MAKING A CALENDAR

A Guide to Design, Layout, and Creativity

Introduction

- Smart calendar application with AI-powered features
- Combines event management with machine learning
- Built using Python with zero external dependencies
- Demonstrates CRUD, ML predictions, and visualization in one system

Problem Statement

- Manual calendar management is time-consuming and error-prone
- No intelligent event categorization or priority suggestions
- Poor visualization leads to schedule conflicts and missed appointments
- Users need automated organization and smart insights

Functional Requirements

- CRUD Operations: Create, read, update, delete events
- ML Predictions: Automatic event categorization and priority assessment
- Visualization: Monthly and daily calendar views
- Data Persistence: JSON file storage with automatic saving
- Input Validation: Date/time format checking and error handling

Non-Functional Requirements

- Performance: Fast response times for all operations
- Usability: Intuitive menu-driven interface
- Reliability: Data persistence and error recovery
- Maintainability: Modular, documented code structure
- Security: Input validation and data integrity

System Architecture

```
Presentation Layer → Business Logic Layer → Data Access Layer
    (Menu UI)        (Calendar Manager)     (JSON Storage)
                     (ML Predictor)
                     (Visualizer)
```

Design Diagrams

Use Case Diagram

- Actor: User

- Use Cases: Add Event, View Calendar, Update Event, Delete Event, Get Predictions, View Simulation

Workflow Diagram

1. Start → Authentication → Main Menu → Select Operation → Process → Save → Return to Menu

Sequence Diagram

```
User → MainApp → CalendarManager → JSON Storage
User → MainApp → EventPredictor → Return Predictions
User → MainApp → Visualizer → Display Output
```

Class Diagram

```

CalendarManager ──┐

EventPredictor ──┤ → MainApp

Visualizer ──────┘

```

ER Diagram

```

Events [id, title, description, date, time, category, priority]

```

Design Decisions & Rationale

- JSON Storage: Chosen for simplicity and no database setup
- Rule-based ML: Selected for interpretability and no training data requirement
- Console Interface: Prioritized functionality over GUI for demo purposes
- Modular Design: Separated concerns for maintainability

Implementation Details

- Language: Python 3.x
- Storage: JSON file-based system
- ML Approach: Keyword-based classification
- Architecture: Layered pattern with separation of concerns
- Error Handling: Comprehensive try-catch blocks

Testing Approach

- Unit Testing: Manual testing of each function

- Integration Testing: End-to-end workflow validation
- Validation Testing: Input format and error handling checks
- Persistence Testing: Data save/load verification

Challenges Faced

- Module naming conflict with Python's built-in calendar
- Date/time parsing and validation complexity
- Balancing simplicity vs feature completeness
- JSON serialization of datetime objects

Learnings & Key Takeaways

- Importance of proper module naming conventions
- Value of separation of concerns in architecture
- Effectiveness of rule-based ML for simple classification
- Benefits of comprehensive error handling

Future Enhancements

- Web-based GUI interface
- Integration with real calendar APIs (Google Calendar)
- Advanced ML models with proper training data
- Multi-user support with authentication
- Mobile application version
- Conflict detection and resolution

References

- Python Official Documentation

- JSON Schema Standards
- Software Architecture Patterns
- Machine Learning Fundamentals