? Negotiating the Boundary of Software Work Through Artifacts and Conversation. *Computer Supported Cooperative Work ,* 18(5-6), pp. 401-443.

Daud, N., Bakar, N. & Rusli, H., 2010. *Implementing rapid application development (RAD) methodology in developing practical training application system.* Kuala Lumpur, Malaysia, IEEE: 2010 International Symposium on Information Technology.

Diao, Z., 2018. *Design and Implementation of Online E-business Website Based on Java.* Jinan, China, Atlantis Press.

Diaz, P., 2018. *Java Did What? Understanding How 2019 Java Licensing Changes Impact You.* [Online]   
Available at: https://www.lakesidesoftware.com/blog/java-did-what-understanding-how-2019-java-licensing-changes-impact-you  
[Accessed 01 May 2020].

Durdik, Z., 2016. *Architectural Design Decision Documentation through Reuse of Design Patterns.* 1st ed. Karlsruhe, Germany: KIT Scientific Publishing.

Eckhardt, J., Vogelsang, A. & Fernandez, D., 2016. *Are "non-functional" requirements really non-functional?: an investigation of non-functional requirements in practice.* New York, United States, ACM - ICSE '16: Proceedings of the 38th International Conference on Software Engineering.

Fowler, F., 2019. *Navigating Hybrid Scrum Environments.* 1st ed. Berkeley, CA: Apress.

Goodwin, D., 2012. *Object Oriented Analysis & Design - Modelling and Simulation, Lecture #3.* Bedfordshire and Buckinghamshire, England: University of Bedfordshire.

Grogono, P. & Shearing, B., 2008. *Concurrent software engineering: Preparing for paradigm shift.* Montreal, Canada, C3S2E: Canadian Conference on Computer Science & Software Engineering.

Gupta, P., Mata-Toledo, R. & Monger, M., 2012. Utilizing ASP.NET MVC in web development courses. *Journal of Computing Sciences in Colleges,* 27(3), pp. 10-14.

Harper, R., 2011. *Some Thoughts on Teaching FP.* [Online]   
Available at: https://existentialtype.wordpress.com/2011/04/17/some-advice-on-teaching-fp/  
[Accessed 1 May 2020].

Hoda, R., Noble, J. & Marshall, S., 2010. *How Much is Just Enough? - Some Documentation Patterns on Agile Projects.* Bavaria, Germany, European Conference on Pattern Languages of Programs (EuroPLoP 2010).

Horntvedt, R. & Åkesson, T., 2019. *Java, Python and Javascript, a comparison,* Kristianstad, Sweden: Högskolan Kristianstad.

Hourani, H., Wasmi, H. & Alrawashdeh, T., 2019. *A Code Complexity Model of Object Oriented Programming (OOP).* Amman, Jordan, Jordan, IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT).

Jan, S. et al., 2016. An Innovative Approach to Investigate Various Software Testing Techniques and Strategies. *IJSRSET ,* 2(2), pp. 682-689.

Kallen, M. & Wrigstrad, T., 2019. *Performance of an OO compute kernel on the JVM: revisiting Java as a language for scientific computing applications.* s.l., MLPR, Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes.

Khosrow-Pour, M. & Ramachandran, M., 2017. Secure Software Development of Cyber-Physical and IoT Systems. In: *Encyclopedia of Information Science and Technology.* Pennsylvania, United States: IGI Global, p. 14.

Kindler, E. & Krivy, I., 2011. Object-oriented simulation of systems with sophisticated control. *International Journal of General Systems,* 40(3), pp. 313-343.

Komara, H., Hendradjaya, B. & Saptawati, G., 2016. *Dynamic generic web pattern for multi platform.* Denpasar, Indonesia, IEEE: International Conference on Data and Software Engineering (ICoDSE).

Kuhrmann, M., 2017. *Hybrid software and system development in practice: waterfall, scrum, and beyond.* New York, United States, Association for Computing Machinery, ICSSP 2017: Proceedings of the 2017 International Conference on Software and System Process.

Kumar, A., 2019. *Web Technology: Theory and Practice.* 1st ed. Boca Raton, Florida, United States: CRC Press.

Kumar, L., Krishna, A. & Rath, S., 2017. The impact of feature selection on maintainability prediction of service-oriented applications. *Service Oriented Computing and Applications ,* Volume SOCA 11, p. 137–161.

Lewis, J. & Neumann, U., 2016. *Performance of Java versus C++.* [Online]   
Available at: http://www.scribblethink.org/Computer/javaCbenchmark.html  
[Accessed 1 May 2020].

Light, M., 2009. *How the Waterfall Methodology Adapted and Whistled,* Stamford, Connecticut, United States: Gartner.

Lyon, D., 2012. The Java Tree Withers. *Computer,* 45(1), pp. 83-85.

Mahalakshmi, M. & Sundararajan, M., 2013. Traditional SDLC Vs Scrum Methodology – A Comparative Study. *International Journal of Emerging Technology and Advanced Engineering,* 3(6), pp. 192-196.

Mahapatra, H. & Goswami, B., 2015. Selection of Software Development Methodology (SDM): A Comparative Approach. *International Journal of Advanced Research inComputer Science and Software Engineering,* 5(3), pp. 58-61.

Mak, S. & Bakker, P., 2017. *Java 9 Modularity.* 1st ed. Sebastopol, CA: O'Reilly Media Inc..

Meng, N. et al., 2018. *Secure coding practices in Java: challenges and vulnerabilities.* Gothenburg, Sweden, ICSE '18: Proceedings of the 40th International Conference on Software Engineering.

Mergel, I., Gong, Y. & Bertot, J., 2018. Agile government: Systematic literature review and future research. *Government Information Quarterly,* 35(2), pp. 291-298.

Moliavko, O., Drozdovskyi, T., Petrychenko, V. & Kopysov, O., 2019. uJVM: Lightweight Java Virtual Machine for embedded systems. *The Journal of Open Source Software,* 4(36), p. 1338.

Nor, M., Jalaldeen, M., Razi, M. & Zakaria, A., 2018. Cloudemy: Step into the Cloud. *Journal of Telecommunication, Electronic and Computer Engineering,* 9(3-5), pp. 135-139.

Ogihara, M., 2018. Interfaces, Inheritance, and Polymorphism. In: *Fundamentals of Java Programming.* Berlin, Germany: Springer, pp. 427-455.

Oracle, 2019. *Oracle Java SE Licensing FAQ.* [Online]   
Available at: https://www.oracle.com/technetwork/java/javase/overview/oracle-jdk-faqs.html  
[Accessed 1 May 2020].

Parlante, N., 2000. *Pointers and Memory,* Stanford, CA: Stanford Computer Science Education Library.

Sarker, I. & Apu, K., 2014. MVC Architecture Driven Design and Implementation of Java. *International Journal of Hybrid Information Technology,* 7(5), pp. 317-322.

Schildt, H., 2018. *Java: The Complete Reference.* 11th ed. New York, United States: McGraw-Hill Education.

Sciore, E., 2019. *Polymorphism.* Berkeley, CA: Apress.

Shaffi, A. & Al-Obaidy, M., 2013. Analysis and comparative study of traditional and web information systems development methodology (WISDM) towards Web development applications. *International Journal of Emerging Technology and Advanced Engineering,* 3(11), pp. 277-282.

Singh, S. & Iyer, J., 2016. Comparative Study of MVC (Model View Controller) Architecture with respect to Struts Framework and PHP. *International Journal of Computer Science Engineering (IJCSE) ,* 5(3), pp. 142-150.

Sommerville, I., 2016. *Software Engineering.* 10th ed. Harlow, UK: Pearson Education.

Sulír, M., 2020. *String Representations of Java Objects: An Empirical Study.* Limassol, Cyprus,, SOFSEM 2020: Theory and Practice of Computer Science.

Su, P., Wang, Q., Chabbi, M. & Liu, X., 2019. *Pinpointing performance inefficiencies in Java.* Tallinn, Estonia, ESEC/FSE 2019: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

Sweeney, M., 2014. *Agile vs Waterfall: Which Method is More Successful?.* [Online]   
Available at: https://clearcode.cc/blog/agile-vs-waterfall-method/  
[Accessed 1 May 2020].

Tempero, E., Yang, H. & Noble, J., 2013. *What Programmers Do with Inheritance in Java.* Montpellier, France, European Conference on Object-Oriented Programming, Springer.

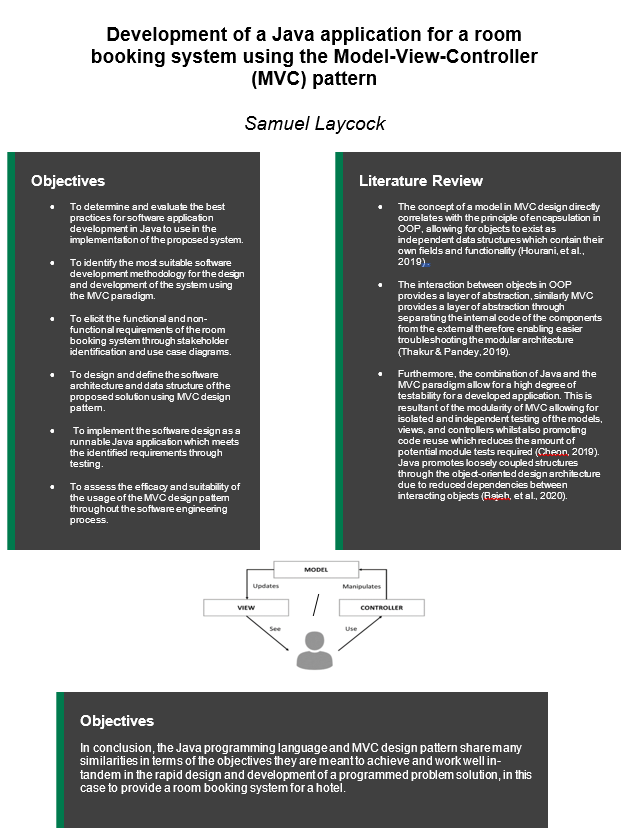
Thakur, R. & Pandey, U., 2019. The Role of Model-View Controller in Object Oriented Software Development. *Nepal Journal of Multidisciplinary Research,* 2(2), pp. 1-6.

Wiegers, K. & Beatty, J., 2013. *Software Requirements.* 3rd ed. Washington, United States: Microsoft Press.

Yusop, O. et al., 2019. Experiences in Software Development Model Selection and MVC Design Pattern Implementation for Kuala Lumpur City Hall Traffic Offence Management System. *Open International Journal of Informatics (OIJI) ,* 7(1), pp. 26-36.

# Chapter 10 – Appendices

## Appendix A – Poster



## Appendix B – Questionnaire

**Interview Questionnaire for Dissertation “Development of a Java application for a room scheduling system using the Model-View-Controller (MVC) pattern”**

|  |  |
| --- | --- |
| **Participant Name:** | Sharon Holland |
| **Date:** | 17/04/2020 |

**Disclaimer**

This interview questionnaire is being conducted for the purposes of sourcing primary data for the elicitation of software requirements for Samuel Laycock’s dissertation at Edge Hill University.

The steps following describe how the rights of the participant were expressed to them, the purpose and reasoning behind the exercise, and the consent of the participant.

* The results of this interview will only be used by the interviewer (Samuel Laycock) for the derivation of requirements for creating a software solution. This interview, along with its results and transcription, will only be observed by the interviewer and the supervising staff of the Department of Computing at Edge Hill University. They will not be shared outside of these boundaries without the explicit consent of the participating interviewee.
* The results of this interview will be stored and processed following the regulations imposed by GDPR and the Data Protection Act regulations. The results of the interview will be only used in the methods specified above and will only be stored for the time period that they are necessarily needed and will only be accessed by the specified parties.
* The participant holds the right to withdraw from the interview process at any time and revoke their consent for their participation to be used. They can achieve this by contacting the dissertation author.
* The participant holds the right to refuse to answer any questions posed by the interviewer at any time.
* The purpose of this questionnaire is to elicit the requirements to develop a Java application. The application will be used to create a program which implements the Model-View-Controller design paradigm into the structure of its software.

**Question 1.**

|  |  |
| --- | --- |
| **Interviewer:** | *“In a brief explanation, can you explain what you want the system to be able to do?”* |
| **Participant:** | *“I want to be able to manage the bookings for a hotel using some sort of software. I want to be able to manage and view each of the rooms I have in the building. I want to be able to take bookings, be it in person or over the phone, from customers.*  *I want the program to let me update and delete bookings, in case of any errors were added to the booking or if they customer needed to cancel a previous booking they had made.*  *If it’s possible, I want customers to be able to book multiple rooms at once rather than making lots of individual bookings as it would make my life easier, I think.”* |

**Question 2.**

|  |  |
| --- | --- |
| **Interviewer:** | *“Do you have any preferences, in regard to technologies used, in the creation of the application solution?”* |
| **Participant:** | *“Not really, I don’t know enough about them to really comment on that. I just want some sort of database which can store the information of the system which I can add to, update, or delete information about the bookings from it.*  *I just want a stable system which I can easily use without much hassle to manage the business. If it’s possible again, to be able to do it on other devices such as a tablet would make my life easier, so I don’t have to be sat at a desk all the time.”* |

|  |  |
| --- | --- |
| **Participant Signature:** | Sharon Holland |
| **Date:** | 02/05/2020 |

## Appendix C – Use Cases

|  |  |
| --- | --- |
| **Room Booking System – Add Room** | |
| **Actor(s):** | System User |
| **Description:** | Add a new room to the system’s database. |
| **Data:** | Room characteristics. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which inserts user input into database as a new record. |

|  |  |
| --- | --- |
| **Room Booking System – List Rooms** | |
| **Actor(s):** | System User |
| **Description:** | Loads a table with a list of room records from system database. |
| **Data:** | N/A |
| **Stimulus:** | Opening Room list form |
| **Response:** | Gets the room records from database and inputs them into a table to be displayed to the user. |

|  |  |
| --- | --- |
| **Room Booking System – Update Room** | |
| **Actor(s):** | System User |
| **Description:** | Updates the features of an existing room in the database. |
| **Data:** | Room characteristics. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which updates the room record of the room which is being updated. |

|  |  |
| --- | --- |
| **Room Booking System – Delete Room** | |
| **Actor(s):** | System User |
| **Description:** | Deletes an existing room from the database. |
| **Data:** | Room identifier. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which deletes a specified room from the database. |

|  |  |
| --- | --- |
| **Room Booking System – Create Booking** | |
| **Actor(s):** | System User |
| **Description:** | Creates an overall booking record for a customer so they can make individual room bookings. |
| **Data:** | Customer details, booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which creates a booking record. |

|  |  |
| --- | --- |
| **Room Booking System – Delete Booking** | |
| **Actor(s):** | System User |
| **Description:** | Deletes an existing booking record from the database, for whatever reason. |
| **Data:** | Booking identifier |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which deletes a specific booking record |

|  |  |
| --- | --- |
| **Room Booking System – Add Customer to Booking** | |
| **Actor(s):** | System User |
| **Description:** | Customer record added when upon the creation of a booking. |
| **Data:** | Customer details, booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which creates a customer record |

|  |  |
| --- | --- |
| **Room Booking System – Load Existing Customer to Booking** | |
| **Actor(s):** | System User |
| **Description:** | Previous customer details are loaded to make a new booking |
| **Data:** | Customer identifier |
| **Stimulus:** | Button press by user |
| **Response:** | Customer details are loaded into the booking. |

|  |  |
| --- | --- |
| **Room Booking System – Add Room to Booking** | |
| **Actor(s):** | System User |
| **Description:** | Add a Room to a Booking. |
| **Data:** | Room details, room booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | Room booking record is created, displayed in the booking. |

|  |  |
| --- | --- |
| **Room Booking System – Add Multiple Rooms to Booking** | |
| **Actor(s):** | System User |
| **Description:** | Add multiple rooms to a singular booking. |
| **Data:** | Room details, room booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | Room booking record are created, displayed in the booking in a table. |

|  |  |
| --- | --- |
| **Room Booking System – Delete Room from Booking** | |
| **Actor(s):** | System User |
| **Description:** | Delete a room from a booking. |
| **Data:** | Room identifier, booking identifier. |
| **Stimulus:** | Button press by user |
| **Response:** | Room booking is removed from table, removed from database. |

|  |  |
| --- | --- |
| **Room Booking System – Update Booking** | |
| **Actor(s):** | System User |
| **Description:** | Update the details of an existing booking. |
| **Data:** | Booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | SQL statement which updates the fields of a specific booking record. |

|  |  |
| --- | --- |
| **Room Booking System – Add Room to Existing Booking** | |
| **Actor(s):** | System User |
| **Description:** | Add additional rooms to an existing booking. |
| **Data:** | Booking details, room booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | Room booking added to database, appears in the booking’s list of rooms. |

|  |  |
| --- | --- |
| **Room Booking System – Delete Room from Existing Booking** | |
| **Actor(s):** | System User |
| **Description:** | Delete rooms from an existing booking. |
| **Data:** | Booking details, room booking details. |
| **Stimulus:** | Button press by user |
| **Response:** | Room booking is removed from the booking in the database, removed from the booking’s list of rooms. |

|  |  |
| --- | --- |
| **Room Booking System – List all Bookings** | |
| **Actor(s):** | System User |
| **Description:** | Displays a list of all existing bookings. |
| **Data:** | N/A |
| **Stimulus:** | Open booking list form |
| **Response:** | Gets the booking records from database and inputs them into a table to be displayed to the user. |

## Appendix D – Testing Plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Title** | **Test Description** | **Requirement Addressed** | **Test Data** | **Result** |
| T-001 | Navigate Menu | The user should be able to navigate to the other forms to access the rest of the functionality of the application. | N/A | Load the application and click the navigational buttons. | The GUI forms open as expected. |
| T-002 | Add Room | The user should be able to add a new Room record to the system. | FR1 | **ROOMNUMBER:**  “Room 1”  **NUMBER OF BEDS:**  2  **ROOM PRICE:**  99.99  **FLOOR NUMBER:**  “Ground”  **DESCRIPTION:**  “Deluxe Double”  **ENSUITE:**  TRUE | Room added to the Room table. |
| T-003 | List Rooms | The user should be able to view a form which lists all the existing rooms in the database. | FR4 |  | Current rooms in system are correctly displayed in the table. |
| T-004 | Load Room | The user should be able to click on one of the rooms listed and load a form containing all the selected Room’s fields for updating or deletion. | FR2, FR3, FR4 |  | By clicking in the room in the room list, the manage room form opened and populated the selected rooms’ details into the fields correctly. |
| T-005 | Update Room | The user should be able to update the fields of an existing Room. | FR2 | **ROOMNUMBER:**  “Room 1”  **NUMBER OF BEDS:**  3  **ROOM PRICE:**  199.99  **FLOOR NUMBER:**  “First”  **DESCRIPTION:**  “Standard Triple”  **ENSUITE:**  FALSE | By changing the contents of the fields and clicking “Update Room”, the room’s fields were updated in the database to reflect the changes made by the user. |
| T-006 | Delete Room | The user should be able to delete an existing Room record. | FR3 | N/A | The currently selected room record was deleted from the database. |
| T-007 | Add Room Bookings from new Booking | The user should be able to add room bookings to a new booking. | FR8, FR9 | Several Rooms were re-added to the system to test further functionality.  **ROOMNUMBER:** “Room 1”  **NUMBER OF BEDS:** 2  **ROOM PRICE:** 99.99  **FLOOR NUMBER:** “Ground”  **DESCRIPTION:** “Deluxe Double”  **ENSUITE:** TRUE  **ROOMNUMBER:** “Room 2”  **NUMBER OF BEDS: 1**  **ROOM PRICE: 7**9.50  **FLOOR NUMBER:** “Ground”  **DESCRIPTION:** “Master Suite”  **ENSUITE:** TRUE  **ROOMNUMBER:** “Room 3”  **NUMBER OF BEDS**: 4  **ROOM PRICE:** 129.00  **FLOOR NUMBER:** “First”  **DESCRIPTION:** “Family Style”  **ENSUITE:** TRUE  **ROOMNUMBER:** “Room 4”  **NUMBER OF BEDS:** 2  **ROOM PRICE:** 69.69  **FLOOR NUMBER:** “First”  **DESCRIPTION:** “Economy Twin”  **ENSUITE:** FALSE    The Room 2 and Room 4 were added to the new Booking.  **ROOMNUMBER:** “Room 2”  **GUESTNUMBER:** 2  **CATERED:** TRUE  **ROOMNUMBER:** “Room 4”  **GUESTNUMBER:** 2  **CATERED:** FALSE | Room bookings are made for Room 2 and Room 4, which appear in the list of room bookings when creating a booking. |
| T-008 | Delete Room Booking to new Booking | The user should be able to delete a room booking from a new booking. | FR10 | Delete the Room 2 Booking. | Room 2 booking is removed from the list of new booking. |
| T-009 | List Room Bookings in new Booking | The user should be able to see a list of Room Bookings in the new Booking. | FR9 | **ROOMNUMBER:** “Room 2”  **GUESTNUMBER:** 2  **CATERED:** TRUE | Room 4 booking appears in the list of Room Bookings for the booking as expected. |
| T-010 | Add Customer to new Booking | The user should be able to input a customer’s details to a new booking. | FR6 | **MOBILE NUMBER:** “07572333222”  **FIRST NAME:** “David”  **LAST NAME:** “Smoke”  **EMAIL:** “sdav@email.com” | New Customer details can be entered the form as expected. |
| T-011 | Add Booking | The user should be able to finalise and create a new booking. | FR5 | **MOBILE NUMBER:** “07572333222”  **FIRST NAME:** “David”  **LAST NAME:** “Smoke”  **EMAIL:** “sdav@email.com”  **ARRIVAL DATE:** 01/05/2020  **DEPARTURE DATE:** 08/05/2020  **ROOMNUMBER:** “Room 2”  **GUESTNUMBER:** 2  **CATERED:** TRUE | As expected, a new booking record was created, associated with a customer and room booking records. |
| T-012 | Fetch existing Customer | The user should be able to fetch the details of a previous customer given their mobile number. | FR7 | Insertion of a new booking so that the customer can be recalled.  **MOBILE NUMBER:** “07572140326”  **FIRST NAME:** “Samuel”  **LAST NAME:** “Laycock”  **EMAIL:** “laycock@email.com”  **ARRIVAL DATE:** 09/05/2020  **DEPARTURE DATE: 11**/05/2020  **ROOMNUMBER:** “Room 3”  **GUESTNUMBER: 1**  **CATERED:** TRUE | As expected,  the details of Samuel Laycock were called by entering the mobile number and clicking the “Check existing customer” button. |
| T-013 | List Bookings | The user should be able to view a form which lists all the existing bookings in the database | FR15 | N/A | As expected, clicking Manage Bookings on the menu generated a list of all current bookings. |
| T-014 | Load Booking | The user should be able to click on one of the rooms listed and load a form containing all the selected Booking’s fields for updating or deletion, as well as the room bookings related to that record. | FR11, FR12, FR13, FR14 | Click on Booking number 1, should load the booking, customer, and room booking details:  **MOBILE NUMBER:** “07572140326”  **FIRST NAME:** “Samuel”  **LAST NAME:** “Laycock”  **EMAIL:** “laycock@email.com”  **ARRIVAL DATE:** 09/05/2020  **DEPARTURE DATE: 11**/05/2020  **ROOMNUMBER:** “Room 3”  **GUESTNUMBER: 1**  **CATERED:** TRUE | As expected, clicking on the booking number 1 record in the table prompted a “manage booking” form with the populated data of the selected booking. |
| T-015 | Add Room Booking to existing Booking | The user should be able to add a room booking to an existing booking. | FR12 | Add an additional booking to the previously created and loaded booking, booking number 1:  **BOOKING NUMBER:** 1  **ROOMNUMBER:** “Room 1”  **GUESTNUMBER:** 2  **CATERED:** FALSE | As expected, adding a new room booking to an existing record added a new room booking in the database and updated the list of current room bookings on the room booking form. |
| T-016 | Delete Room Booking from existing Booking | The user should be able to delete a room booking from an existing booking. | FR13 | Delete “Room 3” room booking in Booking 1. | As expected, the Room 3 room booking was deleted from the database for booking 1 and was removed from booking 1’s list of room booking in the manage booking form. |
| T-017 | List Room Bookings in existing Booking | The user should be able to see a list of Room Bookings in an existing Booking. | FR11, FR15 | Click on Booking number 1, should load the booking, customer, and room booking details:  **MOBILE NUMBER:** “07572140326”  **FIRST NAME:** “Samuel”  **LAST NAME:** “Laycock”  **EMAIL:** “laycock@email.com”  **ARRIVAL DATE:** 09/05/2020  **DEPARTURE DATE: 11**/05/2020  **ROOMNUMBER:** “Room 3”  **GUESTNUMBER: 1**  **CATERED:** TRUE | As expected, clicking on the booking number 1 record in the table prompted a “manage booking” form with the populated data of the selected booking. |
| T-018 | Update Booking | The user should be able to update the fields of an existing Booking. | FR11 | Update the dates of the Booking Number 2 booking record.  **BOOKING NUMBER:** 2  **ARRIVALDATE:** 01/05/2020 -> 21/05/2020  **DEPARTUREDATE:** 15/05/2020 -> 31/05/2020 | As expected, the dates of the booking were updated in the database which can be seen through the view bookings menu additionally. |
| T-019 | Delete Booking | The user should be able to delete an existing Booking record. | FR14 | Deletion of Booking number 2 Record. | As expected, the booking record was deleted from the database and no longer appears in the table displaying all the booking records. |