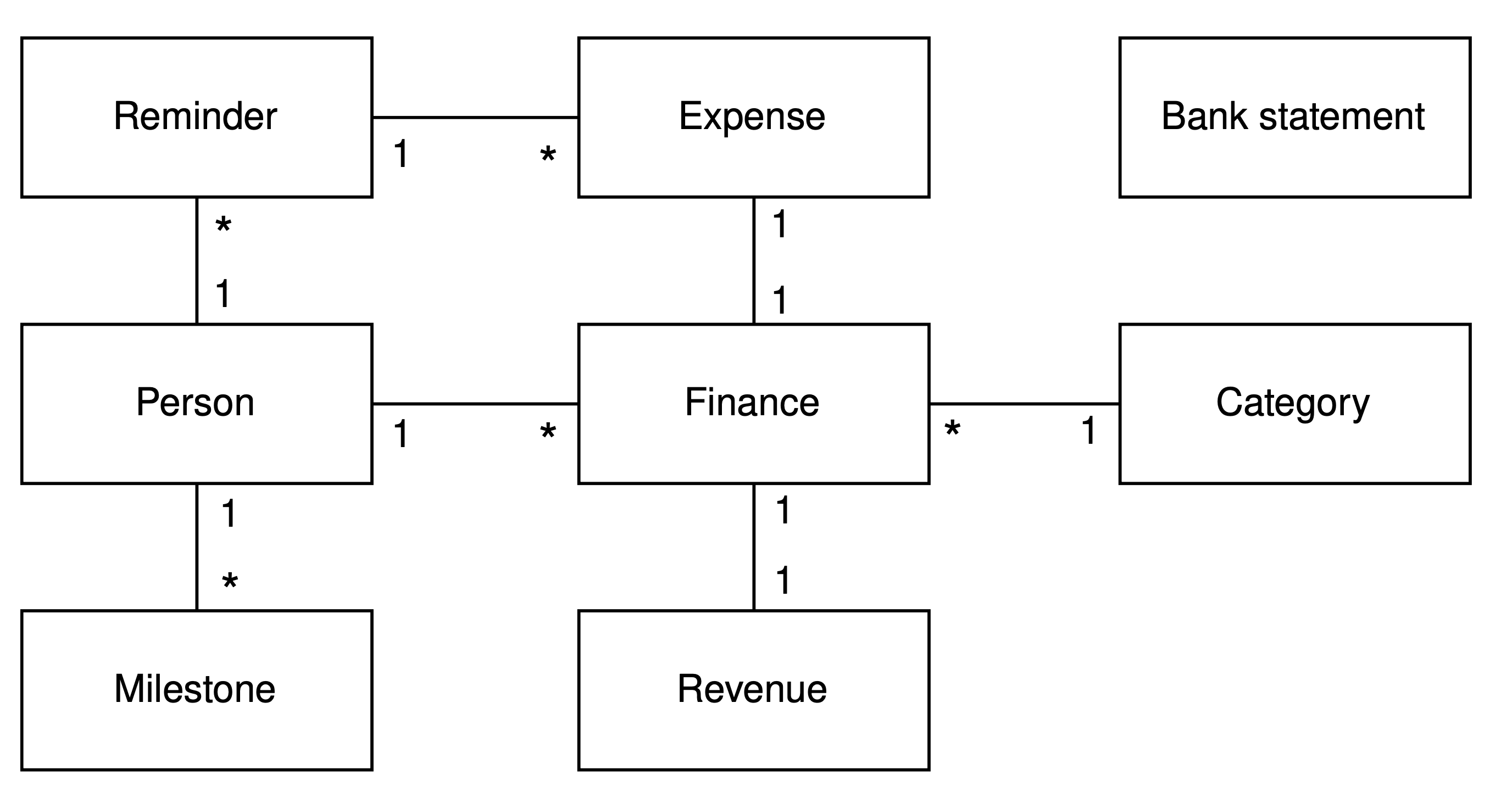
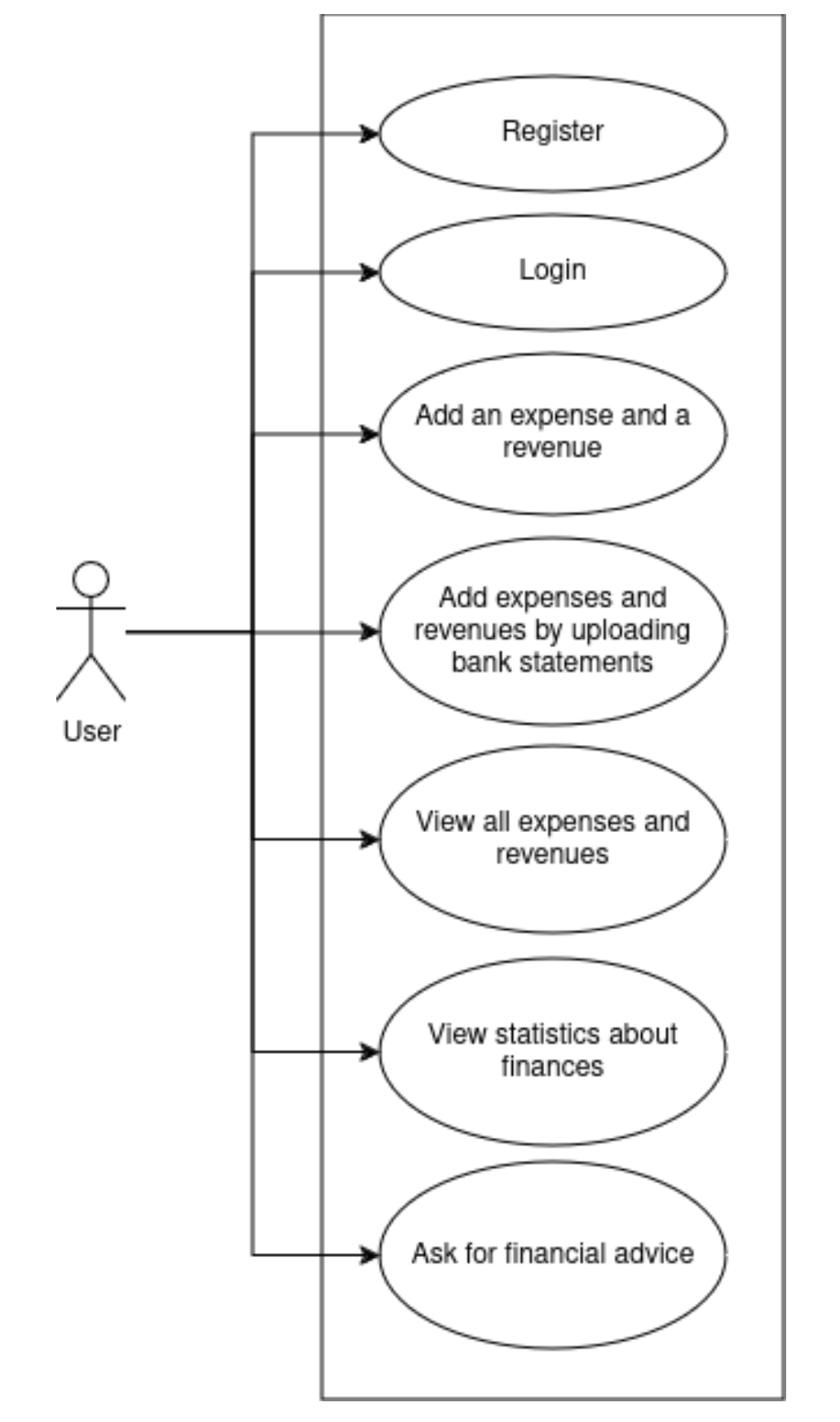
Textual information model

|  |  |
| --- | --- |
| Type | Definition |
| Person | Someone who tracks Finances |
| Finance | Person’s Revenue or Expense |
| Expense | Spent money by Person |
| Revenue | Earned money by Person |
| Bank statement | Person’s information about their Finances from a certain bank |
| Category | A group of Finances |
| Milestone | An achievement of Person that can be reached by defined amount of money |
| Reminder | A message reminding Person about upcoming Expenses |

Graphical information model 

Use case diagram



## 1. Use Case: Register

**Actor:** Guest (unauthenticated user)

**Description:** Allows a new user to create an account by supplying required personal details.

|  |  |
| --- | --- |
| **Preconditions** | **The user is not already registered or logged in.** |
| **Trigger** | User clicks “Register” on the welcome screen. |

### Main Success Scenario

1. User navigates to the **Register** form.
2. User provides **email**, **password**, **full name**, and any other required fields.
3. System validates inputs (e.g. email format, password strength).
4. System creates a new **User** record in the database.
5. System sends a confirmation (e.g. HTTP 201 Created) and may dispatch a verification email/SMS.
6. User receives confirmation and is redirected to the **Login** screen.

### Alternative Flows

* **3a. Invalid input:**
  + 3a1. If validation fails (e.g. email already in use, weak password), system returns error messages.
  + 3a2. User corrects the input and resubmits.

### Postconditions

* A new user account exists in the database (status: unverified or active).
* User can now log in.

## 2. Use Case: Login

**Actor:** Registered User

**Description:** Allows a registered user to authenticate and gain access to their personalized dashboard.

| **Preconditions** | User must have a valid registered account (and completed verification if required). | | **Trigger** | User clicks “Login” and submits credentials. |

### Main Success Scenario

1. User enters **email** and **password** on the **Login** form.
2. System validates credentials against stored user records.
3. System generates a **JWT** token and attaches user-specific claims.
4. System returns HTTP 200 OK with the token and basic user info.
5. Client stores the token (e.g. in localStorage) and redirects user to the **Financial Dashboard**.

### Alternative Flows

* **2a. Invalid credentials:**
  + 2a1. System returns HTTP 401 Unauthorized with an error message.
  + 2a2. User may retry or request password reset.

### Postconditions

* User is authenticated and holds a valid JWT for subsequent requests.

## 3. Use Case: Add an Expense and a Revenue

**Actor:** Authenticated User

**Description:** Enables manual input of individual financial transactions (expenses or revenues).

| **Preconditions** | User is logged in with a valid JWT. | | **Trigger** | User selects “Add Transaction” in the dashboard. |

### Main Success Scenario

1. User opens the **Add Transaction** form.
2. User selects **type** (Expense or Revenue), **amount**, **date**, **category**, and optionally a **description**.
3. System validates the data (e.g. non-negative amount, valid date).
4. System creates a new **Transaction** record linked to the user.
5. System returns HTTP 201 Created with the transaction details.
6. Dashboard updates to reflect the new transaction and recalculates balances.

### Alternative Flows

* **2a. Missing required field:**
  + 2a1. System highlights missing fields and prevents submission.
  + 2a2. User completes missing information.

### Postconditions

* A new Transaction exists in the database.
* User’s current balance and statistics update accordingly.

## 4. Use Case: Add Expenses and Revenues by Uploading Bank Statements

**Actor:** Authenticated User

**Description:** Allows bulk import of transactions via bank statement files.

| **Preconditions** | User is logged in. File format supported (PDF, CSV). | | **Trigger** | User selects “Upload Statement” in the dashboard. |

### Main Success Scenario

1. User clicks **Upload Statement**.
2. User selects a statement file (e.g. CSV or PDF) from their device.
3. System receives the file (multipart/form-data).
4. System parses the statement and extracts transaction lines.
5. System validates each transaction (date, amount, description).
6. System saves all valid transactions in bulk for that user.
7. System returns HTTP 200 OK with a summary: imported count, skipped lines, errors.
8. Dashboard refreshes to show newly imported transactions.

### Alternative Flows

* **4a. Parsing error:**
  + 4a1. If file format is unrecognized or corrupted, system returns HTTP 400 Bad Request with details.
  + 4a2. User corrects the file or chooses another.
* **4b. Partial import:**
  + 4b1. Some lines fail validation—system imports valid lines, reports errors for failed ones.

### Postconditions

* Multiple Transaction records added.
* User’s balance and statistics updated in bulk.

## 5. Use Case: View All Expenses and Revenues

**Actor:** Authenticated User

**Description:** Displays a paginated, filterable list of all recorded transactions.

| **Preconditions** | User is authenticated. | | **Trigger** | User navigates to “Transactions” page. |

### Main Success Scenario

1. Client sends GET /transactions?from=YYYY-MM-DD&to=YYYY-MM-DD&page=X&size=Y.
2. System validates JWT and query parameters.
3. System queries the **TransactionRepository** with optional date filters.
4. System returns HTTP 200 OK with a paginated list of Transaction DTOs.
5. Client renders the list in a table with sorting, filtering controls.

### Alternative Flows

* **5a. No transactions found:**
  + System returns an empty list with HTTP 200 OK—client displays “No transactions found.”

### Postconditions

* User sees an up-to-date ledger of all financial activity in the chosen period.

## 6. Use Case: View Statistics about Finances

**Actor:** Authenticated User

**Description:** Provides summarized statistical insights (totals, averages, category breakdowns).

| **Preconditions** | User is authenticated and has existing transactions. | | **Trigger** | User clicks “Statistics” in the dashboard. |

### Main Success Scenario

1. Client requests GET /transactions/statistics?from=…&to=….
2. System aggregates data (sum of expenses, revenues, per-category totals, time-series).
3. System returns HTTP 200 OK with a **StatisticsResponse** JSON.
4. Client displays charts (bar, pie, line) and summary cards (e.g. total spent, total earned).

### Alternative Flows

* **6a. Insufficient data:**
  + If fewer than N transactions exist, system may return a minimal summary with a “Too little data” note.

### Postconditions

* User gains insight into spending trends and income patterns.

## 7. Use Case: Ask for Financial Advice

**Actor:** Authenticated User

**Description:** Sends a user’s query to an AI-backed advisory service and returns personalized recommendations.

| **Preconditions** | User is logged in and has some transaction history. | | **Trigger** | User selects “Get Advice” and submits a text question. |

### Main Success Scenario

1. User enters a question or goal (e.g. “How can I save more?”) into the **Advice** form.
2. Client sends POST /advice with the text payload.
3. System forwards the request to the OpenAI client (FinancialAdviserService).
4. OpenAI returns a generated advice text.
5. System wraps it in a **FinancialAdviceResponse** DTO.
6. System returns HTTP 200 OK with the advice.
7. Client displays the advice; user may save or rate it.

### Alternative Flows

* **7a. AI service error:**
  + 7a1. If OpenAI API fails or times out, system returns HTTP 503 Service Unavailable with a retry suggestion.

### Postconditions

* User receives personalized financial guidance based on their query.

These scenarios map one-to-one with the controllers and services in the codebase, and they can be used to drive both functional tests and UI wireframes. Let me know if you’d like additional diagrams, test case templates, or sample request/response payloads!

Sequence diagram for bank statement uploading process

