



VIT<sup>®</sup>  
—  
BHOPAL

**NAME: ISHIKA**

**REGISTRATION NO.:25BCE10999**

**COURSE FACULTY: SMT.PAVITHRA**

# Personal Notes & Assignment Deadline Tracker

## 1. Introduction

- AI-powered academic organization system
- Python-based desktop application
- Combines task management with ML intelligence

## 2. Problem Statement

- Students struggle with deadline management
- No intelligent prioritization of academic tasks
- Lack of predictive risk assessment for assignments

## 3. Functional Requirements

- Notes CRUD operations with categorization
- Assignment tracking with deadlines and priorities
- ML risk prediction and classification
- Automated work scheduling
- Search and data visualization

## 4. Non-functional Requirements

- Performance: <2 second operations
- Reliability: 99% uptime with data persistence
- Usability: Intuitive menu-driven interface
- Security: Input validation and secure storage

## 5. System Architecture

Layered Architecture:

Presentation → Application → Domain → Data

(Menu UI) → (Services) → (Models) → (JSON Storage)

## 6. Design Diagrams

### ○ Use Case Diagram

Student → Manage Notes → Manage Assignments → Get ML Insights → View Analytics

### ○ Workflow Diagram

Start → Login → Main Menu → Select Feature → Process → Save → Continue/Exit

### ○ Sequence Diagram

User → UI → Service → Repository → Storage  
←      ←      ←      ←

### ○ Class/Component Diagram

TrackerApp → NoteRepo + AssignmentRepo + MLService + Config

### ○ ER Diagram

Notes[id,title,content,priority] ↔  
Assignments[id,title,due\_date,status,priority]

## 7. Design Decisions & Rationale

- **JSON Storage:** Chosen for simplicity and portability
- **Menu-based UI:** Optimal for educational demonstration

- **Custom ML:** Rule-based for transparency and control
- **Modular Design:** Easy maintenance and extensibility

## 8. Implementation Details

- **Language:** Python 3.8+
- **Storage:** JSON files for notes and assignments
- **ML Algorithm:** Weighted scoring system for risk prediction
- **Architecture:** Repository pattern with service layers

## 9. Testing Approach

- **Unit Testing:** Individual component validation
- **Integration Testing:** End-to-end workflow verification
- **User Acceptance:** Sample student scenarios
- **Data Persistence:** Restart and data recovery tests

## 10. Challenges Faced

- Date parsing and validation complexity
- JSON serialization/deserialization issues
- ML model accuracy tuning
- Error handling across multiple layers

## 11. Learnings & Key Takeaways

- Importance of modular architecture
- Value of comprehensive input validation
- Benefits of separation of concerns
- ML doesn't require complex algorithms for practical use

## 12. Future Enhancements

- Web-based interface with React frontend
- Mobile application development

- Cloud synchronization
- Advanced AI with neural networks
- Integration with calendar APIs
- Multi-user collaboration features

## 13. References

- Python Official Documentation
- Software Design Patterns (Repository, Service Layers)
- Academic papers on student time management
- ML literature on risk prediction algorithms