PHASE 2/DEVELOPMENT PHASE: HABIT TRACKING APP

- Course: DLBDSOOFPP01 - Object Oriented and Functional Programming with Python
- Name: Oluwatoyin Eniola Ipadeola
- Mat Number: 92130758
- Date:10/08/2025

- 📂 Repository: github.com/ipadeolaoluwatoyin7880/habit_tracker

PROJECT OVERVIEW & TECHNOLOGY STACK

Requirements:

- **Python Version**

```
Python 3.7 or higher
```

- **Install Dependencies:**

```
pip install -r requirements.txt
```

- **Setup:**
- **Clone the Repository:**

```
git clone https://github.com/ipadeolaoluwatoyin7880/habit_tracker.git
```

```
cd habit_tracker
```

- **Start the Application:**

```
python main.py
```

Project Goal: menus

- Develop a robust habit tracking application
- Demonstrate OOP and FP programming paradigms
- Create modular, testable, and extensible code

Version Control: Git & GitHub

Technology Stack:

- **Language:** Python 3.11+
- **Database:** SQLite with built-in sqlite3 module
- **Testing:** Pytest framework
- **CLI:** Questionary library for interactive

PROJECT STRUCUTRE AND RESPONSIBILITY

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
```

RESPONSIBILITIES

The application is structured into modules, each with a 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, and 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.

OBJECT-ORIENTED DESIGN

Design Patterns:

- Factory Pattern: HabitFactory.create_habit_from_db()

1. HabitFactory creates habit objects from database data.
2. Decouples object creation from business logic

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

- Repository Pattern: DatabaseHandler

1. DatabaseHandler abstracts data access.
2. Business logic independent of storage technology

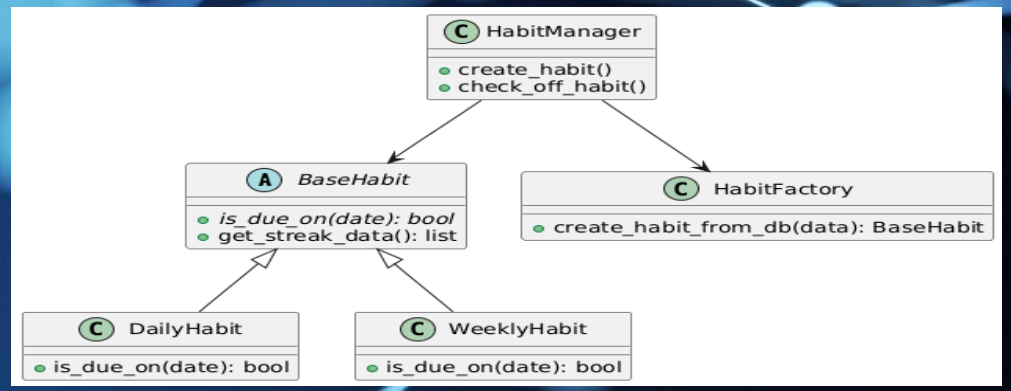
```
class DatabaseHandler:
    # CRUD operations for all entities
    def save_habit(), get_habits_for_user(), etc.
```

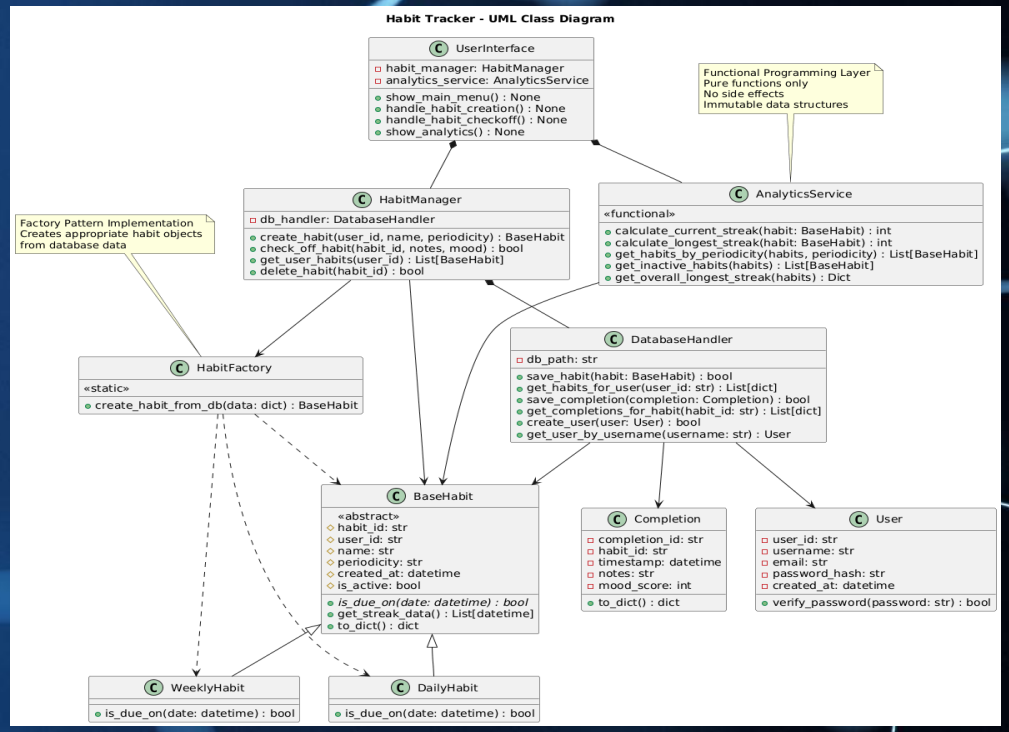
- Service Layer: HabitManager

1. HabitManager and AnalyticsService Orchestrate complex workflows.
2. Clean separation between domain and application logic

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

UML Class Diagram





OBJECT-ORIENTED PROGRAMMING IMPLEMENTATION

OOP Principles Applied:

- ✅ **Inheritance:** Daily/Weekly habits extend BaseHabit
- ✅ **Polymorphism:** Different is_due_on() implementations
- ✅ **Encapsulation:** Private attributes with public methods
- ✅ **Abstraction:** Base class defines interface
- ✅ **Composition:** Habits contain Completion objects

Key Features:

- Factory pattern for habit creation
- Enum for type-safe periodicity
- Clean separation of concerns

```python
from abc import ABC, abstractmethod
from datetime import datetime, timedelta

class BaseHabit(ABC):
    def __init__(self, name: str, periodicity: str):
        self.name = name
        self.periodicity = periodicity
        self.created_at = datetime.now()
        self.is_active = True

    @abstractmethod
    def is_due_on(self, date: datetime) -> bool:
        pass

class DailyHabit(BaseHabit):
    def is_due_on(self, date: datetime) -> bool:
        # Due every day
        return True

class WeeklyHabit(BaseHabit):
    def is_due_on(self, date: datetime) -> bool:
        # Due on specific weekday
        return date.weekday() == self.target_weekday
```

FUNCTIONAL PROGRAMMING IMPLEMENTATION

🧮 FP Analytics: Pure & Predictable

Analytics Module Features:

- Pure functions for streak calculations
- Immutable data processing
- Higher-order functions: filter(), map(), reduce()
- Comprehensive statistics generation

Core FP Functions:

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

FP Benefits:

- 🧪 Easy testing and debugging
- 🔄 Predictable behavior
- 📊 Composability and reusability

```
from typing import List, Dict
from datetime import datetime, timedelta

def calculate_current_streak(completions: List[datetime]) -> int:
    """Pure function to calculate current streak"""
    if not completions:
        return 0

    sorted_dates = sorted(completions, reverse=True)
    streak = 1 if datetime.now().date() == sorted_dates[0].date() else 0

    for i in range(1, len(sorted_dates)):
        if (sorted_dates[i-1].date() - sorted_dates[i].date()).days == 1:
            streak += 1
        else:
            break

    return streak

def get_habits_by_periodicity(habits: List[BaseHabit],
                              periodicity: str) -> List[BaseHabit]:
    """Filter habits by periodicity using FP"""
    return list(filter(lambda habit: habit.periodicity == periodicity,
habits))
```

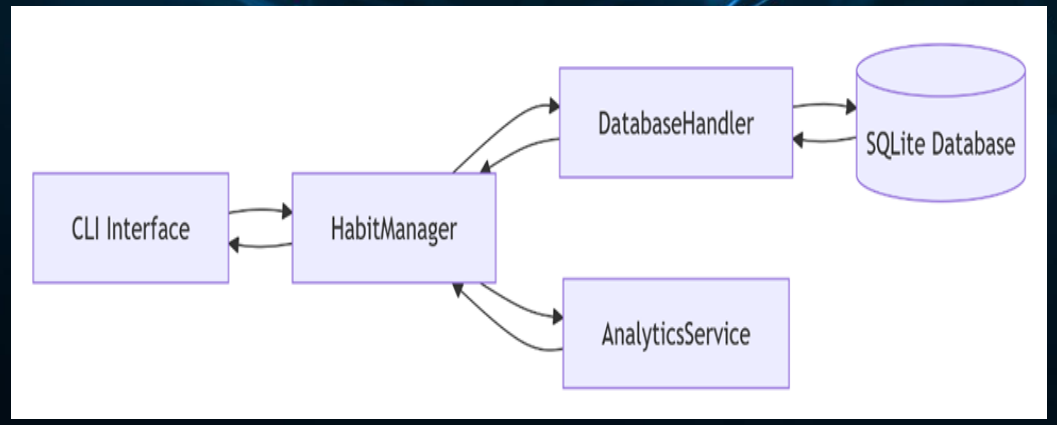
DATABASE DESIGN & PERSISTENCE

```sql
-- Users: Secure authentication
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY,
    username TEXT UNIQUE,
    password_hash TEXT  -- SHA-256 + salt
);

-- Habits: OOP relationships
CREATE TABLE habits (
    habit_id INTEGER PRIMARY KEY,
    user_id INTEGER REFERENCES users,
    periodicity TEXT CHECK(periodicity IN ('daily', 'weekly'))
);

-- Completions: Track habit progress
CREATE TABLE completions (
    completion_id INTEGER PRIMARY KEY,
    habit_id INTEGER REFERENCES habits,
    timestamp DATETIME,
    mood_score INTEGER CHECK(mood_score BETWEEN 1 AND 10)
);
```

DATA FLOW



💾 Robust Data Management:

Normalized table structure:

- **Users (user_id, username, password, email,created_at)**
- habits (habit_id, name, periodicity, created_at, is_active)
- completions (completion_id, habit_id, timestamp, notes, mood_score)

Design Decisions:

- SQL injection, prevention with parameterized queries
- Foreign, Unique, Check keys constraint for data integrity
- Soft deletion (is_active flag) for data preservation
- Password hashing with SHA-256 + salt

USER INTERFACE DESIGN

💻 Interactive CLI Experience

Questionary Integration:

- Interactive menu system
- Input Validation and error handling.
- Rich User experience with emojis and colors

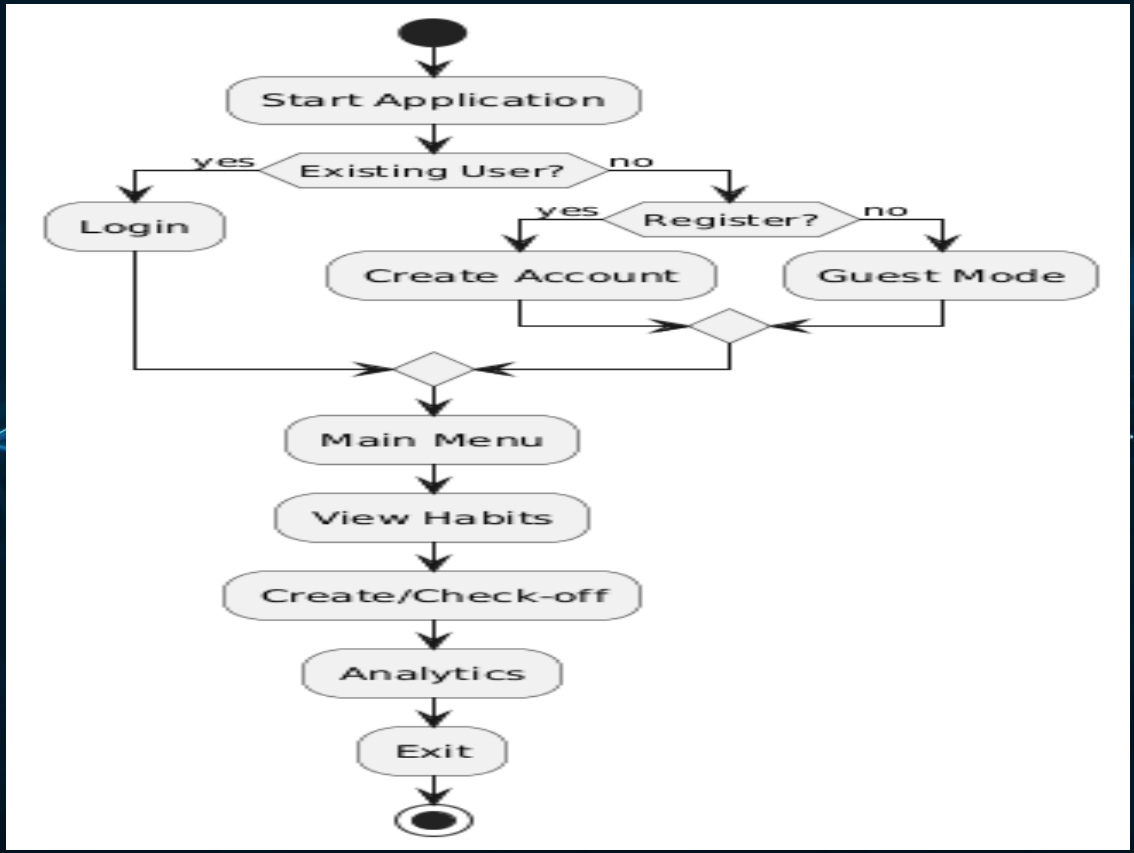
Feature Categories:

- Habit Management (Create, View, Delete)
- Habit Tracking (Check-off with notes/mood)
- Analytics Dashboard (Streaks, Statistics)
- Completion History (Recent activity)

Accessibility:

- Guest mode for quick testing
- Demo account with sample data
- Intuitive navigation patterns

User Flow



APPLICATION USAGE

STARTING THE APPLICATION

Run the Application

```
python main.py
```

```
PS C:\Users\PC\PycharmProjects\habit_tracker> python main.py
✅ Database initialized successfully at: habits.db
✅ Database initialized successfully at: habits.db
🚀 Welcome to the Habit Tracker!
==================================================
? How would you like to proceed? (Use arrow keys)
 » Login
   Register
   Continue as Guest
   Exit
```

Authentication Options

Login: Use existing credentials (demo account: demo/demo123)

Register: Create a new account

Guest Mode: Use the pre-configured demo account

SEED WITH SAMPLE DATA

Initialized the database with sample data

```
python seed_data.py
```

```
PS C:\Users\PC\PycharmProjects\habit_tracker> python seed_data.py
✅ Database initialized successfully at: habits.db
⚠️  Demo user may already exist: UNIQUE constraint failed: users.email
✅ Using existing demo user with ID: 1
⚠️  Habit 'Morning Meditation' may already exist: UNIQUE constraint failed:
habits.user_id, habits.name
⚠️  Habit 'Evening Journal' may already exist: UNIQUE constraint failed:
habits.user_id, habits.name
⚠️  Habit 'Weekly Planning' may already exist: UNIQUE constraint failed:
habits.user_id, habits.name
⚠️  Habit 'Exercise 3x' may already exist: UNIQUE constraint failed:
habits.user_id, habits.name
⚠️  Habit 'Read 10 Pages' may already exist: UNIQUE constraint failed:
habits.user_id, habits.name

📊 Generating sample completion data...

🎉 Sample data seeded successfully!
📊 Created 0 habits with 0 completion records
🔑 You can now login with:
   Username: 'demo'
   Password: 'demo123'
```

Demo Account

For quick testing, use the pre-loaded demo account:

Username: demo

Password: demo123

This account comes with sample habits and 4 weeks of tracking data.

```
🚀 Welcome to the Habit Tracker!
==================================================
? How would you like to proceed? Login
? Enter your username: demo
? Enter your password: ********
✅ Welcome back, demo!
? What would you like to do? (Logged in as: demo) (Use arrow keys)
 » View All Habits
   Create New Habit
   Check Off Habit
   Analytics Dashboard
   View Completions
   Switch User
   Exit
```

TESTING STRATEGY

```
Terminal   Local   Local (2)
tests/test_analytics.py::TestAnalyticsService::test_calculate_current_streak_weekly PASSED [  5%]
tests/test_analytics.py::TestAnalyticsService::test_calculate_longest_streak_daily PASSED [  7%]
tests/test_analytics.py::TestAnalyticsService::test_calculate_longest_streak_weekly PASSED [ 10%]
tests/test_analytics.py::TestAnalyticsService::test_get_habits_by_periodicity PASSED [ 12%]
tests/test_analytics.py::TestAnalyticsService::test_get_overall_longest_streak PASSED [ 15%]
tests/test_analytics.py::TestAnalyticsService::test_get_inactive_habits PASSED [ 17%]
tests/test_analytics.py::TestAnalyticsService::test_get_habits_streak_summary PASSED [ 20%]
tests/test_analytics.py::TestAnalyticsService::test_pure_functions_no_side_effects PASSED [ 22%]
tests/test_db.py::TestDatabaseHandler::test_init_database PASSED [ 25%]
tests/test_db.py::TestDatabaseHandler::test_create_user PASSED [ 27%]
tests/test_db.py::TestDatabaseHandler::test_verify_user_credentials PASSED [ 30%]
tests/test_db.py::TestDatabaseHandler::test_save_habit PASSED [ 32%]
tests/test_db.py::TestDatabaseHandler::test_get_habits_for_user PASSED [ 35%]
tests/test_db.py::TestDatabaseHandler::test_save_completion PASSED [ 37%]
tests/test_db.py::TestDatabaseHandler::test_completion_validation PASSED [ 40%]
tests/test_db.py::TestDatabaseHandler::test_soft_delete_habit PASSED [ 42%]
tests/test_db.py::TestDatabaseHandler::test_user_password_hashing PASSED [ 45%]
tests/test_habit.py::TestHabitClasses::test_daily_habit_creation PASSED [ 47%]
tests/test_habit.py::TestHabitClasses::test_weekly_habit_creation PASSED [ 50%]
tests/test_habit.py::TestHabitClasses::test_habit_activation_deactivation PASSED [ 52%]
tests/test_habit.py::TestHabitClasses::test_daily_habit_is_due_on PASSED [ 55%]
tests/test_habit.py::TestHabitClasses::test_weekly_habit_is_due_on PASSED [ 57%]
tests/test_habit.py::TestHabitClasses::test_habit_check_off PASSED [ 60%]
tests/test_habit.py::TestHabitClasses::test_get_last_completion_date PASSED [ 62%]
tests/test_habit.py::TestHabitClasses::test_habit_factory_daily PASSED [ 65%]
tests/test_habit.py::TestHabitClasses::test_habit_factory_weekly PASSED [ 67%]
tests/test_habit_manager.py::TestHabitManager::test_create_habit PASSED [ 70%]
tests/test_habit_manager.py::TestHabitManager::test_create_habit_validation PASSED [ 72%]
tests/test_habit_manager.py::TestHabitManager::test_get_all_habits PASSED [ 75%]
```

```python
# Example unit test for FP analytics
def test_calculate_current_streak():
    # Arrange
    completions = [
        datetime(2024, 1, 3),
        datetime(2024, 1, 2),
        datetime(2024, 1, 1)  # Streak of 3 days
    ]

    # Act
    streak = calculate_current_streak(completions)

    # Assert
    assert streak == 3

# Example integration test
def test_habit_creation_workflow():
    user = create_test_user()
    habit = habit_manager.create_habit(user.id, "Exercise", "daily")
    assert habit.name == "Exercise"
    assert habit.periodicity == "daily"
```

Test Categories:

- Unit Tests: 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)

Run the comprehensive test suite

```
Run all tests
    python -m pytest tests/ -v
```

TECHNICAL INNOVATIONS & CONCLUSION

Production-Ready Features

Python 3.12+ Modern Features:

- Custom datetime adapters for SQLite
- Modern type hints throughout codebase
- Context managers for resource management

Performance Optimizations:

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

Key Achievements:

- ✅ Clean Architecture with layered design
- ✅ Advanced OOP Patterns implemented
- ✅ Pure Functional Programming for analytics
- ✅ Comprehensive testing (40+ tests)
- ✅ Security & data integrity
- ✅ Windows compatibility

Future Ready: Solid foundation for enhancements and scaling