Donation Tracker System

Presented by :

Omar Hussien Taha	*221008562*
Ahmed Abdelbaset	*221018000*
Mohammed Ahmed	*221018082*
Ahmed Osama	*221008669*
Mohammed Gomaa	*221018546*

Presented To :

Dr / Shady Zahran

Eng / Sohila Abdallah

Table of contents

Table Of Figures

Donation Tracker System

1. Analysis

1.1 Problem Statement Form

1.1.1 Problem Definition

In today's world, the rise of donation campaigns is often met with skepticism from potential donors due to the lack of transparency regarding how their money is used. This uncertainty results in hesitation and a significant drop in charitable contributions. Therefore, we propose the development of a donation tracking system that ensures transparency by allowing donors to monitor where and how their money is being utilized.

1.1.2 Motivation

The main motivation behind this project is to restore donor confidence by creating a reliable and transparent digital system. Many people refuse to donate because they suspect fraud or mismanagement. By building this system, we aim to eliminate doubt and provide a clear view of the donation journey, from payment to allocation.

1.1.3 Objectives

- Accept and record donation transactions.
- Collect and store donor names, donation amounts, and timestamps.
- Allow administrators to add, modify, or delete donation records.
- Visually represent donation statistics using graphs and charts.
- Show the allocation process and final destination of donations.
- Provide a user-friendly and accessible interface for all user types.
- Implement relevant object-oriented design patterns to ensure a scalable and maintainable system.

1.1.4 Requirements

Functional Requirements:

- Accept new donations.
- View a list of all donations.
- Search donations by donor name, amount, or campaign.
- Admin privileges to add/edit/delete donation records.
- Graphical representation of total and individual donation statistics.
- Show donation usage history to the donor.

Non-Functional Requirements:

- Simple and intuitive graphical user interface (GUI).
- Secure data handling and protection from tampering.
- High performance to handle a large number of transactions.
- Future extensibility (e.g., support for new campaigns, notification system, PDF reports).

1.1.5 Constraints

- Requires a strong and reliable database (e.g., PostgreSQL or MySQL).
- Needs a high-performance server to handle real-time operations.
- Must implement various learned design patterns such as Singleton, Factory, Observer, etc.
- The system must remain simple and easy to use for all user levels, including non-technical users.
- Project should be delivered within a limited development timeframe.

2.1Design Patterns

1. Abstract Factory Pattern

We used the Abstract Factory Pattern to manage different types of services (like indoor, outdoor, VIP tables, delivery, and takeaway). This allows consistent creation of related service objects without specifying their concrete classes.

Key Components:

Component	Description
Abstraction	Service interface declaring getFess() to retrieve service fee.
Implementation	IndoorTables, OutdoorTables, VIPTables, deliveryService, takeAwayService implement the interface.
Concrete Implementations	These classes define specific logic for service fees.
(Optional) Refined Abstraction	Can be extended (e.g., AdvancedService) to add more methods.

2. Prototype Pattern

We used the Prototype Pattern to clone existing order objects easily, especially when creating similar orders multiple times.

Key Components:

Component	Description
Prototype (Interface)	Declares cloneObject method (e.g., Invoice).
Concrete Prototype	Implements cloning logic (e.g., orderCloned).
Client	Uses the prototype to create new object copies.
Product	Structure holding product data (e.g., ProductOrderFinal).

3. Factory Pattern

We applied the Factory Pattern to simplify the creation of different food items such as CheeseBeard and Salad through a common interface.

Key Components:

Component	Description
Product	Defines object interface (e.g., Product).
Concrete Product	Implements product logic (e.g., CheeseBeard, Salad).
Creator	Declares method to return product (e.g., Order).
Concrete Creator	Overrides factory method (e.g., RestaurantOrder).

4. Singleton Pattern

We used the Singleton Pattern in database connection to ensure only one instance is used across the system.

Key Components:

Component	Description
Singleton Class	Manages single instance (e.g., DatabaseConnection).
Private Constructor	Prevents external instantiation.
Static Instance	Holds the single object.
Public Static Method	Returns the global instance (e.g., getInstance()).
Resource Access	Provides methods like getConnection() to access DB.

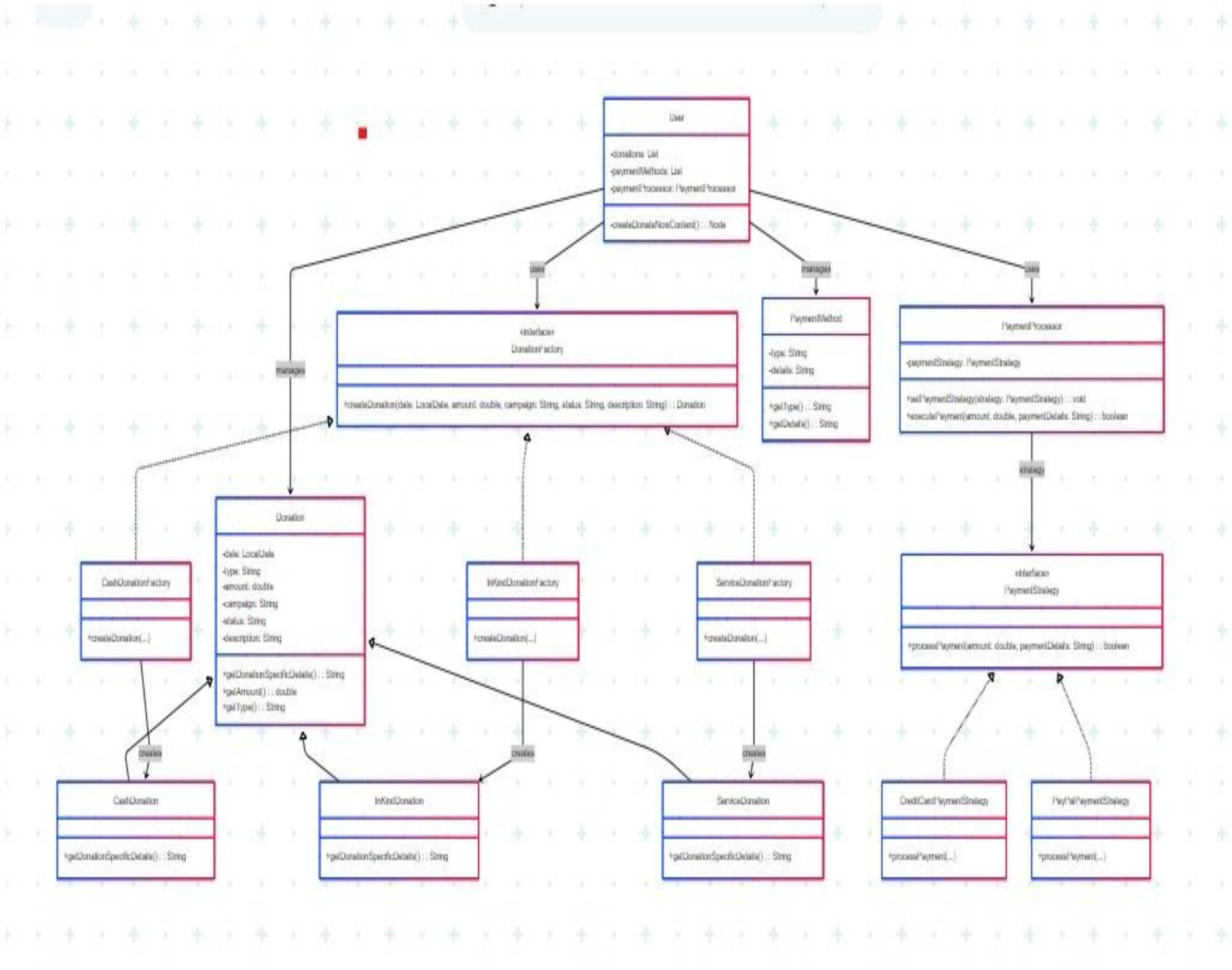
5. Builder Pattern

We used the Builder Pattern to build complex customer orders in a step-by-step approach.

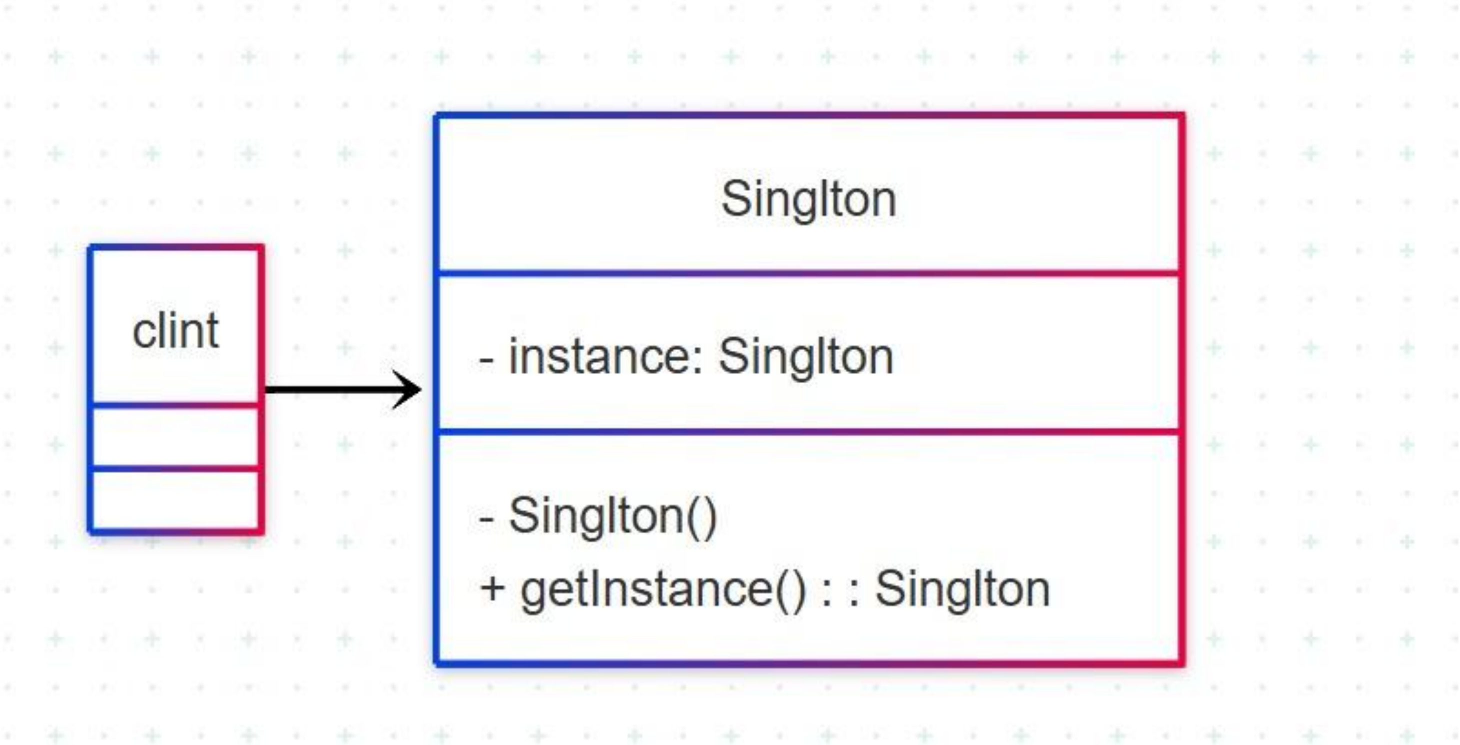
Key Components:

Component	Description
Product	The object being built (e.g., OrderCus).
Builder Interface	Defines building steps (e.g., BuilderFactory).
Concrete Builder	Implements the builder (e.g., orderBuilder).
Director	(Not explicitly used, but can manage build steps).
Final Product	Snapshot of full order (e.g., ProductOrderFinal1).

3.1The UML.



Project UML [1].



Project UML [2].

4.1Structure OverView

Class	Role
User	Represents a user with profile info (ID, name, email, phone, address)
Donation (Abstract)	Base class for donations (date, type, amount, status, etc.)
CashDonation	Represents a cash donation with amount and description
InKindDonation	Represents an in-kind donation (items donated)
ServiceDonation	Represents a service donation (description of service)
DonationFactory	Interface to create donation objects using factory method
CashDonationFactory	Creates a CashDonation object

Class	Role
InKindDonationFactory	Creates an InKindDonation object (ignores amount)
ServiceDonationFactory	Creates a ServiceDonation object (ignores amount)
Notification	Handles notifications (date, title, message)
PaymentMethod	Stores payment method type and details (Visa, PayPal...)
PaymentStrategy	Interface defining how payments are processed
CreditCardPaymentStrategy	Implements credit card payment processing
PayPalPaymentStrategy	Implements PayPal payment processing
PaymentProcessor	Sets and executes payment using the selected strategy