Sprint 1 – Development Requirements

Space Reservation System

Contents

1 Overview 1
2 Use of AI 1
3 System Requirements 2
4 Development Requirements 2
4.1 Project Management 2
4.2 Testing 3
4.3 Coding 3
4.4 Version Control 4
4.5 Individual Team Members 4
5 Time Log Instructions 4
5.1 *Individual Hours* Tab 4
5.2 *Activity Codes* Tab 5
5.3 *Time Log* Tab 5

1 Overview

You will build a stand-alone desktop application in Java (JavaFX) based on the user stories that have been provided while adhering to the constraints and requirements in the next four sections.

The focus of this project is on software engineering practices including object-oriented design, testing, version control, and team collaboration.

Read this document carefully, preferably as a Team.

2 Use of AI

In this sprint, you are allowed to use AI as a development assistant. However, each team member is responsible for understanding, testing, modifying, and explaining any AI-generated code included in the project. The goal is to use AI as a development assistant, not a replacement developer. Teams must be able to explain the design and implementation of all submitted code during demonstrations and code reviews.

An incremental approach will lead to a better system

AI works best when you use it to build small pieces that you understand and connect together. If you ask it to build everything at once, you'll get code that you don't understand, can't fix, and can't extend without making it worse.

The recommended workflow is:

Step 1: Understand the User Story – Read the user story and acceptance tests carefully. Identify inputs, outputs, and constraints before using AI.

Step 2: Break the Story Into Pieces – Do not ask AI to build entire features. Break work into components such as model, logic, UI, and persistence.

Step 3: Ask Targeted Questions – Ask focused questions such as how to implement a class, check time overlap, or create a JavaFX form. Avoid asking for complete systems.

Step 4: Review and Modify Code – Carefully read all generated code. Rename, simplify, and remove anything you do not understand.

Step 5: Integrate Into Your System – Ensure the code fits your MVC structure and works with existing components. Do not paste code randomly.

Step 6: Test Everything – Write JUnit tests for normal, edge, and invalid cases. Verify behavior matches acceptance tests.

Step 7: Refactor – Improve structure, remove duplication, and ensure proper separation of concerns.

3 System Requirements

1. The system is written as a stand-alone JavaFX desktop application.
2. The system is object-oriented (OO) and should utilize best practices for OO design.
3. The system should utilize the MVC architecture (Model, View, Controller). Controllers coordinate system operations and user requests. Models contain application data and business rules. Views are responsible only for presentation and user interaction.
4. The system utilizes JSON files for persistence. Data is loaded into memory as objects while the application is running. The system does not utilize a database.

4 Development Requirements

4.1 Project Management

5. You will use a GitHub *Project* to manage your development. Name your *View*: *Sprint 1.* Use *Board* layout (see *View* button on the right). You will have these columns: *To Do, Doing, Done.* You can add columns if it helps your team.
6. You will break work down into small tasks that 1-2 people will work on (some tasks may involve the whole group). These tasks should be estimated to take between 0.5 hours and 4 hours in length.
   - **ALL tasks must be entered in your GitHub Project as Issues. Do not do work on the project without a task.**
   - Reference the Issue in commits and Pull Requests. For example:

     Issue #15: Create Reservation validation

     Branch:
     feature/reservation-validation

     Commit message:
     Added duration validation for reservations (#15)

     Pull Request:
     Implements reservation validation rules.
     Closes #15
   - *Issues* are created in GitHub and added to the *To Do* column. From there, they will be moved to a *Doing* column and finally to the *Done* column.
   - Work must be distributed across all team members. No team member should rely solely on meetings, documentation, or testing activities as their primary contribution.
   - Sometimes a task may be to figure out what needs to be done, in other words, a planning session. The conclusion of that task will generally result in concrete tasks that contribute to the development.

- Sometimes a task may be a placeholder of sorts, broad, open-ended, just so you don't forget about it. Then, when ready to address it, it might be broken down into concrete tasks. In this case, you would remove the initial task once it has been broken down.
- Task/Issue breakdown is iterative. Your group will identify some initial tasks to get started on development. As issues are in progress or completed you will learn more about what you need to do. If a task turns out to be much bigger, or involve things you didn't initially think of, then simply add them as new tasks, possibly closing out the initial task. Thus, Task/Issue breakdown is ongoing.

4.2 Testing

7. **Unit Testing Requirement** – You are required to implement unit tests (using JUnit) for classes containing business logic. View classes and simple data containers without logic generally do not require unit tests. Unit tests should target methods that contain logic (e.g., conditionals, loops, calculations), rather than simple getters, setters, or constructors without logic. Each test should verify that the method behaves correctly under normal conditions, as well as under edge cases and invalid inputs ("sad paths") where the method should handle errors appropriately.

8. **Integration Testing Requirement** – You are required to implement integration tests using JUnit that correspond to acceptance tests involving business logic (as opposed to presentation/UI behavior). Integration tests should not interact directly with JavaFX user interface components. Instead, integration tests should invoke controller methods and verify collaboration among multiple application classes. Integration tests must be placed in separate test classes from your unit tests. Each test method must include a preceding comment identifying the related User Story and Acceptance Test, for example:

   ```
   // US-3, AT-2.
   ```

9. **System Testing Requirement** – You are required to perform system testing by validating the acceptance tests through manual execution of the application. System testing focuses on verifying that the fully integrated system behaves correctly from a user's perspective, including workflows that involve multiple components and user interactions. As part of your submission, you must demonstrate selected acceptance tests during your video demo. During the demonstration, you should clearly identify the User Story and Acceptance Test being tested and show the steps taken along with the resulting system behavior.

4.3 Coding

10. Follow Java best practices and naming conventions and proper OO design. Avoid using static methods for business logic when an object-oriented design is more appropriate.

11. Adhere to MVC design principles as described in Section 3.

12. Persistence must use JSON files.

13. Persistence code must be placed in separate classes and should not be implemented inside model classes. Persistence classes should only be responsible for saving and loading data.

4.4 Version Control

14. Only working and tested code may be merged into the main branch.
15. Your code base must be in packages, such as: model, view, controller, persistence, utility, etc.
16. Use feature branches; do not commit directly to main. For example, branch names are similar to:
    - feature/short-description
    - bugfix/short-description
    - refactor/controller-cleanup
17. All pull requests must be reviewed and approved by at least one team member other than the author before merging.
18. Commit frequently. Each commit should represent a small, meaningful unit of work and include a descriptive commit message.

4.5 Individual Team Members

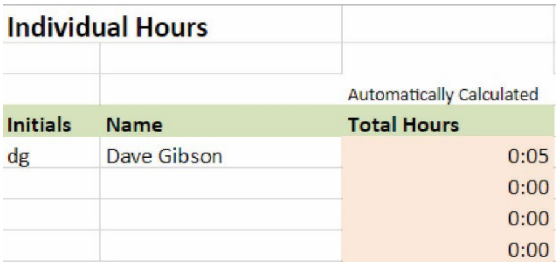
19. All time spent on the project is entered within 48 hours of performing the work into a spreadsheet I will share with your group. You must provide a meaningful description of your work for each entry (1-3 sentences, usually). Directions for the Time Log are in Section 5 (below).
20. Work should be distributed reasonably throughout the sprint rather than concentrated near the deadline. In other words, you should not do a minimal amount of work one week, and then double the next week. This ensures that project is not held back.
21. Each member must contribute meaningful GitHub activity, such as commits, pull request reviews, issue creation/management, testing artifacts, documentation updates, or other project work observable through GitHub.

5 Time Log Instructions

You will record all time that you spend for Sprint 1 using the Time Log Google Sheet that was shared with each member of your team. The spreadsheet has three tabs.

5.1 *Individual Hours* Tab

The **second** tab is "Individual Hours" as shown below. The first time you use the spreadsheet, you should type in your initials and name (the first person should replace my entry). Your total hours will be automatically calculated. Do not alter that field.



Individual Hours		
		Automatically Calculated
Initials	**Name**	**Total Hours**
dg	Dave Gibson	0:05
		0:00
		0:00
		0:00

5.2 *Activity Codes* Tab

The **third** tab is "Activity Codes" as shown below. You will use one of these codes to allocate each increment of time that you contribute to the project.

Activity Codes

- You will record all time (including thinking, meetings, etc) that you spend for this project using the Time Log tab on this spreadsheet. Use the Activity Codes below to indicate how your time was spent.
- Choose the code that best represents the *primary activity* during that time period. You do not need to split time across multiple codes for a single session.

Code	Activity
A	Analysis & Understanding (user stories, requirements, thinking, AI exploration)
I	Implementation (writing, integrating, or modifying code — including AI-assisted coding)
V	Verification (testing, debugging, validation, checking acceptance tests)
R	Refactoring & Improvement (cleaning code, restructuring, fixing design issues)
M	Meetings & Coordination (team discussions, planning, standups)
P	Presentation Preparation (demo, video, required documentation (if any))

5.3 *Time Log* Tab

The **first** tab is "Time Log" as shown below.

- Each time you work on the project you will make an entry by supplying: begin date and time, end date and time, interrupt time, activity code, and description of what you did.
- **For *Description of Activity,* provide enough detail so that the instructor knows exactly what you are working on. 1-3 sentences should usually suffice.**
- The total time and running total time will be automatically calculated.
- The formulas can get corrupted if you delete a row. You can fix it yourself if you can, or I can fix it.
- The date/time field is formatted; however, you don't have to enter these exactly as shown. For example, for the first entry shown below, if you type: "6/5 9 am" will produce the result shown.
- Interrupt time (minutes) is time you were not working. For example, you may have worked from 11 am to 1 pm, and taken 20 minutes for a snack. Thus, for interrupt time you would enter: "0:20". For this to work properly, you must enter "h:mm".
- For group meetings, each person should enter their time

Time Log							
				Automatically Calculated			
Initials	**Beg Date/Time**	**End Date/Time**	**Int.Time**	**Tot Time**	**Run Total**	**Act Code**	**Description of Activity**
	06/05 09:00 AM	06/05 09:00 AM	0:00	0:00	0:00	A	
dg	06/05 09:00 AM	06/05 09:06 AM	0:01	0:05	0:05	A	Project overview in class