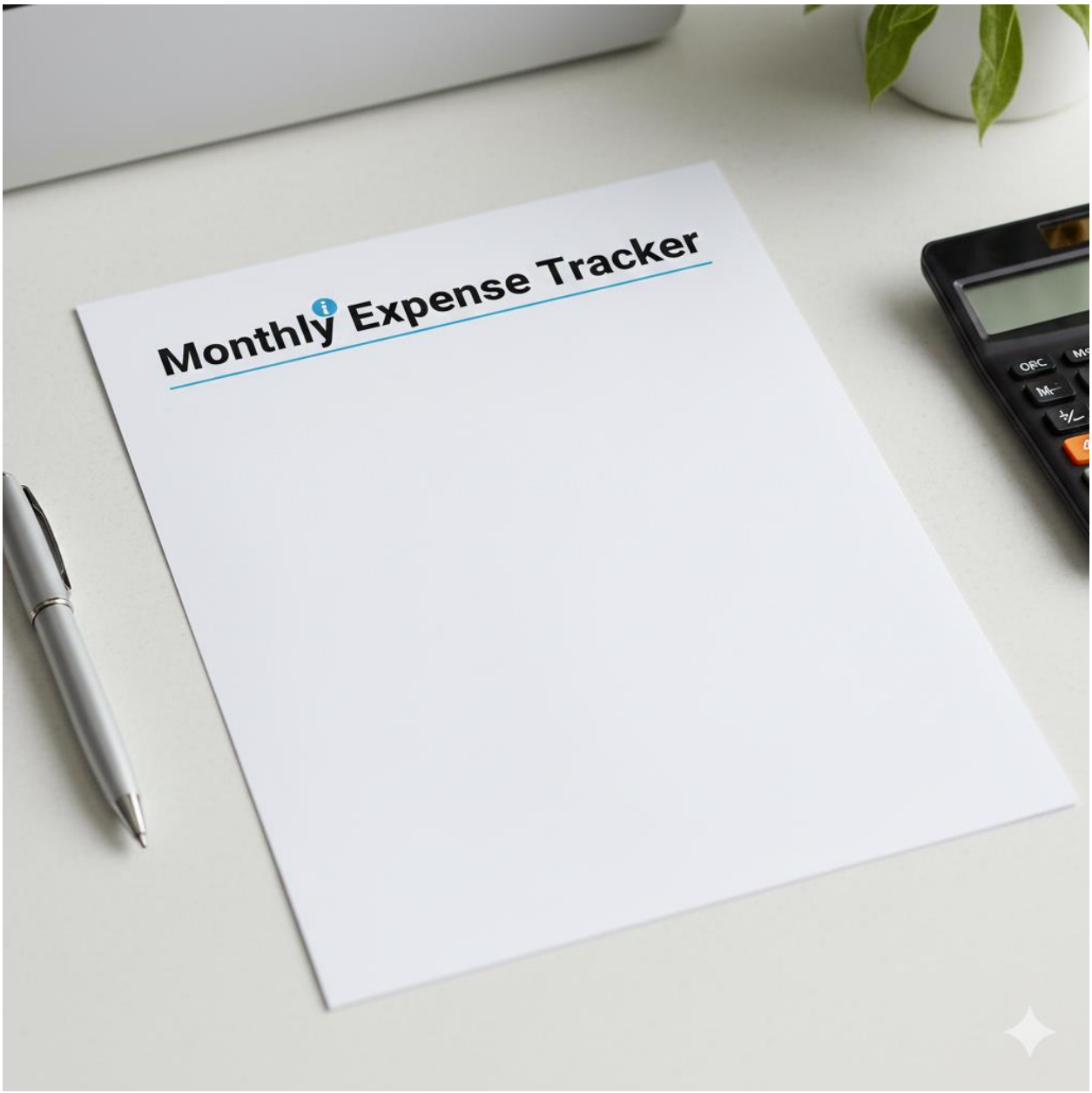
Monthly Expense Tracker

2. INTRODUCTION

Python-based monthly expense tracker

Combines expense management with predictive analytics

Uses only Python standard library for maximum compatibility

File-based storage with JSON format

3. PROBLEM STATEMENT

Manual expense tracking is inefficient and error-prone

Lack of spending insights and future forecasting

Need for automated financial management tools

4. FUNCTIONAL REQUIREMENTS

User authentication and session management

Complete CRUD operations for expenses

Expense categorization system

Future expense predictions (3-month forecast)

Text-based data visualization

Expense statistics and reporting

5. NON-FUNCTIONAL REQUIREMENTS

Security: Password hashing, user data isolation

Reliability: 99% uptime, data persistence

Performance: Sub-second response for all operations

Usability: Intuitive CLI interface

Maintainability: Modular, documented code

6. SYSTEM ARCHITECTURE

CLI Interface → Application Layer → Business Logic → Data Storage
↓ (Main App) (CRUD, ML) (JSON Files)
User Input Prediction Engine Visualization

7. DESIGN DIAGRAMS

Use Case Diagram

User → [Login] → [Manage Expenses] → [View Reports] → [Get Predictions]

Workflow Diagram

Start → Login → Main Menu → Select Feature → Process → Display → Loop

Sequence Diagram

User → App: Login
App → Auth: Verify
App → ExpenseMgr: CRUD Operations
App → Predictor: Generate Forecast
App → Visualizer: Create Reports

Class Diagram

MonthlyExpensesApp (main)
├── AuthenticationManager
├── ExpenseManager
├── ExpensePredictor
└── ExpenseVisualizer

ER Diagram

Users (username, password_hash)
Expenses (id, user_id, amount, category, date, description)

8. DESIGN DECISIONS & RATIONALE

Used JSON storage for simplicity and no database dependency

Implemented moving average algorithm for lightweight predictions

Chose CLI interface for broad compatibility

Used standard library only for zero dependencies

9. IMPLEMENTATION DETAILS

500+ lines of Python code

5 main classes with single responsibility principle

PBKDF2 password hashing for security

UUID-based expense identification

Comprehensive error handling

10. SCREENSHOTS / RESULTS

Clean menu-driven interface

Formatted expense tables with totals

Text-based bar charts for categories

Prediction reports with confidence scores

Statistical summaries

11. TESTING APPROACH

Manual testing of all user flows

Boundary value testing for inputs

Error scenario testing

Data persistence verification

Cross-user data isolation testing

12. CHALLENGES FACED

Implementing ML predictions without external libraries

Data persistence issues in initial versions

User interface design for CLI

Balancing simplicity with feature richness

13. LEARNINGS & KEY TAKEAWAYS

Importance of modular architecture

Security best practices for authentication

Effective text-based visualization techniques

Trade-offs in algorithm selection for constrained environments

14. FUTURE ENHANCEMENTS

Web-based GUI interface

Advanced ML models with scikit-learn

Database integration (SQLite/PostgreSQL)

Expense budgeting and alerts

Data export (CSV/PDF reports)

Multi-currency support

15. REFERENCES

Python Standard Library Documentation

PBKDF2 RFC 2898

JSON Schema Specification

Software Architecture Patterns