Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution affiliated to Anna University)
Rajiv Gandhi Salai, Kalavakkam – 603110

Department
B.Tech Information Technology

UIT3361 Object-Oriented Programming Using Java
And
UIT3311 Database Technology Laboratory

Project Title
Java Standalone Application with Database Connectivity

Task Management and To-Do Application

Name and Reg No of the Project Students
Muskan Kumari V – 3122 24 5002 059
Nithilan S – 3122 24 5002 060
Nithiyasri R – 3122 24 5002 061

Name, Designation and Department of the Project Guide (s)
Dr. K.S. Gayathri - Associate Professor - Information Technology
Dr. R. Swathika - Associate Professor - Information Technology

Signature of the Project Students		**Signature of the Project Guide(s)**	
Muskan Kumari V	-	Dr. K.S. Gayathri	-
Nithilan S	-	Dr. R. Swatika	-
Nithiyasri R	-		

Table of Contents

1. Problem Statement

Design and implement a lightweight, secure, and maintainable desktop **Task Management** application that enables users to create, update, delete, and organize tasks with attributes (title, description, priority, deadline, completion status), supports user accounts and per-user preferences, and persists data reliably on a relational database. The system must be usable offline, easy to install for academic/demo deployment, and designed to allow safe manual control of database schema in production environments. Key functional requirements:

- Create, edit, and delete tasks with title, description, priority, deadline, and completion state.
- User accounts and authentication (email + password).
- User preferences for UI and behavioral choices.
- Task listing and filters (by priority, completion state, deadline sorting).
- Local persistence with transactional guarantees and safe schema management.

Non-functional requirements:

- Cross-platform Java desktop application (Windows, macOS, Linux)
- Simple install (JAR + bundled JRE or run via Maven during development)
- Clean, modern UI implemented with JavaFX and CSS
- Maintainable codebase with separation of concerns (presentation, service, persistence)

Challenges addressed:

- Mapping Java 8+ time types (LocalDate, LocalDateTime) to Oracle TIMESTAMP/DATE types
- Balancing convenience (hibernate auto DDL) and safety (manual DDL for production)

General summary

This Task Management & To-Do desktop application is a lightweight JavaFX application designed for personal productivity. It provides user accounts, task management (CRUD), persistent storage via Hibernate (JPA), and a clean, modern UI. The system is intentionally small but extensible: it demonstrates layered architecture, transaction-safe persistence, and attention to production-ready concerns such as manual DB schema control and secure configuration.

This report documents the problem and solution, provides a complete ER diagram, full DDL, normalization analysis, system requirements, architecture, module descriptions, code snippets, testing plan, and deployment instructions. Appendices contain the full entity source code and helper scripts.

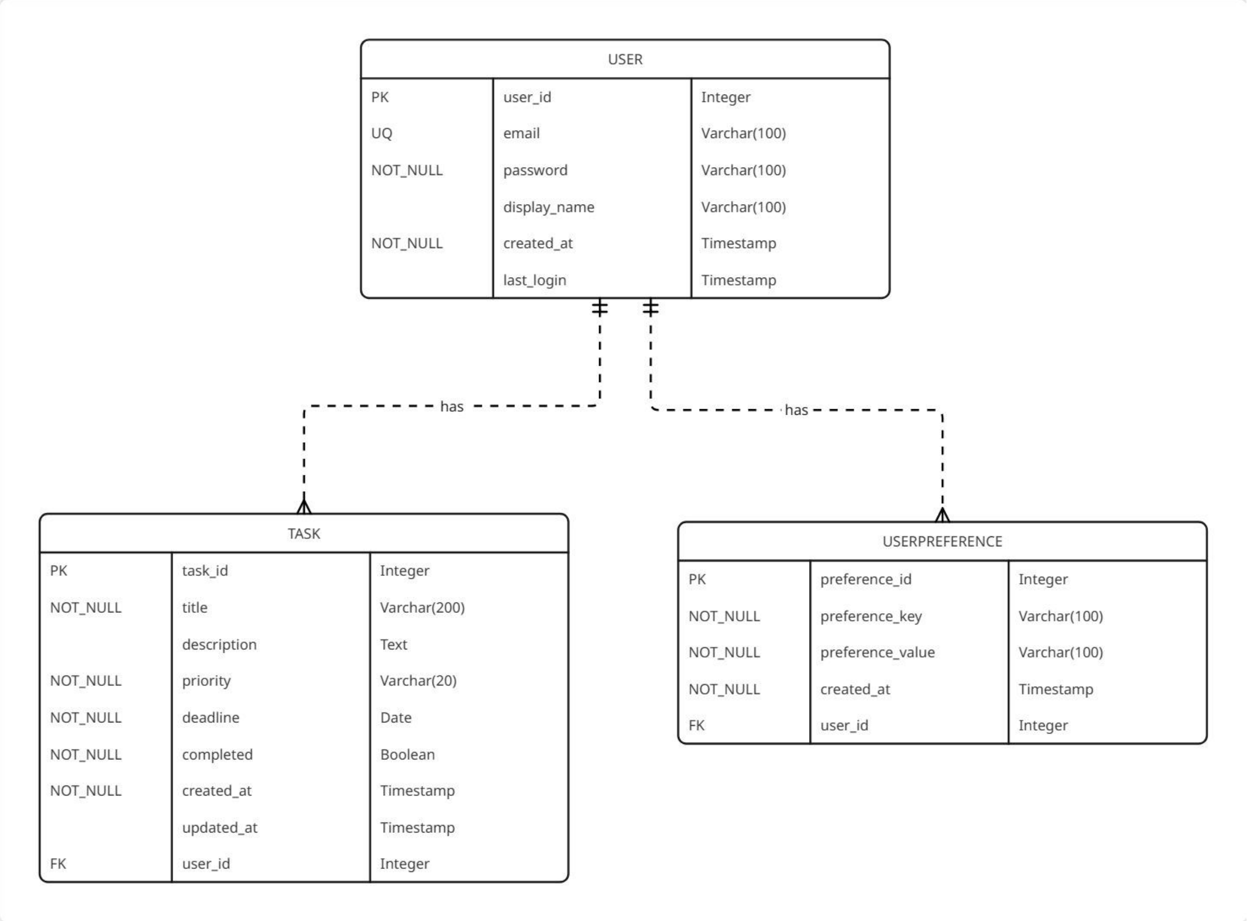
2. ER Diagram

The ER diagram for the application is included in `docs/images/er_diagram.png` and summarized below:

- User (1) — (N) Task
- User (1) — (N) UserPreference

Attributes (selected): -

- User: id (PK), email (unique), password, display_name, created_at, last_login
- Task: id (PK), title, description, priority, deadline, completed, created_at, updated_at, user_id (FK)
- UserPreference: id (PK), preference_key, preference_value, created_at, user_id (FK)



USER		
PK	user_id	Integer
UQ	email	Varchar(100)
NOT_NULL	password	Varchar(100)
	display_name	Varchar(100)
NOT_NULL	created_at	Timestamp
	last_login	Timestamp

TASK		
PK	task_id	Integer
NOT_NULL	title	Varchar(200)
	description	Text
NOT_NULL	priority	Varchar(20)
NOT_NULL	deadline	Date
NOT_NULL	completed	Boolean
NOT_NULL	created_at	Timestamp
	updated_at	Timestamp
FK	user_id	Integer

USERPREFERENCE		
PK	preference_id	Integer
NOT_NULL	preference_key	Varchar(100)
NOT_NULL	preference_value	Varchar(100)
NOT_NULL	created_at	Timestamp
FK	user_id	Integer

3. Database Schema (complete DDL)

The project includes a full Oracle-compatible DDL script: `src/main/resources/sql/create_schema_oracle.sql`.

For convenience, the script is reproduced here:

```
-- Schema and objects for Task Management app (Oracle)
-- Run as a DBA (SYSTEM) or a user with CREATE USER and GRANT privileges.

-- 1) Create an application schema/user (recommended)
-- Run as SYS or SYSTEM:
-- CREATE USER task_app IDENTIFIED BY "PASSWORD";
-- GRANT CREATE SESSION TO task_app;
-- GRANT CREATE TABLE TO task_app;
-- GRANT CREATE SEQUENCE TO task_app;
-- GRANT CREATE VIEW TO task_app;
-- GRANT CREATE PROCEDURE TO task_app;
-- GRANT UNLIMITED TABLESPACE TO task_app; -- optional, or grant specific quo
tas

-- Alternatively, if you want to create objects in an existing schema, skip u
ser creation

-- 2) Connect as the application user (or run the following as that user)
-- CONNECT task_app/PASSWORD<db_connect_string>

-- 3) Create sequences used by JPA annotations
CREATE SEQUENCE TASK_SEQ START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE;
CREATE SEQUENCE USER_SEQ START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE;
CREATE SEQUENCE PREFERENCE_SEQ START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE;

-- 4) Create tables

-- Users table
CREATE TABLE users (
    id NUMBER(10) PRIMARY KEY,
    email VARCHAR2(255) NOT NULL,
    password VARCHAR2(255) NOT NULL,
    display_name VARCHAR2(255),
    created_at TIMESTAMP,
    last_login TIMESTAMP
);

-- Unique constraint on email
ALTER TABLE users ADD CONSTRAINT uq_users_email UNIQUE (email);

-- Tasks table
CREATE TABLE tasks (
    id NUMBER(10) PRIMARY KEY,
    title VARCHAR2(400) NOT NULL,
    description CLOB,
    priority VARCHAR2(50),
    deadline DATE,
    completed NUMBER(1) DEFAULT 0 NOT NULL,
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    user_id NUMBER(10) NOT NULL
);

-- Foreign key to users
ALTER TABLE tasks ADD CONSTRAINT fk_tasks_user FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE;

-- Index for tasks.user_id
CREATE INDEX idx_tasks_user_id ON tasks(user_id);

-- User preferences table
CREATE TABLE user_preferences (
    id NUMBER(10) PRIMARY KEY,
    preference_key VARCHAR2(255) NOT NULL,
    preference_value VARCHAR2(2000),
    created_at TIMESTAMP,
    user_id NUMBER(10) NOT NULL
);

-- Foreign key to users
ALTER TABLE user_preferences ADD CONSTRAINT fk_prefs_user FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE;

-- Index for preferences.user_id
CREATE INDEX idx_prefs_user_id ON user_preferences(user_id);

-- 5) Optional: triggers to set id from sequence if inserts originate outside Hibernate
-- (Hibernate normally fetches NEXTVAL itself; triggers are optional.)

-- Trigger for tasks
CREATE OR REPLACE TRIGGER trg_tasks_before_insert
BEFORE INSERT ON tasks
FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    SELECT TASK_SEQ.NEXTVAL INTO :NEW.id FROM DUAL;
  END IF;
END;
/
```

```
-- Trigger for users
CREATE OR REPLACE TRIGGER trg_users_before_insert
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    SELECT USER_SEQ.NEXTVAL INTO :NEW.id FROM DUAL;
  END IF;
END;
/

-- Trigger for user_preferences
CREATE OR REPLACE TRIGGER trg_prefs_before_insert
BEFORE INSERT ON user_preferences
FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL THEN
    SELECT PREFERENCE_SEQ.NEXTVAL INTO :NEW.id FROM DUAL;
  END IF;
END;
/
-- 6) Optional: enforce boolean semantics for 'completed' column (0/1) via check constraint
ALTER TABLE tasks ADD CONSTRAINT chk_tasks_completed CHECK (completed IN (0,1));

-- 7) Verification queries (run as the application user)
-- List tables
-- SELECT table_name FROM user_tables WHERE table_name IN ('USERS','TASKS','USER_PREFERENCES');

-- Check sequences
-- SELECT sequence_name, last_number FROM user_sequences WHERE sequence_name IN ('TASK_SEQ','USER_SEQ','PREFERENCE_SEQ');

-- Basic sample inserts to test sequences and FK constraints
-- INSERT INTO users(email, password, display_name, created_at) VALUES ('alice@example.com', 'secret', 'Alice', CURRENT_TIMESTAMP);
-- INSERT INTO tasks(title, description, priority, deadline, completed, created_at, user_id) VALUES ('Test task', 'desc', 'High', TO_DATE('2025-12-31', 'YYYY-MM-DD'), 0, CURRENT_TIMESTAMP, 1);

-- End of script
```

4. Database Normalization (detailed analysis and examples)

Normalization rationale:

- The `users` table contains atomic attributes such as email and password; email is declared UNIQUE to prevent duplicates.
- The `tasks` table uses `user_id` as a foreign key; all task attributes are functionally dependent on the primary key `id` (1NF and 2NF satisfied).
- `user_preferences` stores key-value pairs to avoid schema changes when adding new preference items, improving extensibility.

Example demonstrating avoidance of redundancy:

- If preferences were stored as columns in `users` (e.g., `pref_theme`, `pref_notifications`), adding new preference types would require ALTER TABLE statements and potential data migration. The key/value approach isolates preferences and keeps the main `users` table lean.

Normalization trade-offs:

- The key/value model is flexible but can make queries for multiple preferences slightly more complex (joins or aggregation). For our app scale, the simplicity and flexibility outweigh this cost.

5. System Requirements

Recommended development environment:

- Java 17 (LTS) for modern features and long-term support.
- Maven 3.8+, IntelliJ IDEA Community/Ultimate edition, or Visual Studio Code with the Java extension pack.
- Oracle XE for local testing; alternatively use Postgres for easier CI integration.

Production considerations:

- For single-user desktop installs, an embedded database (H2 with file storage) is acceptable. For multi-user or multi-machine deployments, use a managed RDBMS and create a secure dedicated schema.

6. System Design

Component details:

- MainApp (application entry): initializes `ConfigManager`, sets up the `SceneRouter`, and launches JavaFX.
- SceneRouter: centralizes scene switching and sharing the current user session across controllers.
- DatabaseService: single point of contact for all database operations; uses Hibernate sessions and transactions.
- HibernateUtil: lazily builds a singleton `SessionFactory` using `hibernate.cfg.xml` and ensures clean shutdown.

Threading and UI responsiveness:

- Database operations are executed on background threads to avoid blocking the JavaFX Application Thread. Use `Task/Service` from JavaFX concurrency utilities when performing long-running queries.

7. Core Modules (detailed responsibilities)

- Presentation: JavaFX (FXML) views and Controllers (LoginController, DashboardController, TaskController).
- Application: MainApp bootstraps configuration and routing.
- Business logic: DatabaseService singleton handles transactional operations.
- Persistence: Hibernate (configured via hibernate.cfg.xml), entity classes User, Task, and UserPreference.
- Configuration: ConfigManager reads application.properties and environment variables.

Packaging:

- Build a runnable JAR with dependencies using Maven Shade plugin or use a bundled runtime with jlink for smaller size.
- Include the SQL script and report in the repository under `docs/`.

Deployment instructions (developer mode):

```
mvn clean package -DskipTests
mvn javafx:run
```

GitHub Repository Link :
https://github.com/Nithilan77/Task_Management_And_To_Do_Application

---

8. Code Snippets

We used Java to implement the full application stack. Below is a package-by-package map and the important classes, objects and methods. For each class you'll find: purpose, important methods, key interactions, and file path.

8.1 Entry point

- Main class: `com.taskmanager.MainApp`
  - File: `src/main/java/com/taskmanager/MainApp.java`
  - Purpose: JavaFX `Application` subclass. Boots the `ConfigManager`, `DatabaseService` and `SceneRouter`.
  - Important methods:
    - `public void start(Stage primaryStage)` — initializes configuration, database service, scene router and shows the login screen.
    - `public void stop()` — calls `DatabaseService.getInstance().close()` to shutdown Hibernate.
    - `public static void main(String[] args)` — calls `launch(args)` to start JavaFX.

Snippet (startup sequence — small excerpt)

```
// File: src/main/java/com/taskmanager/MainApp.java
@Override
public void start(Stage primaryStage) {
    ConfigManager configManager = ConfigManager.getInstance();
    DatabaseService databaseService = DatabaseService.getInstance();
    SceneRouter.getInstance().init(primaryStage);
}
```

8.2 Configuration

- `com.taskmanager.config.ConfigManager`
  - File: `src/main/java/com/taskmanager/config/ConfigManager.java`
  - Purpose: Loads application properties from `application.properties`, falls back to environment vars and system properties.
  - Important methods (examples):
    - `getDbUrl()`, `getDbUsername()`, `getDbPassword()` — database connection values.
    - `getHibernateHbm2ddl()` — returns configured `hibernate.hbm2ddl.auto` (default `update`).
    - `printConfigStatus()` — useful for debugging during startup.

8.3 Persistence utility

- `com.taskmanager.util.HibernateUtil`
  - File: `src/main/java/com/taskmanager/util/HibernateUtil.java`

  - Purpose: Controls a singleton Hibernate `SessionFactory` using `hibernate.cfg.xml`.
  - Important methods:
    - `public static SessionFactory getSessionFactory()` — lazily builds and returns the `SessionFactory`.
    - `public static void shutdown()` — closes the `SessionFactory`.

Snippet (HibernateUtil.getSessionFactory)

```
// File: src/main/java/com/taskmanager/util/HibernateUtil.java
public static SessionFactory getSessionFactory() {
    if (sessionFactory == null) {
        sessionFactory = new Configuration()
                .configure("hibernate.cfg.xml")
                .buildSessionFactory();
    }
    return sessionFactory;
}
```

8.4 Business & Data Access

- `com.taskmanager.service.DatabaseService` (Singleton)
  - File: `src/main/java/com/taskmanager/service/DatabaseService.java`
  - Purpose: Centralized database operations using Hibernate Sessions/Transactions. All CRUD is here.
  - Pattern: Singleton with `getInstance()`.
  - Key methods for Users:
    - `public User authenticateUser(String email, String password)` — queries for a matching user and updates `lastLogin`.
    - `public User registerUser(String email, String password, String displayName)` — checks for existing email and persists a new `User`.
    - `public User getUserById(int userId)` — fetch by primary key.
  - Key methods for Tasks:
    - `public Task saveTask(Task task)` — `session.merge(task)` in a transaction.
    - `public List<Task> getUserTasks(int userId)` — HQL: `FROM Task WHERE user.id = :userId ORDER BY createdAt DESC`.
    - `public List<Task> getUserTasksByStatus(int userId, boolean completed)`
    - `public List<Task> getUserTasksByPriority(int userId, String priority)`
    - `public void deleteTask(int taskId)` — removes Task if present.
    - `public Task updateTask(Task task)` — merges and sets updated timestamp.
  - Preferences:

        - saveUserPreference, getUserPreference, getUserPreferences.
    - Transaction handling: opens session, begins transaction, try/catch with rollback on exception.

Snippet (authentication and task save examples)

```java
// File: src/main/java/com/taskmanager/service/DatabaseService.java
public User authenticateUser(String email, String password) {
  try (Session session = HibernateUtil.getSessionFactory().openSession()) {
    Transaction tx = session.beginTransaction();
    Query<User> query = session.createQuery(
      "FROM User WHERE email = :email AND password = :password", User.class);
    query.setParameter("email", email);
    query.setParameter("password", password);
    User user = query.uniqueResult();
    if (user != null) { user.setLastLogin(LocalDateTime.now());
session.merge(user); }
    tx.commit();
    return user;
  }
}

public Task saveTask(Task task) {
  try (Session session = HibernateUtil.getSessionFactory().openSession()) {
    Transaction tx = session.beginTransaction();
    session.merge(task);
    tx.commit();
    return task;
  }
}
```

8.5 Entities (JPA)

All entities live under `src/main/java/com/taskmanager/entity/` and are annotated with Jakarta Persistence annotations.

- `com.taskmanager.entity.User` — `User.java`
    - Fields: `id` (sequence USER_SEQ), `email` (unique), `password`, `displayName`, `createdAt`, `lastLogin`, `tasks` (OneToMany), `preferences` (OneToMany).
    - Constructors: default sets `createdAt`, convenience constructor `User(String email, String password, String displayName)`.
    - Methods: getters/setters, `addTask(Task)`, `removeTask(Task)`.
- `com.taskmanager.entity.Task` — `Task.java`
    - Fields: `id` (sequence TASK_SEQ), `title` (not null), `description`, `priority`, `deadline`, `completed` (boolean), `createdAt`, `updatedAt`, `user` (ManyToOne).
    - Lifecycle: `@PreUpdate preUpdate()` updates `updatedAt`.
    - Methods: getters/setters; setting `completed` updates `updatedAt`.

Snippet (Task entity core fields & annotations)

```java
// File: src/main/java/com/taskmanager/entity/Task.java
@Entity
@Table(name = "tasks")
public class Task {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"task_seq")
  @SequenceGenerator(name = "task_seq", sequenceName = "TASK_SEQ",
allocationSize = 1)
  private int id;

  @Column(nullable = false)
  private String title;

  private String description;
  private String priority; // High, Medium, Low
  private LocalDate deadline;
  private boolean completed;
  // createdAt, updatedAt, user relation omitted for brevity
}
```

- `com.taskmanager.entity.UserPreference` — `UserPreference.java`
  - Fields: `id` (sequence PREFERENCE_SEQ), `key`, `value`, `createdAt`, `user` (ManyToOne).

File paths: - `src/main/java/com/taskmanager/entity/User.java` - `src/main/java/com/taskmanager/entity/Task.java` - `src/main/java/com/taskmanager/entity/UserPreference.java`

8.6 Persistence and DBMS integration

The application uses Hibernate (JPA) as the ORM and is configured with `hibernate.cfg.xml` to connect to an Oracle database.

File: `src/main/resources/hibernate.cfg.xml` - Key properties used: - `hibernate.connection.driver_class` — `oracle.jdbc.driver.OracleDriver` - `hibernate.connection.url` — `jdbc:oracle:thin:@localhost:1521:xe` - `hibernate.connection.username` & `hibernate.connection.password` — currently set to `system` / `Meenakshi@10` (replace for production) - `hibernate.dialect` — `org.hibernate.community.dialect.Oracle12cDialect` - `hibernate.hbm2ddl.auto` — `update` (development convenience). See security notes for production changes.

Snippet (`hibernate.cfg.xml` — connection/DDL mode)

```
<!-- File: src/main/resources/hibernate.cfg.xml -->
<property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property name="hibernate.connection.username">system</property>
<property name="hibernate.connection.password">Meenakshi@10</property>
<property
name="hibernate.dialect">org.hibernate.community.dialect.Oracle12cDialect</pr
operty>
<property name="hibernate.hbm2ddl.auto">update</property>
```

How integration is wired: - `HibernateUtil.getSessionFactory()` reads `hibernate.cfg.xml` (line: `new Configuration().configure("hibernate.cfg.xml").buildSessionFactory()`) and builds a `SessionFactory`. - `DatabaseService` uses `HibernateUtil.getSessionFactory().openSession()` to open a `Session`, starts a `Transaction`, executes queries or `persist`/`merge`/`remove`, then commits or rolls back. - Entities are mapped by `@Entity` and listed in `hibernate.cfg.xml` mapping entries.

Manual SQL provided: - `src/main/resources/sql/create_schema_oracle.sql` — complete DDL with sequences (`TASK_SEQ`, `USER_SEQ`, `PREFERENCE_SEQ`), tables (`users`, `tasks`, `user_preferences`), FK constraints and optional triggers.

Snippet (DDL excerpt — sequences & users table)

```
-- File: src/main/resources/sql/create_schema_oracle.sql
CREATE SEQUENCE TASK_SEQ START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE;
CREATE SEQUENCE USER_SEQ START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE;

CREATE TABLE users (
  id NUMBER(10) PRIMARY KEY,
  email VARCHAR2(255) NOT NULL,
  password VARCHAR2(255) NOT NULL,
  display_name VARCHAR2(255),
  created_at TIMESTAMP,
  last_login TIMESTAMP
);
ALTER TABLE users ADD CONSTRAINT uq_users_email UNIQUE (email);
```

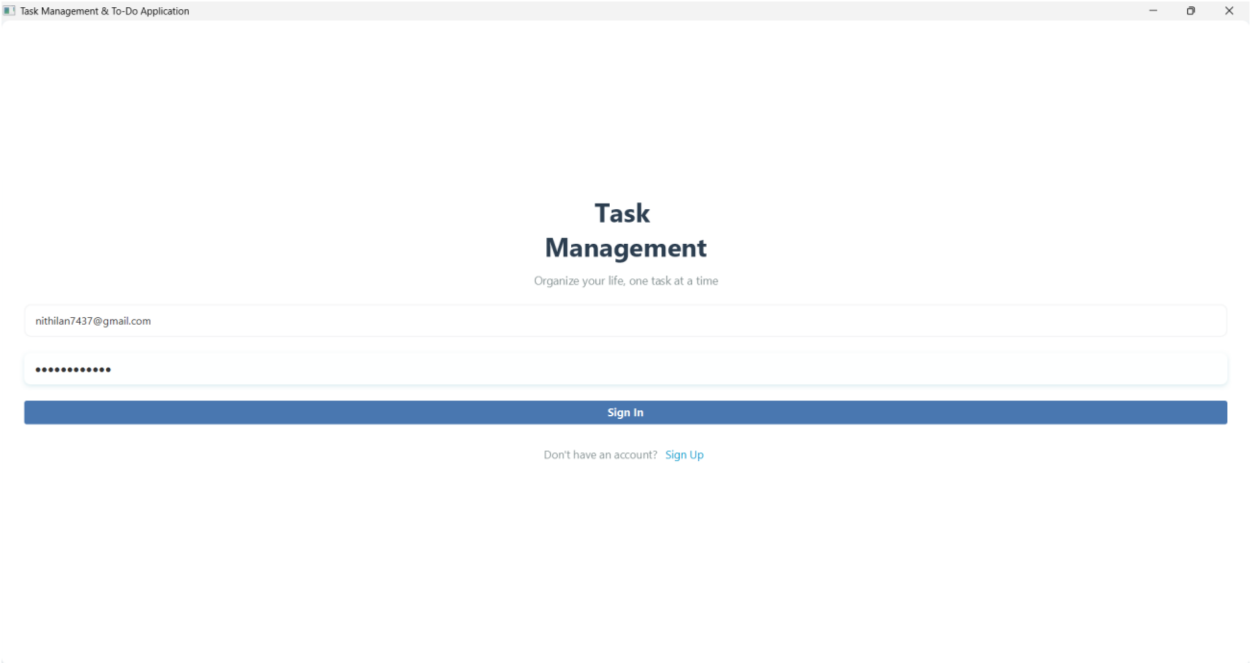
Why manual DDL? - For controlled production deployments, manual DDL minimizes accidental schema changes. `hibernate.hbm2ddl.auto=update` can be helpful during development but it's recommended to use `validate` in production.

Transaction and concurrency notes: - The app opens short-lived sessions for each operation and uses transactions for operations that change the DB. This ensures ACID correctness for each user action.

---

10. Screenshots and GitHub Submission

- ### Login Page

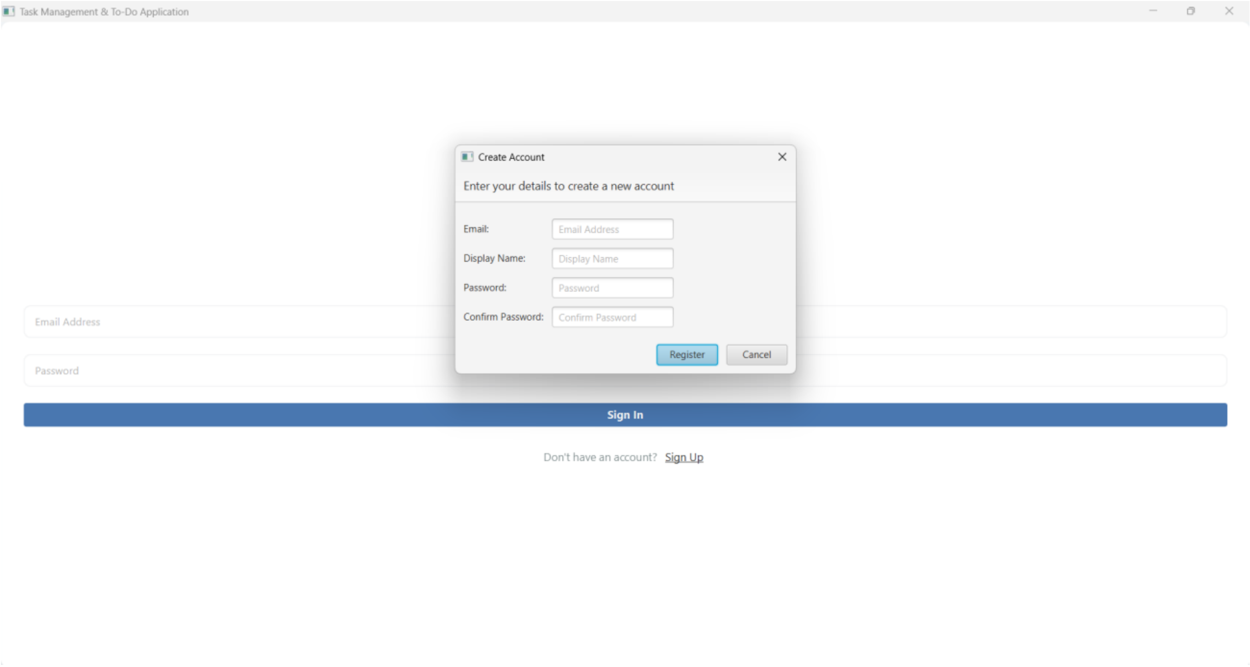

Task Management & To-Do Application
Task
Management
Organize your life, one task at a time

nithilan7437@gmail.com

••••••••••••

Sign In
Don't have an account? Sign Up

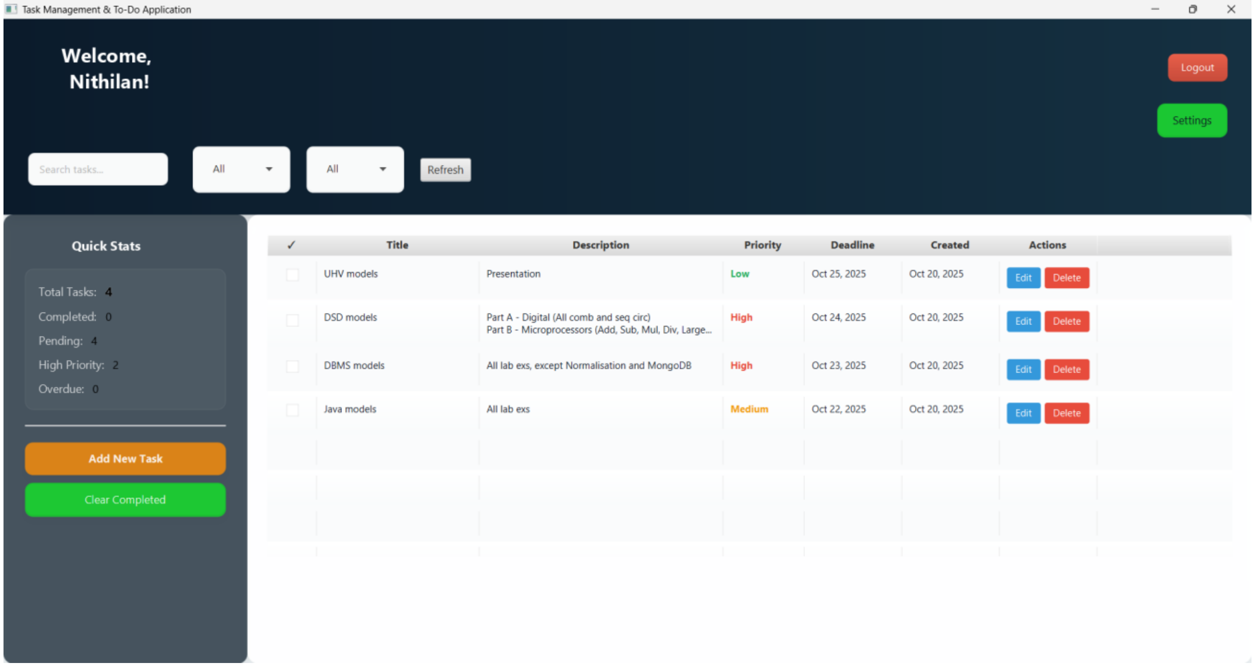
- ### Sign Up Page


Task Management & To-Do Application
Create Account

Enter your details to create a new account

Email: Email Address
Display Name: Display Name
Password: Password
Confirm Password: Confirm Password
Register
Cancel
Email Address
Password
Sign In
Don't have an account? Sign Up

- **Home Page**



Task Management & To-Do Application

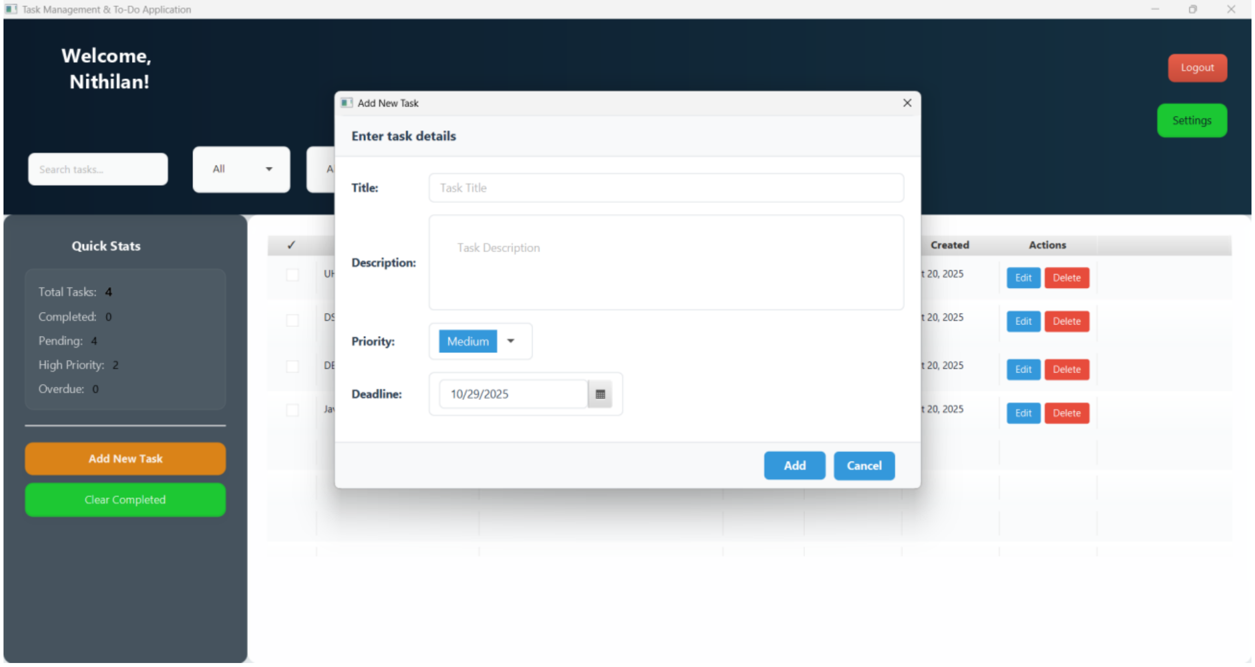
Welcome, Nithilan!

Quick Stats

Total Tasks: 4
Completed: 0
Pending: 4
High Priority: 2
Overdue: 0

✓	Title	Description	Priority	Deadline	Created	Actions
□	UHV models	Presentation	Low	Oct 25, 2025	Oct 20, 2025	Edit Delete
□	DSD models	Part A - Digital (All comb and seq circ) Part B - Microprocessors (Add, Sub, Mul, Div, Large...	High	Oct 24, 2025	Oct 20, 2025	Edit Delete
□	DBMS models	All lab exs, except Normalisation and MongoDB	High	Oct 23, 2025	Oct 20, 2025	Edit Delete
□	Java models	All lab exs	Medium	Oct 22, 2025	Oct 20, 2025	Edit Delete

- **Add New Task**



Task Management & To-Do Application

Welcome, Nithilan!

Add New Task

Enter task details

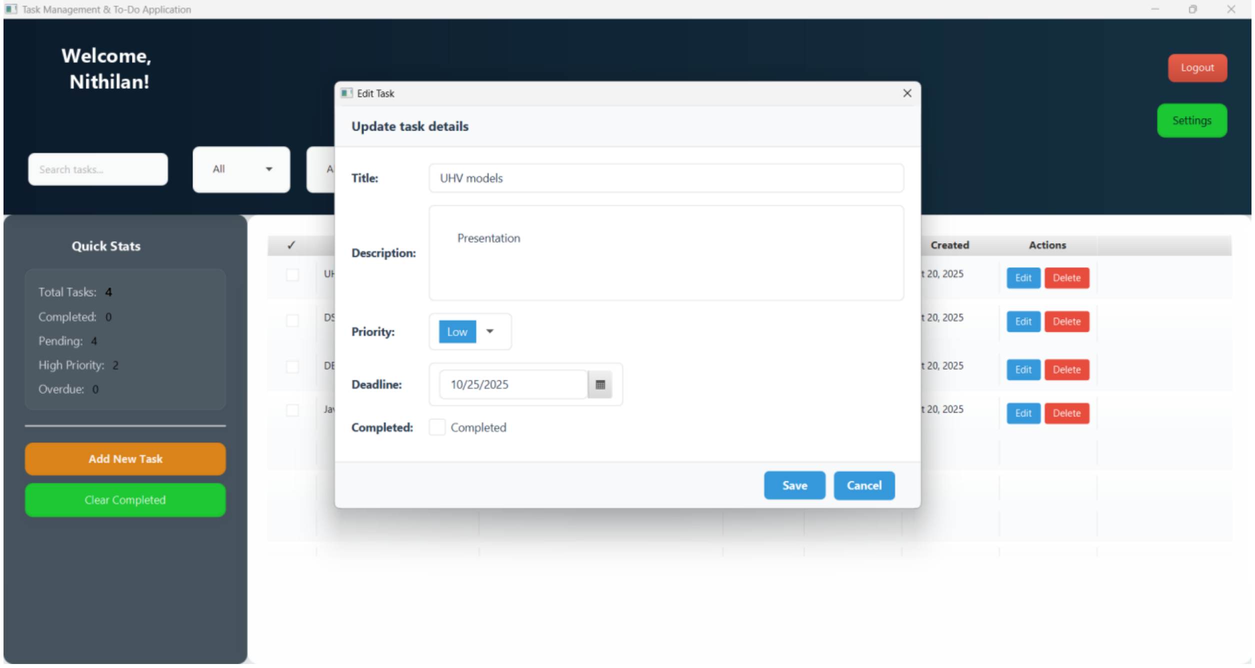
Title: Task Title

Description: Task Description

Priority: Medium

Deadline: 10/29/2025

Edit Task



Welcome, Nithilan!

Quick Stats

Total Tasks: 4
Completed: 0
Pending: 4
High Priority: 2
Overdue: 0

Update task details

Title: UHV models

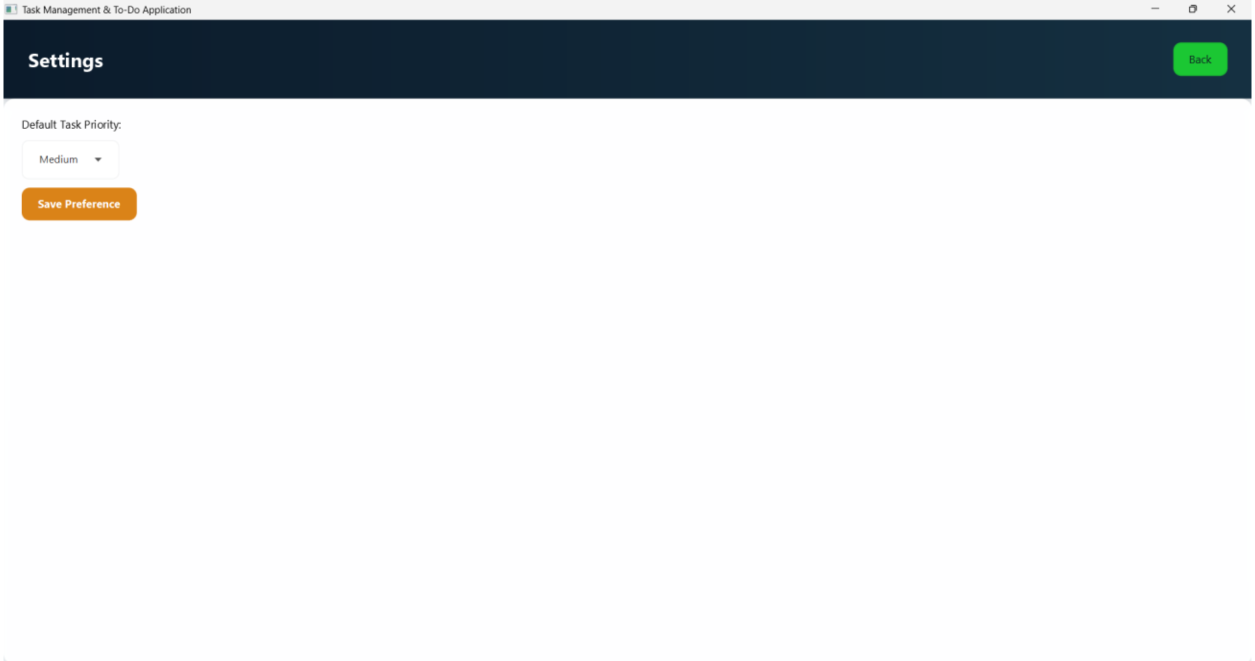
Description: Presentation

Priority: Low

Deadline: 10/25/2025

Completed: □ Completed

Settings (User Preferences)



Settings

Default Task Priority: Medium

Appendices

Appendix A — Full DDL Script

See `src/main/resources/sql/create_schema_oracle.sql` in the repository.

Appendix B — Hibernate Config Recommendation and XML diff

Current snippet in `src/main/resources/hibernate.cfg.xml`:

```
<property name="hibernate.hbm2ddl.auto">update</property>
```

Recommended change to validate-only mode:

```
<property name="hibernate.hbm2ddl.auto">validate</property>
```

Use `validate` in production to ensure the schema matches the mappings without making changes. Remove the property entirely to fully disable any checks.

Appendix D — Full entity sources (abridged)

The main entity classes are in `src/main/java/com/taskmanager/entity/`.

Task.java (abridged):

```
package com.taskmanager.entity;

import jakarta.persistence.*;
import java.time.LocalDate;
import java.time.LocalDateTime;

@Entity
@Table(name = "tasks")
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "task_seq
")
    @SequenceGenerator(name = "task_seq", sequenceName = "TASK_SEQ", allocati
onSize = 1)
    private int id;

    @Column(nullable = false)
    private String title;

    private String description;

    private String priority;

    private LocalDate deadline;
    private boolean completed;

    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    // constructors, getters, setters, @PreUpdate omitted for brevity
}
```

User.java and UserPreference.java are included in the codebase; the full files are included in the repository and this appendix references them by path.