[INITIAL RELEASE]

Project Design Document

Team 1a - JSCOPE

Oliver Gomes Jr. (odg1896@rit.edu)

Patrick Lebeau (pml4324@rit.edu)

Jonathan Ho (jlh5360@rit.edu)

Connor Bashaw (cdb9772@rit.edu)

Project Summary

FundGoodDeeds App (Version 2.0) is a powerful yet user-friendly tool designed to help individuals and families manage their financial needs and funding sources with clarity and precision. By organizing basic needs, bundles of essentials, and funding sources in simple CSV files, the app keeps a daily ledger of fulfilled needs, received funds, and thresholds, making it easy to track spending, income, and net financial impact at a glance. Users can add, edit, or remove needs and funding sources, set daily financial goals, and maintain an accurate record of their financial activity, all through an intuitive interface that ensures data is always up-to-date and error-free.

With FundGoodDeeds, users gain the ability to plan and optimize their finances effortlessly. The app automatically calculates total costs of expenses, tracks income, and net balances, while alerting when expenses exceed predefined thresholds. Bundles of essential items, recurring funding sources, and historical logs provide valuable insights for budgeting and decision-making. Whether you're managing monthly expenses, tracking income sources, or ensuring your essential needs are met efficiently, FundGoodDeeds empowers users to stay organized, financially informed, and confident in their daily financial decisions.

Design Overview

The Design evolved from simple class sketches and noun verb analysis of project requirements to a structured multi-subsystem architecture emphasizing separation of concerns, high cohesion, and low coupling.

Architecture

Model–View–Controller (MVC) separates presentation, business logic, and data.

Patterns

Composite Pattern models *Bundles* of *Needs* uniformly, enabling recursive operations such as total cost or fulfillment status.

Observer Pattern allows the UI and reporting components to react automatically to updates in the data model.

Repository Pattern (custom pattern) allows for abstraction of stored data access. All CRUD (Create, Read, Update, and Delete) operations are handled in one place, to keep our code organized, easier to maintain, and output back to a data store upon request.

Command Pattern allows for our app to invoke common commands for multiple views (or receivers) in the system to pave the way for a CLI and GUI to work synchronously.

Principles

DIP via Dependency Injection allows for extensibility, since all of the objects being passed in any constructor will depend on the same interface, we can interchange the objects without any conflicts.

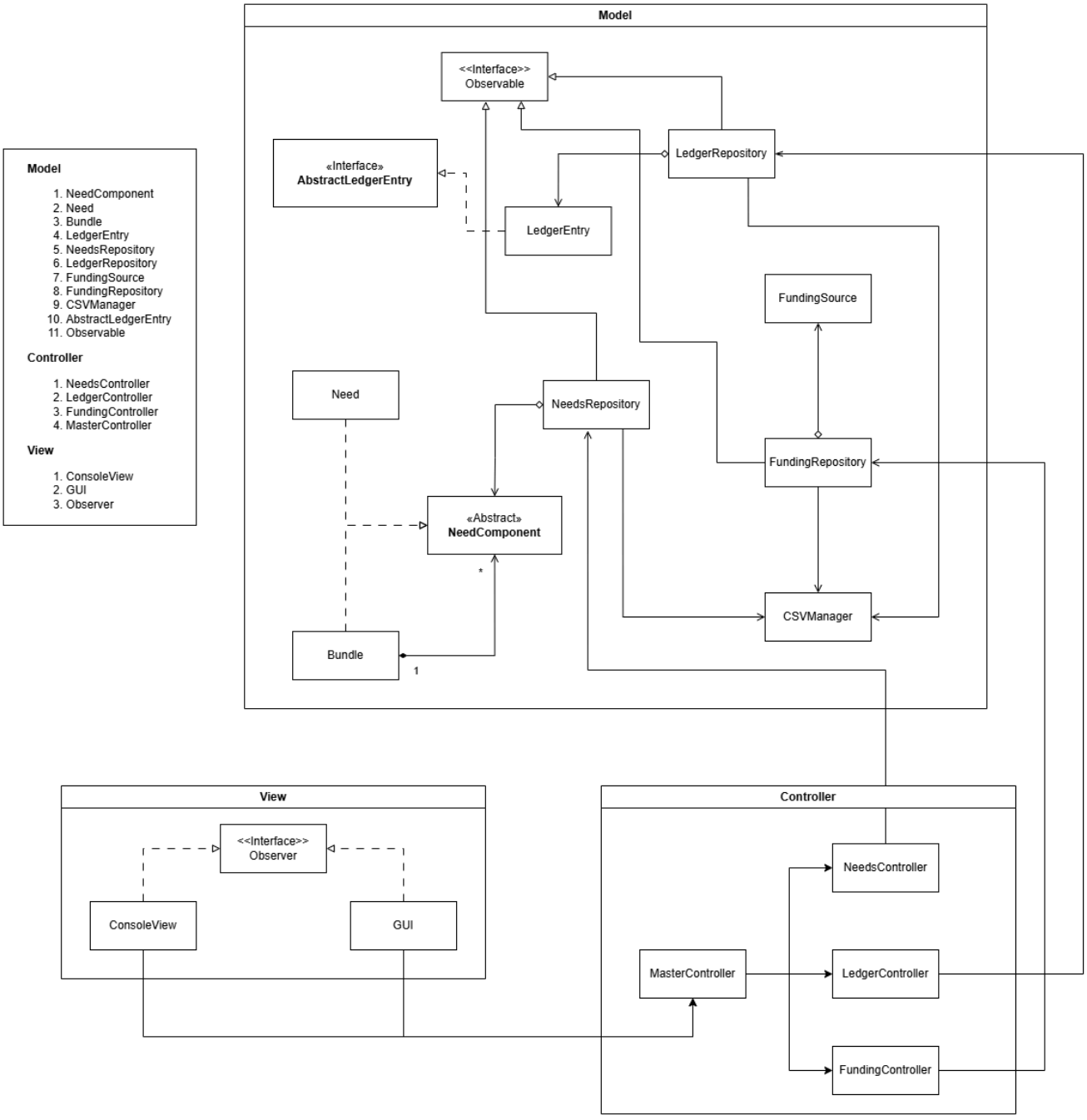
Early prototypes revealed tight coupling between UI and logic, these were then handled through interface abstractions. Dependency inversion was applied to allow testing and future integrations with external API's or databases.

Rejected alternatives include direct database coupling in controllers and hard-coded UI bindings. Assumptions are 1 active donor per session and stable in memory data storage.

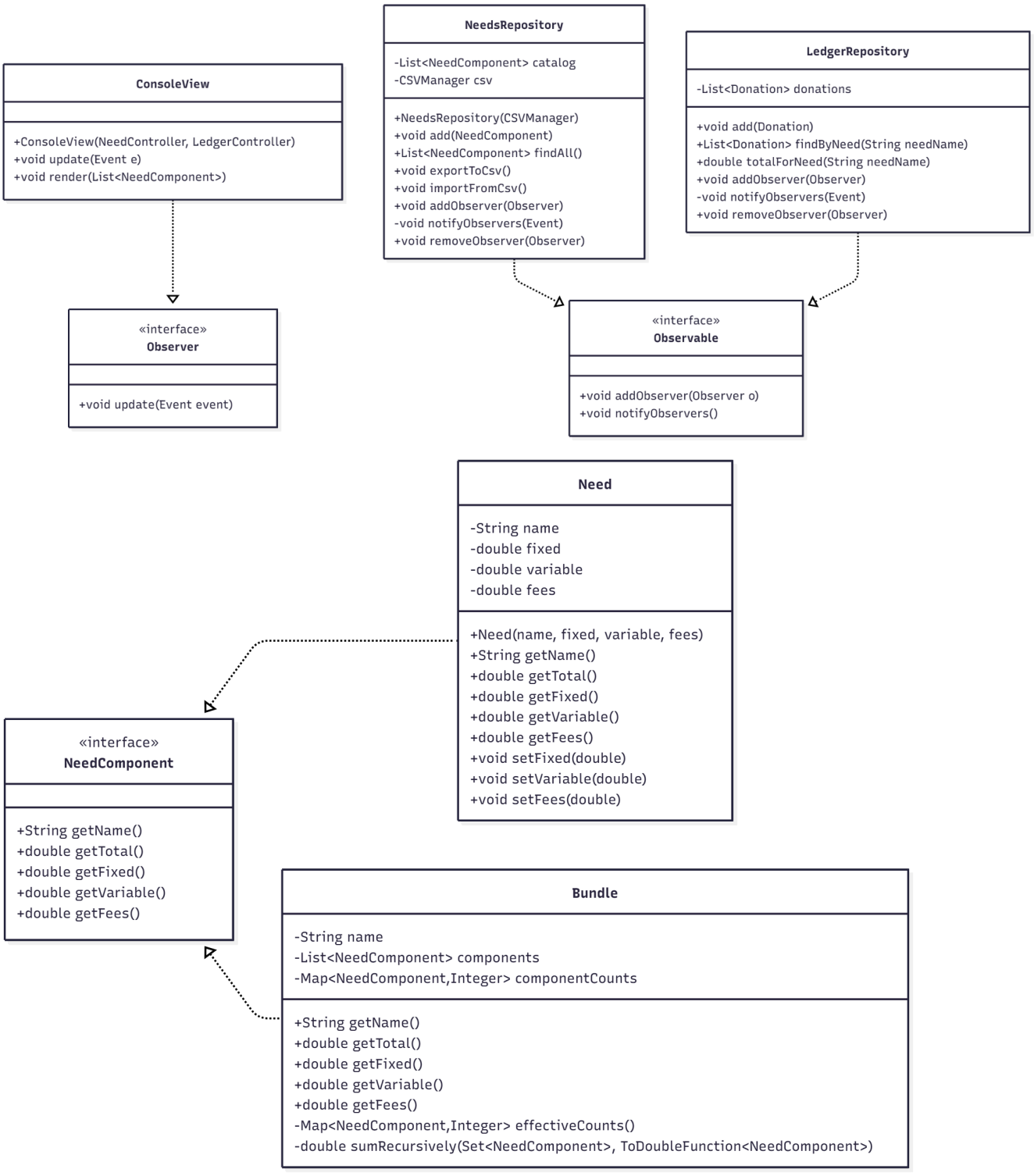
DRY Principle allows for functionality that is expected to be repeated to exist as an abstraction instead of repeated code. Our design holds a csv parser as multiple entities in the system are derived from data files and not always created by the user.

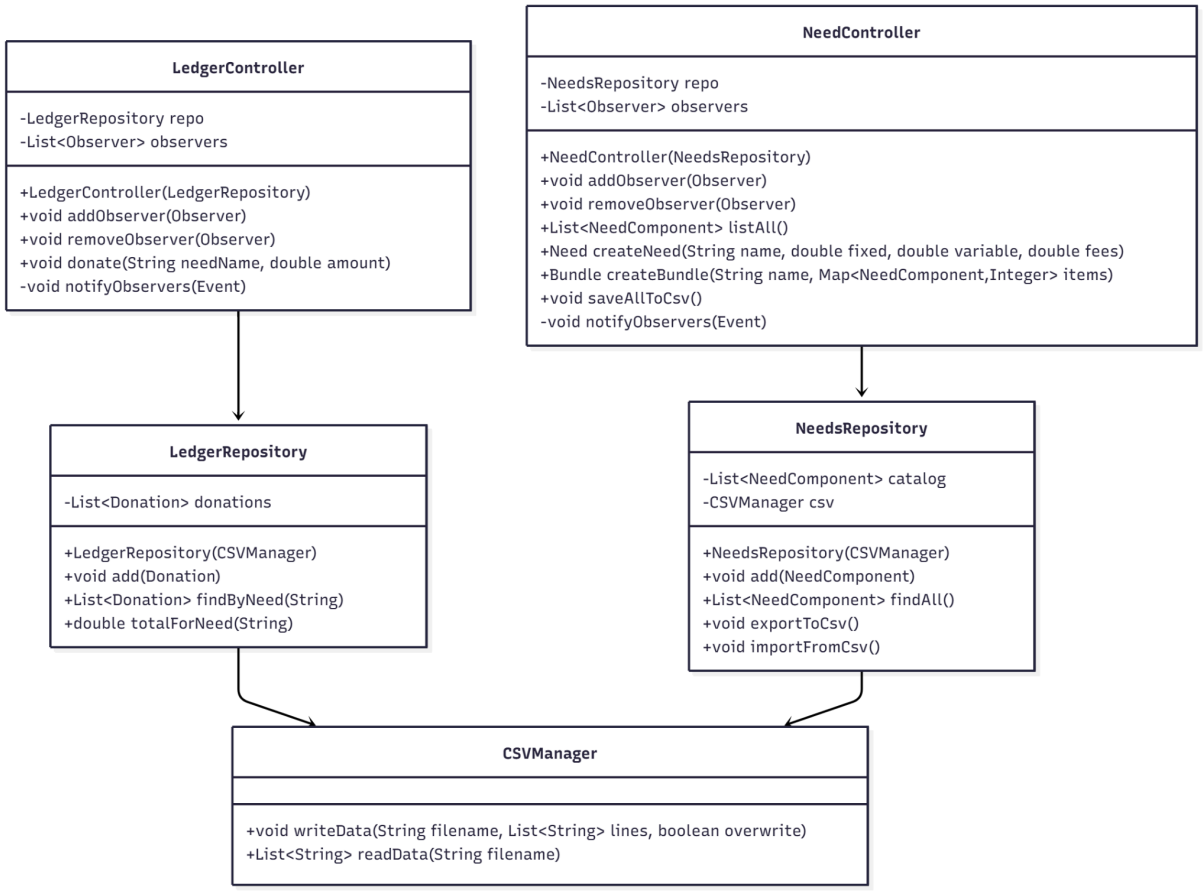
SRP (Single Responsibility Principle) unlike our first iteration, we will ensure our classes in our view are only responsible for displaying information and sending user input to the controllers. We do not want extra processing logic in the view but maintained in our controllers.

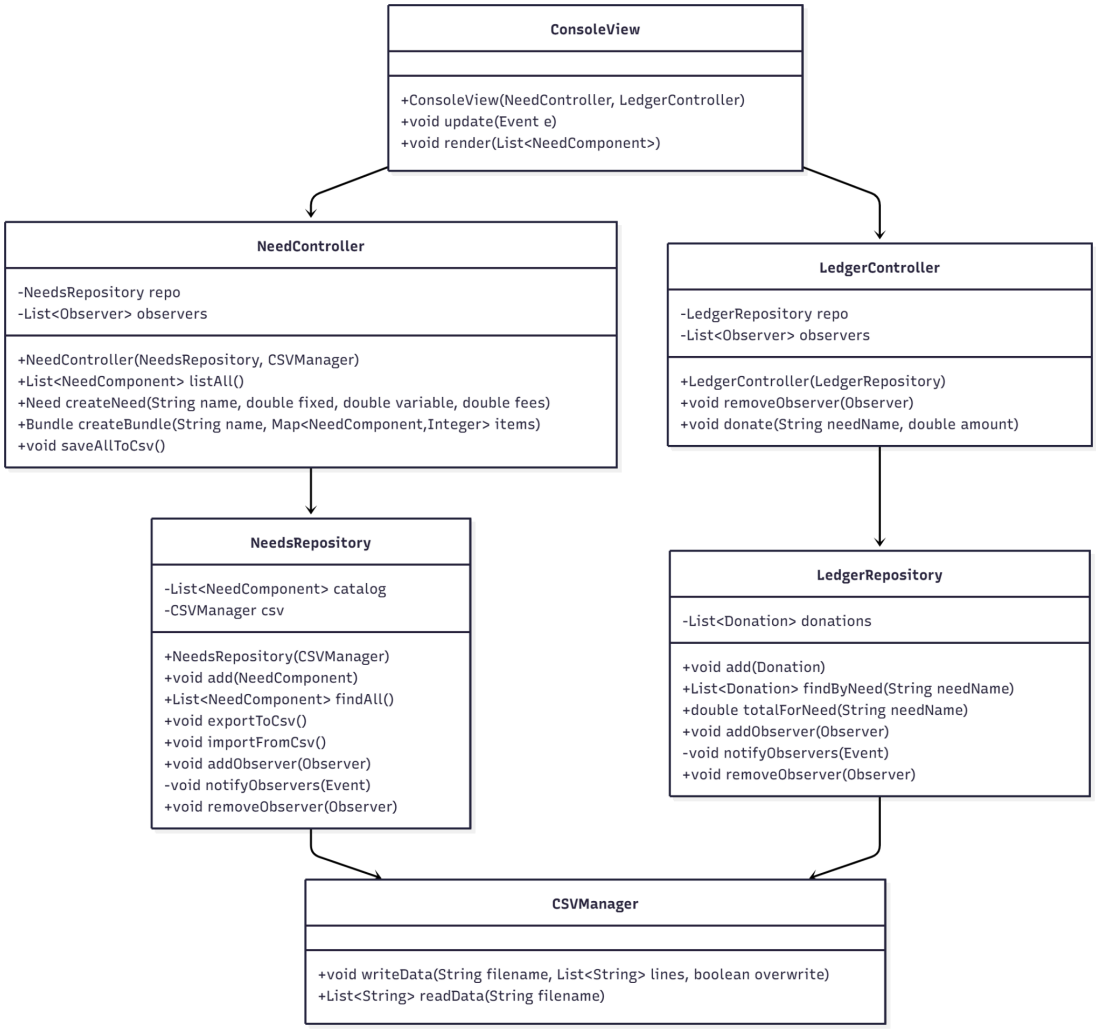
System Architecture (MVC)



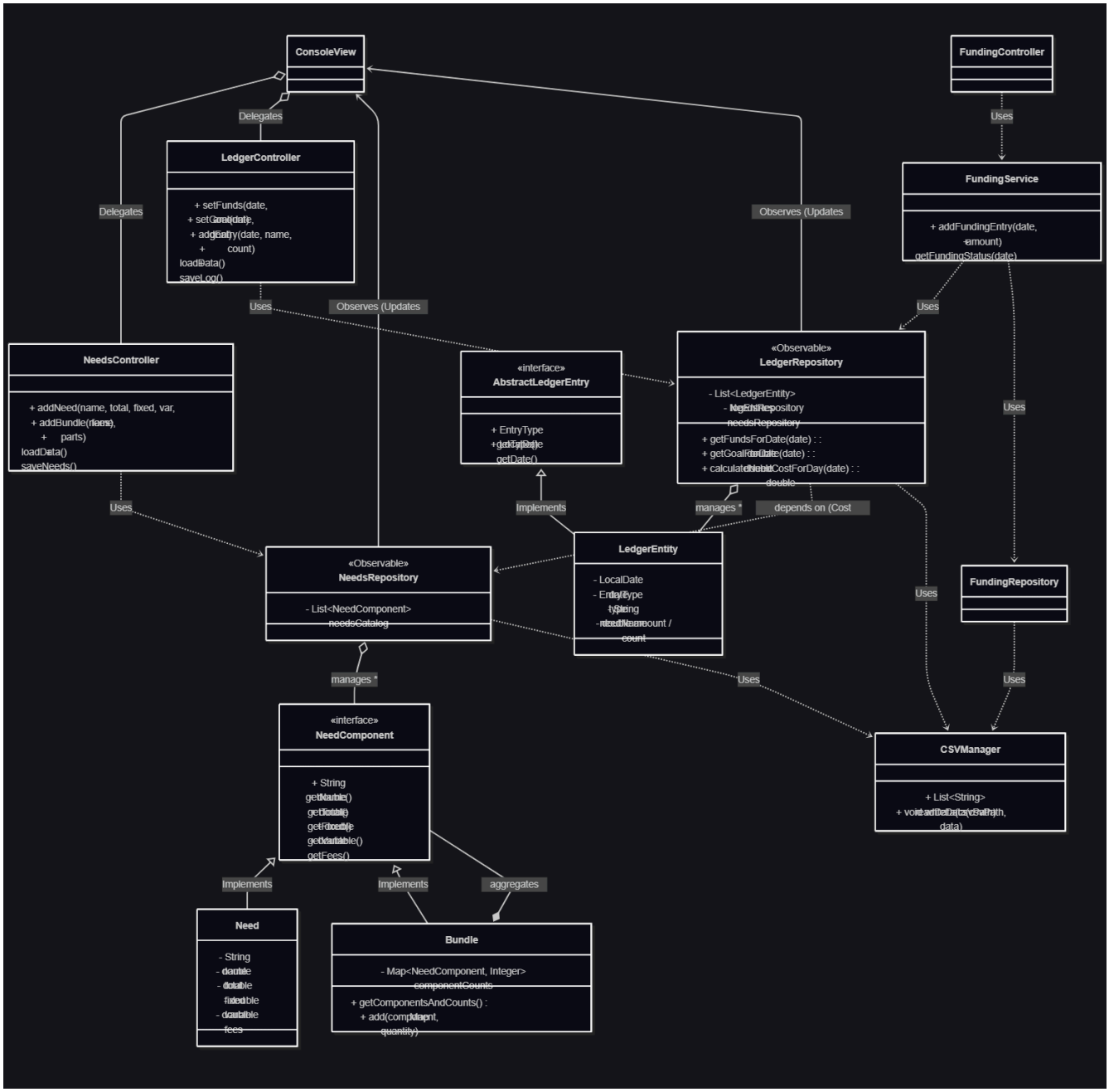
V1

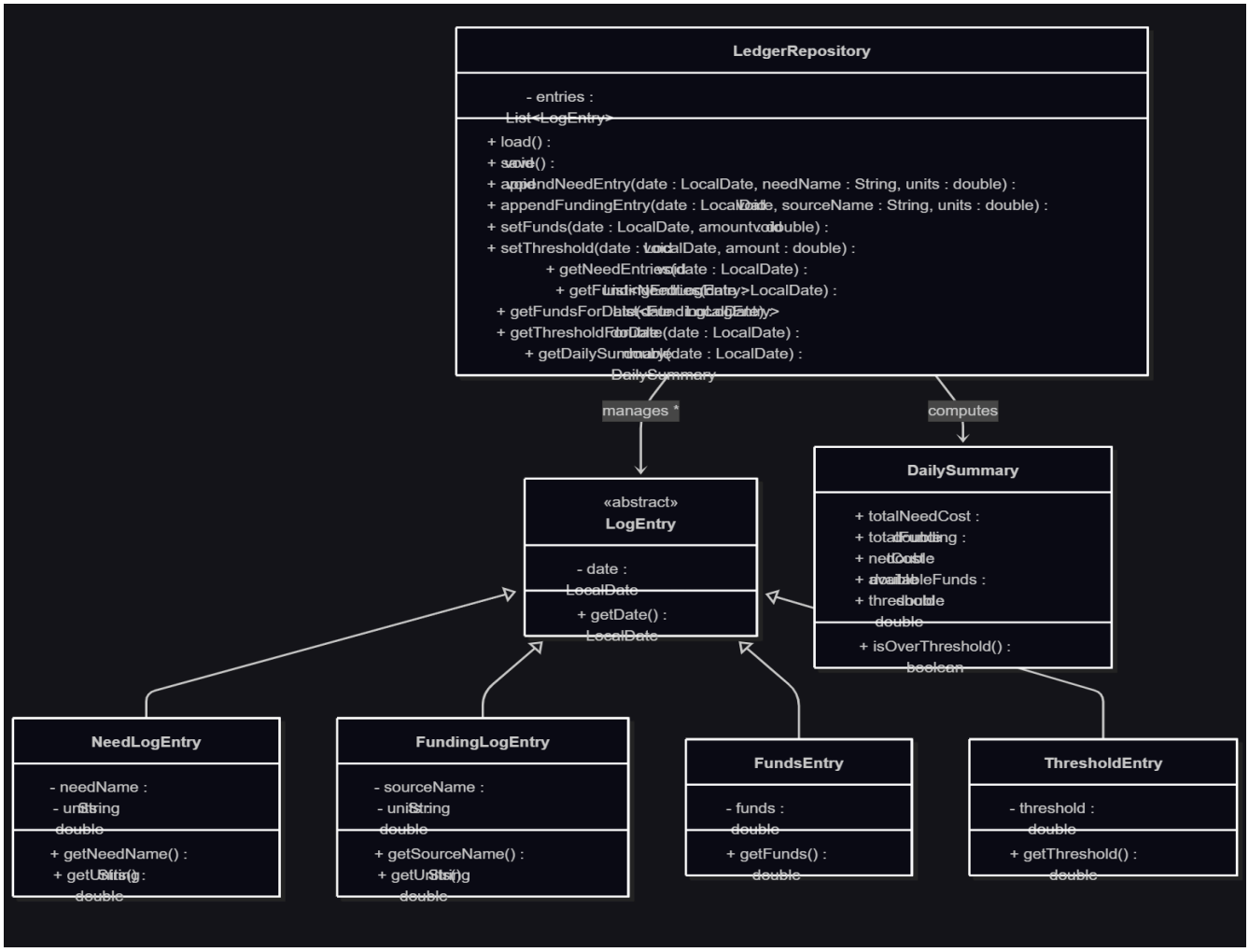

ConsoleView
+ConsoleView(NeedController, LedgerController)
+void update(Event e)
+void render(List<NeedComponent>)
NeedsRepository
-List<NeedComponent> catalog
-CSVManager csv
+NeedsRepository(CSVManager)
+void add(NeedComponent)
+List<NeedComponent> findAll()
+void exportToCsv()
+void importFromCsv()
+void addObserver(Observer)
-void notifyObservers(Event)
+void removeObserver(Observer)
LedgerRepository
-List<Donation> donations
+void add(Donation)
+List<Donation> findByNeed(String needName)
+double totalForNeed(String needName)
+void addObserver(Observer)
-void notifyObservers(Event)
+void removeObserver(Observer)
«interface»
Observer
+void update(Event event)
«interface»
Observable
+void addObserver(Observer o)
+void notifyObservers()
Need
-String name
-double fixed
-double variable
-double fees
+Need(name, fixed, variable, fees)
+String getName()
+double getTotal()
+double getFixed()
+double getVariable()
+double getFees()
+void setFixed(double)
+void setVariable(double)
+void setFees(double)
«interface»
NeedComponent
+String getName()
+double getTotal()
+double getFixed()
+double getVariable()
+double getFees()
Bundle
-String name
-List<NeedComponent> components
-Map<NeedComponent,Integer> componentCounts
+String getName()
+double getTotal()
+double getFixed()
+double getVariable()
+double getFees()
-Map<NeedComponent,Integer> effectiveCounts()
-double sumRecursively(Set<NeedComponent>, ToDoubleFunction<NeedComponent>)


LedgerController
-LedgerRepository repo
-List<Observer> observers
+LedgerController(LedgerRepository)
+void addObserver(Observer)
+void removeObserver(Observer)
+void donate(String needName, double amount)
-void notifyObservers(Event)
NeedController
-NeedsRepository repo
-List<Observer> observers
+NeedController(NeedsRepository)
+void addObserver(Observer)
+void removeObserver(Observer)
+List<NeedComponent> listAll()
+Need createNeed(String name, double fixed, double variable, double fees)
+Bundle createBundle(String name, Map<NeedComponent,Integer> items)
+void saveAllToCsv()
-void notifyObservers(Event)
LedgerRepository
-List<Donation> donations
+LedgerRepository(CSVManager)
+void add(Donation)
+List<Donation> findByNeed(String)
+double totalForNeed(String)
NeedsRepository
-List<NeedComponent> catalog
-CSVManager csv
+NeedsRepository(CSVManager)
+void add(NeedComponent)
+List<NeedComponent> findAll()
+void exportToCsv()
+void importFromCsv()
CSVManager
+void writeData(String filename, List<String> lines, boolean overwrite)
+List<String> readData(String filename)


ConsoleView
+ConsoleView(NeedController, LedgerController)
+void update(Event e)
+void render(List<NeedComponent>)
NeedController
-NeedsRepository repo
-List<Observer> observers
+NeedController(NeedsRepository, CSVManager)
+List<NeedComponent> listAll()
+Need createNeed(String name, double fixed, double variable, double fees)
+Bundle createBundle(String name, Map<NeedComponent,Integer> items)
+void saveAllToCsv()
LedgerController
-LedgerRepository repo
-List<Observer> observers
+LedgerController(LedgerRepository)
+void removeObserver(Observer)
+void donate(String needName, double amount)
NeedsRepository
-List<NeedComponent> catalog
-CSVManager csv
+NeedsRepository(CSVManager)
+void add(NeedComponent)
+List<NeedComponent> findAll()
+void exportToCsv()
+void importFromCsv()
+void addObserver(Observer)
-void notifyObservers(Event)
+void removeObserver(Observer)
LedgerRepository
-List<Donation> donations
+void add(Donation)
+List<Donation> findByNeed(String needName)
+double totalForNeed(String needName)
+void addObserver(Observer)
-void notifyObservers(Event)
+void removeObserver(Observer)
CSVManager
+void writeData(String filename, List<String> lines, boolean overwrite)
+List<String> readData(String filename)

V2


ConsoleView
FundingController
Delegates
Uses
LedgerController
FundingService
Delegates
Observes (Updates
Uses
Observes (Updates
Uses
NeedsController
«interface»
AbstractLedgerEntry
«Observable»
LedgerRepository
Uses
Uses
Implements
manages *
depends on (Cost
LedgerEntity
«Observable»
NeedsRepository
FundingRepository
Uses
Uses
manages *
Uses
CSVManager
«interface»
NeedComponent
Implements
Implements
aggregates
Need
Bundle



- **Need Management Subsystem** – manages creation, retrieval, and status of Needs.
- **Bundle Management Subsystem** – groups related Needs using the Composite pattern.
- **Ledger Subsystem** – records donations, calculates totals, and ensures transaction integrity.
- **Funding Subsystem** – manages the definition, storage, and recording of financial income sources and received funds.
- **User Subsystem** – handles Donor, Admin, and general user operations.
- **UI / Controller Subsystem** – manages user interactions, events, and display updates.

Subsystems

1.1 Need Management

Class Need	
Responsibilities	Represents a single item of aid (description, cost, status).
Collaborators (uses)	LedgerEntity, Bundle, NeedRepository

Class NeedComponent	
Responsibilities	Common interface for Need and Bundle to support Composite operations.

Class NeedRepository	
Responsibilities	Handles loading, retrieving, and saving of Needs/Bundle data to the CSV file.
Collaborators (inheritance)	NeedsController, LedgerController

Class CSVManager	
Responsibilities	Handles the reading and writing of CSV data to the CSV file.
Collaborators (inheritance)	NeedsRepository, LedgerRepository

1.2 Bundle Management

Class Bundle	
Responsibilities	Composite that aggregates multiple NeedComponents. Calculates totals recursively.
Collaborators (uses)	NeedComponent, Need

Class NeedRepository	
Responsibilities	Handles loading, retrieving, and saving of Needs/Bundle data to the CSV file.
Collaborators (inheritance)	NeedsController, LedgerController, CSVManager

Class CSVManager	
Responsibilities	Handles the reading and writing of CSV data to the CSV file.
Collaborators (inheritance)	NeedsRepository, LedgerRepository, FundingRepository

1.3 Ledger subsystem

Class AbstractLedgerEntity	
Responsibilities	An abstraction layer for all Ledger entries to ensure all concrete entries share common methods.

Class LedgerEntity	
Responsibilities	Records all Donation entries and computes totals.
Collaborators (uses)	Need, Bundle, AbstractLedgerEntity

Class LedgerRepository	
Responsibilities	Handles loading, retrieving, and saving of LedgerEntity data into the CSV file.
Collaborators (inheritance)	LedgerController, CSVManager

Class CSVManager	
Responsibilities	Handles the reading and writing of CSV data to the CSV file.

Collaborators (inheritance)	NeedsRepository, LedgerRepository, FundingRepository

1.4 Funding Subsystem

Class FundingSource	
Responsibilities	Represents a single definable source of income (name, amount per unit). Tracks the base value for one unit of funding.

Class FundingRepository	
Responsibilities	Handles loading, retrieving, and saving of FundingSource data into the CSV file. Ensures data integrity and access control for funding sources.
Collaborators (inheritance)	FundingController

Class CSVManager	
Responsibilities	Handles the reading and writing of CSV data to the CSV file.
Collaborators (inheritance)	NeedsRepository, LedgerRepository, FundingRepository

~~1.5 User Subsystem~~

Class User	
Responsibilities	Base class with shared info for all our users.
Collaborators (uses)	ConsoleView

Class Admin	
Responsibilities	Extend user management of system data.
Collaborators	NeedRepository, LedgerRepository

1.6 UI/Controller Subsystem

Class FundGoodDeedsApp	
Responsibilities	Entry Point, initialize UI and subsystems.
Collaborators (uses)	NeedController, LedgerController, ConsoleView

Class NeedController	
Responsibilities	Handles add/update/view Need and Bundle actions.
Collaborators	NeedRepository, NeedComponent, ConsoleView

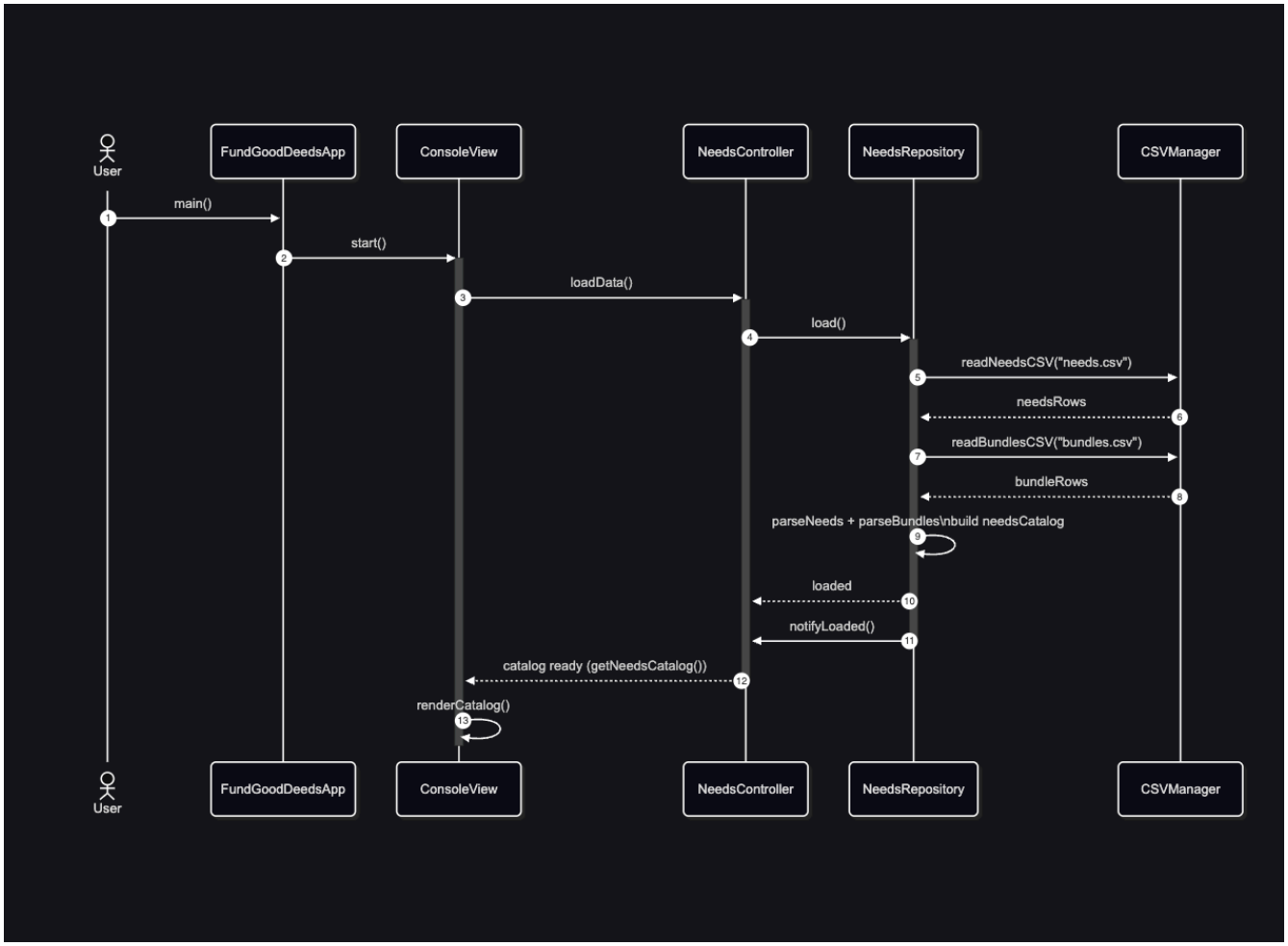
Class LedgerController	
Responsibilities	Handles add/update/view LedgerEntity actions.
Collaborators	LedgerRepository, ConsoleView

Class FundingController	
Responsibilities	Handles add/update/view FundingSource actions.
Collaborators (uses)	FundingRepository, LedgerController, ConsoleView

Class ConsoleView	
Responsibilities	Observes data model changes to refresh the interface.
Collaborators	NeedController, LedgerController, FundingController

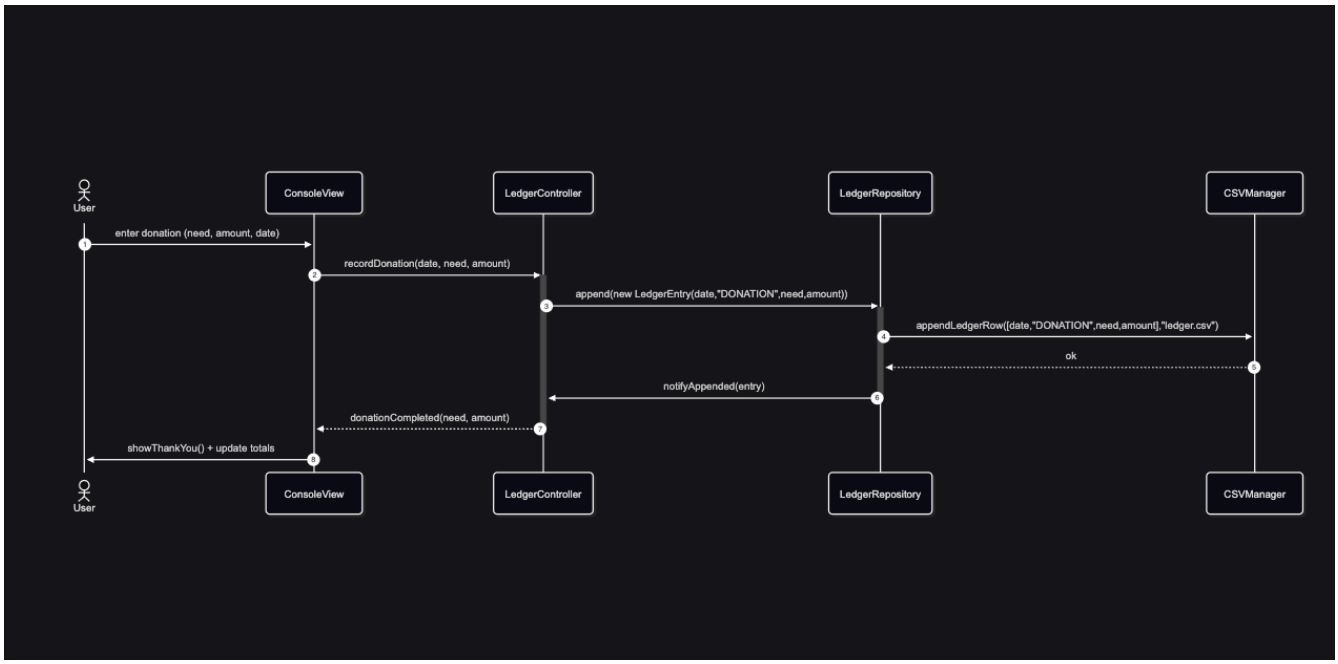
Sequence Diagrams

1.7 Description of labeled Sequence diagram #1 and (what feature / operation / scenario the diagram shows).



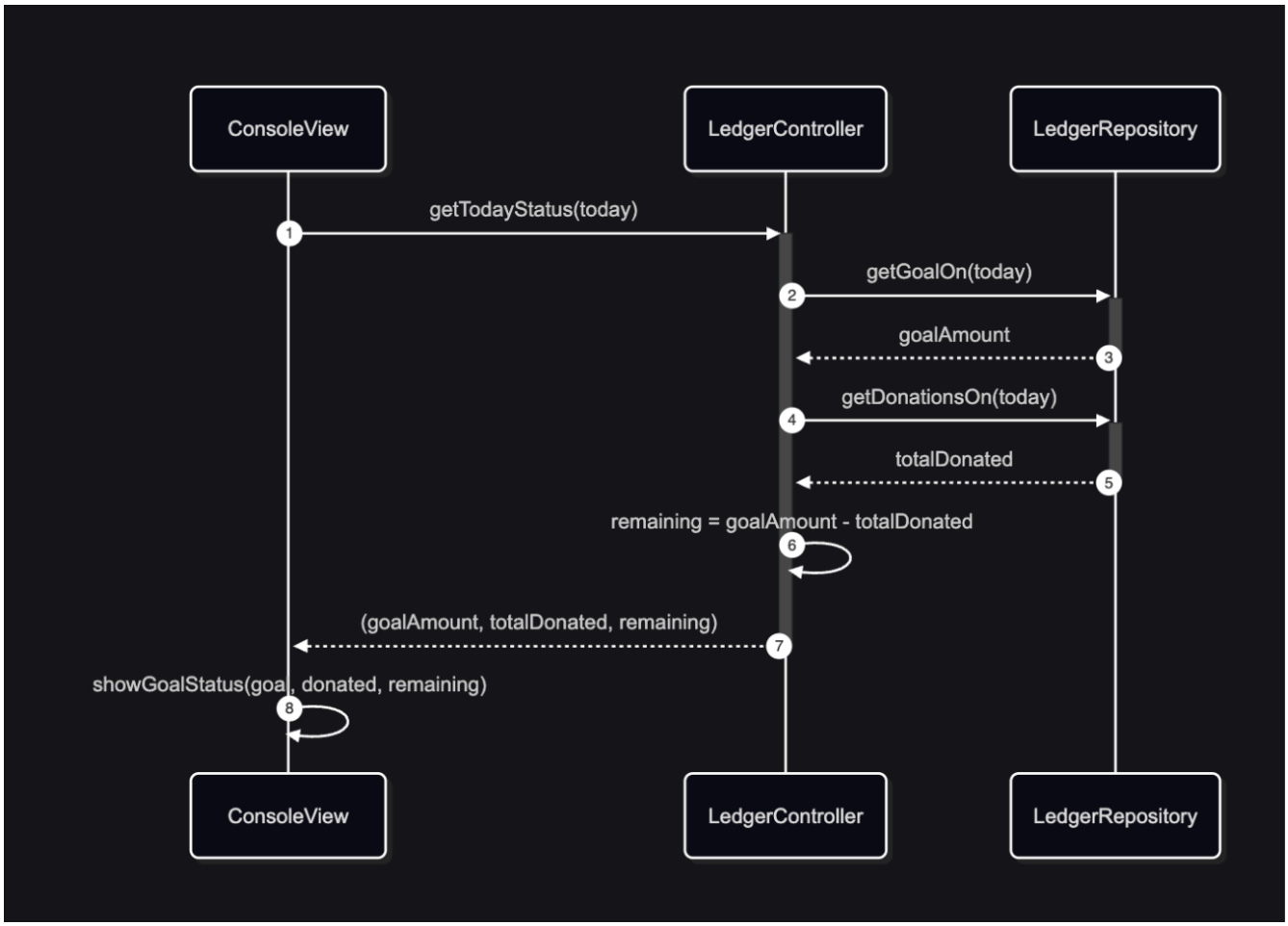
On startup the user launches the app; ConsoleView.start() asks NeedsController.loadData(). The **controller** delegates to NeedsRepository.load(), which uses CSVManager to load needs and bundles, converts rows into objects, and refreshes the in-memory catalog. When loading completes, the **controller** returns control and the view renders the catalog.

1.8 Description of labeled Sequence diagram #2 (what feature / operation / scenario the diagram shows).



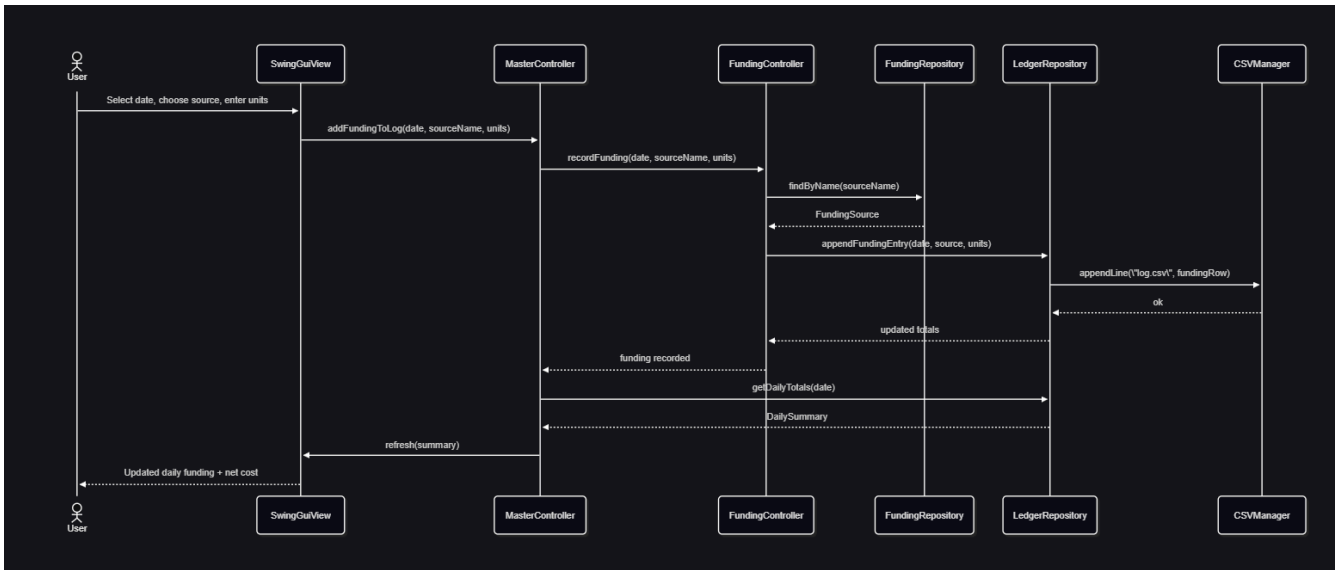
When the user enters a donation, ConsoleView calls LedgerController.recordDonation(date, need, amount). The **controller** constructs a LedgerEntity(type="DONATION") and appends it through LedgerRepository, which persists the row with CSVManager. Once appended, the **controller** notifies the view; the UI renders a thank-you and updated totals.

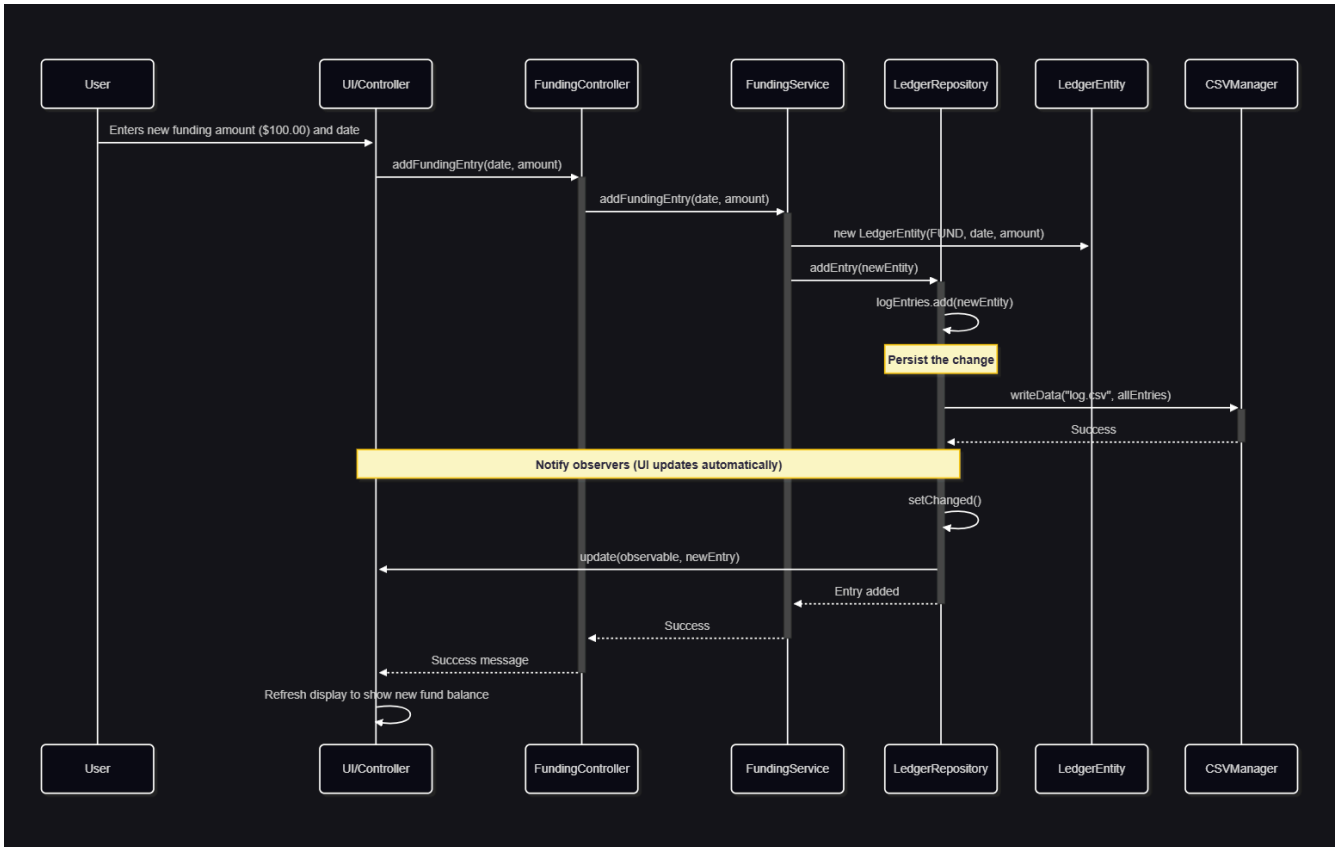
1.9 Description of labeled Sequence diagram #3 (what feature / operation / scenario the diagram shows).

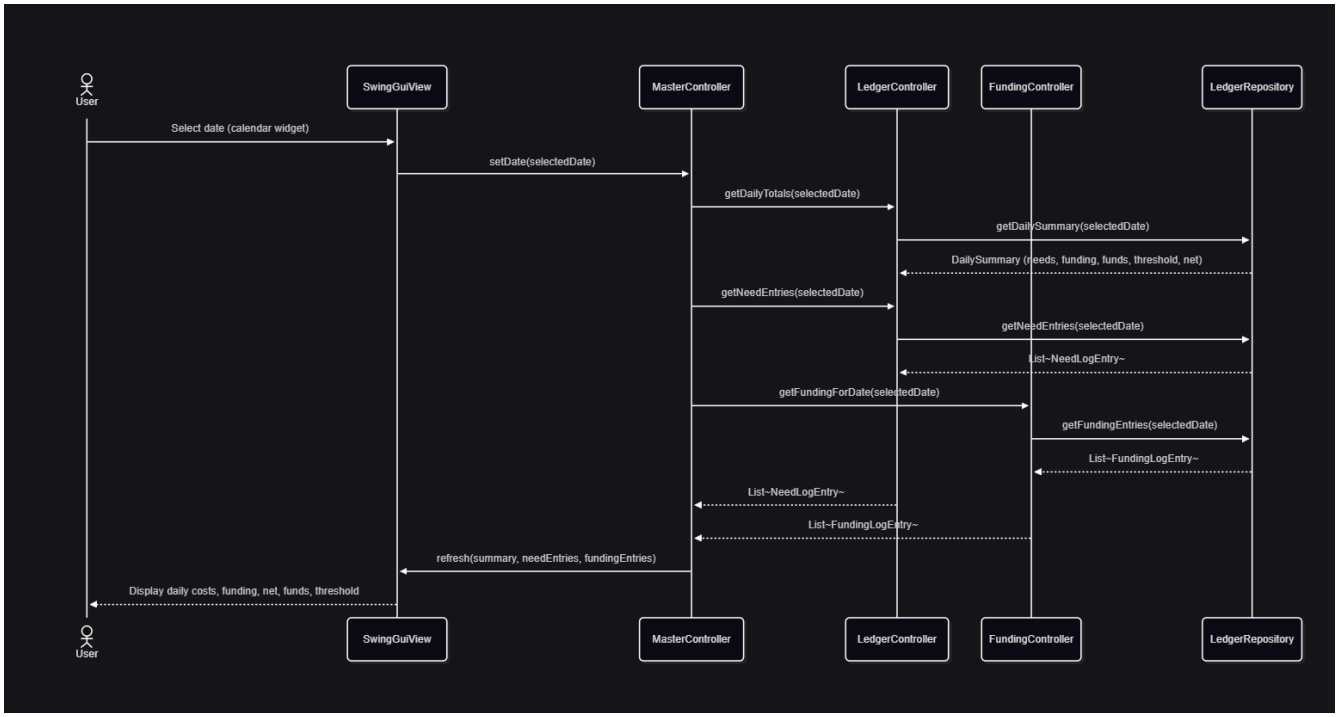


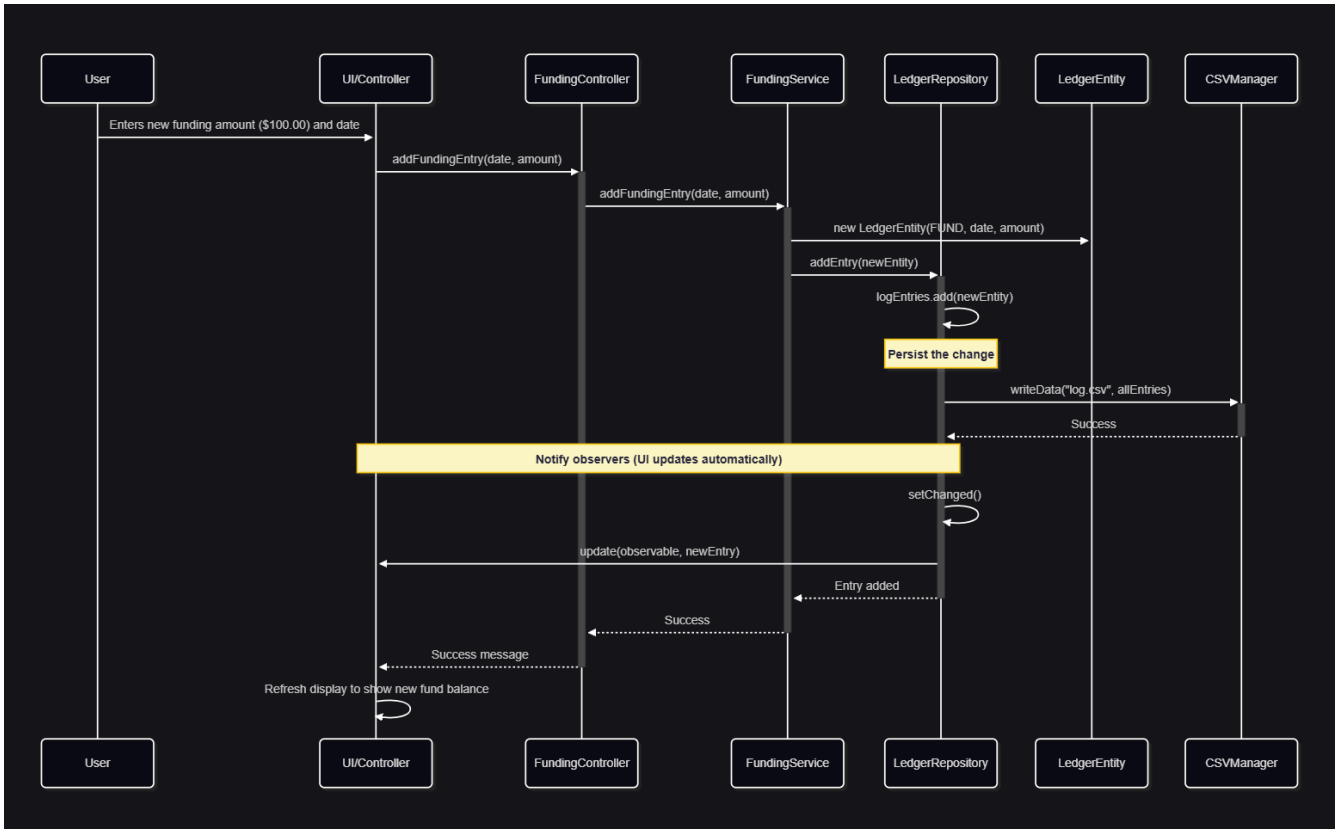
When the user enters a donation, ConsoleView calls LedgerController.recordDonation(date, need, amount). The **controller** constructs a LedgerEntity(type="DONATION") and appends it through LedgerRepository, which persists the row with CSVManager. Once appended, the **controller** notifies the view; the UI renders a thank-you and updated totals.

V2


User
SwingGuiView
MasterController
FundingController
FundingRepository
LedgerRepository
CSVManager
Select date, choose source, enter units
addFundingToLog(date, sourceName, units)
recordFunding(date, sourceName, units)
findByName(sourceName)
FundingSource
appendFundingEntry(date, source, units)
appendLine(\"log.csv\", fundingRow)
ok
updated totals
funding recorded
getDailyTotals(date)
DailySummary
refresh(summary)
Updated daily funding + net cost


User
UI/Controller
FundingController
FundingService
LedgerRepository
LedgerEntity
CSVManager
Enters new funding amount ($100.00) and date
addFundingEntry(date, amount)
addFundingEntry(date, amount)
new LedgerEntity(FUND, date, amount)
addEntry(newEntity)
logEntries.add(newEntity)
Persist the change
writeData("log.csv", allEntries)
Success
Notify observers (UI updates automatically)
setChanged()
update(observable, newEntry)
Entry added
Success
Success message
Refresh display to show new fund balance


User
SwingGuiView
MasterController
LedgerController
FundingController
LedgerRepository
Select date (calendar widget)
setDate(selectedDate)
getDailyTotals(selectedDate)
getDailySummary(selectedDate)
DailySummary (needs, funding, funds, threshold, net)
getNeedEntries(selectedDate)
getNeedEntries(selectedDate)
List~NeedLogEntry~
getFundingForDate(selectedDate)
getFundingEntries(selectedDate)
List~FundingLogEntry~
List~NeedLogEntry~
List~FundingLogEntry~
refresh(summary, needEntries, fundingEntries)
Display daily costs, funding, net, funds, threshold
User
SwingGuiView
MasterController
LedgerController
FundingController
LedgerRepository


User
UI/Controller
FundingController
FundingService
LedgerRepository
LedgerEntity
CSVManager
Enters new funding amount ($100.00) and date
addFundingEntry(date, amount)
addFundingEntry(date, amount)
new LedgerEntity(FUND, date, amount)
addEntry(newEntity)
logEntries.add(newEntity)
Persist the change
writeData("log.csv", allEntries)
Success
Notify observers (UI updates automatically)
setChanged()
update(observable, newEntry)
Entry added
Success
Success message
Refresh display to show new fund balance
User
UI/Controller
FundingController
FundingService
LedgerRepository
LedgerEntity
CSVManager

Pattern Usage

1.10 Observer Pattern

Observer Pattern	
Observer(s)	ConsoleView(view layer), ReportView (optional)
Observable(s)	NeedRepository, Ledger
Notification Method	notifyObservers(), update()
Event Sources	Need status changes, Donation recorded

1.11 Composite Pattern

Composite Pattern	
Component	NeedComponent
Leaf	Need
Composite	Bundle
Client(s)	NeedController, FundGoodDeedsApp

1.12 MVC

MVC	
Model	Need, Bundle, LedgerEntity, FundingSource, LedgerRepository, NeedRepository, FundingRepository
View	ConsoleView, FundGoodDeedsApp display components

Controller	NeedController, LedgerController, FundingController
Glue/Contracts	Simple view models, using NeedComponent interface

1.13 Dependency Injection / Inversion (DI/IoC)

DI / DIP	
Abstractions	NeedComponent, (optionally ILedger, IRepository<T>)
Concrete Implementations	Need, Bundle, LedgerEntity, NeedRepository
Composition Root	FundGoodDeedsApp (wires controllers, repos, observers)
Consumers	Controllers, views using interfaces rather than concretes

1.14 Repository Pattern

Repository Pattern	
Repository	NeedRepository, LedgerRepository, FundingRepository (in-memory R1; DB later)
Aggregate Root	NeedComponent, LedgerEntity, FundingSource
Clients	NeedController, Admin workflows

System Rationale

The system is organized using the Model–View–Controller (MVC) architecture, the Composite pattern, the Observer pattern, the Repository pattern, and Dependency Injection, with clear separation of responsibilities among classes and concurrent updates across various interfaces.

Benefits, a huge separation of concerns. The Model layer includes classes such as NeedComponent, BasicNeed, Bundle, NeedsRepository, LedgerEntity, and LedgerRepository, which manage all data and business logic. The Composite relationship between Bundle and NeedComponent allows bundles to contain both basic needs and other bundles, providing a flexible way to represent complex financial structures without duplicating logic. Repositories isolate file storage and retrieval, allowing other components to focus on logic and presentation. The Controllers (NeedsController and LedgerController) coordinate data flow and enforce rules between the Model and the user-facing View. Importantly, the NeedsController manages the creation and validation of BasicNeed and Bundle objects.

Drawbacks, the MVC architecture requires a controller to be made for each view. In this case, if we add another logging element to the system, we would need to build a controller for it. The repository classes must contain lots of data validation from the CSV files, as the logs are the main source of truth. If they contain errors, then the system fails. The main drawback is that this design introduces more classes and relationships, which increases complexity and coordination effort.

10/10/2025 – Component vs. Composite

Chose Composite (NeedComponent → Need/Bundle) so totals and fulfillment logic work identically for single needs and groups. This avoids duplicate code and simplifies UI/controllers.

10/12/2025 – Central Ledger

Introduced LedgerEntity as the single source of truth for donations/transactions. Keeps financial logic cohesive and auditable. This hopes to prevent scattering donation math across controllers.

10/13/2025 – MVC Separation

Refactored to MVC where controllers handle orchestration, views only render, models encapsulate rules. This reduced coupling and made unit tests for business rules straightforward.

10/14/2025 – Observer for Live Updates

Added Observer so UI auto-refreshes on Need/LedgerEntity changes. This removed manual refresh calls from controllers and clarified responsibilities.

10/15/2025 – Repository Abstraction

Adopted Repository (NeedRepository) to isolate storage concerns. R1 uses in-memory lists; R2 can switch to DB without controller changes.

10/16/2025 – DI at Composition Root

Applied Dependency Inversion by wiring interfaces in FundGoodDeedsApp. Enables mocking NeedRepository/LedgerEntity in tests and future strategy swaps.

10/17/2025 – Donation Allocation Strategies (Future-proofing)

Defined a Strategy interface for bundle allocation. R1 defaults to Equal Split; left hooks for Priority/Single-Need strategies required by stakeholders.

10/18/2025 – Rejected Alternatives

Rejected controller-direct DB calls (tight coupling), and view-driven business logic (violates MVC). Also rejected hard-coded allocation rules (blocks future change).

10/19/2025 – Subsystems & Contracts

Locked subsystem boundaries (Need, Bundle, Ledger, User, UI/Controllers) and interfaces (NeedComponent).

10/20/2025 – CSV Schema v1. Finalized headers for `needs.csv`, `funding.csv`, and `log.csv`; locked row order and delimiter rules to stabilize parsing.

10/22/2025 – Error Handling Pass. Added guard-rails for malformed rows; repository catches/filters bad records and surfaces user-friendly messages to the view.

10/23/2025 – Controller Split. Moved view-triggered logic from repositories into `NeedsController` / `LedgerController` to keep MVC boundaries crisp.

10/24/2025 – Metrics Hook. Added `getDonationsOn(date)` / `getGoalOn(date)` to support daily goal status without duplicating math in the controller.

11/03/2025 – R1 Diagram. Locked Class + 3 Sequence diagrams to match shipped code for submission.