MAT NUMBER: 92130758
COURSE CODE: DLBDSOOFPP01
COURSE: OBJECT-ORIENTED AND FUNCTIONAL PROGRAMMING WITH PYTHON

PORTFOLIO: HABIT TRACKING APP

TUTOR: Mirmehdi Seyedebrahim

🎯 Habit Tracker: Conception Phase Document

1. Project Overview and Goals

Goal: Build a professional habit tracking application demonstrating Object-Oriented Programming (OOP) and Functional Programming (FP) paradigms.

Key Technologies:

- Python 3.12+ with modern features
- SQLite for data persistence
- Questionary for CLI interface
- Pytest for comprehensive testing

Architecture: Layered package structure with clear separation of concerns

2. System Architecture

Package Structure:

```
src/
├── 📊 analytics/           # FP Analytics Layer
│   └── analytics_service.py
├── 💻 cli/                 # Presentation Layer
│   └── user_interface.py
├── 🏗️ data_model/          # Domain Layer
│   ├── habit.py
│   └── completion.py
├── ⚙️ managers/            # Business Logic Layer
│   └── habit_manager.py
└── 🗄️ storage/             # Data Persistence Layer
    └── db.py
```

Layer Responsibilities:

The application is structured into modules, each with clear responsibility:

Data Model (habit.py, completion.py): Defines the core classes of the app.

- BaseHabit, DailyHabit, WeeklyHabit (habits) – represent different types of habits.
- Completion (check-off records) – represents when a habit is checked off (with timestamp, notes, mood).

Storage Layer (db.py): Manages SQLite database.

- Create DatabaseHandler, User, SQLite schema.
- Stores, retrieves and secures data so habits persist across sessions.

Managers (habit_manager.py): Implement the business logic.

- Create HabitManager, HabitFactory.
- Workflow coordination and object creation.

Analytics Service (analytics_service.py): Provides Functional Programming logic:

- Calculate current streak and longest streak,
- Finds inactive habits.
- Filter habits by daily/weekly.

CLI Interface (user_interface.py): Menu-driven interface built with Questionary.

- User interaction and input validation.

3. Core Features

User Management:

- Secure registration/login with password hashing
- Guest mode for quick testing
- Demo account with sample data

Habit Tracking:

- Create daily/weekly habits
- Check-off completions with optional notes/mood scores
- Soft deletion with data preservation

Analytics:

- Current and longest streak calculations
- Habit filtering by periodicity
- Inactive habit identification
- Overall progress statistics

4. Object-Oriented Design

Class Hierarchy:

```
# Abstract base class defining common interface
class BaseHabit(ABC):
    ├── DailyHabit      # Concrete implementation for daily habits
    └── WeeklyHabit     # Concrete implementation for weekly habits
```

Design Patterns Implemented:

Factory Pattern:

```
class HabitFactory:
    @staticmethod
    def create_habit_from_db(data: dict) -> BaseHabit
```

- HabitFactory.create_habit_from_db() creates appropriate objects from database data
- Handles datetime conversion between database and Python objects

Repository Pattern:

```
class DatabaseHandler:
    # CRUD operations for all entities
    def save_habit(), get_habits_for_user(), etc.
```

- DatabaseHandler abstracts all database operations
- Clean API for CRUD operations

Service Layer Pattern:

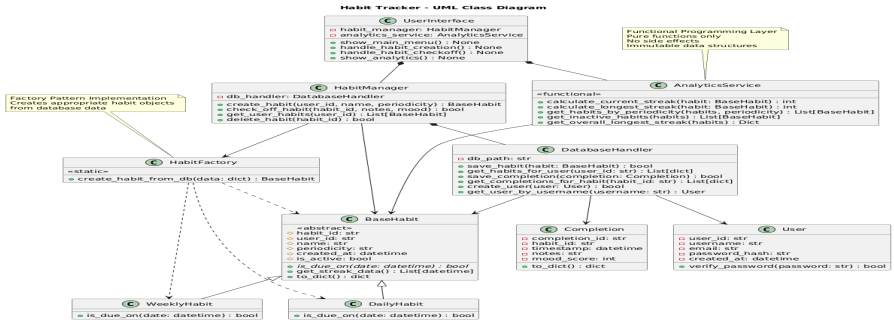
```
class HabitManager:
    # Coordinates operations across components
    def create_habit(), check_off_habit(), etc.
```

- HabitManager coordinates business workflows
- Clear separation between domain and application logic

OOP Principles:

- **Encapsulation:** Private attributes with controlled access
- **Inheritance:** Shared functionality in base class
- **Polymorphism:** Different is_due_on() implementations
- **Abstraction:** Clear interface separation

5. THE UML DIAGRAM



6. Database Design

Schema with Constraints:

Users Table:

- user_id (PK), username (Unique), email (Unique)
- password_hash (SHA-256 with salt), created_at

Habits Table:

- habit_id (PK), user_id (FK), name, periodicity (Check constraint)
- created_at, is_active (Soft deletion flag)

- Unique constraint: (user_id, name)

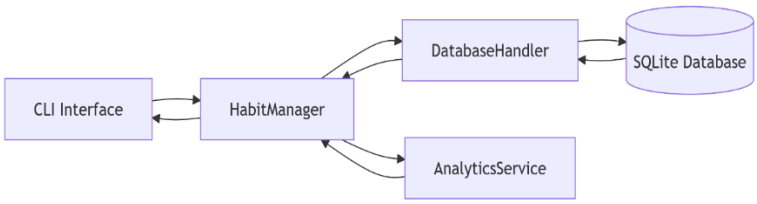
Completions Table:

- completion_id (PK), habit_id (FK), timestamp
- notes, mood_score (Check: 1-10 or NULL)

Data Integrity:

- Foreign key constraints with cascade delete
- Unique constraints prevent duplicates
- Check constraints enforce business rules
- Indexes on frequently queried columns

DATA FLOW



7. Functional Programming Implementation

Pure Functions in AnalyticsService:

Key Functions:

- calculate_current_streak(habit) → int
- calculate_longest_streak(habit) → int
- get_habits_by_periodicity(habits, periodicity) → List[BaseHabit]
- get_overall_longest_streak(habits) → Dict

FP Principles Applied:

Immutability:

- Input data never modified
- New structures created for results
- Predictable state management

Referential Transparency:

- Same inputs → Same outputs
- No hidden dependencies
- Easy testing and reasoning

Higher-Order Functions:

- filter() for habit filtering
- map() for data transformation
- reduce() for streak calculations

Analytics Algorithms:

Daily Streaks:

- Consecutive calendar days with completions
- Gap detection for streak breaks

Weekly Streaks:

- ISO calendar week-based tracking
- Year boundary handling

8. Testing Strategy

Comprehensive Test Suite (40+ Tests):

Test Categories:

- **Unit Tests:** Individual components (OOP models, FP functions)
- **Integration Tests:** Cross-module workflows
- **End-to-End Tests:** Complete user journeys
- **Database Tests:** CRUD operations and constraints

Test Files:

- test_db.py - Database operations (10+ tests)
- test_habit.py - OOP model validation (10+ tests)
- test_analytics.py - FP function correctness (10+ tests)
- test_habit_manager.py - Business logic (8+ tests)
- test_integration.py - End-to-end workflows (5+ tests)

Windows Compatibility:

- Proper database connection cleanup
- File locking awareness
- Graceful degradation when cleanup fails

9. User Interface Design

FLOWCHART

Authentication Options:

- Login with existing credentials
- Register new account
- Continue as Guest (uses demo account)

Main Menu Features:

- View All Habits
- Create New Habit
- Check Off Habit
- Analytics Dashboard
- View Completions
- Switch User and Exit

User Experience:

- Interactive menus with Questionary library
- Visual feedback with emojis and colors
- Input validation and error handling
- Progressive disclosure of features

10. Technical Innovations

Python 3.12+ Compatibility:

- Custom datetime adapters for SQLite
- Modern type hints throughout codebase
- Context managers for resource management
- **Production-Ready Features:**

Security:

- Password hashing with SHA-256 and salt

- SQL injection prevention with parameterized queries
- Input validation at multiple levels

Error Handling:

- Comprehensive exception handling
- Graceful degradation strategies
- Informative error messages

Performance:

- Database indexes on frequently queried columns
- Efficient connection management
- Optimized streak calculation algorithms

11. Sample Data & Demonstration

Predefined Setup:

- **5 Sample Habits** (mix of daily/weekly)
- **4 Weeks of Sample Data** with realistic patterns
- **Demo Account:** username: demo, password: demo123

Edge Case Coverage:

- Month and year boundary crossings
- Streak break scenarios
- Inactive habit identification
- Data validation edge cases

12. Conclusion

This conception outlines a **production-ready habit tracker** that demonstrates:

- **Clean Architecture** with layered package design
- **Advanced OOP Patterns** (Factory, Repository, Service Layer)
- **Pure Functional Programming** for predictable analytics
- **Comprehensive Testing** with 40+ passing tests
- **Windows Compatibility** with robust file handling
- **Security & Data Integrity** through constraints and hashing

The system provides a solid foundation for future enhancements while delivering a reliable, user-friendly habit tracking experience.