**DIGITAL ASSET MANAGEMENT**

**Requirement documentation:**

**1.INTRODUCTION:**

The Digital Asset Management System is developed to streamline the tracking, allocation, reservation, and maintenance of organizational assets. It connects to a MySQL database and provides complete CRUD and tracking operations through a menu-driven Python application. The system strictly adheres to object-oriented design, uses collections, exception handling, and includes unit testing for reliability.

**2.FUNCTIONAL REQUIREMENTS:**

**2.1. Asset Management:**

Handling the full lifecycle of assets from creation to deletion all are performed by this asset management.

**Add asset:** When a new asset like laptop, vehicle, projector is purchased and needs to be registered into the system.

**Update asset:** When existing asset details need to change.If Location is changed when the product is changed to under maintenace to delivered state and status also updated.Ownership is transferred from one person to another.

**Delete Asset**: When an asset is damaged beyond repair, sold, or decommissioned and must be removed from the database

2.2 Asset Tracking:

**Allocate Asset**:  
When an admin wants to assign an asset like laptop to an employee for work use along with the allocation date.

**Deallocate Asset**:  
When the employee returns the asset after a project or when leaving the company and the return date must be recorded and then the data should be deleted.

**Reserve Asset**:  
When an employee wants to book an asset in advance for a certain period .Previously booking a projector for a meeting from June 20–22.

**Withdraw Reservation**:  
When the employee no longer needs the reserved asset and cancels the booking then the allocated data should be returned.

**2.3. Asset Maintenance:**

Asset Maintenance is used to log service or maintenance records for assets.

**Perform Maintenance**:  
When any asset undergoes repair, service, or inspection the data need to be maintained form the admin side.

* Date of maintenance
* Description like what are the process had done.
* Cost

**3.DATABASE DESIGN**

TABLE 1: employee

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| employee\_id | INT (PK) | Unique ID for each employee |
| name | VARCHAR(100) | Name of employee |
| department | VARCHAR(100) | Department |
| email | VARCHAR(100) | Email address |

TABLE 2: assets

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| asset\_id | INT (PK) | Unique ID for each asset |
| name | VARCHAR(100) | Asset name |
| type | VARCHAR(100) | Type (e.g., Laptop, Monitor) |
| serial\_number | VARCHAR(100) | Unique serial number |
| purchase\_date | DATE | Date of purchase |
| location | VARCHAR(100) | Location where the asset is kept |
| status | VARCHAR(50) | available / allocated / maintenance |
| owner\_id | INT (FK) | Employee to whom asset is assigned |

TABLE 3: asset\_allocations

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| allocation\_id | INT (PK) | Unique ID for allocation record |
| asset\_id | INT (FK) | Asset being allocated |
| employee\_id | INT (FK) | Employee assigned |
| allocation\_date | DATE | Date asset was assigned |
| return\_date | DATE | Date asset was returned |

TABLE 4: maintenance\_records

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| record\_id | INT (PK) | Unique maintenance record ID |
| asset\_id | INT (FK) | Asset under maintenance |
| maintenance\_date | DATE | Date of maintenance |
| description | TEXT | Issue and resolution details |

TABLE 5: reservations

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| reservation\_id | INT (PK) | Unique ID for reservation |
| asset\_id | INT (FK) | Reserved asset |
| employee\_id | INT (FK) | Employee requesting reservation |
| reservation\_date | DATE | Date of reservation |

**4. System Architecture**

* **entity/** → Model classes like Asset, Employee, etc.
* **dao/** → Handles direct MySQL queries using mysql.connector
* **service/** → Business logic layer (allocating, maintaining, etc.)
* **exception/** → Custom exception classes for error scenarios
* **util/** → DB connection and utility methods
* **main/** → Entry point for the application with menu-driven interface

**5. Operating Environment**

* **Operating System**: Windows
* **Python Version**: 3.13.4
* **MySQL Version**: 8.0.42
* **IDE**: VS Code

6.NON FUNCTIONAL REGIREMENTS:

Security: Only authorized users (Admin/Employee) can perform actions.

Performance: Database queries must be optimized for speed.

Usability: Menu-driven, easy-to-use application interface.

7.TECHNICAL REGUIREMENTS:

Python: Implement the system using Python language.

SQL Database: Store and retrieve data using SQL queries.

User-Defined Exceptions: Implement and handle custom exceptions EG: AssetNotFound.

8.PRIORITIZE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
| PRIORITY | REQUIREMENT | REASON |
| HIGH | Add / Update / Delete Asset | Core asset operations basic need to be done |
| HIGH | Allocate / Deallocate Asset | Tracks asset use. |
| HIGH | Database Connection Utility | Required for any DB operation to work. |
| HIGH | Exception Handling | Needed to handle invalid data and ensure system doesn’t crash. |
| Medium | Perform Maintenance | Helps track asset health, required before allocation. |
| Medium | Reserve / Withdraw Reservation | Useful for future planning and asset scheduling. |
| Medium | Asset Tracking | Helps admins see usage details; useful for reports. |
| Low | Unit Testing | Ensures system correctness; can be added after core features. |
| Low | Non-functional | Important for long-term, but not first priority during development. |

9. **EXCEPTION HANDLING:**

**AssetNotFoundException**

* + Triggered when an invalid asset ID is given.
  + When a user tries to update or delete an asset that doesn't exist in the database.

**AssetNotMaintainException**

* + Triggered when trying to allocate an asset **that hasn’t had maintenance in the last 2 years**.
  + This enforces asset safety before allocation.

10.UNIT TESTING:

|  |  |
| --- | --- |
| test\_add\_asset\_success | Confirms that a new asset can be added correctly |
| test\_perform\_maintenance\_success | Confirms maintenance can be recorded for an asset |
| test\_reserve\_asset\_success | Confirms reservation logic is working |
| test\_asset\_not\_found\_exception | Ensures AssetNotFoundException is raised for invalid asset ID |
| test\_asset\_not\_maintained\_exception | |  | | --- | |  |  |  | | --- | | Ensures AssetNotMaintainException is raised if asset not maintained in 2 years | |

11.ARCHITECTURE:

User Interface

Main Menu in main module

Service Layer (dao)

Interface: AssetManagementService Implementation: AssetManagementServiceImpl

Entity Layer (entity)

Asset, Employee, Reservation, etc.

MySQL Relational DB

Tables: assets, employees, maintenance\_records, etc.

Database Access Layer

(DBConnUtil + SQL)