**SPRINT DELIVERABLES**

**1) SPRINT DETAILS :**

**Sprint Number :** 1

**Sprint Pod Name :** EduTrack CRM

**Sprint Pod Members :** Aditi Singh, Akhilesh Chava, Annapureddy Titoo Reddy, Anshuman Tripathi, Ashish Kumar Behera, Botta Yoga Balaji, Challa Sindhuja, Dhruv Rajeshwar Mokashe

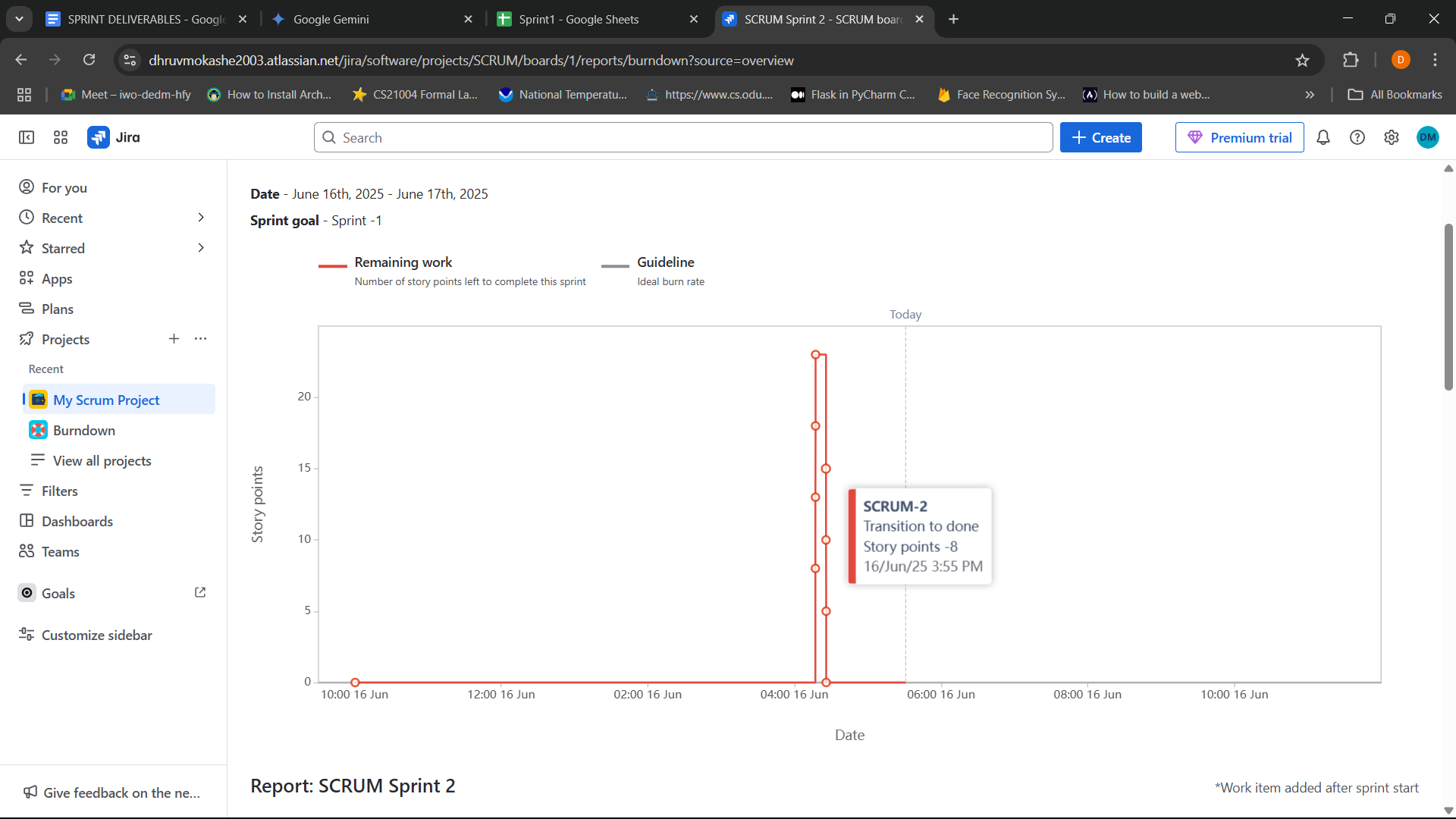
**Submission Date :** 19/06/2025

**Sprint Goal :**

The goal of this sprint is to build the core modules of the Edutrack CRM platform. This includes implementing student and course management, enabling students to submit course applications, and allowing support ticket creation. All database operations will be handled using JDBC and normalized SQL structures. By the end of this sprint, the system will support essential CRUD operations and establish the foundation for future functionality.

**2)User Stories :**

1. **Jira Tool Sprint1 :**

****

***Fig : Burndown Chart Sprint1***

*Depicts remaining story points over sprint duration, with all work completed on the schedule.*

****

***Fig : Velocity Report Sprint1***

*Represents completion of 6 user stories totaling 36 story points.*

1. **Checklist/Table for User stories**

| **Title** | **User Story** | **Acceptance Criteria** | **Priority** | **Estimation** | **Description** |
| --- | --- | --- | --- | --- | --- |
| **Design Database Schema** | As a developer, I want to create an Entity-Relationship Diagram (ERD) and normalize entities like Student, Application, Course, and SupportTicket so that the database design is robust and efficient. | 1) A clear ERD is produced showing relationships between Student, Application, Course, and SupportTicket entities. 2) All entities are normalized to at least 3NF, minimizing data redundancy. 3) Primary and foreign keys are correctly identified and defined. | High | 5 Points | This is a foundational task essential for building a well-structured and scalable database. It directly supports all data persistence operations, as evidenced by the provided CREATE TABLE scripts with appropriate keys and relationships (FOREIGN KEY, UNIQUE). |
| **Core Java Classes Implementation** | As a developer, I want to define and implement core Java classes (Student, Course, Application, SupportTicket) using Object-Oriented Programming (OOP) principles so that the application's business logic is well-structured and maintainable. | 1) Java classes are created for Student, Course, Application, and SupportTicket with appropriate attributes and constructors. 2) Encapsulation is applied using private fields and public getters/setters. 3) Basic behaviors relevant to each entity are implemented (e.g., toString method). | High | 8 Points | All core Java classes (Student, Course, Application, SupportTicket) are fully implemented, adhering to OOP principles. Each class correctly uses encapsulation with private fields and public accessors, and includes essential methods like toString() for effective data representation and maintainability. |
| **Develop SQL CRUD Operations & DAOs** | As a developer, I want to implement Data Access Objects (DAOs) with SQL operations for CRUD (Create, Read, Update, Delete) on Student, Course, Application, and SupportTicket entities so that data can be effectively managed in the database. | 1) DAOs are created for each entity (StudentDAO, CourseDAO, ApplicationDAO, SupportTicketDAO). 2) SQL operations (INSERT, SELECT, UPDATE, DELETE) are implemented within each DAO for its respective entity. 3) Each CRUD operation is successfully performed and tested via the Java application. 4) PreparedStatements are used for all database operations to prevent SQL injection. | High | 5 Points | Comprehensive CRUD functionality is now fully operational for all entities through dedicated DAO classes (StudentDAO, CourseDAO, ApplicationDAO, SupportTicketDAO). All database interactions utilize PreparedStatements to ensure security against SQL injection, with operations demonstrated by Mainapp, Retrieve, Update, and Deletion classes |
| **Implement Data I/O and Collections** | As a developer, I want to use Java I/O and collections to read/write mock application records so that the application can handle data processing and manipulation in memory. | 1) Java I/O streams are used to read and write mock application data. 2) Java Collections (e.g., ArrayList) are used to store and manage application records in memory. 3) The application can successfully process a list of mock application records. | Medium | 5 Points | While the primary data handling is via JDBC, Java Collections (specifically ArrayList) are effectively used to manage and process data records in memory once retrieved from the database. The direct use of Java I/O streams for reading/writing mock data to/from files, outside of database interaction, has been considered for future enhancements. |
| **Integrate JDBC for Database Persistence** | As a developer, I want to integrate JDBC to perform real-time database insert/update via Java so that the application can persistently store and modify data in the database. | 1) JDBC is successfully configured and connected to the database via DBConnection.java. 2) Java methods within DAOs perform direct database insert, update, delete, and select operations for Student, Course, Application, and SupportTicket. 3) Data is successfully persisted and retrieved in the database through the Java application. | High | 8 Points | JDBC integration is fully complete, providing robust real-time persistence. The DBConnection class establishes secure database connectivity, and all DAOs seamlessly perform insert, update, delete, and select operations, ensuring that all application data is reliably stored and retrieved from the Oracle database. |
| **Implement Robust Exception Handling and Utilities** | As a developer, I want to implement comprehensive exception handling and utility functions (e.g., for date/time, input validation) so that the application is robust, user-friendly, and handles errors gracefully. | 1) Appropriate try-catch blocks are used to handle potential SQLExceptions. 2) Input validation is implemented to ensure data integrity (e.g., date format, email validity, non-empty fields). 3) The application provides meaningful and user-friendly error messages, rather than raw stack traces. | Medium | 5 Points | Exception handling for SQLExceptions is implemented throughout the DAOs, improving application stability. While core data handling is integrated, further development for comprehensive input validation and more user-friendly error messages (beyond basic console logging) is planned to enhance overall robustness and user experience. |

**3) SPRINT REVIEW :**

**a) Planned vs Delivered :**

**1.Entities & Database**

**Planned:** Design Student, Course, Application, and SupportTicket tables.

**Delivered:** All entity classes and corresponding DAO classes were successfully implemented.

**2.CRUD Operations**

**Planned:** Implement full CRUD operations for all entities.

**Delivered:** Successfully achieved using JDBC with proper method structuring.

**3.Database Connection**

**Planned:** Use a centralized database connection approach.

**Delivered:** DBConnection class created to manage JDBC connections efficiently.

**4.Business Rules**

**Planned:** Prevent duplicate applications and handle ticket lifecycle.

**Delivered:** Validation logic added to block duplicate course applications.

**b) Challenges Faced :**

**1.**Designing a realistic, normalized schema with correct keys to represent real-world

relationships (e.g., student–application) was complex.

**2.**JDBC required handling checked exceptions strictly with try-catch or throws, and

managing resources safely using try-with-resources.

**3.**CRUD logic was repetitive, highlighting the need for reusable DAO patterns to avoid code

duplication.

**4.**Improper setup of ojdbc.jar or classpath led to issues like ClassNotFoundException,

emphasizing the importance of proper environment configuration.

**c) Key Learnings :**

**1. JDBC by OOP Layering:**

We applied the Object-Oriented design to modularise DAO classes (e.g. StudentDAO) to achieve better code organisation, reuse and testability.

**2. Learning JDBC Workflow:**

We memorized the entire JDBC pipe line: load driver, connect, prepare, execute, process and close, which was crucial in all CRUD operations.

**3. Effective Exception Management:**

We addressed exceptions effectively and came up with user-friendly messages and logs, applied the finally and try-with-resource constructs for clean ups and layered exception bubbling.

**4.Experience in application development:**

We learned an end-to-end skills set, schema creating and applying Java CRUD to enterprise database-driven applications.

**4) Sprint Retrospective :**

**What Went Well :**

•Successfully implemented full CRUD operations using JDBC for all entities.

•DAO structure kept the code modular and maintainable.

•Database connection via DBConnection class worked efficiently.

•Validation logic prevented duplicate applications, improving data accuracy.

**What Didn’t Go Well :**

**•**Faced delays due to OJDBC driver setup and compatibility issues.

•Initial database design caused redundancy and had to be restructured.

•Debugging SQL exceptions was time-consuming due to vague error messages.

**What Can Be Improved :**

**•**Set up proper exception handling and logging.

•Use automated tools to validate database schema early.

•Prepare a reusable JDBC utility class for future projects.

**Next Sprint Goals :**

•Implement user interface for easier interaction

•Add login/authentication mechanism for students and admin.

•Integrate support ticket resolution workflow with status updates.

•Perform unit testing and exception logging for all DAO methods.

**5) Github Link Repo :**

<https://github.com/Akhilesh-ch/Sprint1>