Project – Getting Started Guide

Contents

1 Step 1: Create the Team Repository (One Person Only) 1
2 Step 2: Everyone Clones and Verifies Setup 2
3 Step 3: Read All User Stories (As a Team) 2
4 Step 4: Break Three User Stories into Tasks 2
5 Step 5: Start Development (One Task at a Time) 3
6 Step 6: Use Pull Requests 3
7 Step 7: Work Iteratively 3
8 Step 8: Track Your Time 3
Appendix 1 How to Analyze a User Story 4

This guide walks your team through exactly what to do first and how to begin turning user stories into working software.

1 Step 1: Create the Team Repository (One Person Only)

One team member is responsible for the initial setup of the project.

1. Create the following package (folder) structure inside the project:

```
- model
- view
- controller
- persistence
- test
```

2. Add a simple starter class:

```
public class Reservation {
    public static void main(String[] args) {
        System.out.println("Sprint project started");
    }
}
```

3. Commit and push this initial project

At this point:

- Everyone should be able to clone the repo
- The project should compile and run
- You have a clean starting point

2 Step 2: Everyone Clones and Verifies Setup

Each team member should:

1. Clone the repository
2. Open it in your IDE (e.g., IntelliJ)
3. Run the Reservation class
4. Confirm the message prints successfully

If you cannot run the program, fix setup issues now before moving forward.

3 Step 3: Read All User Stories (As a Team)

Before coding anything:

- Read all user stories and acceptance tests together.
- Make sure everyone understands the overall purpose of the system.
- Identify the major features and how the user stories relate to one another.
- Make sure everyone understands the inputs, outputs, and constraints described in the stories.

Do not start designing, creating tasks, or writing code yet.

4 Step 4: Break Three User Stories into Tasks

Start with an MVP (Minimum Viable Product), meaning a simple working version of your system before adding complexity. Focus first on a basic working flow (e.g., view spaces and create a simple reservation), then add validation rules and additional features incrementally. Do NOT build everything at once. I would recommend:

1. Select three relatively small user stories to begin development. Or, ask AI what you should start with. The selected stories should establish the core structure of the system and avoid unnecessarily complex logic early in development.
2. Analyze the first story using the Appendix .
3. Break the user story into tasks that take 0.5-4 hours using AI. Load the Development Requirements into your AI agent so that it knows the type of system you are building and how testing takes place, *etc*.
4. Create each task as a GitHub Issue and add to the Project Board in the “To Do” column.
5. Repeat for the other selected stories.
6. Assign work to team members.

5 Step 5: Start Development (One Task at a Time)

For each task:

1. Pick a task from *To Do*
2. Move it to *Doing*
3. Review the related User Story, Acceptance Tests, and Issue description.
4. Create a branch: feature/short-description
5. Implement the task
   - If using AI, provide the Development Requirements (might already be loaded) and relevant User Story so that recommendations align with the project constraints and architecture.
6. Write or update tests.
7. Verify the task satisfies the relevant acceptance tests.
8. Commit

6 Step 6: Use Pull Requests

1. Push your branch to GitHub
2. Create a Pull Request. Reference the related Issue in the Pull Request
3. Get a an approval from at least one team member other than the author.
4. Merge into the main branch
5. Move the Issue to *Done*

7 Step 7: Work Iteratively

Continue with the remaining user stories using small tasks.

- Do not attempt to build the entire system at once.
- Complete one task at a time.
- Get features working before adding additional complexity.
- Revisit and refine your task list as you learn more about the project.
- Continuously test your changes and verify acceptance criteria.

8 Step 8: Track Your Time

- Record all work in the Time Log spreadsheet within 48 hours of performing it.
- Include coding, meetings, planning, testing. If you are logging time for a meeting, each participant should log time.
- Add a short description – **Provide enough detail so that the instructor knows exactly what you are working on. 1-3 sentences should usually suffice.**

Appendix

Appendix 1 How to Analyze a User Story

Before writing any code, you must clearly understand what the user story is asking. Use the following questions to guide your analysis.

1. What are the Inputs?
    - What data does the user provide?
    - What fields are required?
    - What format should the data be in?
2. What are the Outputs?
    - What should the system do or display?
    - What confirms success?
3. What are the Rules (Business Logic)?
    - What conditions must be true?
    - What validations are required?
4. What are the Edge Cases?
    - What unusual or boundary situations might occur?
5. What are Invalid (Sad Path) Cases?
    - What should happen when something goes wrong?
6. Which Acceptance Tests Apply?
    - Which acceptance tests describe the required behavior?
    - Which tests represent normal, edge, and invalid cases?
7. What Classes Might Be Needed?

    Start thinking in terms of MVC:

    - **Model:** What data objects represent this?
    - **Controller:** What logic processes the request?
    - **View:** What UI is needed?