FlowMate

FINAL PRESENTATION

Team 5

Della Corte Alessio
a.dellacorte23@studenti.unisa.it

Pecoraro Sara
s.pecoraro19@studenti.unisa.it

Sabatino Ester
e.sabatino14@studenti.unisa.it

Siddiqa Ayesha
a.siddiqa1@studenti.unisa.it

Third Sprint – Review Summary

What We Delivered

We implemented advanced automation features, including external program execution, time-based triggers (day, month, and specific date), and a complete counter management system with comparison and arithmetic actions. Rule repetition logic and program-based triggers were also added.

Original Commitments

Our objective was to complete User Stories US14–US17 and US20–US25, focusing on advanced trigger types, counter-based logic, rule repetition, and integration with external programs.

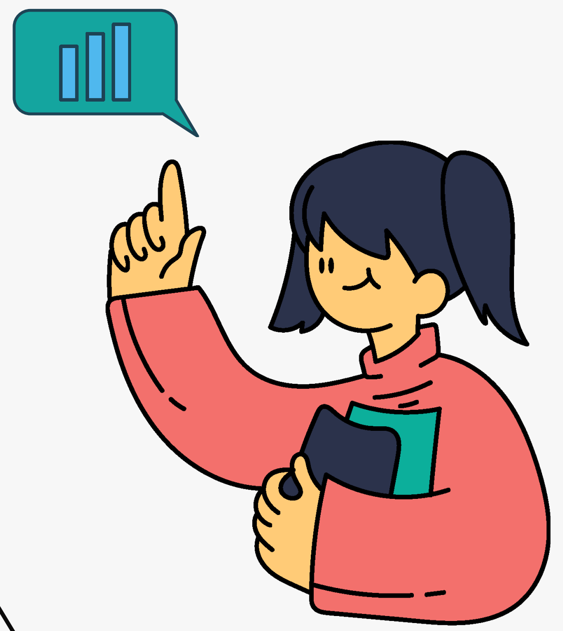
Velocity Achieved

We achieved 77 Story Points, covering almost all planned User Stories.

Quality Assurance

We ensured reliability and correctness by implementing comprehensive unit tests for counters, repetition logic, and external program execution, validating both normal behavior and edge cases.

What Went Well



Clean commenting

The team focused on meaningful code commenting, targeting complex logic rather than obvious operations. This ensured codebase maintainability and full compliance with our Definition of Done (DoD).

User Tasks Division

While one team member developed the logic and verified it through JUnit tests, another designed the GUI. This ensured that by the time the Controller was ready for integration, the underlying Trigger/Action logic was already tested and stable.

Client-Driven Enhancements

Following the last Sprint Review, we integrated new requirements directly requested by the client. Specifically, we enhanced the File Exists and File Exceeds triggers with time-based activation logic, going beyond the initial User Epics.

WHAT DIDN’T WORK

CONFUSING INTERPHACES

Initially, we prioritized functional logic over a unified Design System, resulting in some visual inconsistencies across the GUI modules.

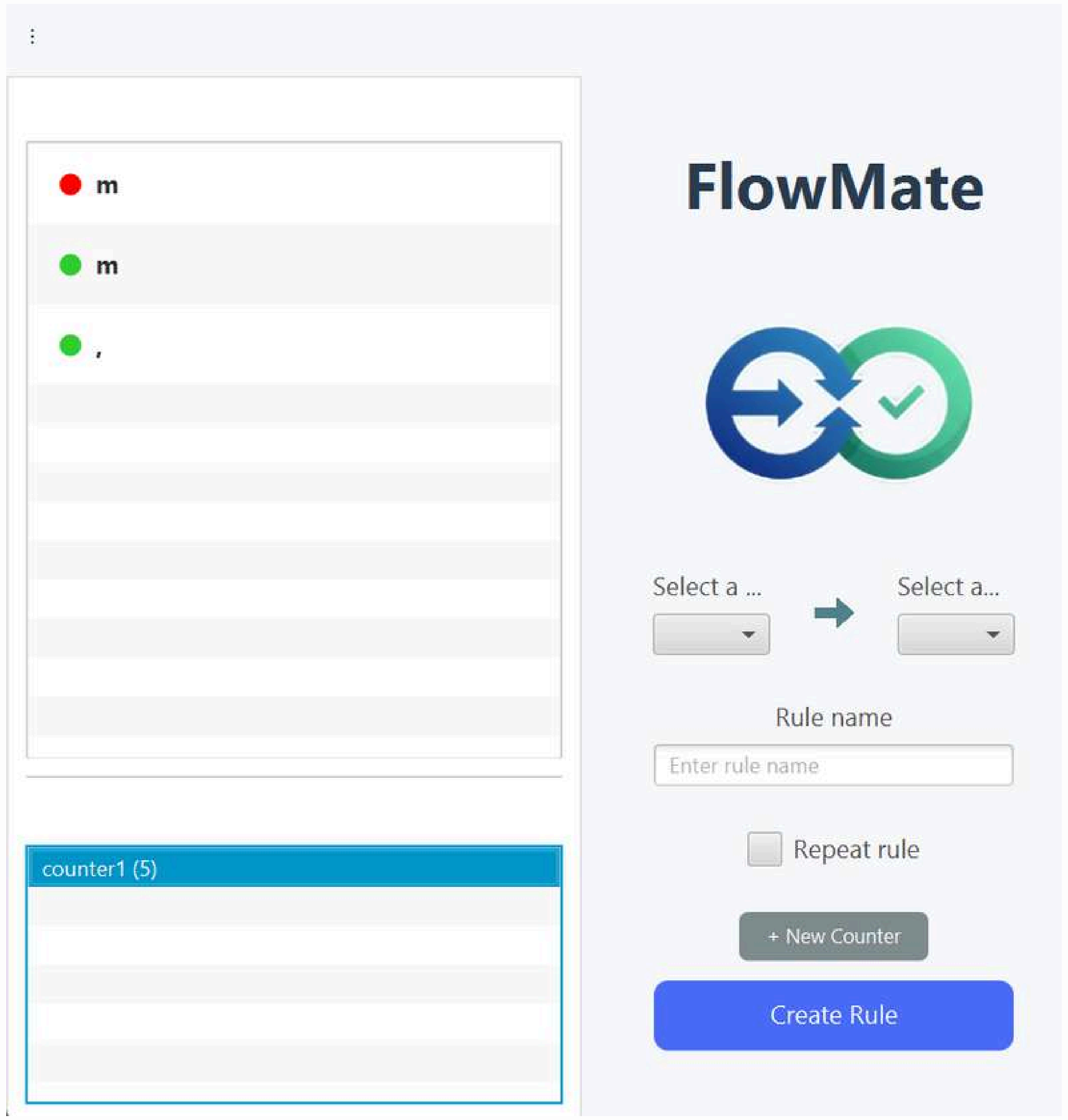
THE IMPORTANCE OF INTEGRATING

Some User Stories, like the File Exceeds Trigger, are code-complete and tested but haven't reached the 'Done' status due to missing integration in the main flow. This lack of coordination led to accidental regressions.

Sprint Retrospective

START	MORE OF	LESS OF	KEEP DOING	STOP
	Communicating bugs or errors that are met during the testing of the application.	Doing the tasks without deciding who has to integrate them in the main flow of the application.	Commenting the code wherever it is needed.	Editing a class without communicating with the rest of the team, in order to be able to push/pull whenever a team member wants.

Counters

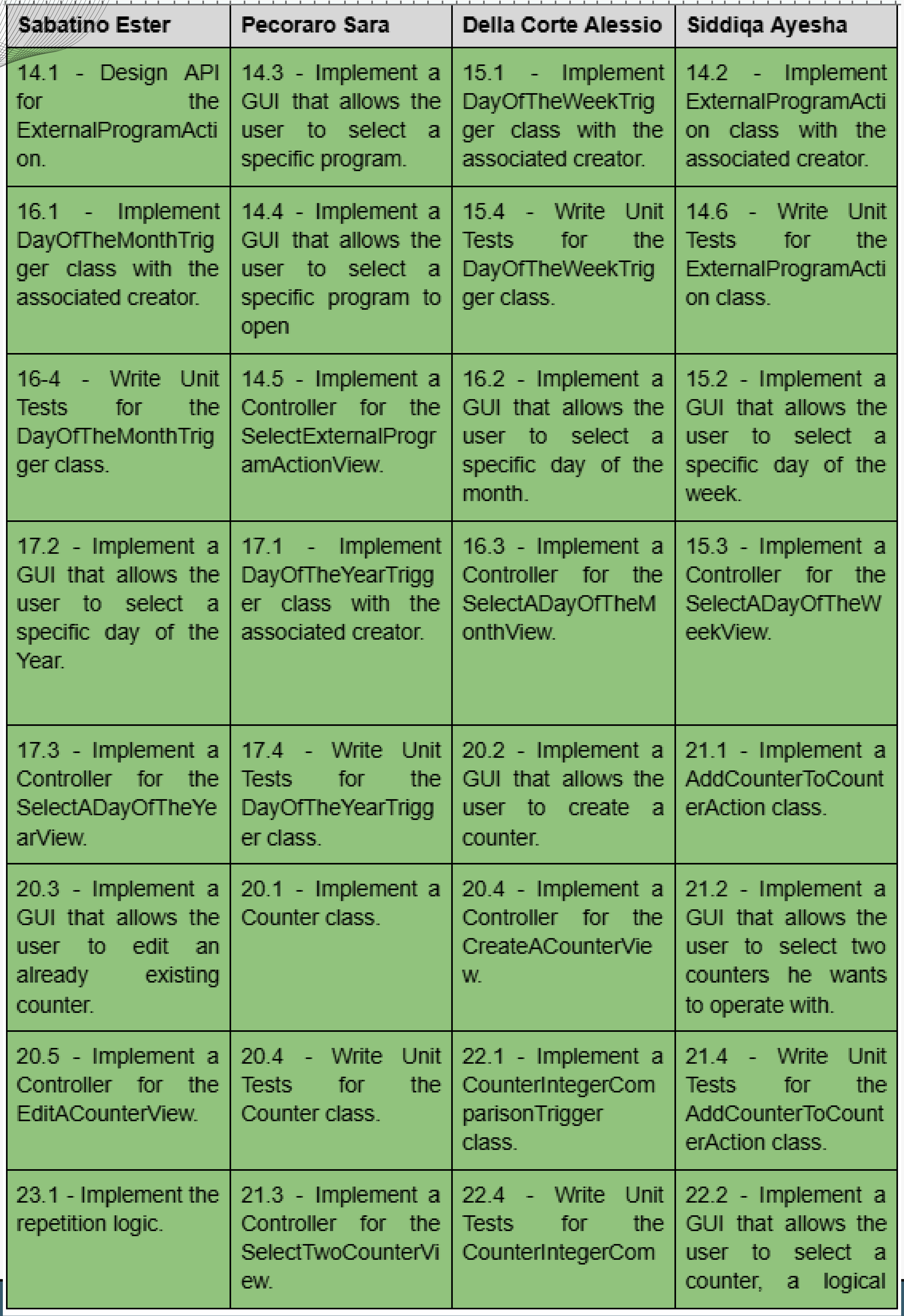
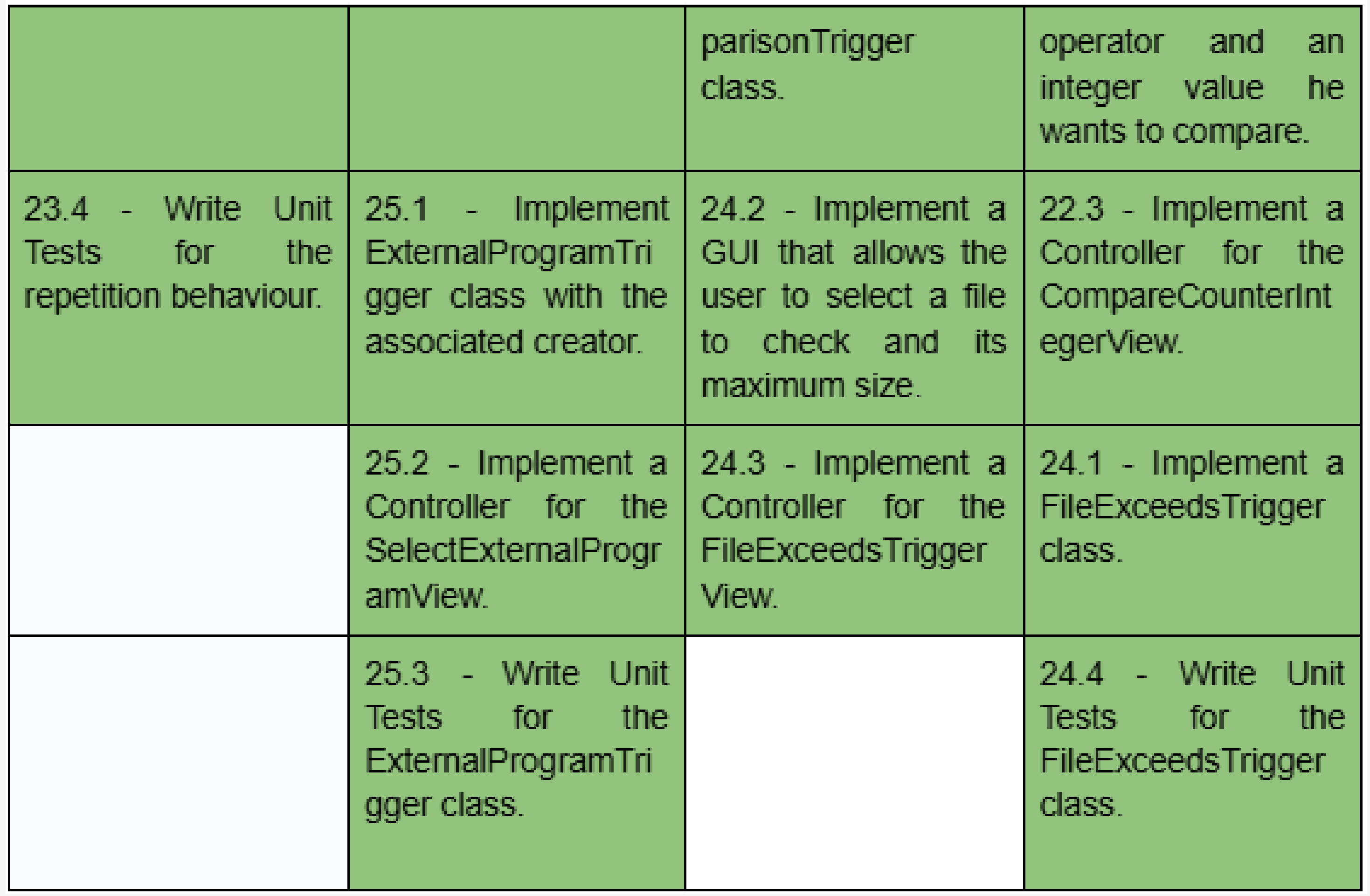


The primary feature developed during this final sprint was the introduction of Counters, which function as global integer variables within the application. Given the project's remaining time, we prioritized just a reduced number of User Stories.

We implemented a specific trigger and a dedicated action. The Counter Comparison Trigger enables users to build logical equations while the Add Counter to Counter Action allows to change the value of one counter according to another ones'.

To ensure a complete user experience, we implemented almost all the same functionalities implemented for the rules: the user can see the list of counters created within the Sidebar, but he can also edit them by double-clicking on any of them.

Updated Sprint Backlog

Sabatino Ester	Pecoraro Sara	Della Corte Alessio	Siddiqa Ayesha
14.1 - Design API for the ExternalProgramAction.	14.3 - Implement a GUI that allows the user to select a specific program.	15.1 - Implement DayOfTheWeekTrigger class with the associated creator.	14.2 - Implement ExternalProgramAction class with the associated creator.
16.1 - Implement DayOfTheMonthTrigger class with the associated creator.	14.4 - Implement a GUI that allows the user to select a specific program to open	15.4 - Write Unit Tests for the DayOfTheWeekTrigger class.	14.6 - Write Unit Tests for the ExternalProgramAction class.
16-4 - Write Unit Tests for the DayOfTheMonthTrigger class.	14.5 - Implement a Controller for the SelectExternalProgramActionView.	16.2 - Implement a GUI that allows the user to select a specific day of the month.	15.2 - Implement a GUI that allows the user to select a specific day of the week.
17.2 - Implement a GUI that allows the user to select a specific day of the Year.	17.1 - Implement DayOfTheYearTrigger class with the associated creator.	16.3 - Implement a Controller for the SelectADayOfTheMonthView.	15.3 - Implement a Controller for the SelectADayOfTheWeekView.
17.3 - Implement a Controller for the SelectADayOfTheYearView.	17.4 - Write Unit Tests for the DayOfTheYearTrigger class.	20.2 - Implement a GUI that allows the user to create a counter.	21.1 - Implement a AddCounterToCounterAction class.
20.3 - Implement a GUI that allows the user to edit an already existing counter.	20.1 - Implement a Counter class.	20.4 - Implement a Controller for the CreateACounterView.	21.2 - Implement a GUI that allows the user to select two counters he wants to operate with.
20.5 - Implement a Controller for the EditACounterView.	20.4 - Write Unit Tests for the Counter class.	22.1 - Implement a CounterIntegerComparisonTrigger class.	21.4 - Write Unit Tests for the AddCounterToCounterAction class.
23.1 - Implement the repetition logic.	21.3 - Implement a Controller for the SelectTwoCounterView.	22.4 - Write Unit Tests for the CounterIntegerComparisonTrigger class.	22.2 - Implement a GUI that allows the user to select a counter, a logical operator and an integer value he wants to compare.
23.4 - Write Unit Tests for the repetition behaviour.	25.1 - Implement ExternalProgramTrigger class with the associated creator.	24.2 - Implement a GUI that allows the user to select a file to check and its maximum size.	22.3 - Implement a Controller for the CompareCounterIntegerView.
	25.2 - Implement a Controller for the SelectExternalProgramView.	24.3 - Implement a Controller for the FileExceedsTriggerView.	24.1 - Implement a FileExceedsTrigger class.
	25.3 - Write Unit Tests for the ExternalProgramTrigger class.		24.4 - Write Unit Tests for the FileExceedsTrigger class.

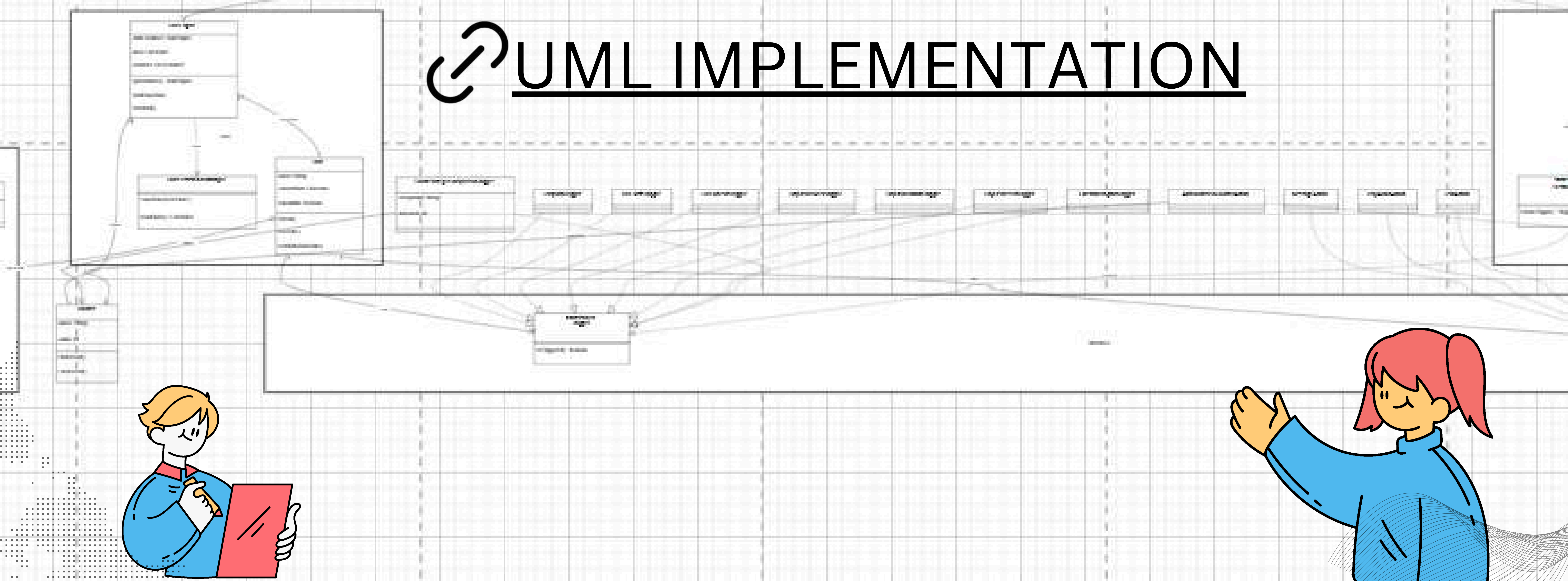
COLOUR LEGEND

🟩 COMPLETED	🟨 CARRIED OVER	🟥 NOT STARTED YET

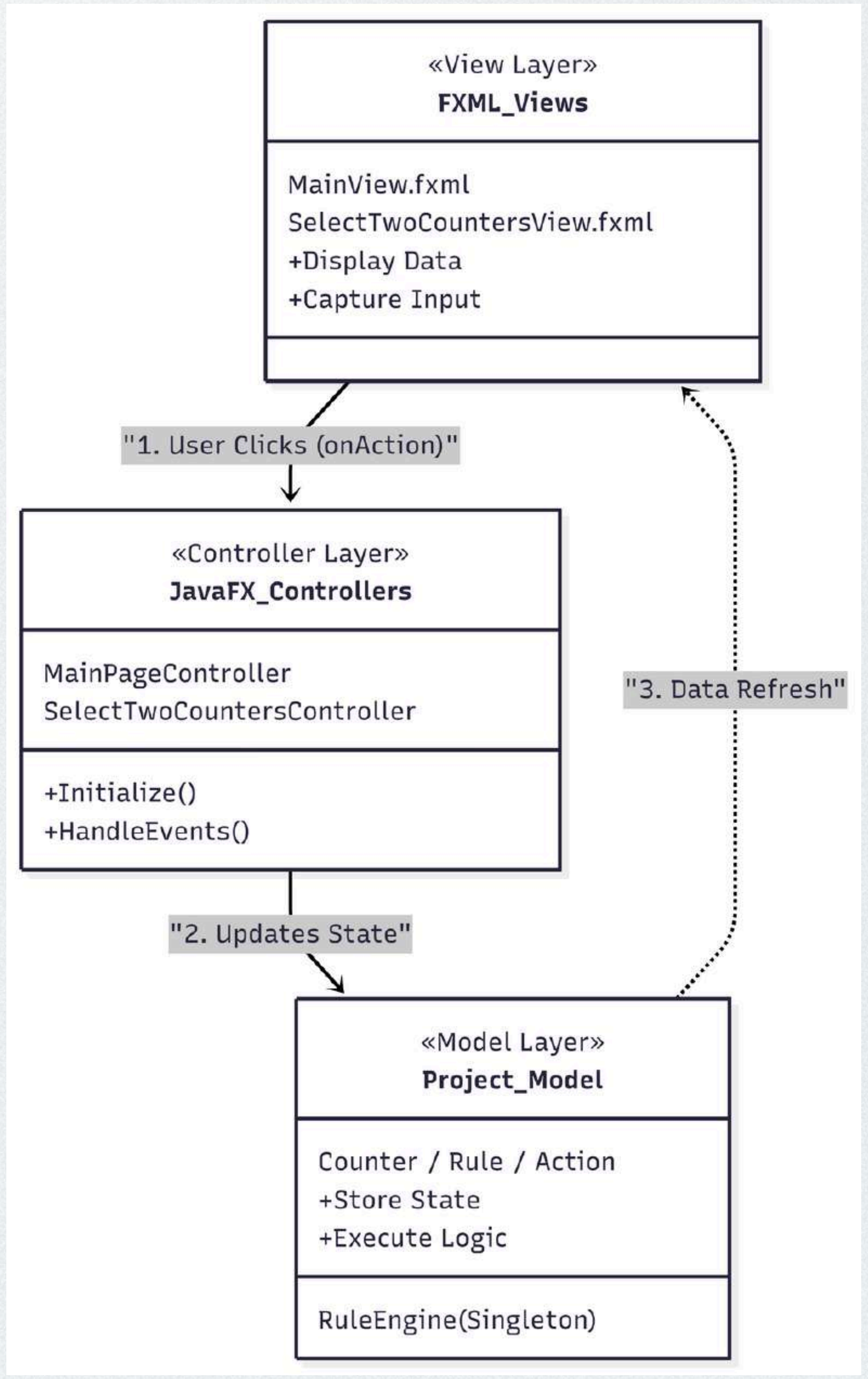
LET'S TAKE A LOOK AT THE FINAL REVIEW OF OUR PROJECT

Implementation Achieved

UML IMPLEMENTATION



The MVC and its application in FlowMate

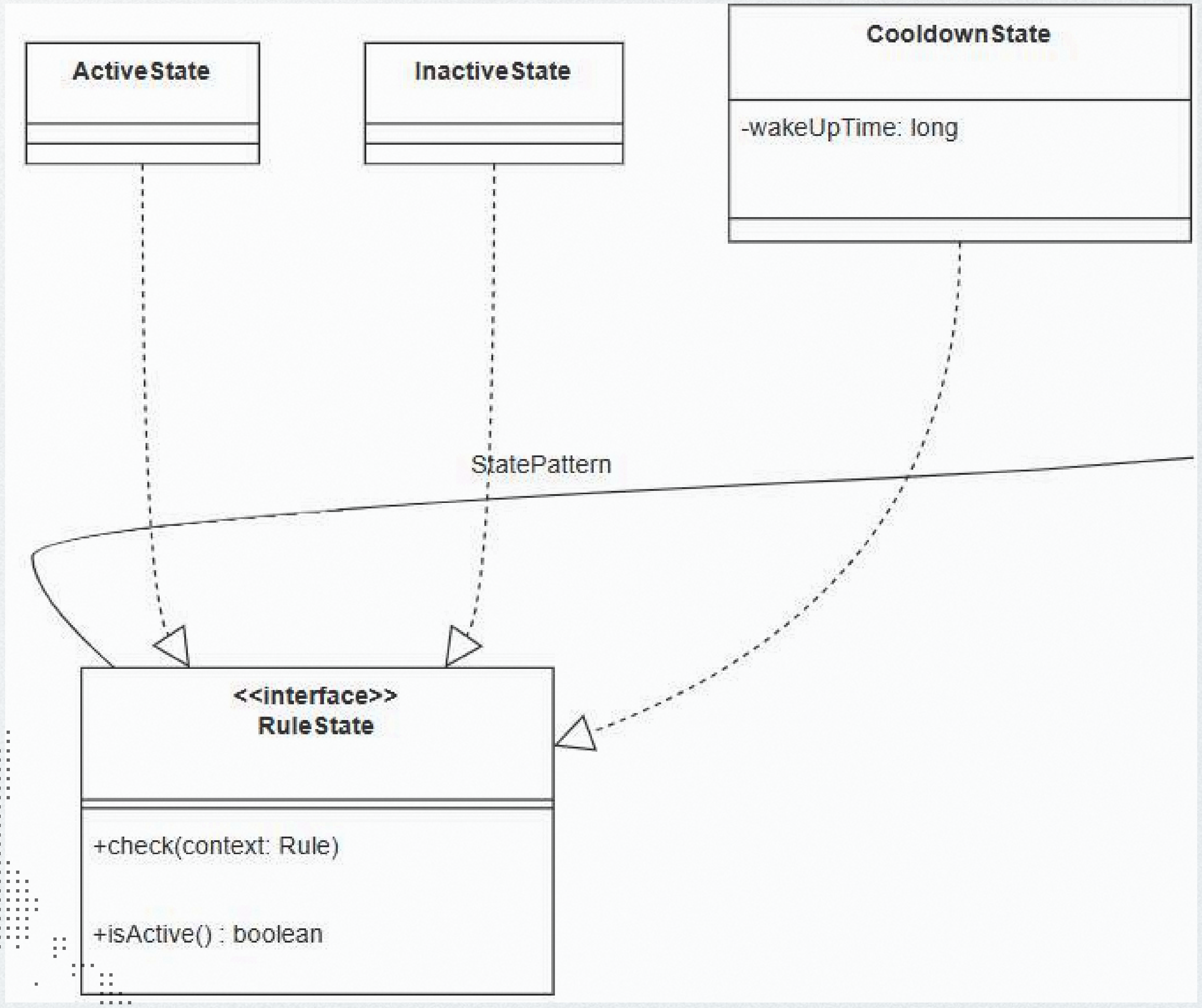


The Model → RuleEngine, Rule, Counter
We use a centralized RuleEngine to store all Rule and Counter objects. Whether a counter is modified by a Trigger or an Action, the Model ensures data consistency across the entire app.

The View → FXML Modularization
Every screen (e.g., the Main Page or the "Select Counter" popup) has its own .fxml file. This allows us to modify the UI design without risking breaking the backend Java logic.

The Controller → The Bridge
Controllers like MainPageController act as the glue. They inject dependencies (like passing Action objects to popups) and use the Factory Pattern to create complex objects based on simple user selections.

The State Pattern and its application in FlowMate



We needed a way to manage the complex lifecycle of a Rule without polluting the Rule class with messy boolean flags or nested if-else statements. We set 3 states:

ON

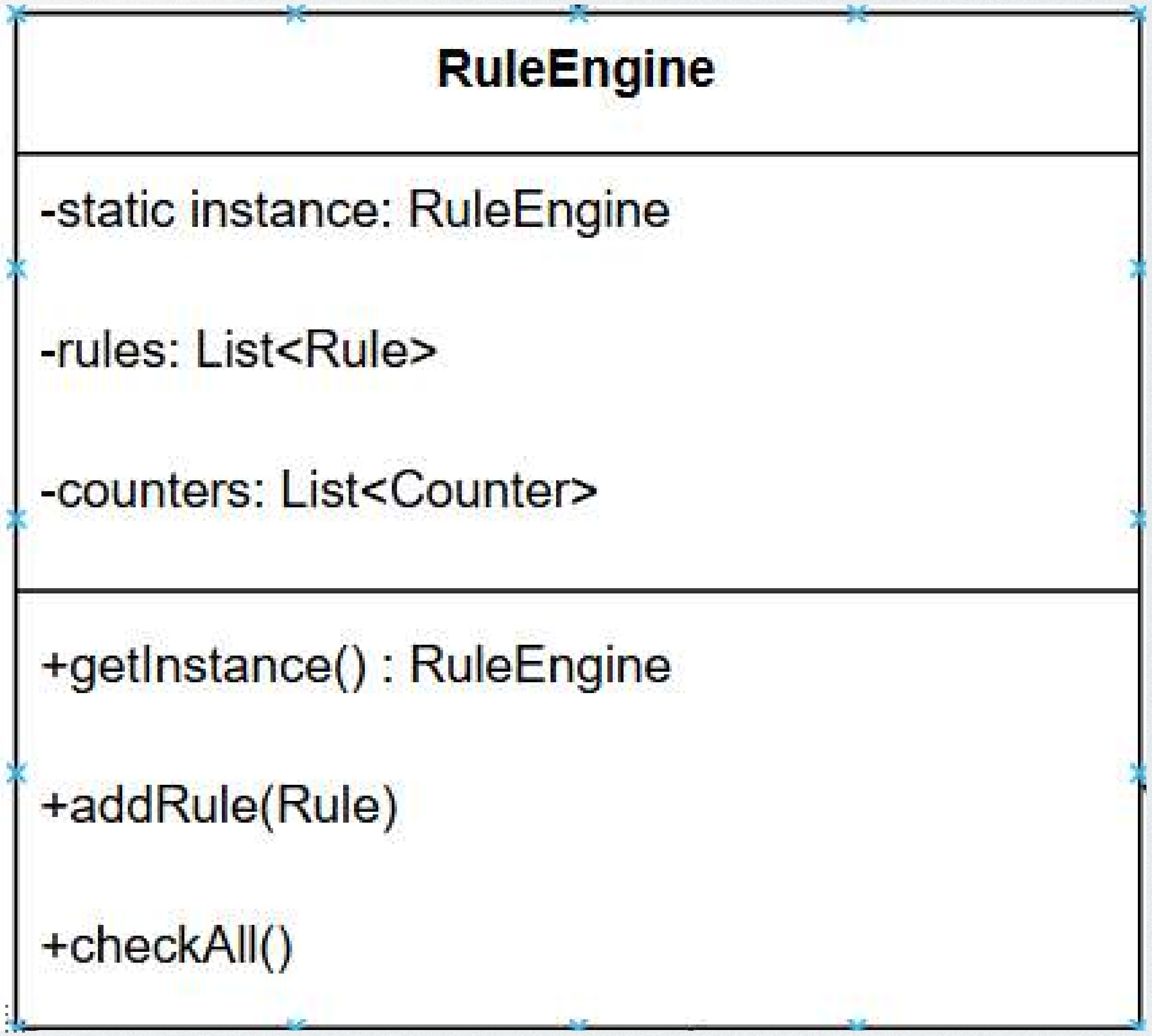
Active State: The rule is "alive." It actively monitors triggers and executes actions. Crucially, it manages the logic to transition itself into Cooldown or Inactive.

OFF

Inactive State: The rule is effectively off. It acts as a "Null Object," consuming zero processing power.

Cooldown State: The rule is temporarily "sleeping." It isolates the timing logic, waiting for the defined duration to pass before automatically waking the rule back to Active.

The Singleton Pattern and its application in FlowMate

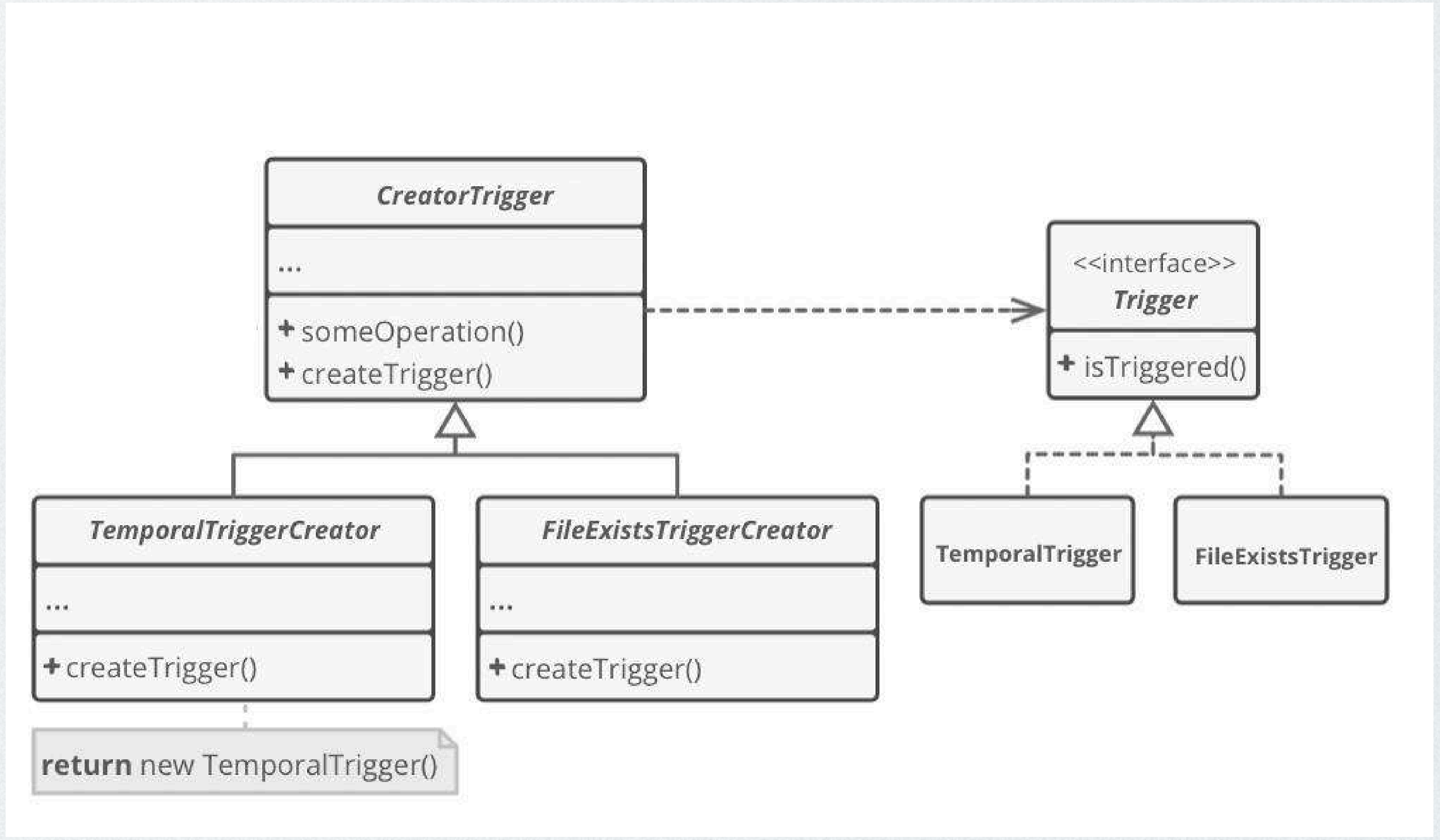


The RuleEngine applies the Singleton pattern to ensure that only one instance of the rule management engine exists within the application.

The pattern is implemented by using a private constructor and a static instance reference, accessed through the getInstance() method. This method is synchronized to guarantee thread safety and prevent multiple instances from being created in concurrent environments.

As a result, the scheduler and the rule evaluation logic are initialized only once, ensuring consistent and centralized control of the system.

The Factory Method Pattern and its application in FlowMate



Creator class → CreatorTrigger

The Creator class declares the factory method that returns new product objects.

Product → Trigger

The Product declares the interface, which is common to all objects that can be produced by the creator and its subclasses.

Concrete Creators → ex. TemporalTriggerCreator

Concrete Creators override the base factory method so it returns a different type of product.

Concrete Products → ex. TemporalTrigger

Concrete Products are different implementations of the product interface.

JUnit Testing

Tools

The reliability of the application was ensured using JUnit 5 (Jupiter) as the primary testing framework. This tool allowed the team to implement a suite of automated Unit Tests to validate the core logic of Triggers, Actions, and State transitions.

Testing Criteria

Our testing strategy focused on validating the core logic of every functional class to cover all possible method outcomes. We deliberately excluded IDE-generated boilerplate, such as getters and setters, as well as the Controller classes, which were verified through manual graphical testing. This targeted approach allowed us to ensure high functional coverage by concentrating exclusively on the application's most critical components.

Examples

For example, we conducted an exhaustive analysis of the Counter Integer Comparison Trigger. Our test suite validates all logical operators,By simulating cases where the comparison is satisfied, not satisfied, or at the exact boundary, we guaranteed that the trigger evaluates the counter's state with absolute precision.

Handling Real-World Edge Cases

File System Resilience

MoveFileAction
Handles non-existent source files gracefully.

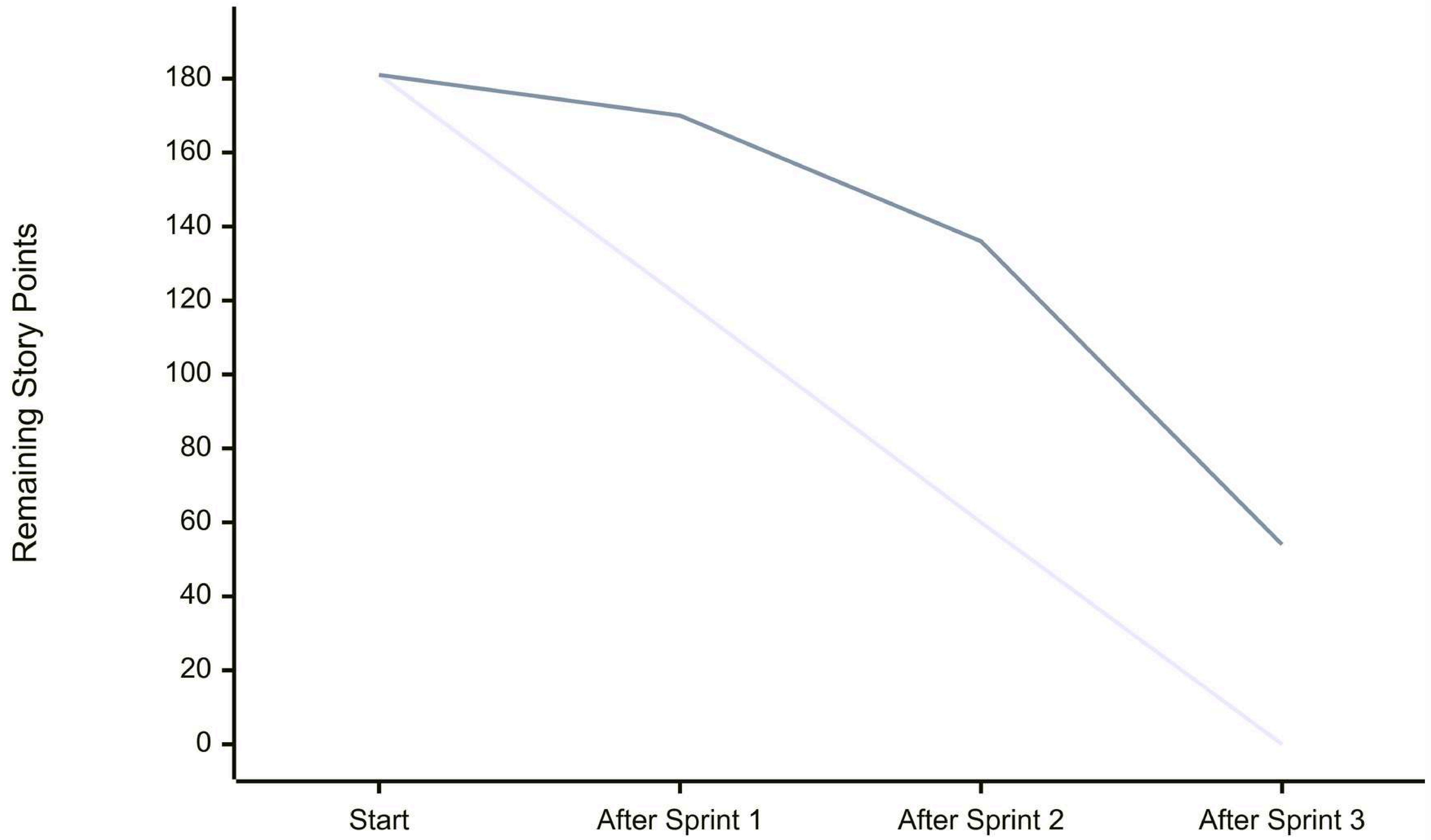
DeleteFileAction
Checks file existence before attempting deletion to avoid exceptions

Input Validation

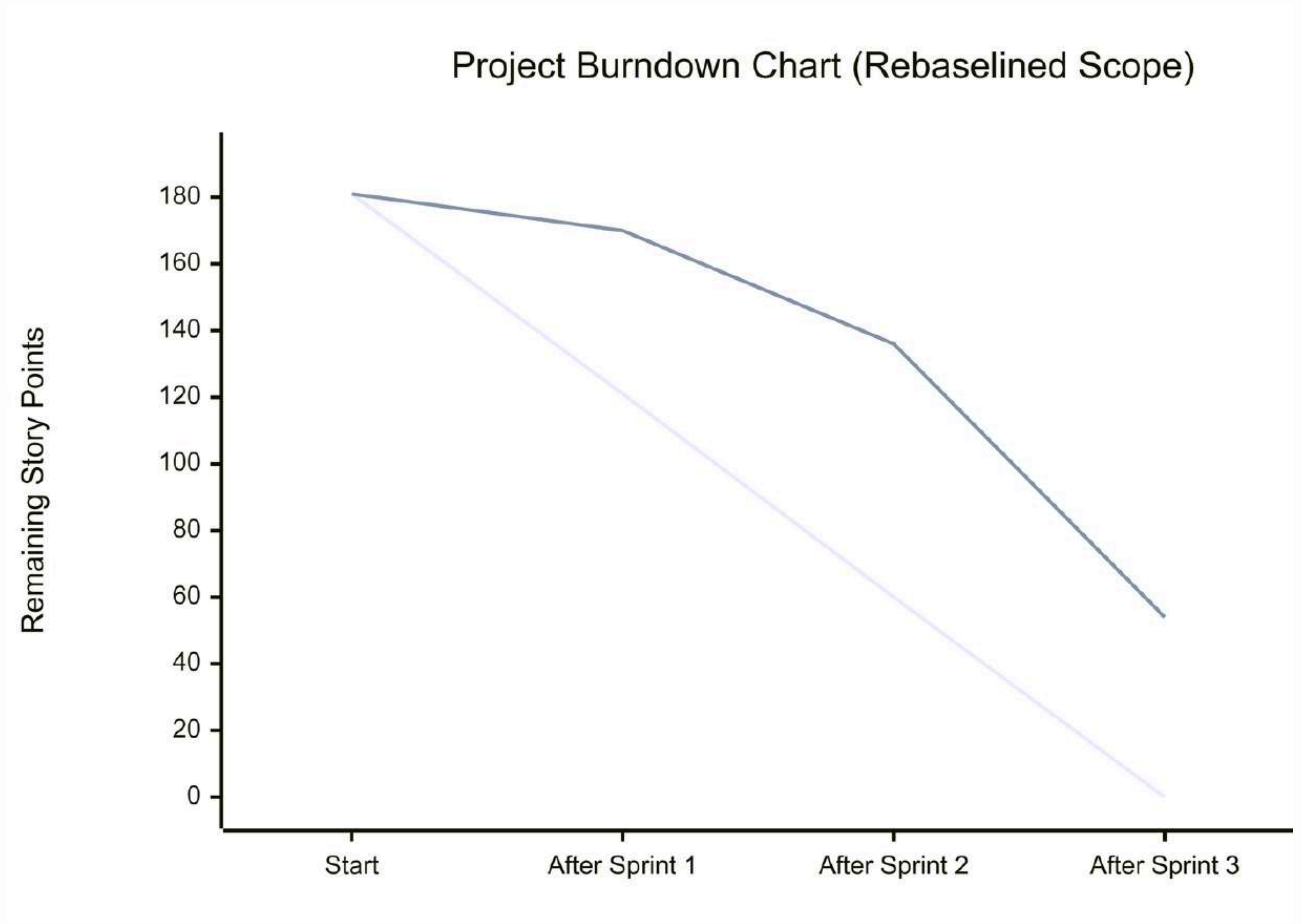
MessageAction
Validates string length and empty inputs before object creation.

ExternalProgramTrigger
Adapts commands based on the OS (Windows vs Unix).

Project Burndown Chart (Rebaselined Scope)



Analyzing Our Velocity Trend

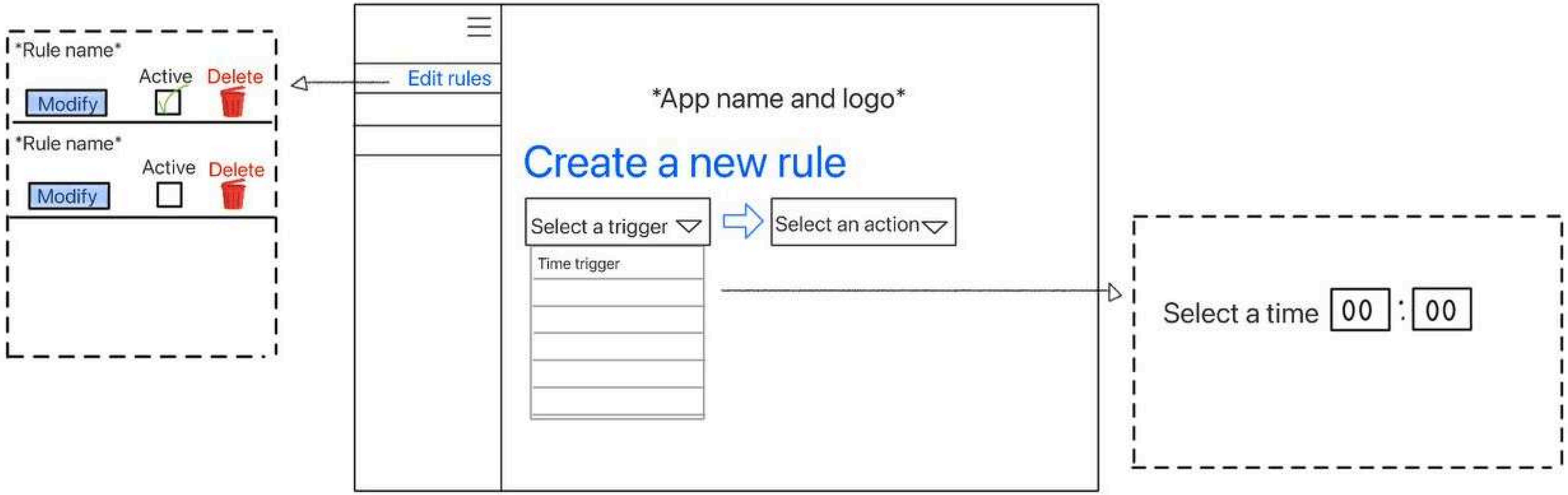


Velocity was initially lower than the Ideal Trend. We deliberately chose to first select few User Stories to concentrate building a solid collaboration flow.

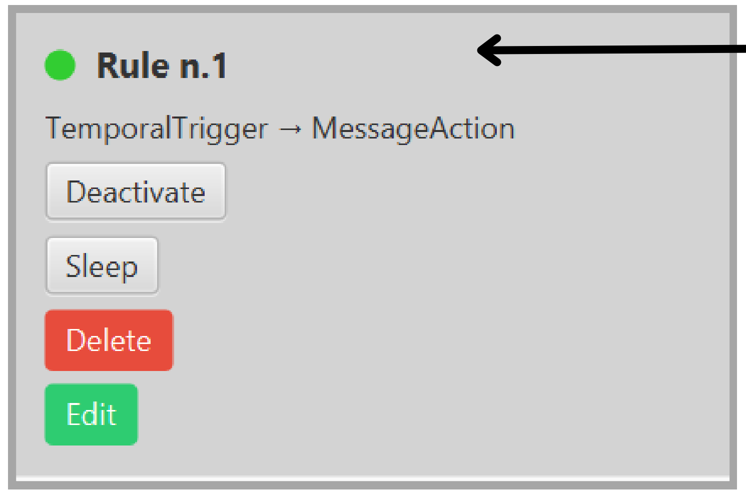
The gap narrowed as we mastered the team mechanisms and the patterns to use. Daily stand-ups and code reviews became efficient, reducing merge conflicts and clarifying tasks.

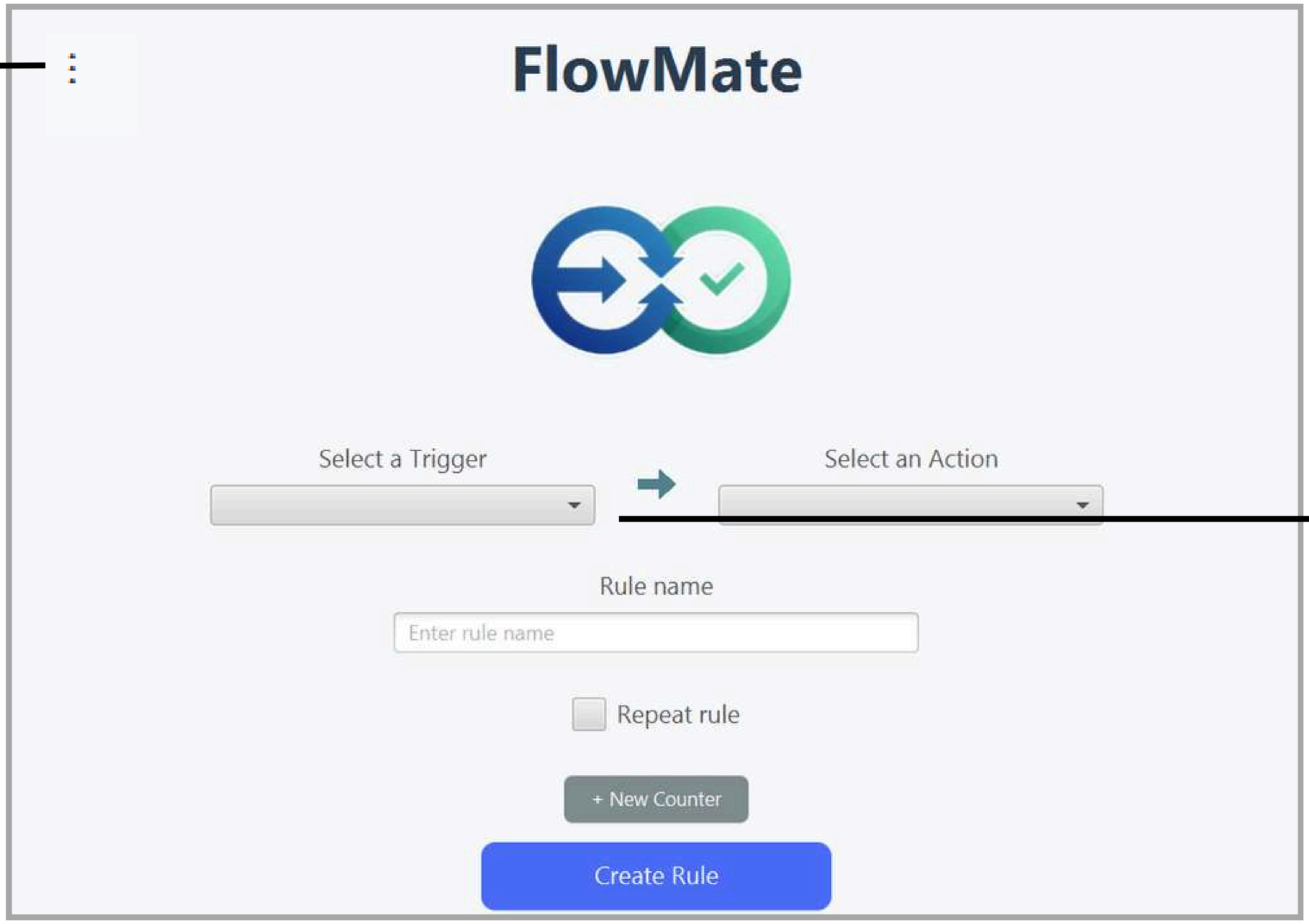
The final acceleration demonstrates the value of our initial design. With the core architecture solid, implementing the remaining User Stories became a rapid task of extending existing interfaces, requiring minimal refactoring.

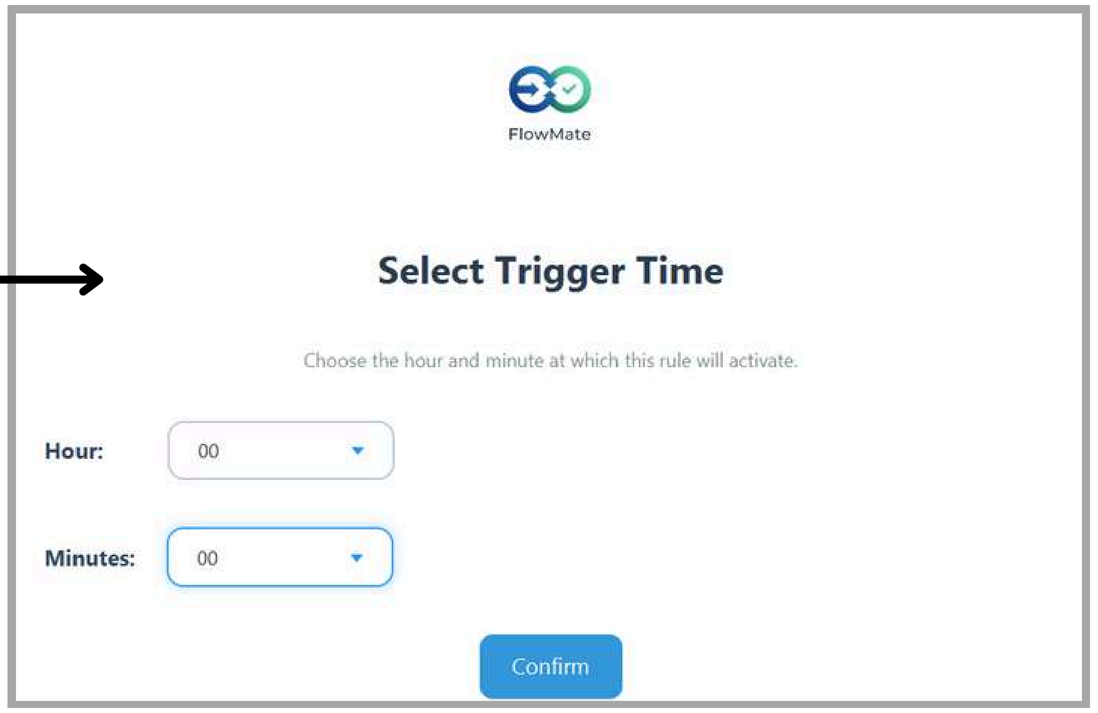
First Mockup vs Final Design



First Mockup vs Final Design






FlowMate
Select Trigger Time
Choose the hour and minute at which this rule will activate.
Hour:
00
Minutes:
00
Confirm

Product Roadmap & Scalability

Short term

Composite Triggers (AND/OR)

Implementing logical operators by extending the existing Trigger interface. The current Composite pattern structure already supports this evolution.

Mid Term

Plugin System

Leveraging our Factory Pattern to allow third-party developers to add new Actions (e.g., "Send Email").

Long Term

Cloud Synchronization

Moving from local persistence to a cloud synchronization to allow rules connection across different devices.

FlowMate

THANK YOU FOR THE ATTENTION!