

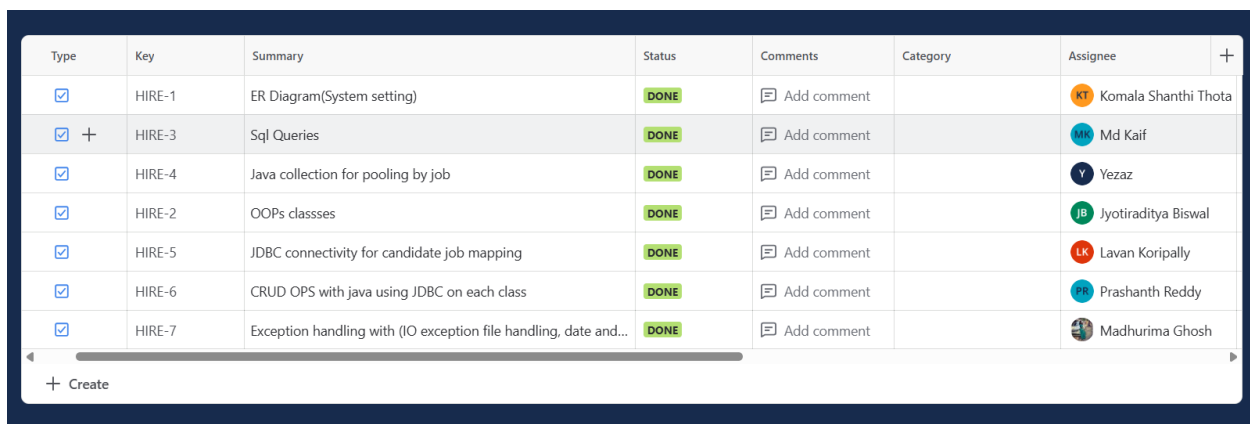
# Sprint Deliverables

## 1. Sprint Goal

In the Sprint we focus on mainly for the backend system that use for the recruitment platform. This involved designing the overall data model, implementing object-oriented structures, connecting with the database, and preparing for real-time application workflows like candidate-job matching, resume uploads, and interview tracking.

## 2. User Stories:

### Jira Tool Burndown chart:



Type	Key	Summary	Status	Comments	Category	Assignee	
<input checked="" type="checkbox"/>	HIRE-1	ER Diagram(System setting)	DONE	Add comment		Komala Shanthi Thota	
<input checked="" type="checkbox"/> +	HIRE-3	Sql Queries	DONE	Add comment		Md Kaif	
<input checked="" type="checkbox"/>	HIRE-4	Java collection for pooling by job	DONE	Add comment		Yezaz	
<input checked="" type="checkbox"/>	HIRE-2	OOPs classes	DONE	Add comment		Jyotiraditya Biswal	
<input checked="" type="checkbox"/>	HIRE-5	JDBC connectivity for candidate job mapping	DONE	Add comment		Lavan Koripally	
<input checked="" type="checkbox"/>	HIRE-6	CRUD OPS with java using JDBC on each class	DONE	Add comment		Prashanth Reddy	
<input checked="" type="checkbox"/>	HIRE-7	Exception handling with (IO exception file handling, date and...	DONE	Add comment		Madhurima Ghosh	

+ Create

*Fig.1: users assign with each task*

## 1. Design an ER Diagram for Candidate, Job, Interview, and Recruiter:

We began by identifying all the key entities involved in a recruitment workflow. Using this understanding, we crafted an Entity-Relationship (ER) diagram that included:

- **Candidate:** Stores profile details and resumes.
- **Job:** Contains job descriptions, skills required, and deadlines.
- **Interview:** Records scheduled discussions with time and feedback.

The relationships between these entities (e.g., one candidate applies to many jobs, one job has many interviews) were carefully structured to normalize the schema and avoid redundancy.

## 2. Implement OOP Classes for Candidate and Job Using Interfaces and Inheritance

After designing the schema, we implemented the Candidate and Job entities in Java using solid Object-Oriented Programming (OOP) principles. We introduced a **Person interface** for shared methods. Candidate implements Person, and includes fields for ID, name, email, phone, and

resume path. Job class encapsulates details like job title, skills, salary, deadline, and recruiter info. By applying **interfaces and inheritance**, we ensured that the code is modular, reusable, and easily extendable.

### □ 3. Create SQL Scripts and Perform CRUD Operations for Candidate and Job

We translated our ER design into SQL tables and wrote scripts to:

- Create tables for Candidate, Job, and their mapping.
- Perform **Create, Read, Update, and Delete (CRUD)** operations.
- Manage candidate-job relationships using foreign keys.

We faced some Oracle-specific challenges like ORA-00913 (too many values) and ORA-01727, which we resolved through careful debugging and schema refinement.

### 4. Use Java Collections and Generics to Manage Applicant Pools for Each Job

To simulate real-time recruitment, we built in-memory data structures: Used `HashMap<Job, List<Candidate>>` to store which candidates applied to which job. Utilized **Java Generics** to maintain type safety and performance. Allowed dynamic addition and retrieval of applicants for any job using CLI options. This helped bridge the application logic before full DB integration.

### 5. Integrate Java JDBC to Map Candidates to Jobs in Oracle Database

A major milestone was successfully establishing **JDBC connectivity** with Oracle SQL. Wrote JDBC logic to insert and retrieve data from the CandidateJobMap table. Managed driver loading, connection setup, statement execution, and graceful error handling. Validated all mappings with console outputs to ensure accurate storage of candidate-job relationships. This connected our Java backend with persistent storage, making the system stateful.

### 6. Add Exception Handling, Resume Upload via File I/O, and Interview Logging with Date/Time:

For better user experience and robustness:

1. Implemented **File I/O** to simulate resume uploads, storing resume paths in the DB.
2. Used **try-catch blocks** extensively to handle SQL, I/O, and input format exceptions gracefully.
3. Captured interview logs with **LocalDateTime** and **DateTimeFormatter**, saving interview scheduling data into the DB and displaying formatted results.

## **Github Link Repo :**

<https://github.com/jyotiraditya0607/HireSmartCTS>

## **Planned vs Delivered:**

### **1. Entities & Database**

**Planned:** we Design schema for the tables including Candidate, Job, Application, and Interview, ensuring proper primary key and foreign key relationships and normalization.

**Delivered:** Successfully implemented all tables with relevant entity classes in Java, along with corresponding classes. Relationships like Candidate–Application and Job–Interview were accurately modeled in both schema and code.

### **2. CRUD Operations**

**Planned:** Develop full Create, Read, Update, and Delete functionalities for all modules using JDBC.

**Delivered:** All modules now support robust CRUD operations, implemented via structured DAO methods for each entity, ensuring clean separation of concerns.

### **3. Database Connectivity**

**Planned:** Maintain a single-point database connection for all JDBC operations.

**Delivered:** A centralized DBConnection utility class was implemented to manage Oracle JDBC connections, making the code reusable and consistent.

### **4. Business Logic & Validations**

**Planned:** Include logic to prevent duplicate applications and ensure valid scheduling of interviews.

**Delivered:** Validations were built into the DAO layers to prevent redundant entries and maintain data integrity, particularly in the Application and Interview modules.

## **Challenges Faced:**

### **1. Designing ER Schema**

Modeling real-world relationships such as Candidates applying for multiple jobs, and jobs

receiving multiple applications, required careful planning. Ensuring proper foreign keys and avoiding redundancy posed a challenge during the initial schema design.

## **2. Exception Handling**

JDBC requires explicit handling of `SQLException`, which made the code verbose. Managing resources safely through `try-with-resources` helped avoid memory leaks but added complexity.

## **3. JDBC Driver and Classpath Issues**

Errors like `ClassNotFoundException` due to missing or misconfigured `ojdbc.jar` files slowed down early development. Setting the correct classpath in Eclipse was critical but non-intuitive. Also we face so many error in the like error in path and password error like that.

## **4. Input Format Parsing**

Date/time inputs (especially for interviews) caused `DateTimeParseException` when user input didn't match the expected format. Handling this gracefully required additional validation logic.

# **Learnings**

In the Sprint we learn a lot from the building code from simple to logically and implementing multiple concepts and also integrated with other technologies like Oracle database and SQL.

## **1. JDBC with OOP Principles**

We structured the backend using App classes like `Candidate` and `Job`, aligning with object-oriented practices. This improved modularity and made testing/debugging much easier.

## **2. Understanding End-to-End JDBC Flow**

From loading the driver to closing connections, we fully grasped the JDBC pipeline:  
`DriverManager` → `Connection` → `PreparedStatement` → `ResultSet` → `Close`.

## **3. Exception Management Techniques**

We practiced exception concepts, wrapping exceptions in meaningful outputs, and using `finally/try-with-resources` blocks to manage database resource cleanup errors effectively.

## **4. Practical Database Design**

Creating normalized schemas with proper relationships enhanced our understanding of real-world enterprise data modeling. Implementing foreign keys and join queries was a valuable hands-on experience.

## **5. Handling User Input and Parsing**

We built robustness into the system by validating user input, especially for date/time fields and dropdown-like entries, ensuring smoother runtime experiences.

## **Sprint Retrospective**

### **What Went Well**

- Full CRUD operations were implemented successfully for each module.
- DB connection via a centralized DBUtils class kept the code efficient and reusable.
- Validation logic worked correctly to avoid duplicate job applications.
- The DAO structure made the codebase modular and easy to extend.

### **What Didn't Go Well**

- Time was lost in resolving ojdbc driver issues and setting up the classpath in Eclipse.
- Debugging SQL errors was difficult due to unclear error messages from Oracle.
- Schema design had to be revised mid-way due to overlooked relationships and redundancies.

### **Items for next Sprint**

Develop a simple command-line or GUI interface for smoother user interaction.

Add login and authentication features for admin and recruiter roles.

Introduce interview evaluation tracking and candidate status updates.

Write JUnit test cases and log exception flows for better debugging and coverage.