





PRESENTATION



BUDGET APP IMPLEMENTATION

This is the implementation of the for the budget app
based sufficient design pattern and applying SOLID
principles



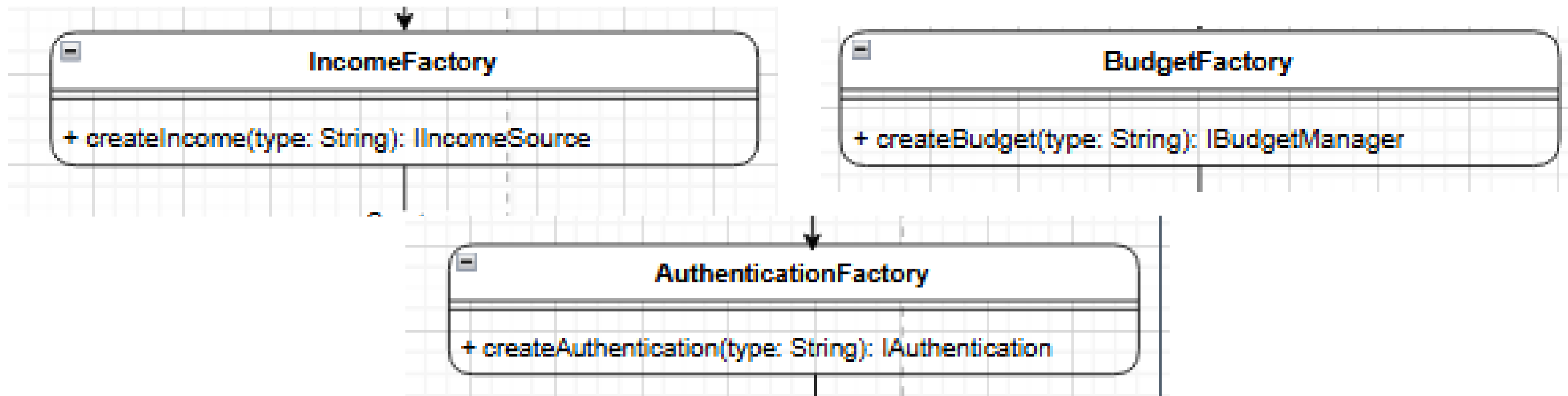
DESIGN

- **Factory Pattern**
- **Strategy pattern**
- **Template Pattern**

FACTORY PATTERN

Classes: AuthenticationFactory, IncomeFactory, BudgetFactory, GoalFactory

Explanation: Used to create instances of authentication methods, income sources, budget managers, and goals without specifying their exact classes. This allows for flexible object creation (e.g., createAuthentication(), createIncome()).

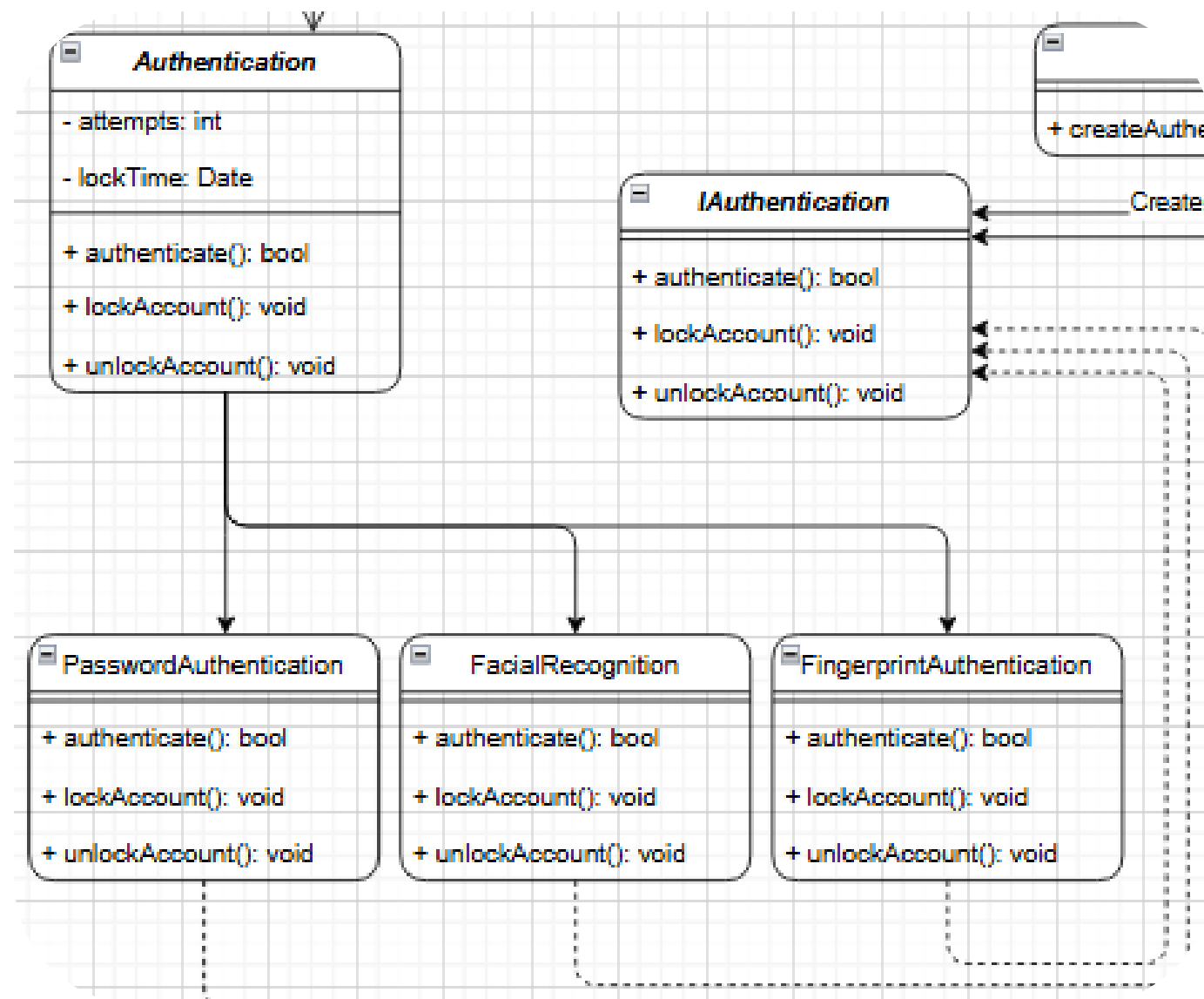


STRATEGY PATTERN

Classes: FacialRecognition, FingerprintAuthentication, PasswordAuthentication

Explanation:

An interface (or abstract class) defines a common method that all algorithms must implement. Concrete classes implement this interface, each providing a different strategy. A context class uses the interface to call the selected strategy dynamically.

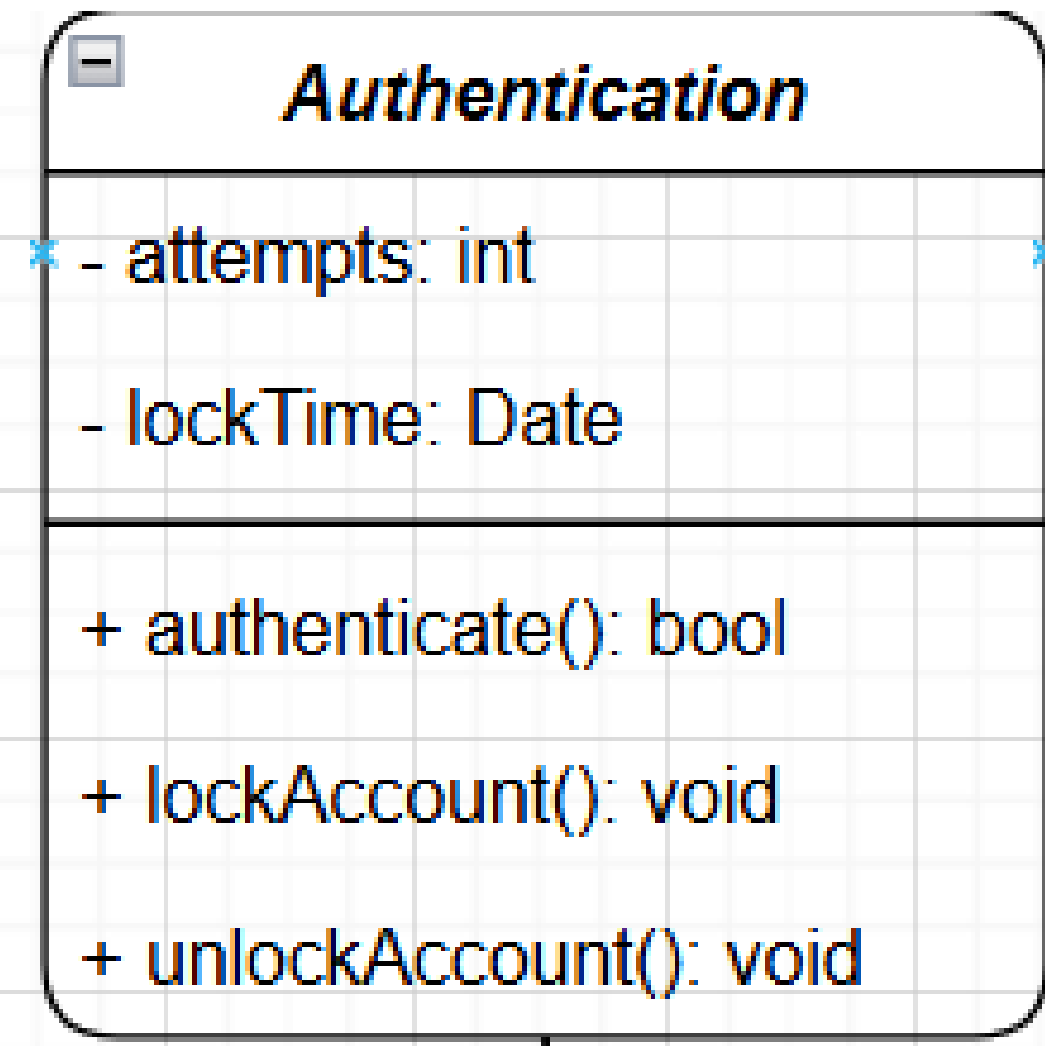
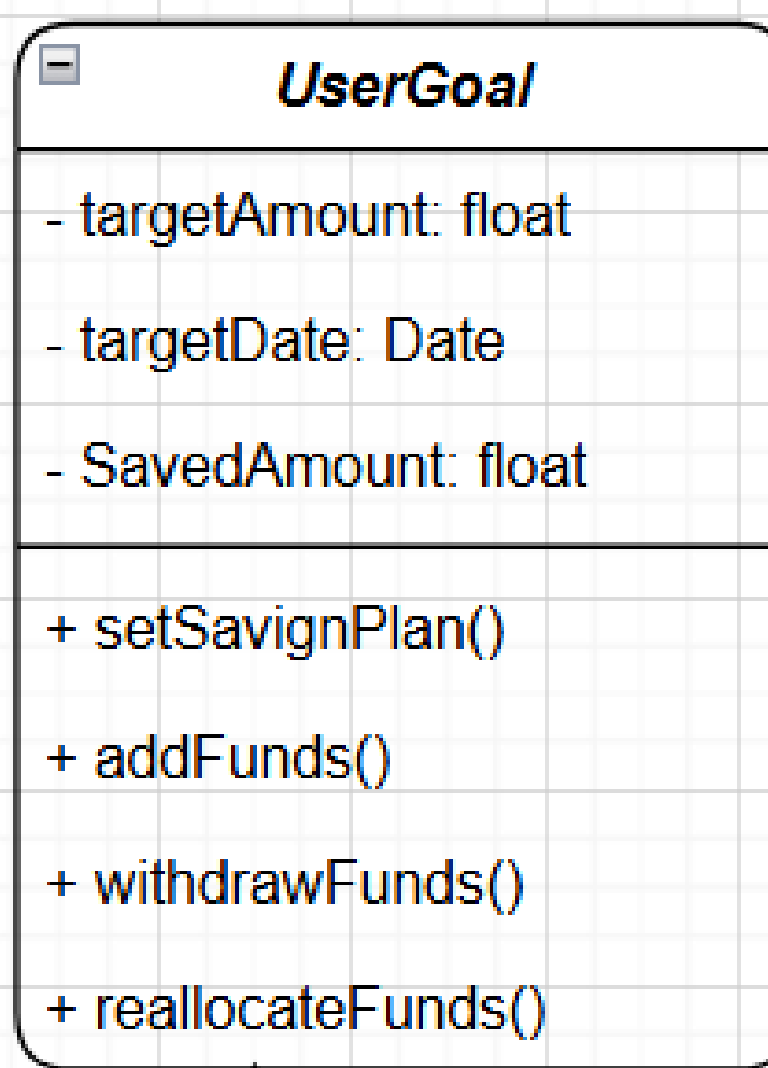
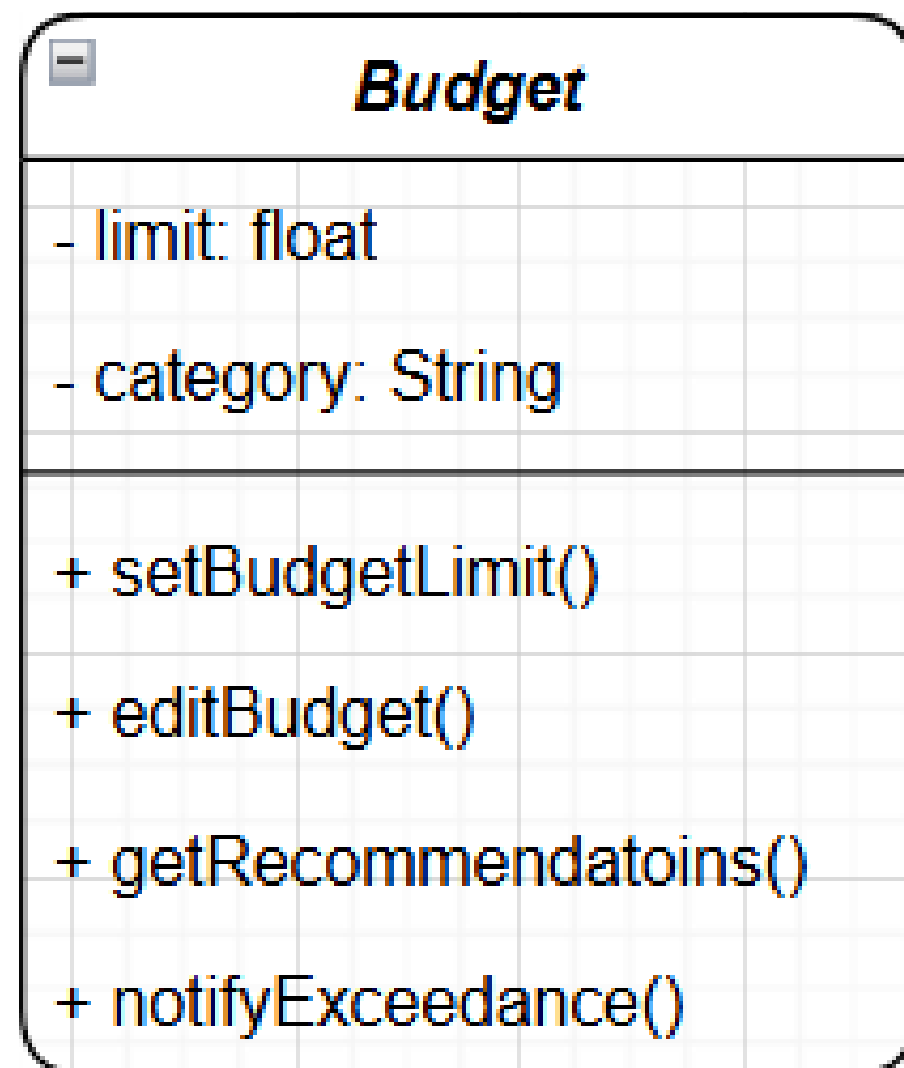


TEMPLATE PATTERN

Classes: Authentication, Income, Budget, UserGoal

Explanation:

The basic idea: The abstract class sets the main steps (template) for the algorithm, saying “do this, then do that, then do this.” However, the small details like “exactly how to do this” are left for the subclasses to complete. This helps you reuse the same code in multiple places (code reuse) without needing to change the basic steps.



SOLID PRINCIPLES

LISSKOV SUBSTITUTION PRINCIPLE (LSP)
OPEN/CLOSED PRINCIPLE (OCP)
SINGLE RESPONSIBILITY PRINCIPLE (SRP)

[UML Link](#)

SINGLE RESPONSIBILITY PRINCIPLE (SRP)

Explanation: A class should have only one reason to change, meaning it should have only one responsibility.

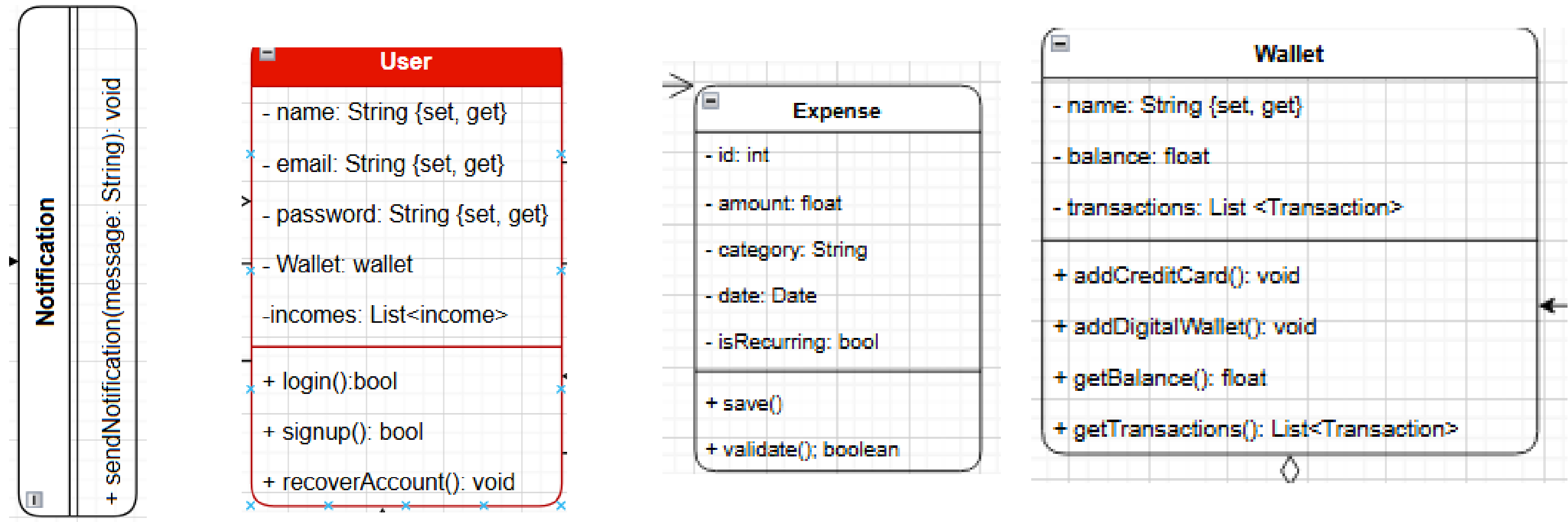
Examples:

User: Focuses solely on user account management (e.g., login(), signup()).

Notification: Handles only sending notifications (sendNotification()).

Wallet : manage external wallets

Expense manage expenses

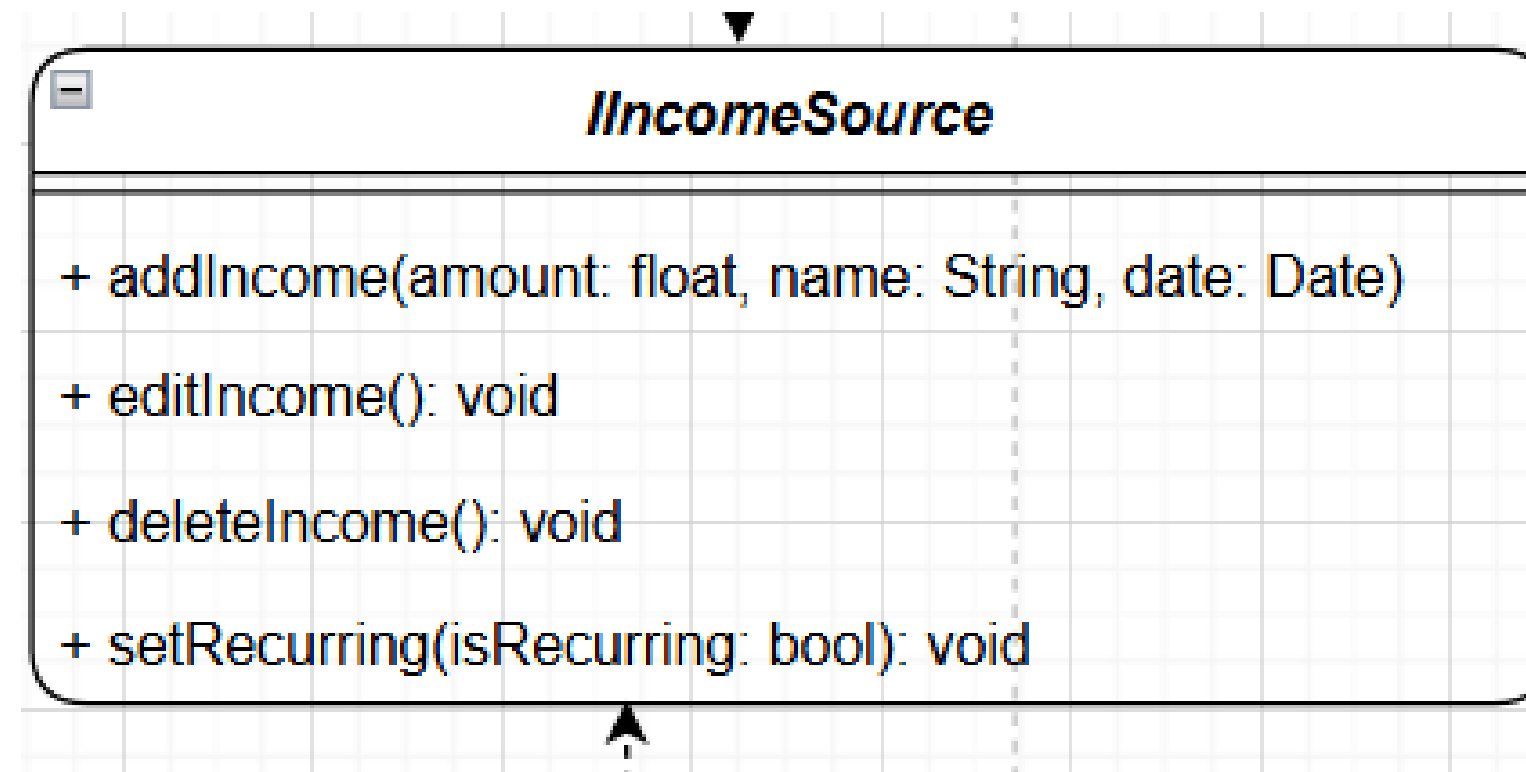
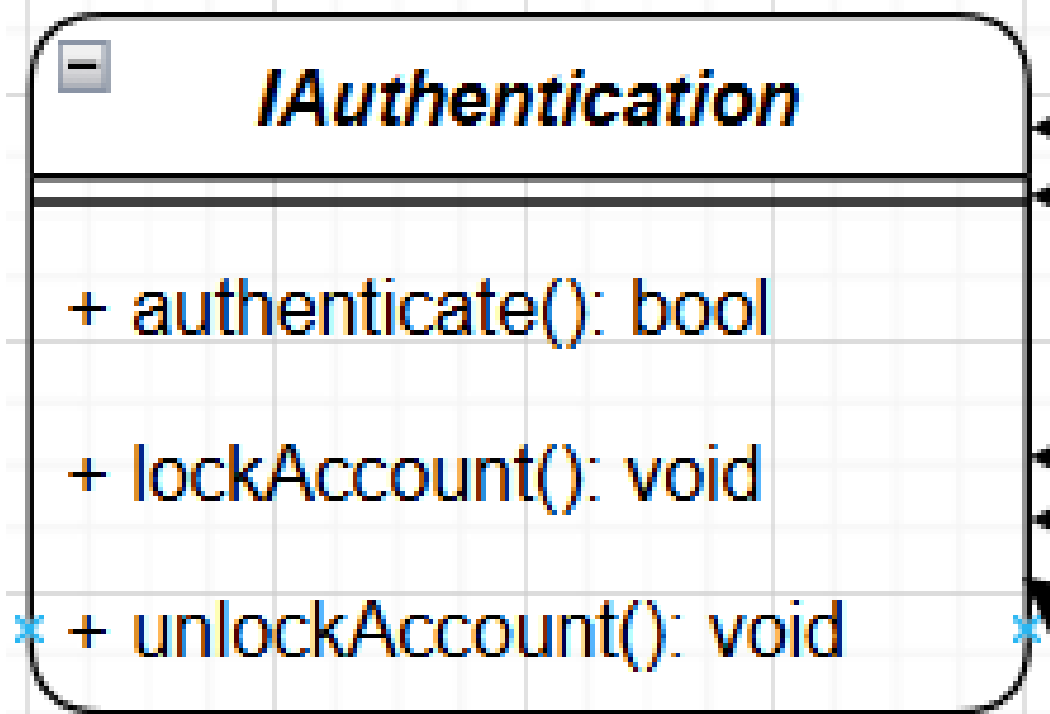


(OPEN/CLOSED PRINCIPLE (OCP)

- **Explanation:** Classes should be open for extension but closed for modification.
- Examples:

IAuthentication interface with implementations (FacialRecognition, FingerprintAuthentication, PasswordAuthentication) allows adding new authentication types without changing the interface.

IncomeSource with **ExternalIncome** enables new income types to be added via extension.



LISKOV SUBSTITUTION PRINCIPLE (LSP)

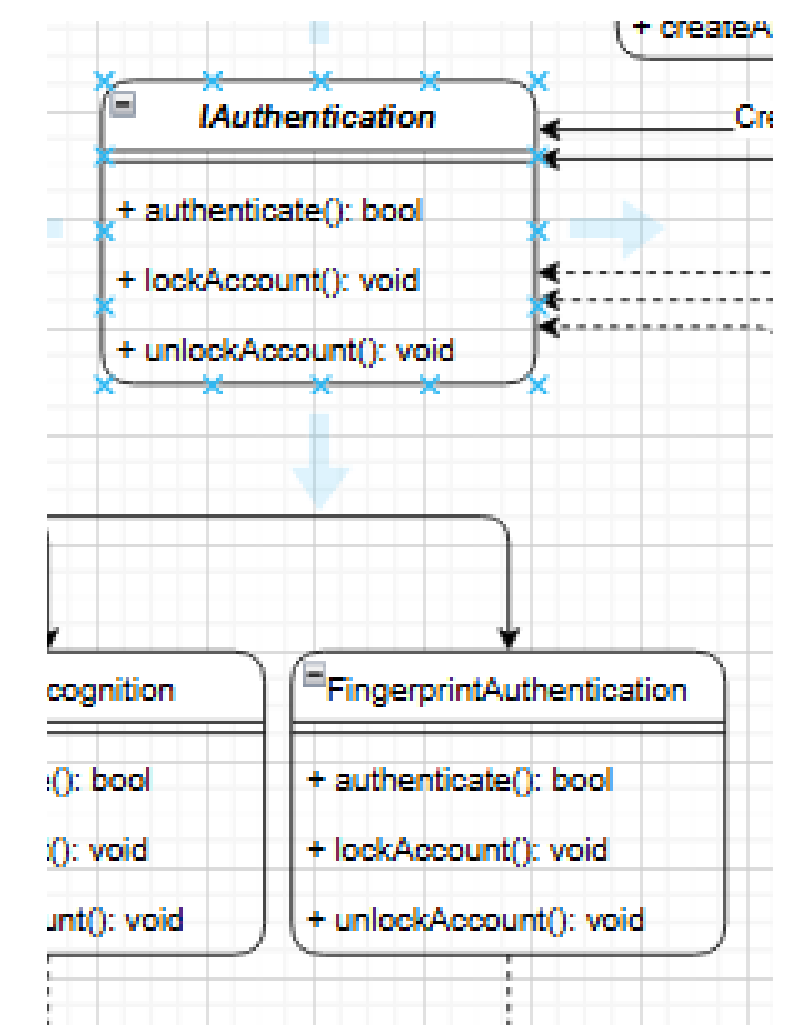
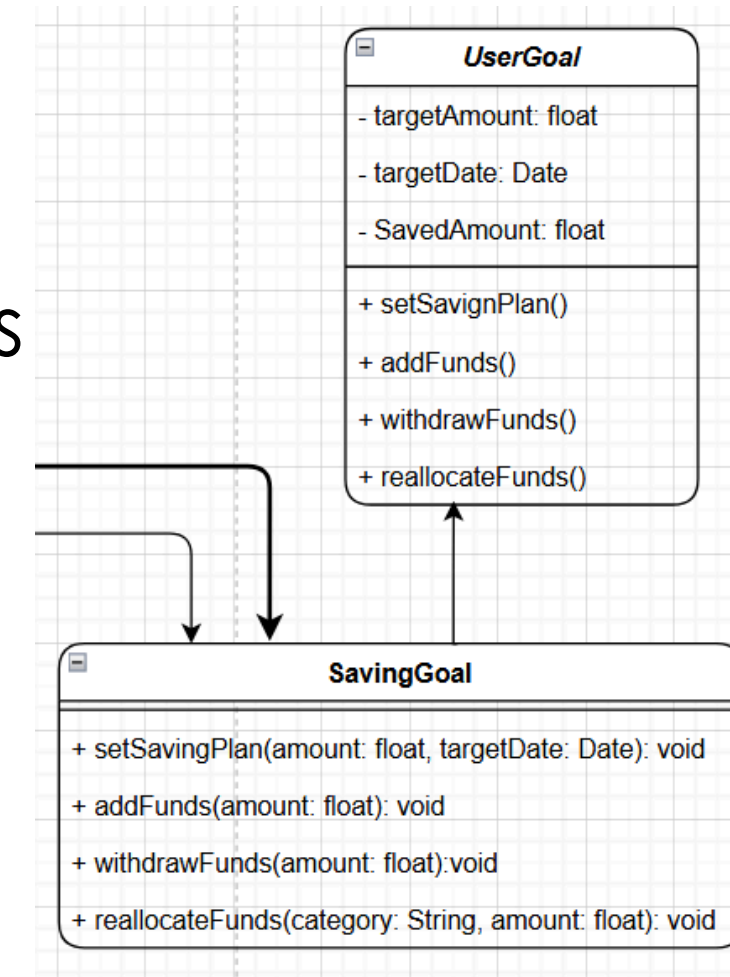
Explanation:

Subtypes must be substitutable for their base types without altering the program's correctness.

Examples:

SavingGoal can replace **UserGoal** (abstract) in any context, ensuring **setSavingPlan()** works consistently.

ExternalIncome can substitute for **Income** or **IncomeSource**, maintaining the expected behavior of income operations.



ARCHITECTURE DIAGRAM - C4 MODEL

SYSTEM CONTEXT DIAGRAM

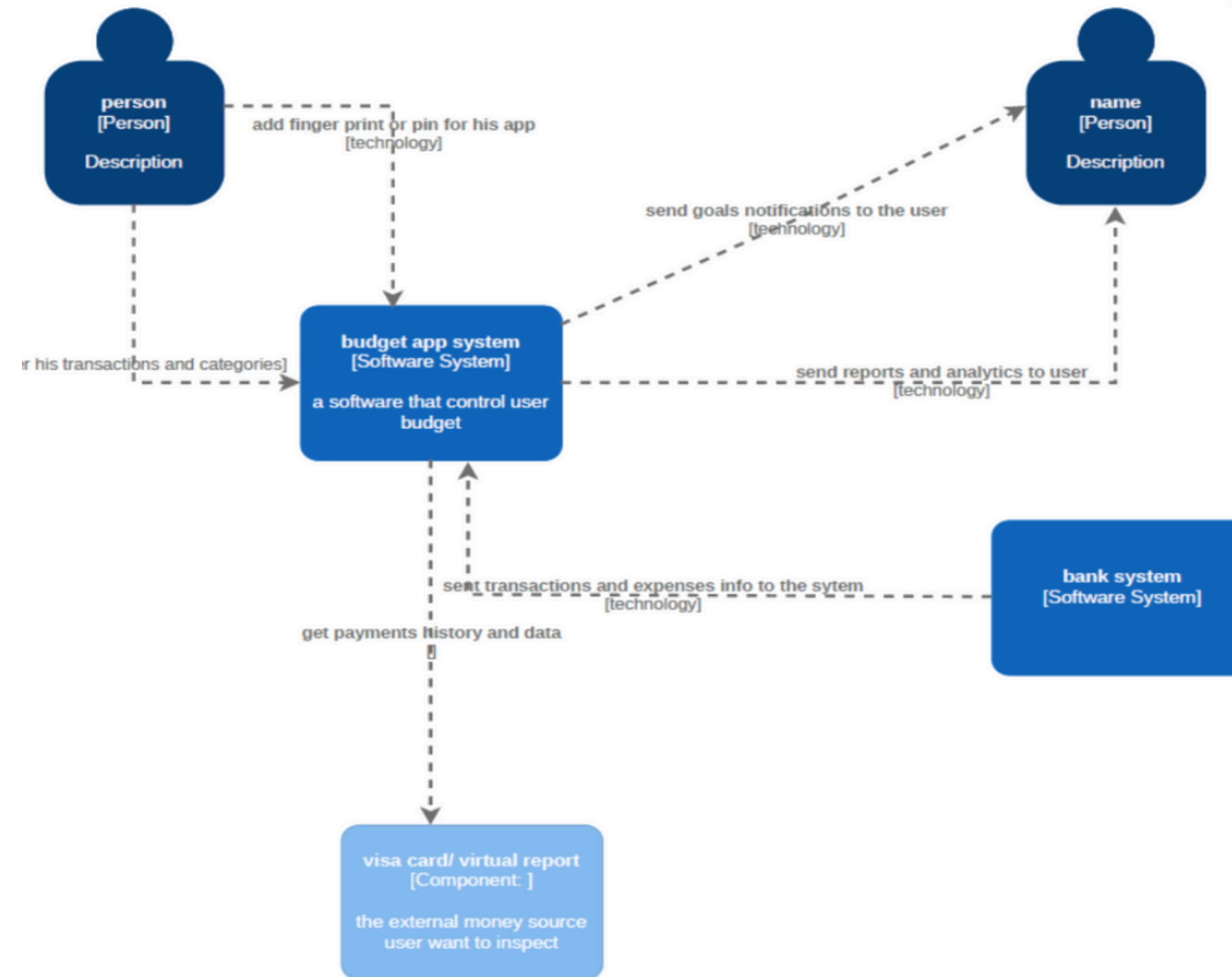
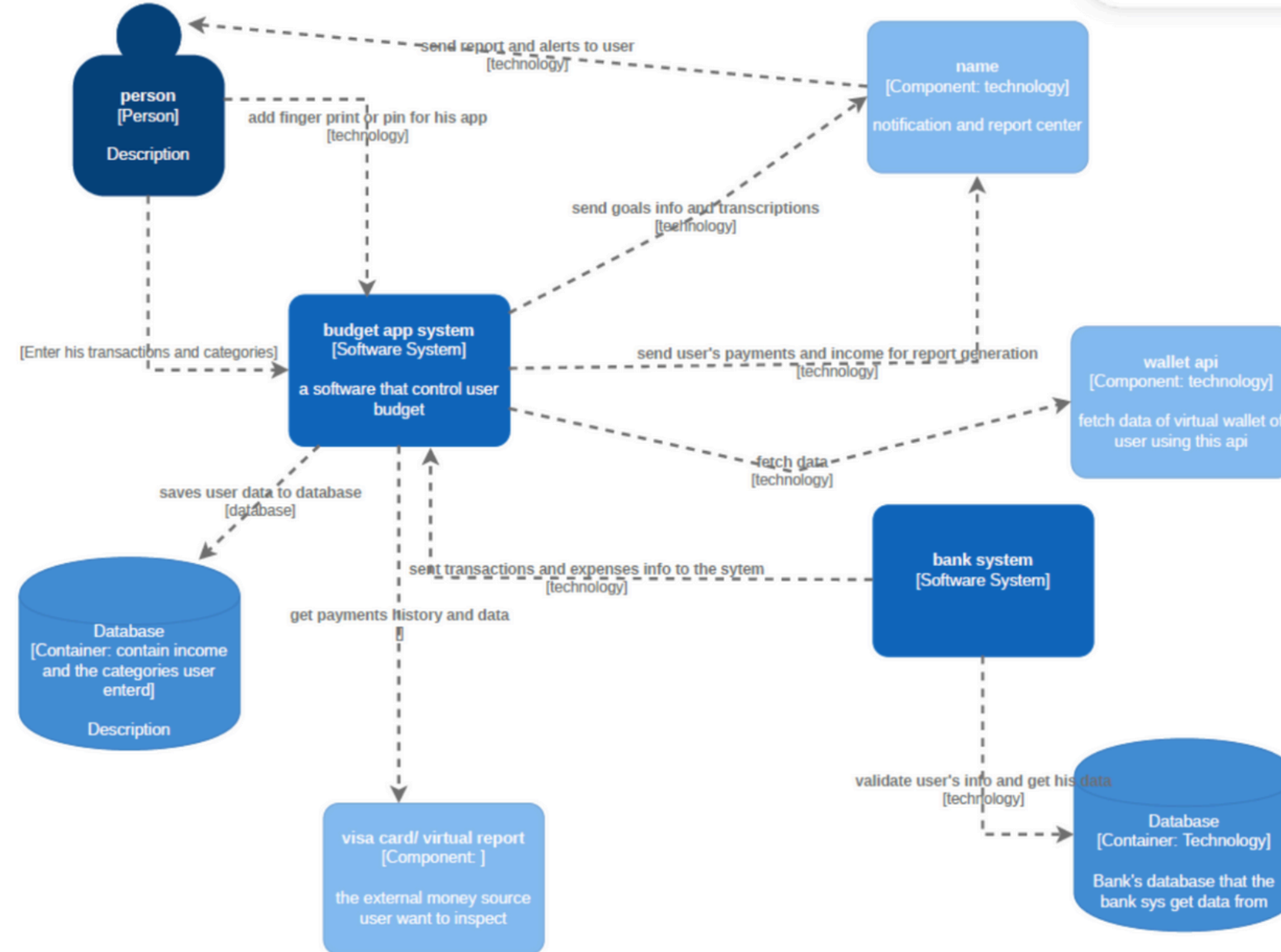


diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with

ARCHITECTURE DIAGRAM - C4 MODEL

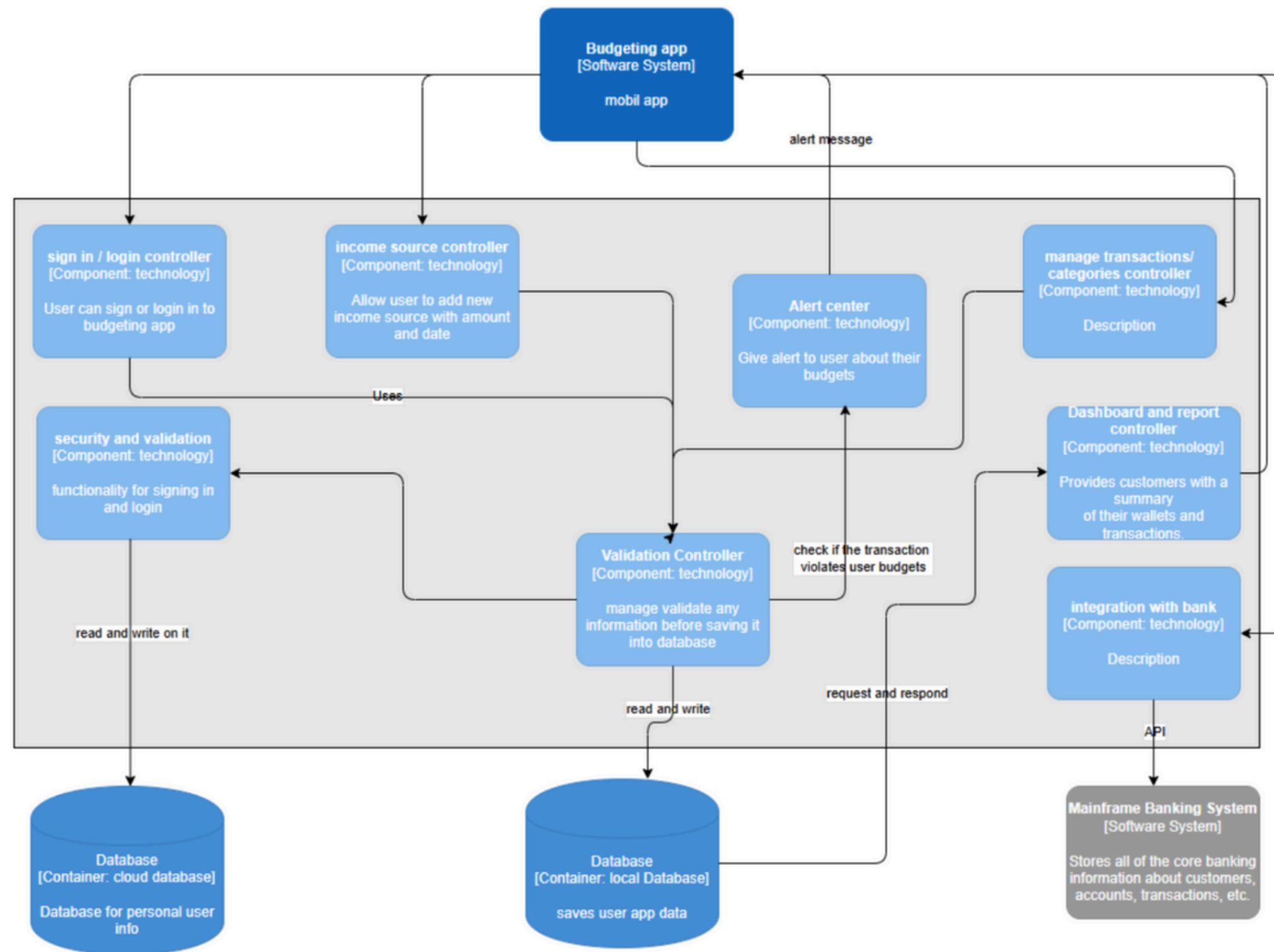
CONTAINER DIAGRAM



The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it

ARCHITECTURE DIAGRAM - C4 MODEL

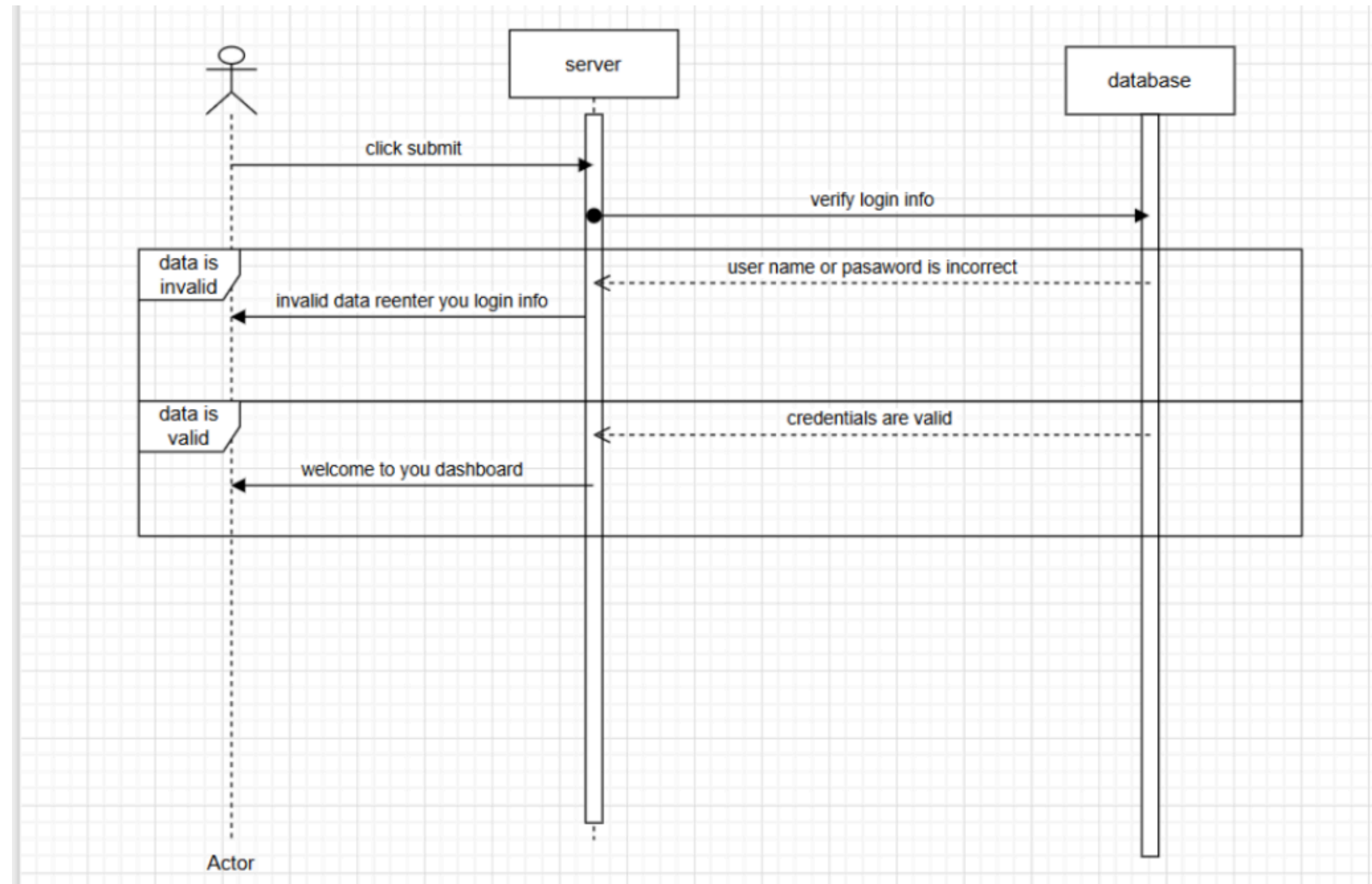
COMPONENT DIAGRAM



the Component diagram shows how a container is made up of a number of “components”, what each of those components are, their responsibilities and the technology/implementation details

SEQUENCE DIGRAM

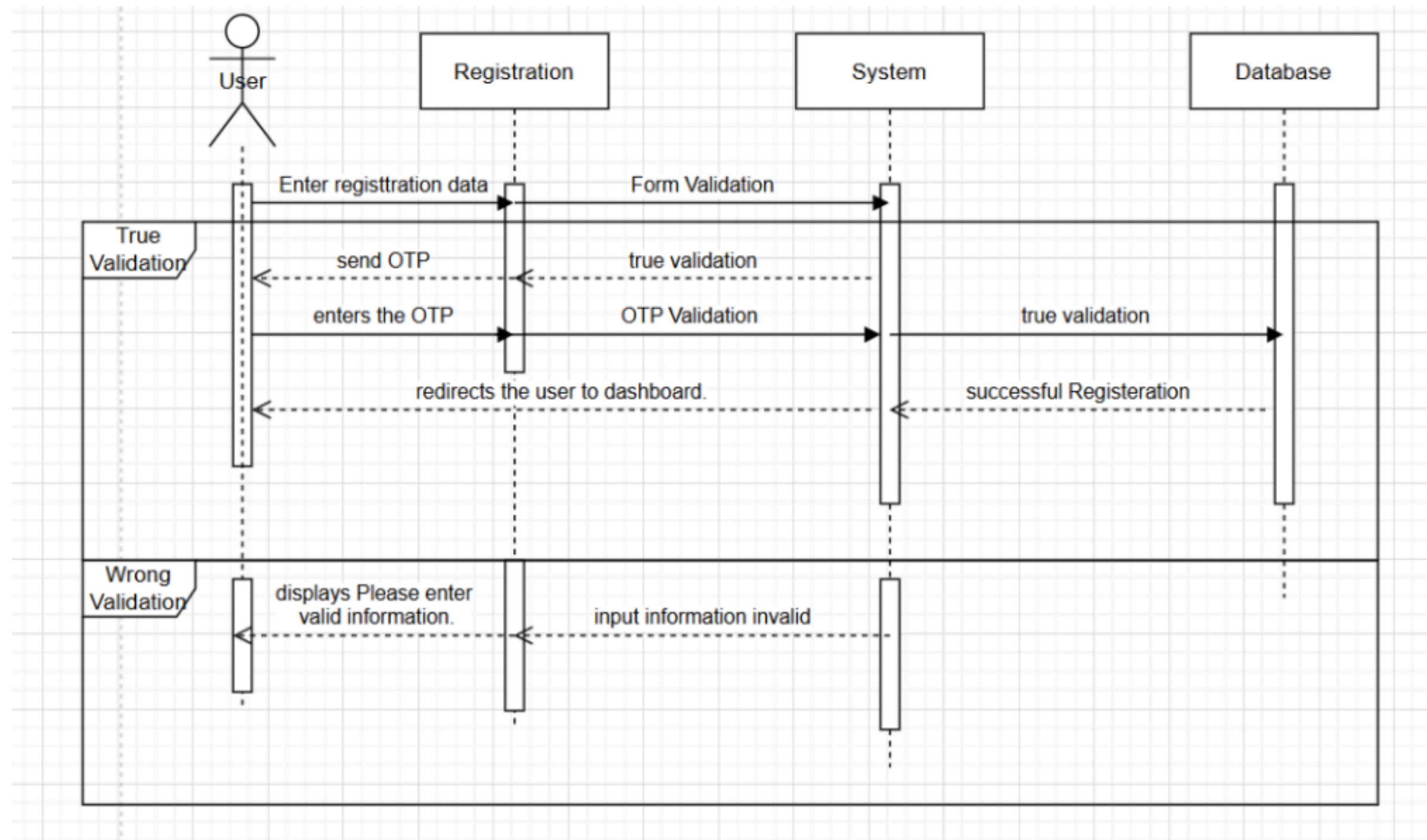
USER STORY 1: LOGIN PROCESS



login process where the user submits credentials, the server verifies them with the database, and responds based on whether the login data is valid or invalid.

SEQUENCE DIGRAM

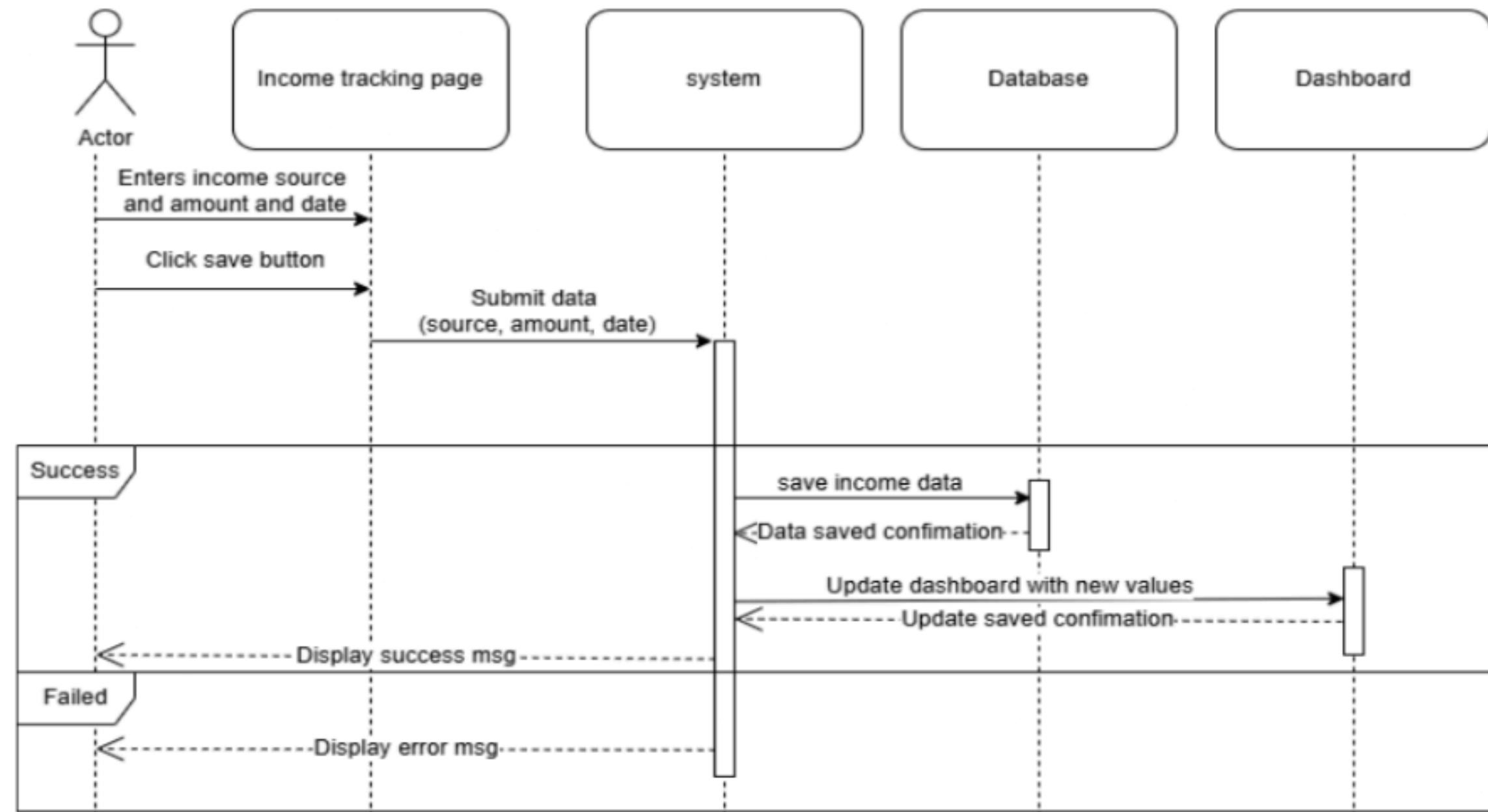
USER STORY 2: SIGNUP PROCESS



the user registration process, where form data is validated and an OTP is used for verification, leading to either successful registration or an error message for invalid input.

SEQUENCE DIGRAM

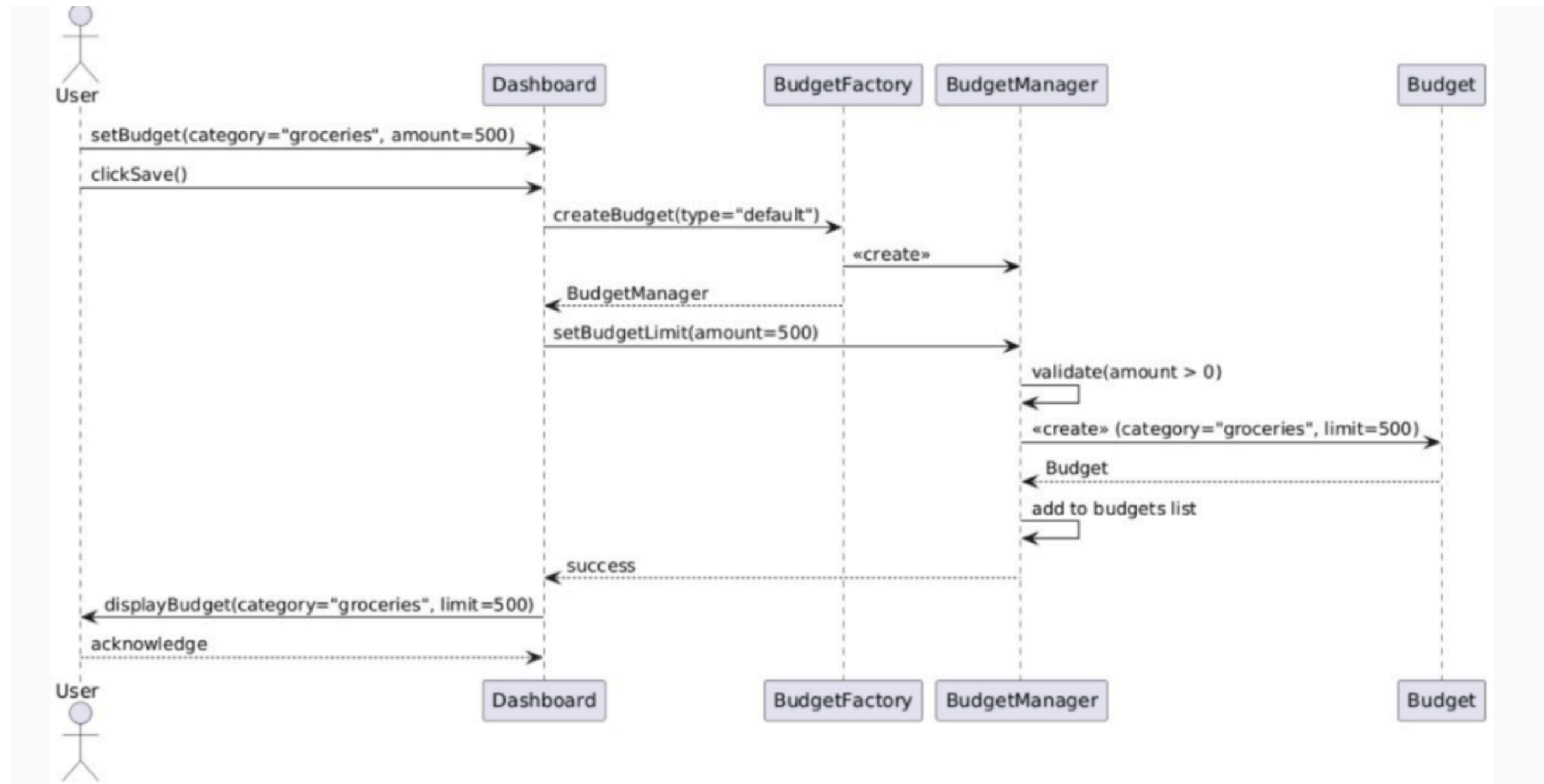
USER STORY 3: ADD INCOME SOURCE AND DISPLAY IT INTO DASHBOARD



the process of adding an income source, saving it to the database, and displaying it on the dashboard, including success and error handling.

SEQUENCE DIGRAM

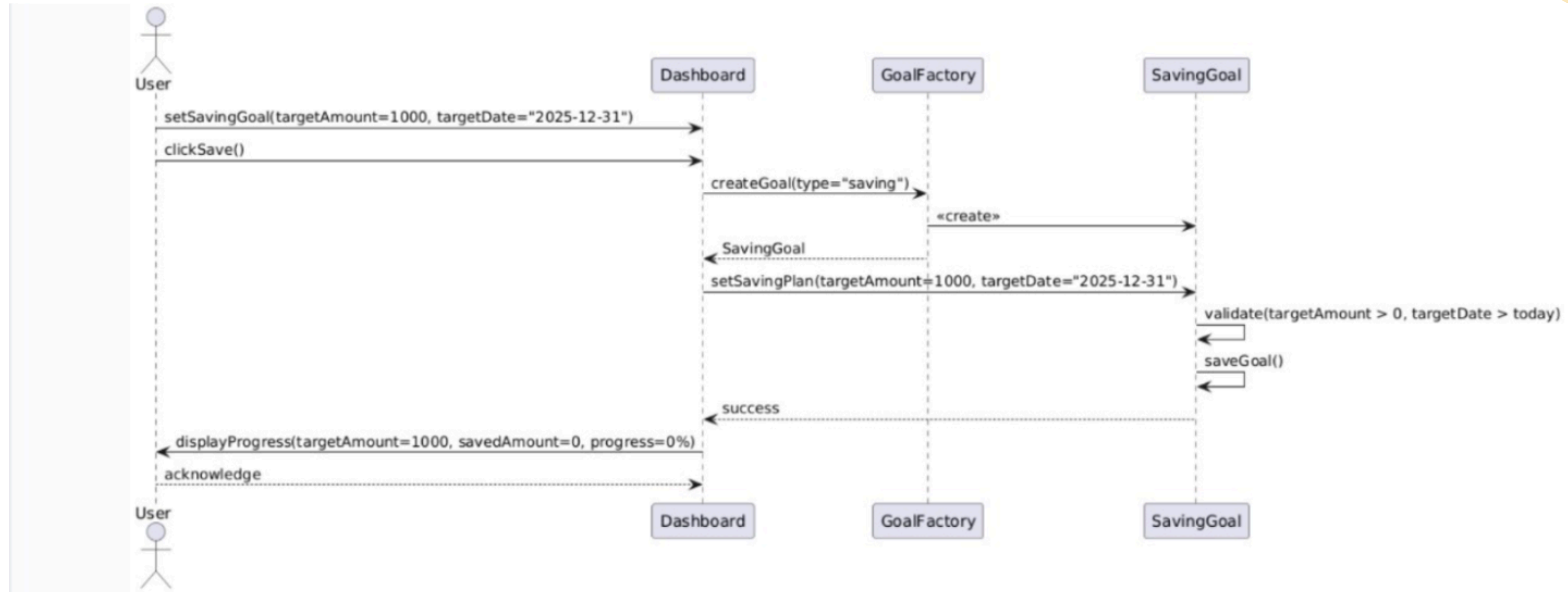
USER STORY 5: SET BUDGET



A user setting a budget (e.g., \$500 for groceries), validating the amount, creating the budget, adding it to the budgets list, and displaying it on the dashboard for confirmation.

SEQUENCE DIGRAM

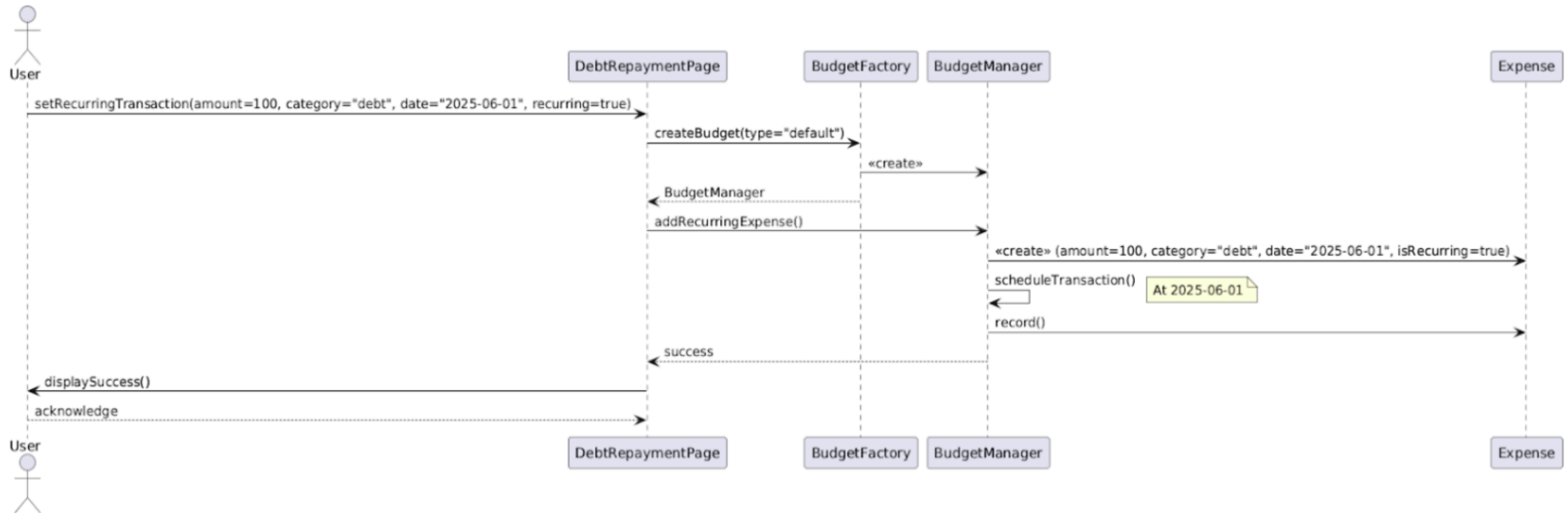
USER STORY 6 : SAVING GOALS



A user setting a savings goal (e.g., \$1000 by December 31, 2025), validating the inputs, saving the goal, and displaying the progress the dashboard.

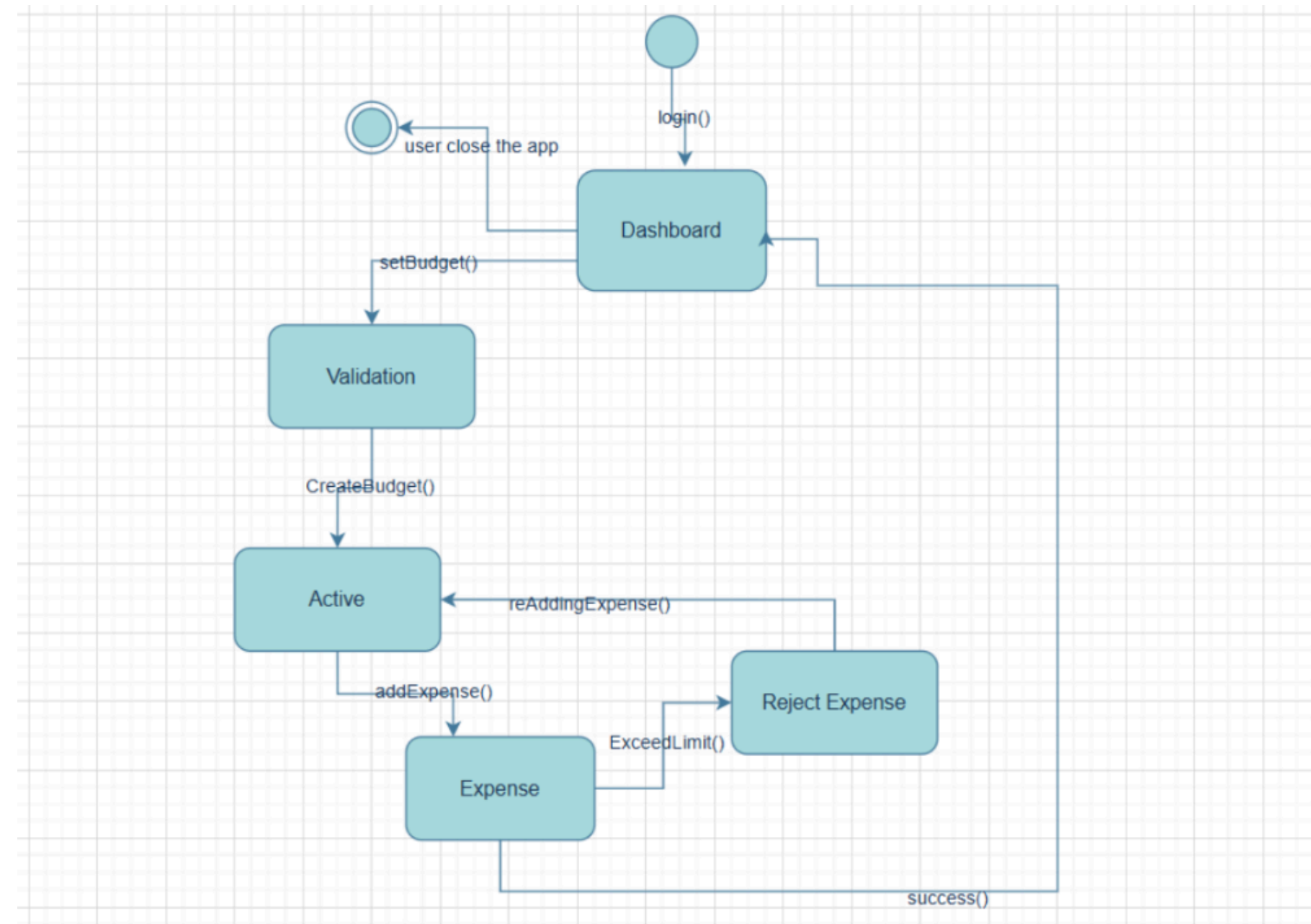
SEQUENCE DIGRAM

USER STORY 9:TRANSACTIONS



1. User submits recurring transaction details.
2. System validates and creates the transaction via BudgetManager.
3. Transaction is scheduled and recorded.
4. Success message is displayed to the user.

STATE DIAGRAM

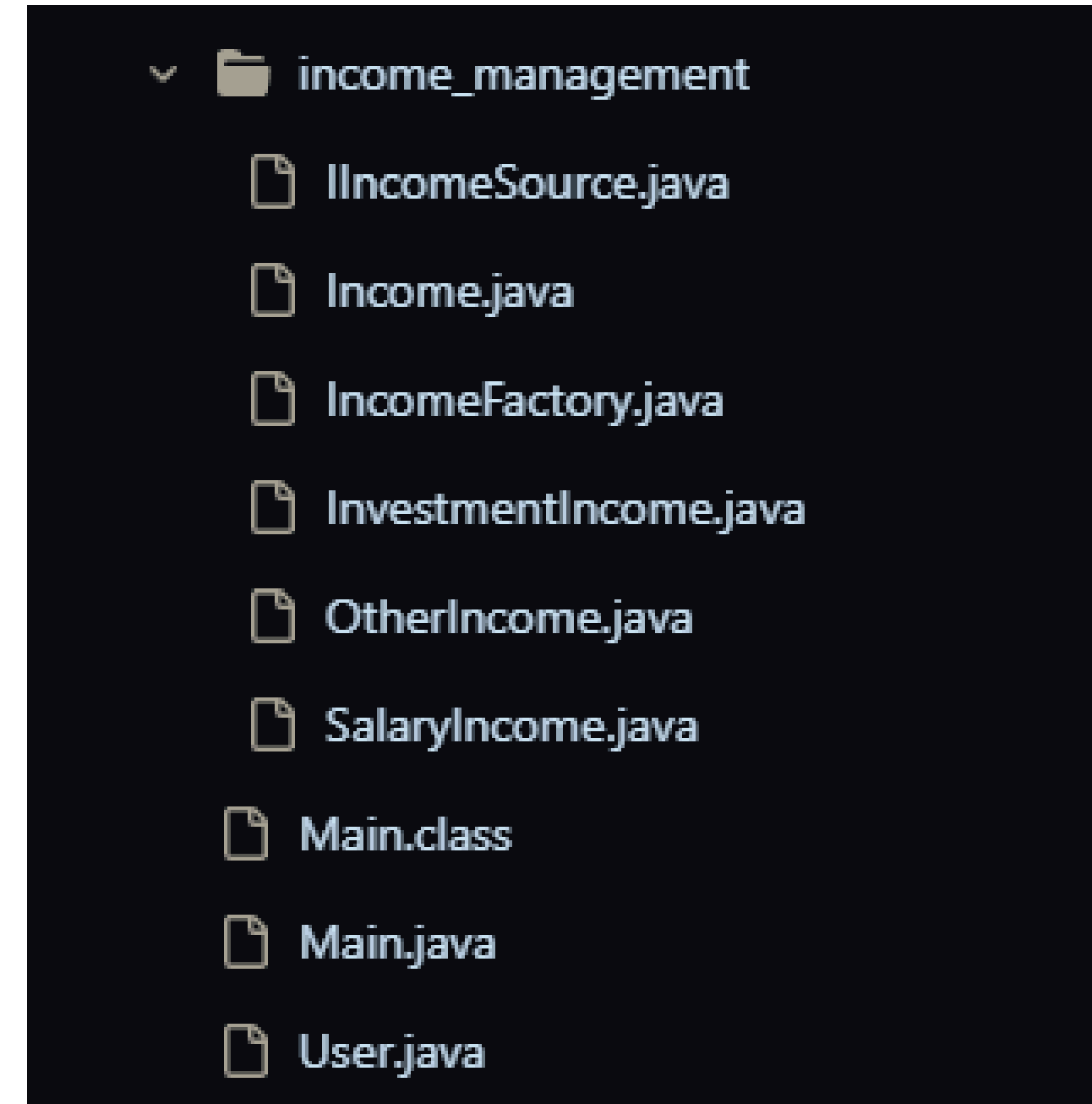
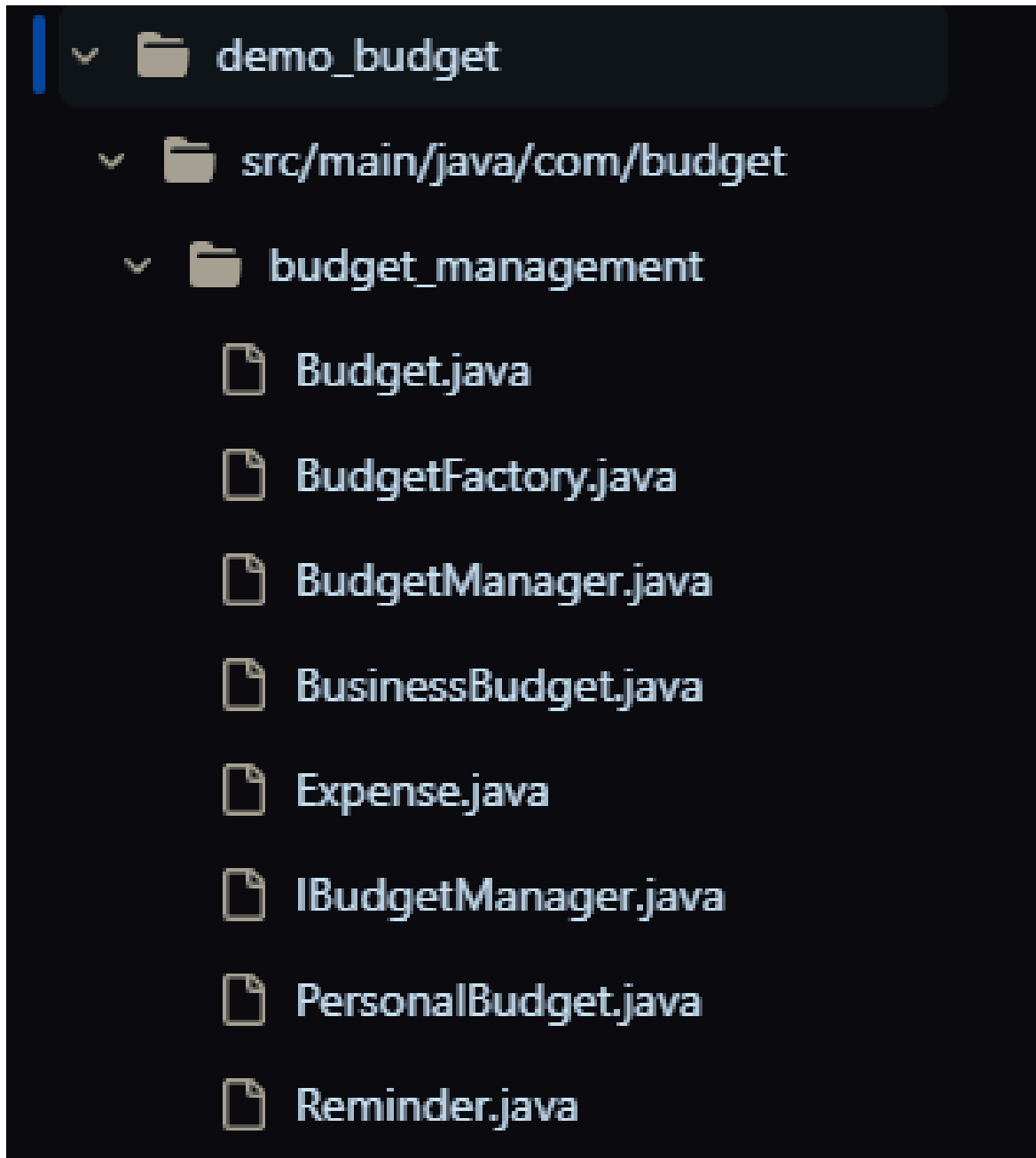


1. User closes the app after setting a budget.
2. System validates and creates the budget (`CreateBudget()`).
3. Tracks active expenses (`reAddingExpense()`, `addExpense()`).
4. Checks for budget limit violations (`ExceedLimit()`).
5. Updates the dashboard accordingly.

CONCLUSION

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis vel dolor ante. Nullam feugiat egestas elit et vehicula.
- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis vel Mina ante Maher. Nullam feugiat egestas elit et vehicula.

CODE OVERVIEW



[GitHub](#)

SOFTWARE ENGINEERING AGENTS - A SNAPSHOT

- **OVERVIEW:** SUPERVISED AGENTIC MODES (E.G., CHOP, PROMPT-TO-CODE) IN IDES TRANSFORM CODING WITH AI-DRIVEN PRODUCTIVITY AND HUMAN OVERSIGHT.
- **CURRENT STATE:** TOOLS LIKE CURSOR, CLINE, WINDSURF, AND GITHUB COPILOT EXCEL IN MODIFYING CODE, RUNNING TESTS, AND RESPONDING TO IDE FEEDBACK, POWERED BY MODELS LIKE CLAUDE'S SONNET.
- **BENEFITS:** FASTER CODING, TASK AUTOMATION, AND CONTEXT-AWARE PERFORMANCE FOR WELL-DEFINED TASKS.
- **CHALLENGES:** COMPLACENCY RISKS, LIMITED SCOPE FOR COMPLEX TASKS, AND UNRELIABLE BENCHMARKS.
- **BEST PRACTICES:** FOCUS ON SMALL TASKS, USE STRUCTURED CODEBASES, AND ENFORCE RIGOROUS REVIEWS.
- **FUTURE:** A STEPPING STONE TO AUTONOMOUS AGENTS, PENDING ADVANCEMENTS IN AI REASONING.
- **CONCLUSION:** SUPERVISED MODES BOOST EFFICIENCY AND PAVE THE WAY FOR FUTURE AI-DRIVEN DEVELOPMENT.

SLIDE: MARKITDOWN - STRUCTURED CONTENT SIMPLIFIED

- OVERVIEW:

MARKITDOWN CONVERTS PDF, HTML, POWERPOINT, AND WORD DOCUMENTS INTO STRUCTURED MARKDOWN, ENHANCING READABILITY AND LLM COMPATIBILITY.

- KEY FEATURES:

PRESERVES DOCUMENT STRUCTURE (HEADINGS, SECTIONS) FOR CONTEXT-RICH INPUT IN AI APPLICATIONS LIKE RAG WORKFLOWS.

- BENEFITS:

ENABLES STRUCTURE-AWARE CHUNKING FOR ACCURATE LLM RESPONSES, INTEGRATES WITH DEVELOPER WORKFLOWS VIA CLI, AND SUPPORTS DOCUMENTATION NEEDS.

- USE CASE:

PREPROCESSES COMPLEX DOCUMENTS TO MAINTAIN SECTION INTEGRITY, IMPROVING QUERY ACCURACY IN AI SYSTEMS.

- CONCLUSION:

MARKITDOWN STREAMLINES DOCUMENT CONVERSION, EMPOWERING AI APPLICATIONS AND EFFICIENT CONTENT MANAGEMENT FOR DEVELOPERS.

SLIDE: RAILWAY - SIMPLIFYING APP DEPLOYMENT

- **OVERVIEW:** RAILWAY IS A MODERN CLOUD PLATFORM FOR DEPLOYING FULL-STACK APPS, OFFERING AN ALTERNATIVE TO HEROKU AND VERCEL WITH A FOCUS ON SIMPLICITY AND MANAGEMENT.
- **KEY FEATURES:**
 - EFFORTLESS DEPLOYMENT WITH AUTO-GENERATED DOCKERFILES AND DOCKER HUB SUPPORT.
 - CUSTOMIZABLE SETTINGS FOR FLEXIBLE CONTROL.
 - COMPREHENSIVE TOOLS FOR MANAGING SETTINGS, SECRETS, AND ENVIRONMENTS.
 - SEAMLESS CLI AND API INTEGRATION WITH EXISTING WORKFLOWS.
 - BUILT-IN MONITORING FOR PERFORMANCE AND HEALTH INSIGHTS.
 - RELIABLE, TRANSPARENT PLATFORM WITH CLEAR DOCUMENTATION.
- **CONCLUSION:** RAILWAY STREAMLINES APP DEPLOYMENT AND LIFECYCLE MANAGEMENT, PROVIDING A HASSLE-FREE SOLUTION FOR DEVELOPERS FROM CODE TO PRODUCTION