## 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)

A diagram of a computer

AI-generated content may be incorrect.

## 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 quotas  
  
-- Alternatively, if you want to create objects in an existing schema, skip user 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)

* Authentication module: secure password storage is recommended (hashed + salted using BCrypt). Currently the project stores raw passwords — improve by integrating BCrypt (BCrypt library) for production.
* Task management module: provides filtering by priority/completion and ordering by deadline/created date.
* Preferences module: persists user settings; used at startup to apply UI theme and default sort order.

## 8. Code Snippets (extended)

* HibernateUtil (simplified):

public class HibernateUtil {  
 private static SessionFactory sessionFactory;  
 public static synchronized SessionFactory getSessionFactory() {  
 if (sessionFactory == null) {  
 Configuration cfg = new Configuration().configure();  
 StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder()  
 .applySettings(cfg.getProperties());  
 sessionFactory = cfg.buildSessionFactory(builder.build());  
 }  
 return sessionFactory;  
 }  
}

* DatabaseService: sample query for tasks by priority:

public List<Task> getUserTasksByPriority(int userId, String priority) {  
 try (Session s = HibernateUtil.getSessionFactory().openSession()) {  
 return s.createQuery("from Task t where t.user.id = :uid and t.priority = :p", Task.class)  
 .setParameter("uid", userId)  
 .setParameter("p", priority)  
 .list();  
 }  
}

## 9. Testing Plan and Sample Test Cases

Testing strategy:

* Unit tests: isolate service-layer logic (mock SessionFactory or use an in-memory DB such as H2 with a Hibernate config profile).
* Integration tests: spin up a transient Oracle/Postgres instance (Docker) and run migrations or manually provision schema before tests.
* Manual acceptance tests: run the application, register user, create tasks, edit tasks, delete tasks; verify referential integrity.

Sample test cases (manual):

1. Register and login

* Steps: Open app -> Register with email/password -> Login
* Expected: user created in users table; login succeeds; dashboard loads with empty task list.

1. Create task and mark complete

* Steps: Add task with title/description/priority/deadline -> Verify in UI and DB -> Mark complete
* Expected: tasks.completed toggles from 0 to 1; updated\_at changes.

1. Cascade delete

* Steps: Create user with tasks -> Delete user via DB -> Expected: tasks deleted by ON DELETE CASCADE constraint.

## 10. Screenshots and GitHub Submission

Screenshots:

* **Login Page**

A computer screen shot of a task management

AI-generated content may be incorrect.

* **Sign Up Page**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Home Page**

A screenshot of a computer

AI-generated content may be incorrect.

* **Add New Task**

A screenshot of a computer

AI-generated content may be incorrect.

* **Edit Task**

A screenshot of a computer

AI-generated content may be incorrect.

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**](https://github.com/Nithilan77/Task_Management_And_To_Do_Application)

## 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", allocationSize = 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.