SOFTWARE ENGINEERING 2 PROJECT SPRING 2025

A Personal Budget Tracker

Table of Contents

1. Introduction. 2
   A) Purpose. 2
   B) Project Scope. 2
   C) Glossary and Abbreviations. 3
   D) List of the System Stakeholders. 3
2. Functional Requirements. 3
   A) Functional Requirements for User. 3
3. Non-Functional Requirements 7
   A) The General Types/Categories 7
   B) Specifications 7
   C) Fit Criteria for Every Non-Functional Requirement 8
   D) How would the above-mentioned Non-Functional Requirements affect the System's overall Architecture? 8
4. Design & Implementation Constraints 8
5. System Evolution 9
   A) Anticipated changes 9
   B) How should any anticipated changes in the future affect the system design? 9
6. What are the requirements discovery approaches that you'll rely on? 9
7. What requirements validation techniques will you employ/use? 9
8. Functional Diagrams. 10
   A) Use-Case Diagram(s). 10
   B) Package Diagram 10
9. Structural & Behavioral Diagrams 10
   A) System Architecture 10

B) Activity Diagrams. 11
C) Class Diagram 1. 14
D) Sequence Diagram(s). 15
E) ERD 18
Package Diagram 19
F) Which strategy (or strategies) did you use to implement the use cases? 19
G) Define a design smell, highlight a design smell in your design, and suggest how it can be avoided. 19

1. Introduction.

A) Purpose.

The Personal Budget Tracker is designed to empower individuals to take control of their personal finances by simplifying the recording, categorization, and visualization of income and expenses. By providing an intuitive interface to log transactions and monitor spending patterns, the system helps users make informed budgeting decisions, avoid overspending, and achieve their financial goals.

B) Project Scope.

The scope of this project covers the following:

- **User Interaction**
  - Secure user registration and login
  - Dashboard displaying current balance, recent transactions, and visual summaries
- **Transaction Management**
  - Creation, editing, and deletion of income and expense entries
  - Assignment of each entry to a predefined category (e.g., Food, Transport, Utilities, Entertainment)
- **Budget Oversight**
  - Definition of monthly budget limits per category
  - Automatic calculation of remaining budget for the current month
- **Reporting & Visualization**

- Tabular and graphical (bar chart, pie chart) views of spending by category
- Monthly summary report showing total income, total expenses, and net balance

C) Glossary and Abbreviations.

- DB: Database
- UI: User Interface
- AOP: Aspect-Oriented Programming
- OCL: Object Constraint Language
- API: Application Programming Interface
- ERD: Entity-Relationship Diagram
- SDD: Software Detailed Design
- SRS: Software Requirements Specification
- UX: User Experience

D) List of the System Stakeholders.

1. Users: The daily users of the application.
2. Guests: Any user visits the website without authentication.

2. Functional Requirements.

A) Functional Requirements for User.

FR-1: User Registration

User Requirements Specification	System Requirements Specification	Priority
The system shall allow a new user to create an account by providing a unique email and password.	1. Email must be unique and valid format. 2. Password must meet minimum strength rules (e.g. 8+ characters, at least one digit).	**Must have**

FR-2: User Login & Authentication

User Requirements Specification	System Requirements Specification	Priority

The system shall authenticate users via email/password and maintain a secure session.	1. Invalid credentials produce an appropriate error message. 2. Successful login redirects to the user's Dashboard.	**Must have**

Update Profile

User Requirements Specification	System Requirements Specification	Priority
The system must allow the user to update his profile information.	1. The system must allow user to enter his first name, last name, username, email and birth date. 2. The system must validate the input. 3. The system must check the username and email against the DB. 4. If the username and email are found, the system will update the user to the DB else show error.	**Must have**

Change password

User Requirements Specification	System Requirements Specification	Priority
The system must allow the user to change his password.	1. The system must allow user to press "change password" button 2. The system must allow user to enter his current password and new password. 3. The system must validate the input.	**Must have**

	4. The system must check the current password against the DB. 5. If the current password is valid, the system will update the password to the DB else show error.	

Delete Profile

User Requirements Specification	System Requirements Specification	Priority
The system must allow the user to delete his account.	1. The system must allow user to press the "Delete Account" button in the profile page UI. 2. The system must ask him if he sure. 3. If the user pressed "Delete" button the system must delete his account else cancel the process.	**Should have**

FR-4: Add Transaction

User Requirements Specification	System Requirements Specification	Priority
The user shall be able to record a new transaction specifying: - **Date (default to today)** - **Amount** - **Type (Income or Expense)** - **Category (select from user-defined list)** - **Optional Note**	1. Amount must be a positive number.	**Must have**

Edit Transaction

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to modify any field of a previously recorded transaction.	1. All validation rules from FR-4 apply. 2. Changes save and immediately reflect in summaries/charts.	**Should have**

Delete Transaction

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to permanently remove a transaction.	1. Deletion updates all dependent totals and visualizations.	**Must have**

FR-7: Manage Categories

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to view, add, rename, or delete transaction categories.	1. Deleting a category prompts to reassign or purge existing transactions.	**Must have**

FR-8: Define Monthly Budgets

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to set a spending limit per category for each calendar month.	1. Budget values must be non-negative. 2. Users can update budget at any time; changes apply to the current month only.	**Should have**

FR-9: Dashboard Summary

User Requirements Specification	System Requirements Specification	Priority
Upon login, users see a Dashboard with: - **Current month's total income, total expenses, and net balance** - **List of the five most recent transactions** - **Expenses By Category** - **Monthly Expenses** - **Income vs Expenses**		**Should have**

FR-10: Monthly Summary Report

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to view a detailed, tabular report for any selected month showing:		**Must have**

FR-11: Expense Charts

User Requirements Specification	System Requirements Specification	Priority
Users shall be able to view interactive charts for any selected month:		**Should have**

FR-12: Validation & Error Handling

User Requirements Specification	System Requirements Specification	Priority
All user inputs shall be validated on both client and server; errors are displayed with clear, actionable messages.		**Must have**

3. Non-Functional Requirements

A) The General Types/Categories

- Performance requirements.
- Security requirements.
- Usability & Humanity requirements.
- Maintainability & Support requirements.

B) Specifications

Requirement Specification	Category/Type	Fit Criteria
The system must perform fast because of use of Java.	Performance.	The response time should be less than 1 second.

Requirement Specification	Category/Type	Fit Criteria
The system can be maintained easily because of the use of object-oriented programming and AOP.	Maintainability.	Codebase should be modular and well-documented to facilitate maintenance tasks.

Requirement Specification	Category/Type	Fit Criteria
The system will be secure because of the use of a secure database.	Security.	Regular security audits should not reveal any vulnerabilities in the database.

Requirement Specification	Category/Type	Fit Criteria
The system will be easy to use because of the use of a user-friendly interface.	Usability.	Users should be able to perform common tasks without extensive training or assistance.

C) Fit Criteria for Every Non-Functional Requirement

- Maintainability: Regular code reviews should indicate that maintenance tasks are completed efficiently.
- Security: Security audits should report no major vulnerabilities, and encryption mechanisms should be verified by penetration testing.
- Usability: Usability testing should confirm that users can complete tasks efficiently without significant training or errors.
- Performance: Performance testing should demonstrate that the system can handle expected loads without degradation in response times or reliability.

D) How would the above-mentioned Non-Functional Requirements affect the System's overall Architecture?

- The use of object-oriented programming for maintainability may influence the system's architecture by promoting modular design and encapsulation of components.
- Ensuring security through encryption and access control would require implementing robust authentication mechanism, potentially affecting the system's architecture to incorporate secure data storage and access layers.
- Usability considerations may influence the design of the user interface, affecting architectural decisions related to layout, navigation, and interaction patterns.

4. Design & Implementation Constraints.

1. Technology Stack Limitations
- Using microservices led to difficulties in implementing the architecture.
2. Architectural trade-offs
- Architectural trade-offs may be necessary to balance constraints and system requirements while maintaining performance, reliability, and usability.

5. System Evolution.

A) Anticipated changes.

- Hardware Evolution: Advancements in hardware technology may lead to changes in computing power, storage capacity, and networking capabilities.
- Changing User Needs: User requirements and preferences may evolve over time due to shifts in technology trends, industry standards, or user demographics.
- Scaling Requirements: As user bases grow or business needs change, the system may need to scale to accommodate increased demand for performance, reliability, and scalability.

B) How should any anticipated changes in the future affect the system design?

- Continuous Improvement: Adopt agile development practices, such as iterative development, continuous integration, and automated testing, to facilitate rapid adaptation to changing requirements and feedback from users.
- Modularity and Flexibility: Design the system with modularity and flexibility to accommodate changes in hardware, technology, and user needs. Use design patterns and architectural principles (e.g., Microservices) to decouple modules and facilitate future changes.

6. What are the requirements discovery approaches that you'll rely on?

1. Observation:
- Observing users in their natural environment to understand their workflows, tasks, and challenges.
2. Prototyping and Mockups:
- Creating prototypes or mockups of the system interface or key features to gather feedback and validate requirements before finalizing the design.
3. Use Cases and Scenarios:
- Developing use cases, user stories, and scenarios to capture functional requirements from the perspective of end-users and stakeholders, focusing on specific tasks or interactions.

7. What requirements validation techniques will you employ/use?

1. User Acceptance Testing (UAT):
- Involving end-users in the testing process to validate whether the system meets their requirements and fulfills their usability and functional needs.
2. Requirement Prioritization:

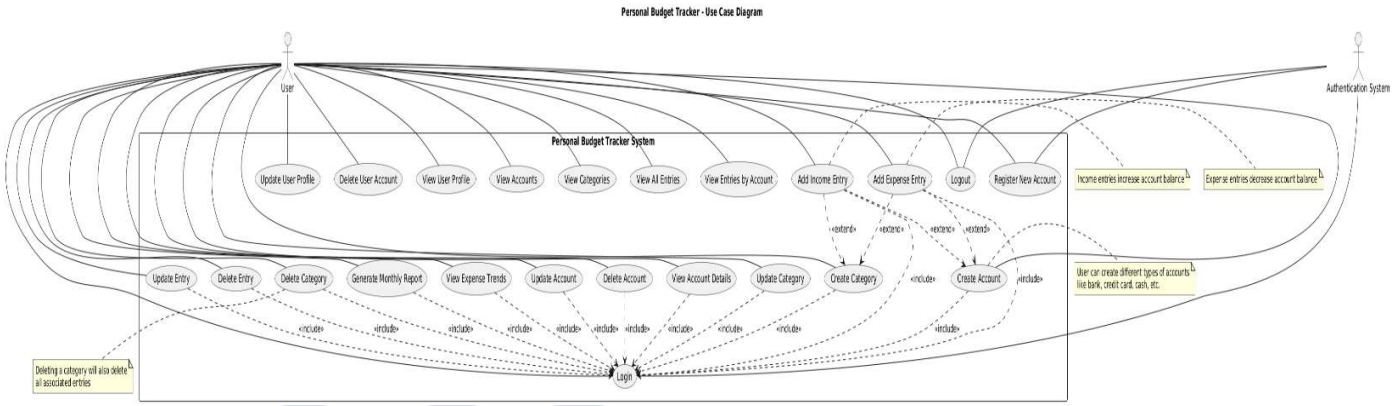
- Prioritizing requirements based on their importance, value, and feasibility to focus on implementing high-priority features first and ensure that essential functionality is delivered within project constraints.

3. Validation Criteria:

- Defining clear and measurable validation criteria for each requirement, specifying the conditions under which a requirement is considered satisfied.

8. Functional Diagrams.

A) Use-Case Diagram(s).



B) Package Diagram

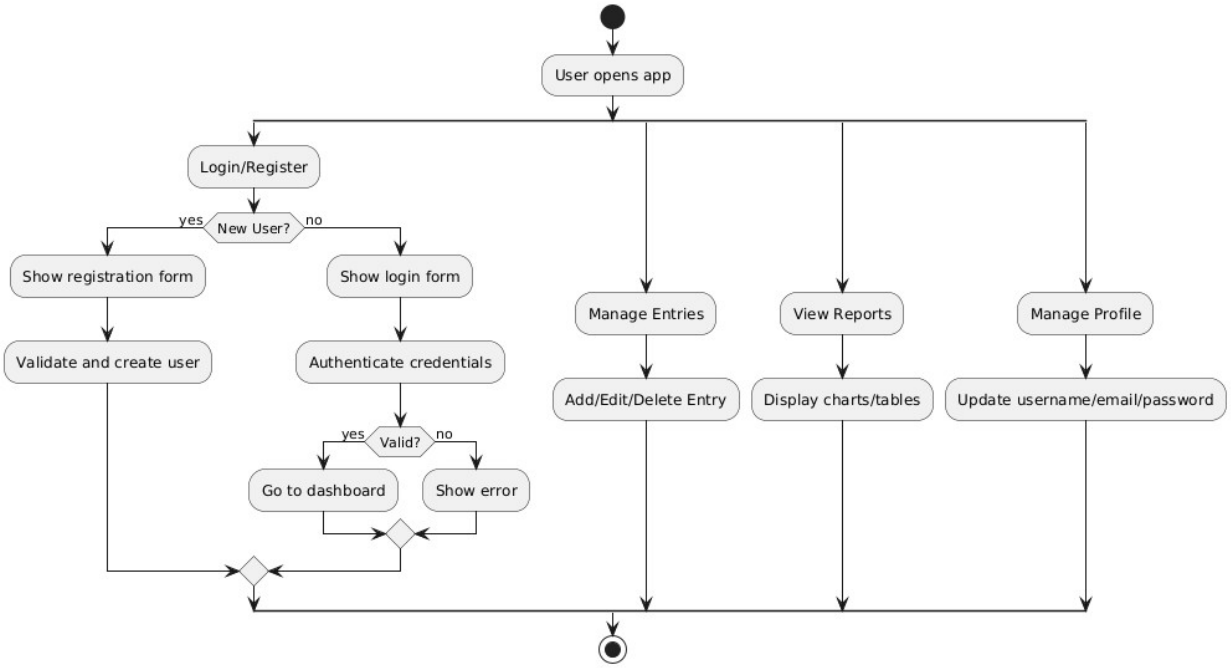
9. Structural & Behavioral Diagrams.

A) System Architecture.

In our project we used Model-View-Controller pattern and we selected this pattern because of the flexibility, maintainability, Reusability, Testability and Scalability.

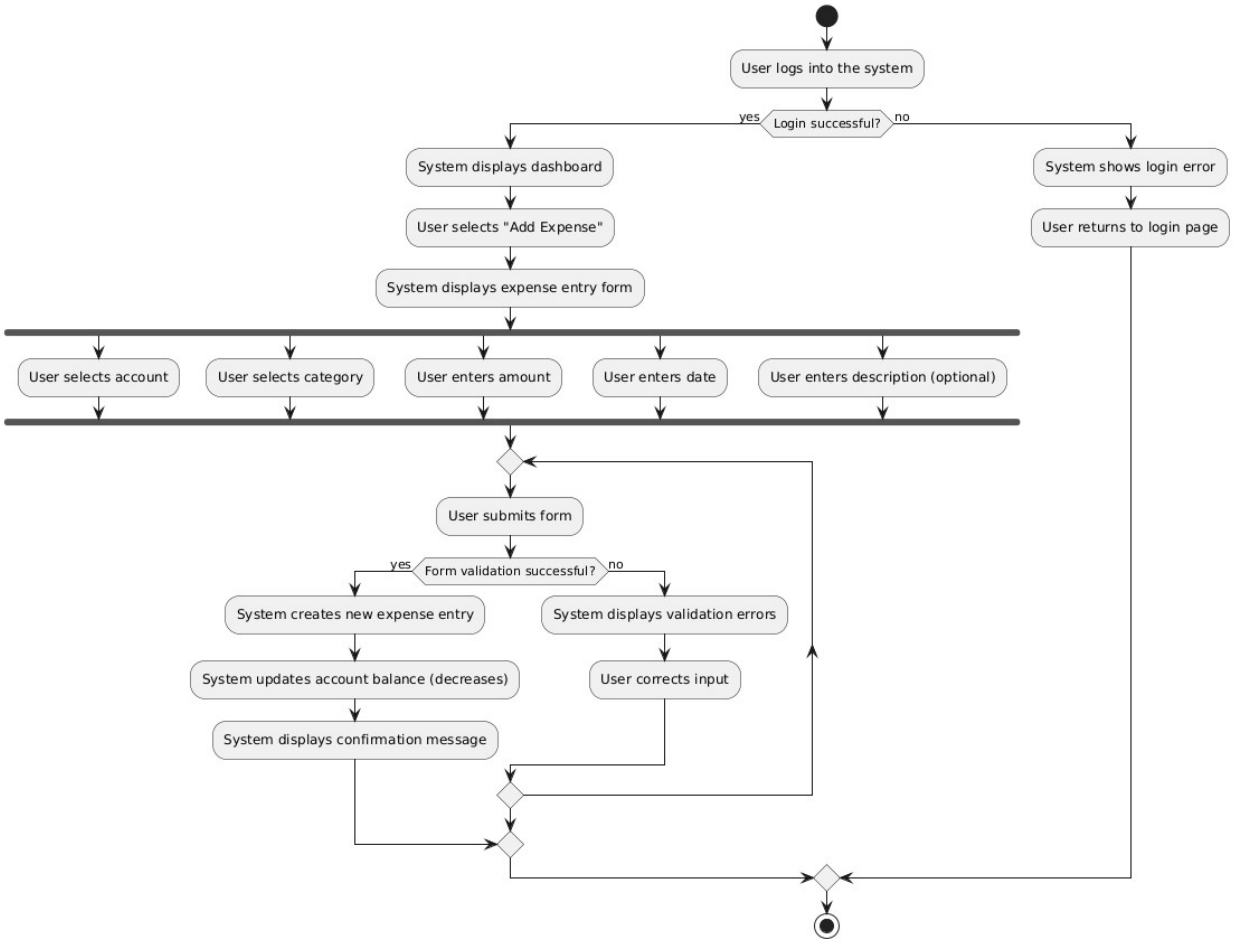
كلية الحاسبات و الذكاء الصناعي

B) Activity Diagrams.

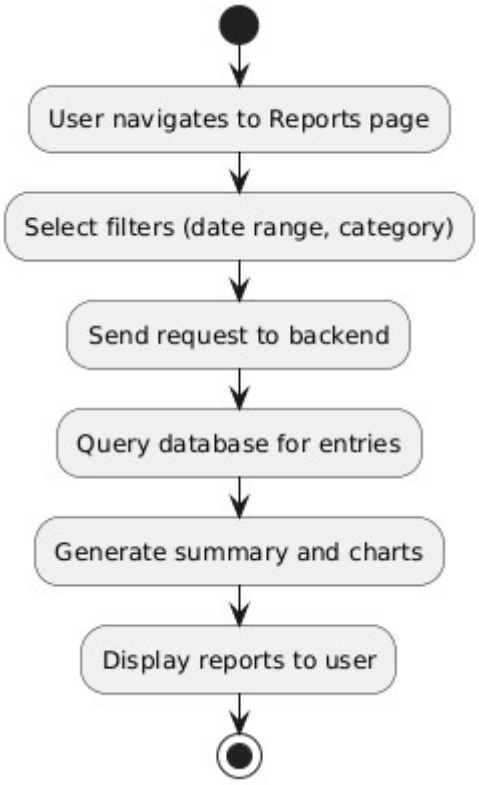


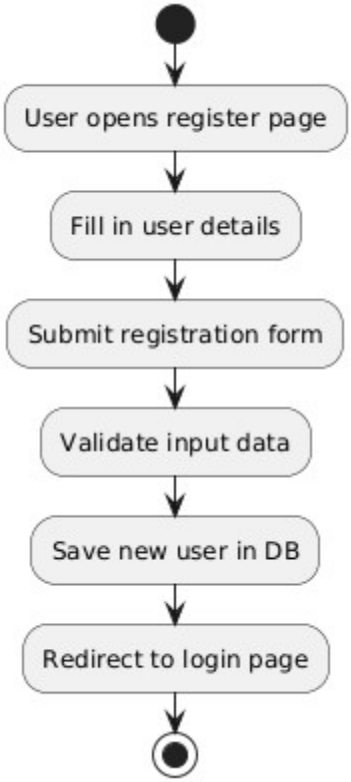
كلية الحاسبات و الذكاء الصناعي

Personal Budget Tracker - Activity Diagram: Adding an Expense

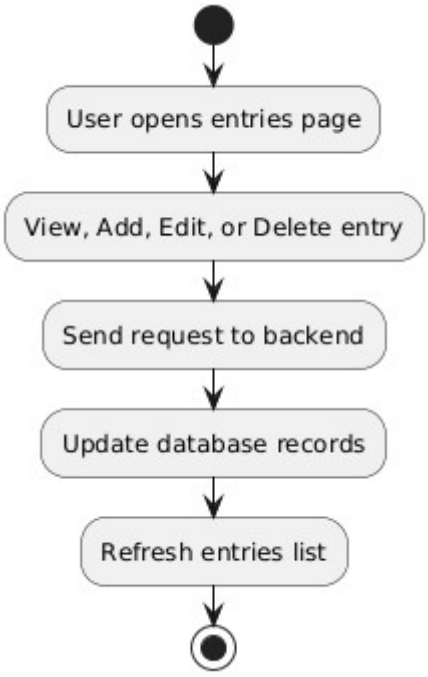


كلية الحاسبات و الذكاء الصناعي

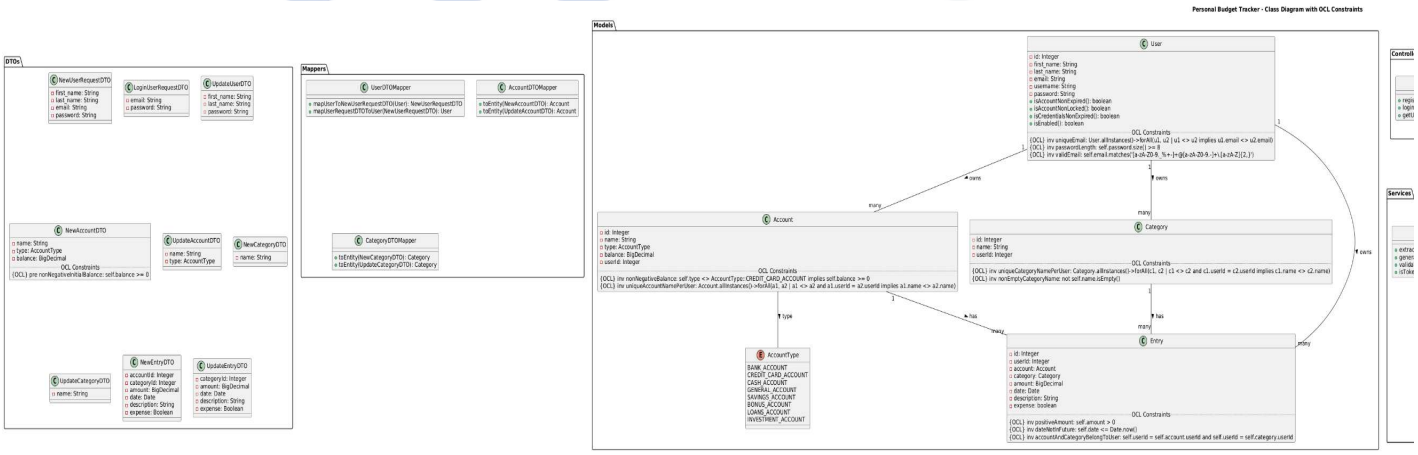




كلية الحاسبات و الذكاء

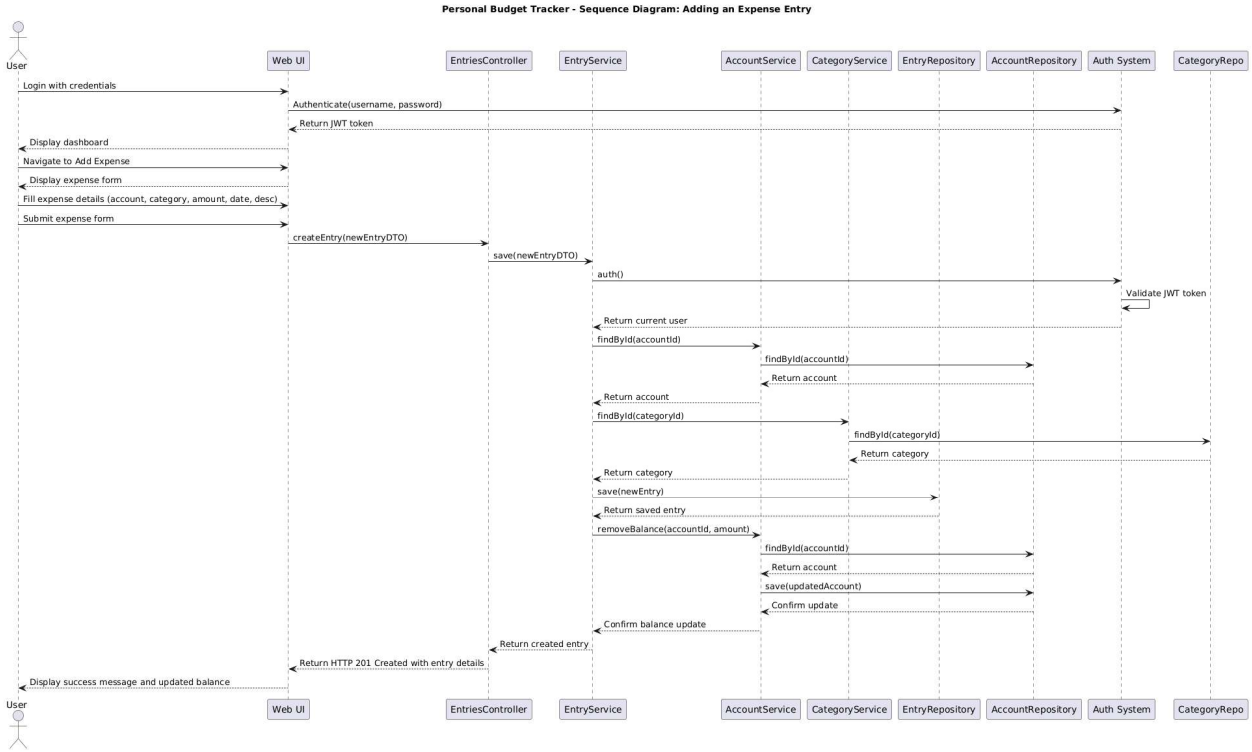


C) Class Diagram.

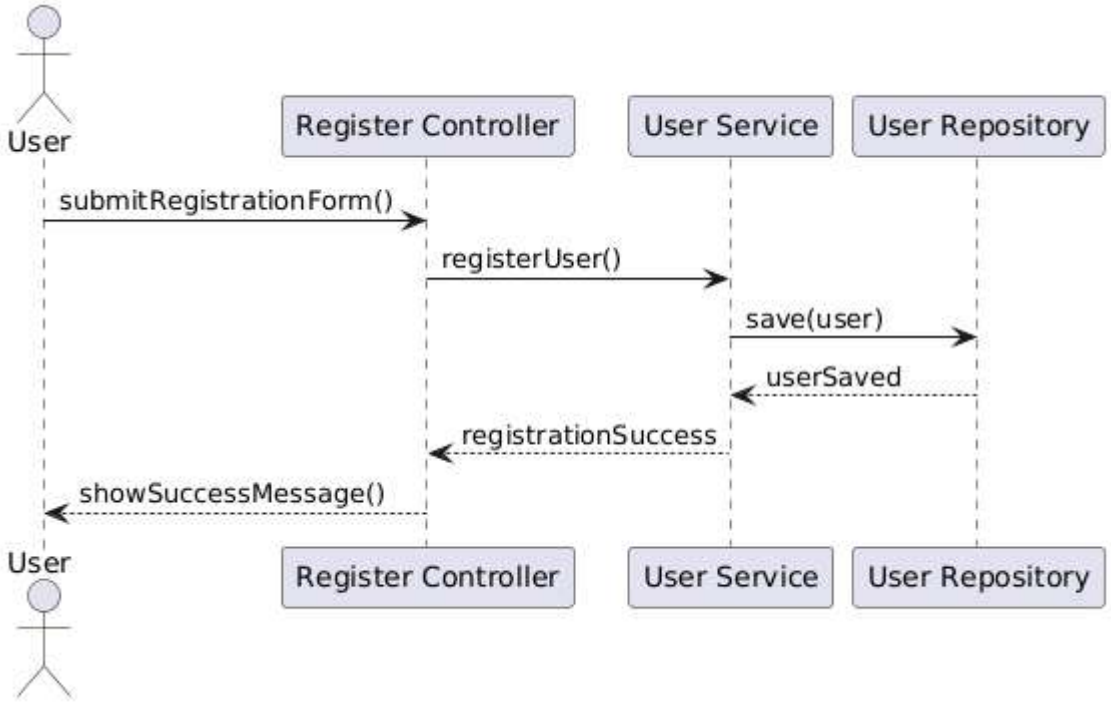


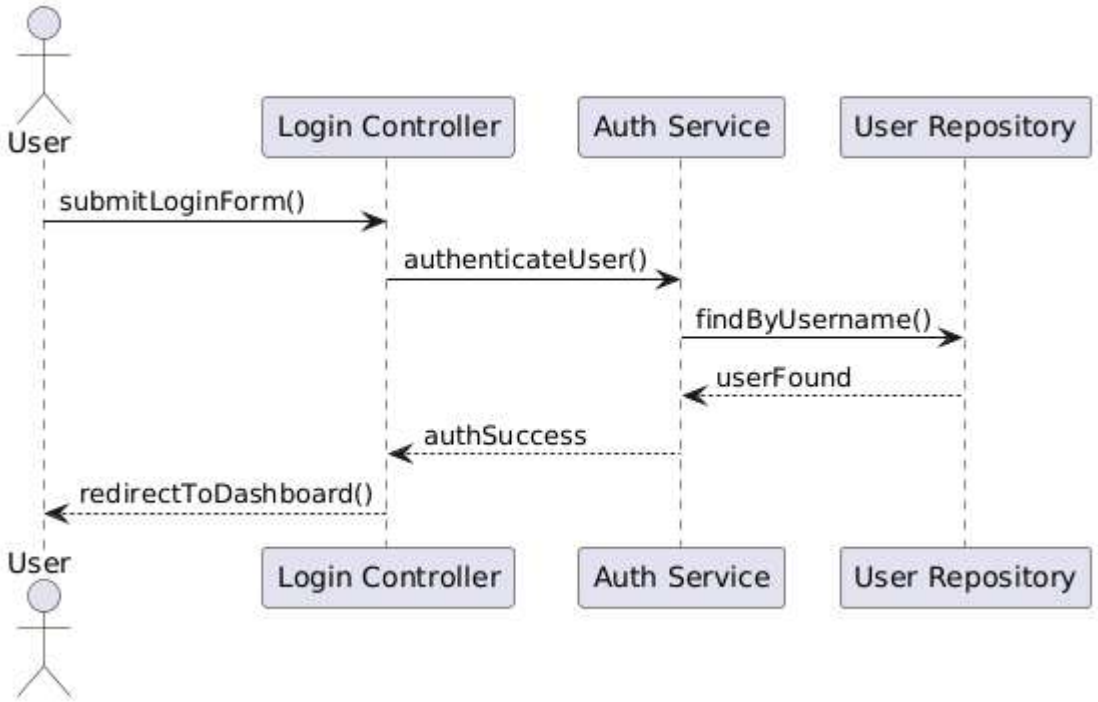
كلية الحاسبات و الذكاء الصناعي

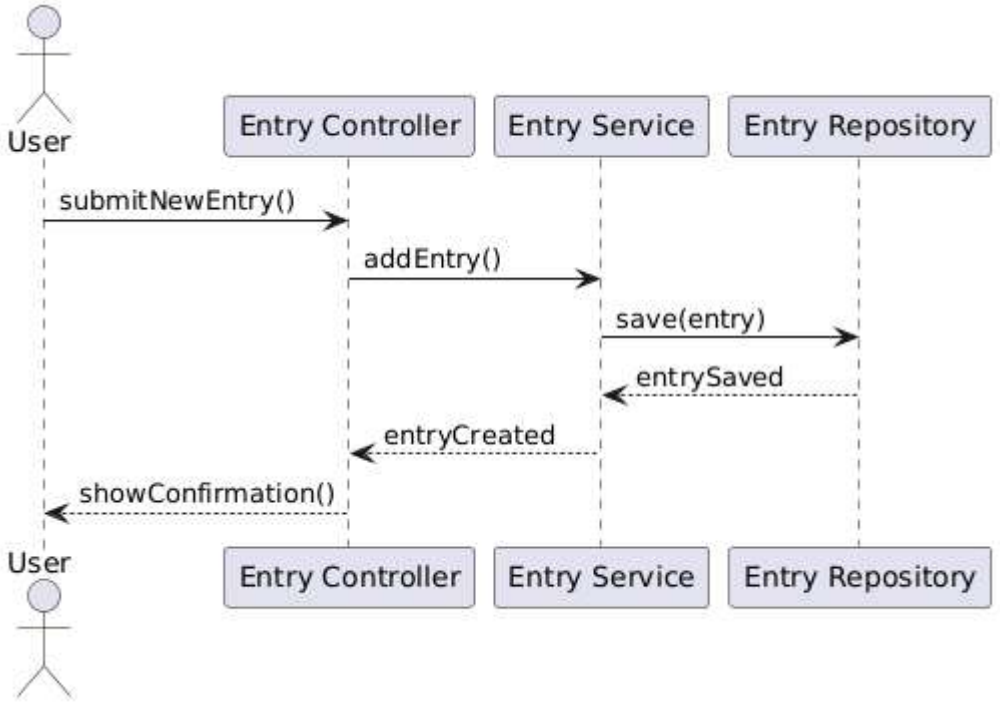
D) Sequence Diagram(s).

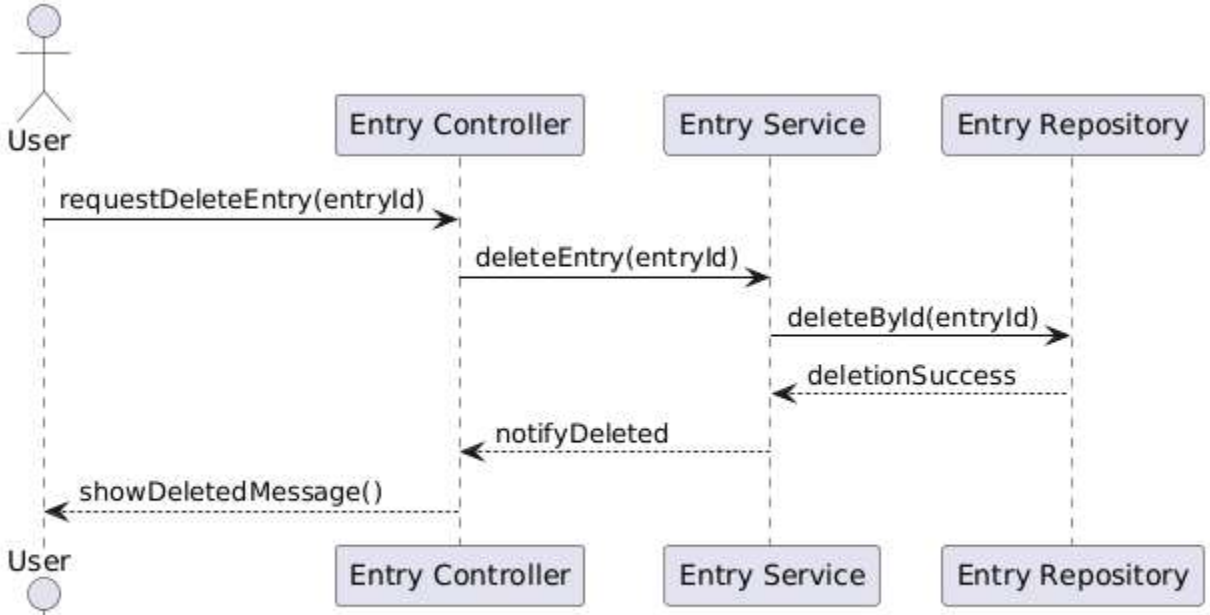


كلية الحاسبات و الذكاء الصناعي

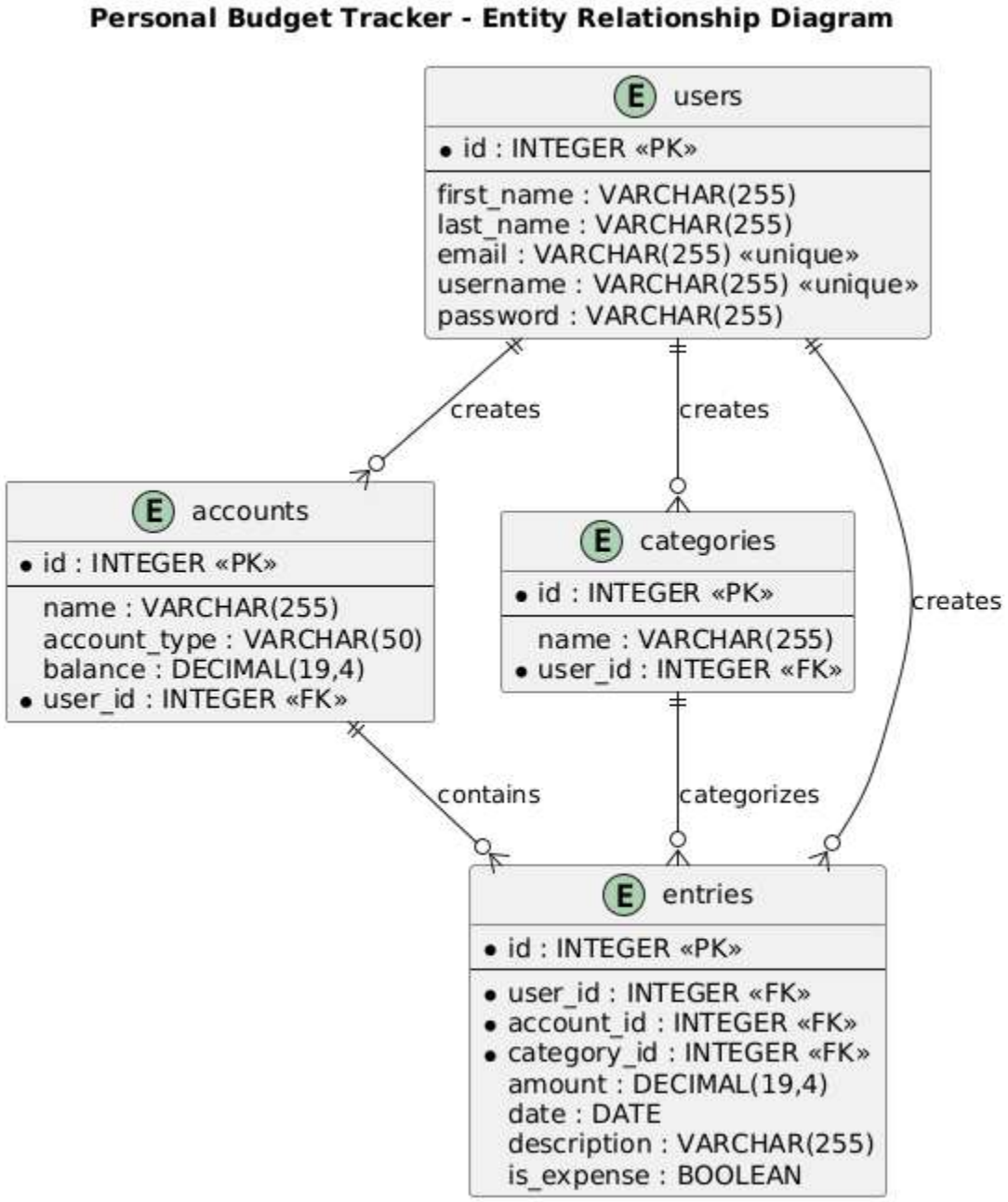

User
Register Controller
User Service
User Repository
submitRegistrationForm()
registerUser()
save(user)
userSaved
registrationSuccess
showSuccessMessage()
User
Register Controller
User Service
User Repository


User
Login Controller
Auth Service
User Repository
submitLoginForm()
authenticateUser()
findByUsername()
userFound
authSuccess
redirectToDashboard()
User
Login Controller
Auth Service
User Repository


User
Entry Controller
Entry Service
Entry Repository
submitNewEntry()
addEntry()
save(entry)
entrySaved
entryCreated
showConfirmation()
User
Entry Controller
Entry Service
Entry Repository

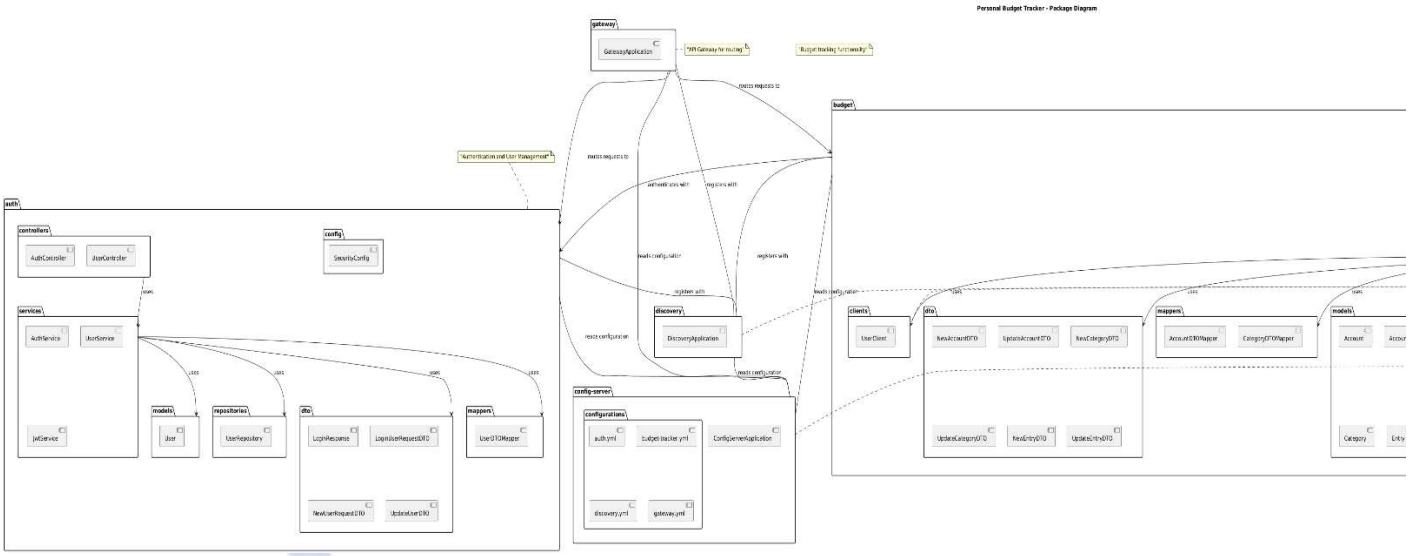

User
Entry Controller
Entry Service
Entry Repository
requestDeleteEntry(entryId)
deleteEntry(entryId)
deleteById(entryId)
deletionSuccess
notifyDeleted
showDeletedMessage()
User
Entry Controller
Entry Service
Entry Repository

E) ERD



كلية الحاسبات و الذكاء الصناعي

Package Diagram



F) Which strategy (or strategies) did you use to implement the use cases?

We used Actor Class strategy because it's advantage in modularity and the clear responsibility of each actor but we also found some disadvantages in managing interactions between multiple actor classes may require additional coordination and communication, which can introduce complexity.

G) Define a design smell, highlight a design smell in your design, and suggest how it can be avoided.

There was a potential design smell is the violation of the Open/Closed Principle (OCP) in voting system and we avoided it by applying strategy pattern, There was another violation of Single Responsibility Principle (SRP) in Admin and User classes (God Object) and we fixed that by breakdown this classes into multiple classes.

Thanks For Reading ❤️

كلية الحاسبات و الذكاء الصناعي