BookKeeper - Project Description and Requirements Specification Document

Project:

Campus Booking System

Prepared for:

Lori Hogan

Software Engineering Instructor

CP3490

Team Members:

Justin Feehan, Morgan Feehan

Date:

October 14th 2025

1.0 - Introduction

1.1 - Purpose

- This document outlines the requirements and expectations of the BookKeeper application. BookKeeper is a software tool designed with the goal of aiding both students and staff of College of the North Atlantic in creating and managing room bookings for on-campus events.

1.2 - Scope

- BookKeeper is a program designed as a school project. While it is ultimately intended to be a functional program, it is not designed with the intention of large-scale adoption in a professional manner. The scope of this project will be limited by its primary intentions of being learning experience for the team developing it.

1.3 - Definitions and Acronyms

- BookKeeper - The application being developed.
- CNA - College of the North Atlantic
- GUI - Graphical User Interface
- Admin - Administrator

2.0 - Project Overview

2.1 - Product Goals

- Bookings that can be created, edited, or deleted by the user that created them.
- Users are able to view details of bookings created by others.
- Bookings will automatically avoid overlapping based on room number and desired time slots.
- Users are able to view existing bookings using a GUI.
- Administrative users are able to manage and modify created bookings.

2.2 - Product Features

- User account creation
- Booking creation / modification GUI
- Booking information GUI (For view other user's bookings)
- Existing Bookings Schedule view GUI
- Administrative user GUI (Allows administrative users to modify existing bookings)

2.3 - Target Users and Personas

- CNA Students - Able to view booking information
- CNA Staff - Able to create event bookings
- System Administrators - Able to modify existing bookings

2.4 - Assumptions and Constraints

- BookKeeper will not be properly implemented in a professional manner, and will be developed as a proof-of-concept student project.
- User account creation, management, and security will be limited, as outlined in section 1.2.

3.0 - Functional Requirements

3.1 - User Account Creation

- Users must be able to create accounts with a username and password.
- Users accounts must be defined as Student, Staff, or Admin accounts.
- Users must be able to log in and log out of their accounts.

3.2 - User Account Management

- Users must be able to delete their own account.
- Users must be able to change their own password.
- Administrative users must be able to change the passwords and delete the accounts of any existing user.

3.3 - Booking Creation

- Staff and Admin users must be able to create bookings.
- Bookings must account for room number and time to avoid overlapping.
- Bookings should include Room Number, Time Start, Time End, Creation Date, Booking Author, and Event Name.

3.4 - Booking Management

- Staff accounts must be able to view, modify, and delete event bookings that they have created in the past.
- Admin accounts must be able to view, modify, and delete event bookings created by any user in the past.

3.5 - Booking Schedule View

- A button should allow users to open a schedule view GUI.
- All accounts should be able to access the schedule view GUI.

- The schedule view GUI will display individual bookings according to time and room number.
- Users must be able to click on event bookings to view information about the booking.
- Staff users should be able to modify and delete their own bookings using schedule view.
- Administrative users should be able to modify and delete all bookings using schedule view.

3.6 - Administrative User GUI

- Admin users should be able to open an Admin Actions GUI
- The Admin Actions GUI should let admin users view and modify existing user accounts.

4.0 - Non-Functional Requirements

4.1 - Performance Requirements

- User / Booking information should be loaded within 2 seconds of logging in
- Additional GUIs (Schedule GUI and Admin GUI) should be loaded within 5 seconds of attempting to open them

4.2 - Usability Requirements

- BookKeeper should be GUI based, allowing for simple user interaction
- BookKeeper should be primarily controlled using the mouse pointer, only requiring keyboard input when entering information.
- Users should be able to create a booking within 2 minutes of finalizing account creation.

4.3 - Reliability Requirements

- BookKeeper should automatically save booking data after creation / modification to minimize unexpected data loss.
- User confirmations should be implemented when creating or modifying data to minimize errors in information entry. (i.e. “Are you sure?” textboxes)

4.4 - Security Requirements

- User passwords should be encrypted.
- Password recovery should only be performed by trusted admin users.

4.5 - Scalability Requirements

- BookKeeper should be able to accommodate as many user accounts as is needed.
- Bookings that have ended should be erased after 30 days to save on available space.
- BookKeeper should be designed in such a way to ensure modularity, allowing for future improvements and modifications to the program's code.

5.0 - Environmental Requirements

- A computer capable of running the Java Virtual Environment and the BookKeeper Program.
- Sufficient storage space for users to be able to download and install the program.
- Sufficient storage space for the program to store newly created user information and booking information.

6.0 - Software Requirements

6.1 - Development Environment

- Programming Language: Java
- Integrated Development Environment (IDE): IntelliJ Idea

6.2 - Operating System Requirements

- Compatible with Windows 10 or later
- Optional compatibility with MacOS or Linux using the Java Virtual Environment

6.3 - Installation Requirements

- Java Runtime Environment installed on the user computer
- Sufficient user permissions to read / write to application critical directories

7.0 - Potential Risks

7.1 - Technical Risks

- Limited Security implementation could increase the risk of security related issues.

- Unsaved data could be lost or corrupted in the event of unexpected crashes or program closures.
- Bugs or unintended behaviours could arise due to lack of experience in the development team.

7.2 - Project Management Risks

- Development team inexperience could lead to features needing to be cut or scaled down to meet required deadlines.
- Limited time for quality assurance testing could lead to an increase in undiscovered bugs.
- Miscommunications between team members could create issues in scheduling and planning of the project.

7.3 - Operational Risks

- Inexperienced users or administrators could accidentally delete or modify existing bookings without intending to.
- New users may struggle with GUI systems if they are not designed efficiently.
- The program may encounter unexpected issues on older systems, or systems with unexpected changes to the Java Virtual Environment.

8.0 - Time Budget

Phase	**Estimated Duration**	**Estimated Completion**
Software Requirement Specification	-	October 14
System Design	2 Weeks	October 28
Environment Setup	1 Week	November 4
Core Functionality Development	2 - 4 Weeks	November 25
Testing and Debugging	1 Week	December 1
Final Implementation and Documentation	1 - 2 Weeks	December 5

9.0 - Future Considerations

- Considerations should be given towards future modifications and improvements to the BookKeeper program. The program should be designed

modularly to ensure that future additions can be made without risking the core functionality of the program.
- Further considerations into a wide-scale implementation of the program for official use may be made at a future date.

10.0 - Glossary

- **Admin (Administrator)** - User role with permissions to manage all bookings and user accounts.
- **Booking** - A record representing a reserved room and time slot.
- **GUI (Graphical User Interface)** - A graphical representation of BookKeeper's data that allows users to interact and change said data.
- **Java Virtual Environment / Java Runtime Environment** - The virtual environment through which applications developed in Java can be executed.