**SPRINT NO:** 01

**SPRINT POD NAME:** GREENSERVE CRM

**POD MEMBERS:**

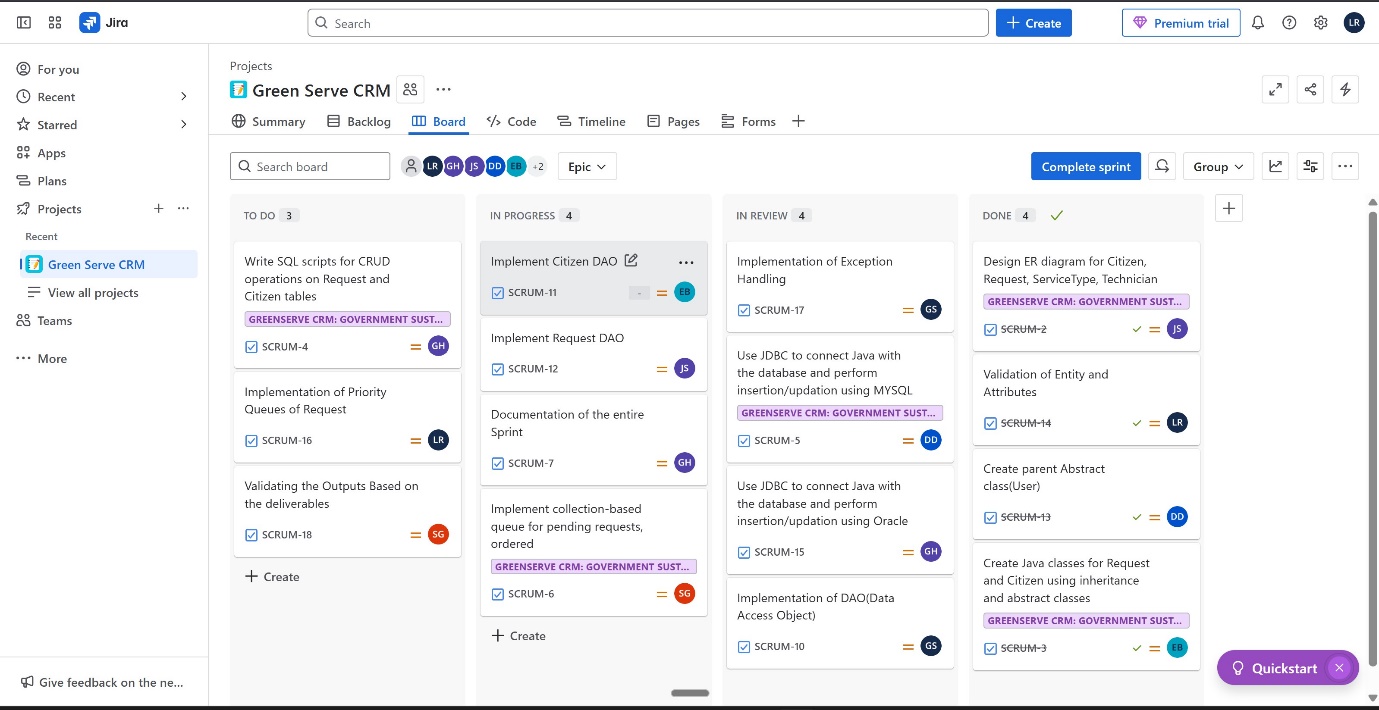
1. Divyansh Dubey
2. Ekta Bisht
3. Ganta Lokesh Reddy
4. Gummuluri Harshitha
5. Gulshan Kumar Sahu
6. Gurrala Sushmitha
7. Jaddu Chaitanya Shankar Srinath

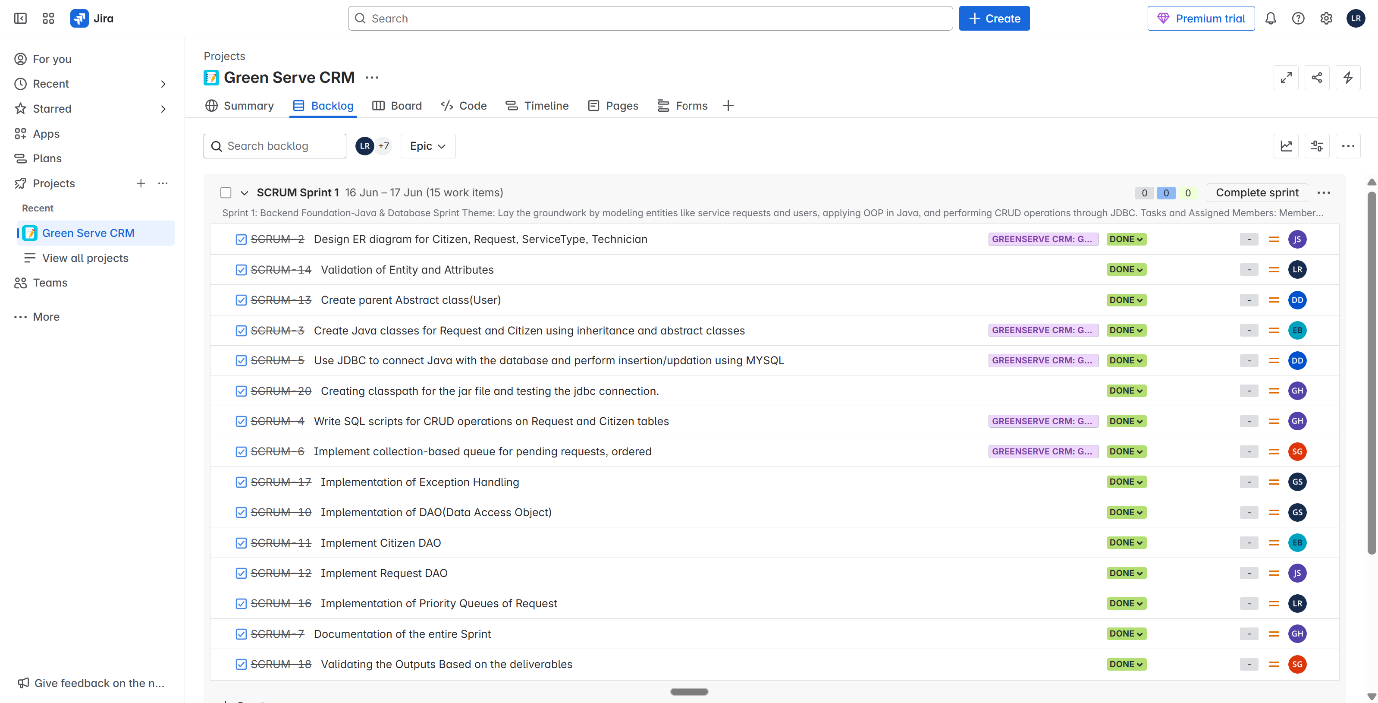
**SUBMISSION DATE:** 19-06-2025

**SPRINT GOAL:**

Establishing a solid backend foundation for the GreenServe CRM—Service Request & Sustainability Tracker—is the aim of Sprint 1. Developing Java classes based on object-oriented principles, integrating JDBC for data persistence, designing a normalized database schema, and putting in place features for generating, retrieving, updating, and managing service requests are all included in this. In order to provide dependable and accurate service delivery, appropriate exception handling and timestamp mechanisms will be included, along with a priority-based collection system to manage outstanding requests.

**USER STORY COMPLETION:**

****

****

**CODE SUBMISSION:**

<https://github.com/gummuluri-harshitha/Sprint-1>

**SPRINT REVIEW:**

What we Planned V/S What we delivered:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **What Was Planned** | **What Was Delivered** |
| **Database Schema (ERD)** | Design and normalize tables for **Citizen**, **Request**, **ServiceType**, and **Technician** | Implemented and normalized tables for **Citizen**, **Request**, **ServiceType**, and **Technician** |
| **Java OOP and Modularity** | Develop Java classes with inheritance and abstract classes for **Request** and **Citizen** | Implemented **Citizen** (with parent class) and **Request** (with inheritance) following OOP principles |
| **JDBC Integration** | Perform CRUD operations (Create, Retrieve, Update, Delete) through JDBC | Implemented CRUD operations (Create, Retrieve, Update, Delete) with JDBC |
| **Collection-Based Pending Request Queue** | Implement a collection (queue) for pending requests, ordered by priority | Implemented a Priority Queue to manage pending requests by their priority |
| **Exception Handling and Date/Time API** | Handle exceptions gracefully and track submission dates with Java Date/Time API | Exception handling implemented; submission timestamp recorded with *java.time.LocalDate* |
| **Sample Input/Output Logs** | Output Demonstration | |  | | --- | |  |  |  | | --- | | Console-based interface created for easier testing | |
| **Quality of Documentation** | Provide clear and comprehensive technical documentation | |  | | --- | | README includes setup, features, screenshots, and testing instructions | |

**CHALLENGES FACED:**

1. Designing a normalized database schema that maintains data integrity.
2. Implementing object-oriented principles while ensuring code reusability.
3. Establishing a reliable JDBC connection and gracefully handling database exceptions.
4. Managing and retrieving requests based on priority in a collection.
5. Integrating Java’s Date/Time API for accurate timestamp recording.

**LEARNINGS:**

1. Improved understanding of database normalization and foreign key constraints
2. Gained practical experience using Java code to implement object-oriented concepts (encapsulation, inheritance).
3. Enhanced technical proficiency with JDBC connections, SQL operations, and handling associated exceptions with grace.
4. Learned how to use Java collection classes, such as Priority Queue, to effectively arrange and rank requests.
5. Learn how to manage timestamps and keep track of submission dates using Java's Date/Time API.
6. Had the opportunity to understand how crucial robustness and error handling are when creating enterprise applications.
7. Obtained a comprehensive understanding of a backend application's end-to-end flow, including data storage, business logic, and the ways in which various components work together.
8. Enhanced interpersonal skills and worked as a team, particularly when working on database structures and coding standards.

**RETROSPECTRIVE NOTES:**

**What went well?**

* Database schema was designed and normalized successfully.
* Implemented object-oriented principles (inheritance, encapsulation) effectively in Java.
* Managed to perform CRUD operations smoothly with JDBC.
* Exception handling mechanisms were implemented gracefully.
* Proper usage of a priority queue for requests.
* Gained valuable technical skills and team collaboration experience.
* Documentation is **clear and beginner-friendly**

**What didn’t Go well?**

* It was a little difficult to set up primary/foreign keys and design relationships.
* Error handling and JDBC operation implementation were more difficult and prone to errors.
* At first, it was unclear how to integrate the collection (queue) with JDBC data.
* A few team members had trouble comprehending the Date/Time API in Java and how to use it.
* Manual input of IDs was tedious and error-prone.
* Time estimates were a little off—it took longer than anticipated to deliver every component within the sprint.

**Action Items for next sprint:**

* To develop a service request form that is responsive in HTML5 and CSS3.
* To enable dynamic service option updating and put JavaScript validation into practice.
* To delegate the proper roles (Admin, Technician, Coordinator) to the Salesforce Developer Organization.
* To generate unique Salesforce objects (Citizen\_\_c, ServiceRequest\_\_c, Technician\_\_c, and Feedback\_\_c).
* To specify sharing guidelines, permission sets, profiles, and Organization-Wide Defaults (OWD).
* To set up record types and page layouts for various service requests (pollution, waste, and solar).
* To create screenshots that show record-type configurations, roles, and permission settings.
* To verify functionality through configuration reviews and team demonstrations.