Progetto ingegneria del software

MyWallet

MATTEO COLOMBO

CAMILLA MAZZOLENI

ANDREA ROTA

PIER FRANCESCO SORGIOVANNI

Obbiettivo

Lo scopo del progetto è creare una piattaforma per la gestione automatizzata di portafogli di investimento di privati

- Un’ interfaccia grafica mostra l’andamento del portafoglio
- Un algoritmo di trading segnala all’utente ogni volta che trova un investimento o una vendita vantaggiosa
- L’utente può autorizzare o negare la transazione con un semplice click

Difficoltà incontrate

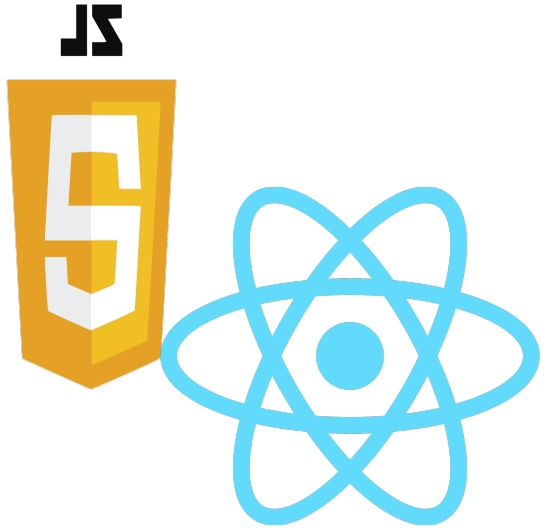
Coordinamento dei due sotto-team (backend e frontend)

Utilizzo di Docker su Windows

Conflitti nell’utilizzo di Git

Paradigma di programmazione

Backend



Frontend

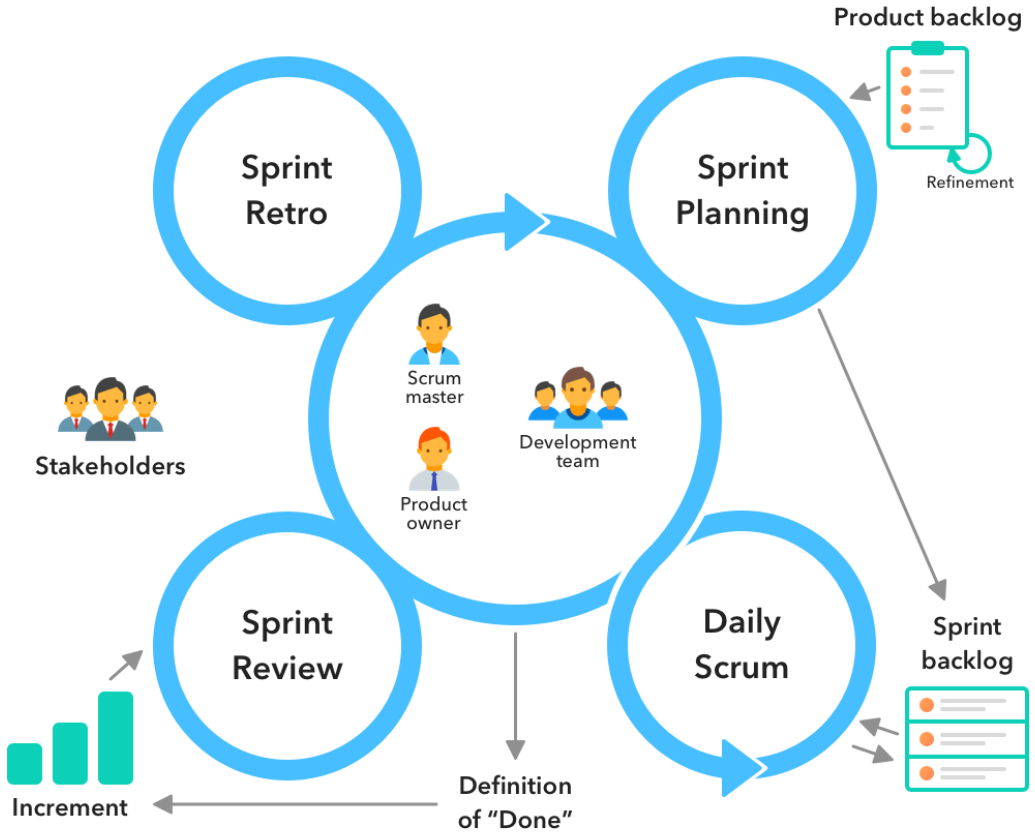
Altro

Software Configuration Managment

Per il progetto abbiamo utilizzato Github

KANBAN	Il progetto è stato gestito con kanban
BRANCH	Ogni componente del gruppo ha creato il proprio branch
PULL REQUEST	Per ogni azione di merge, è stata creata una pull request che doveva essere assegnata ad un reviewer e approvata da esso
ISSUE	Per ogni funzionalità, bug o modifica è stata aperta una issue

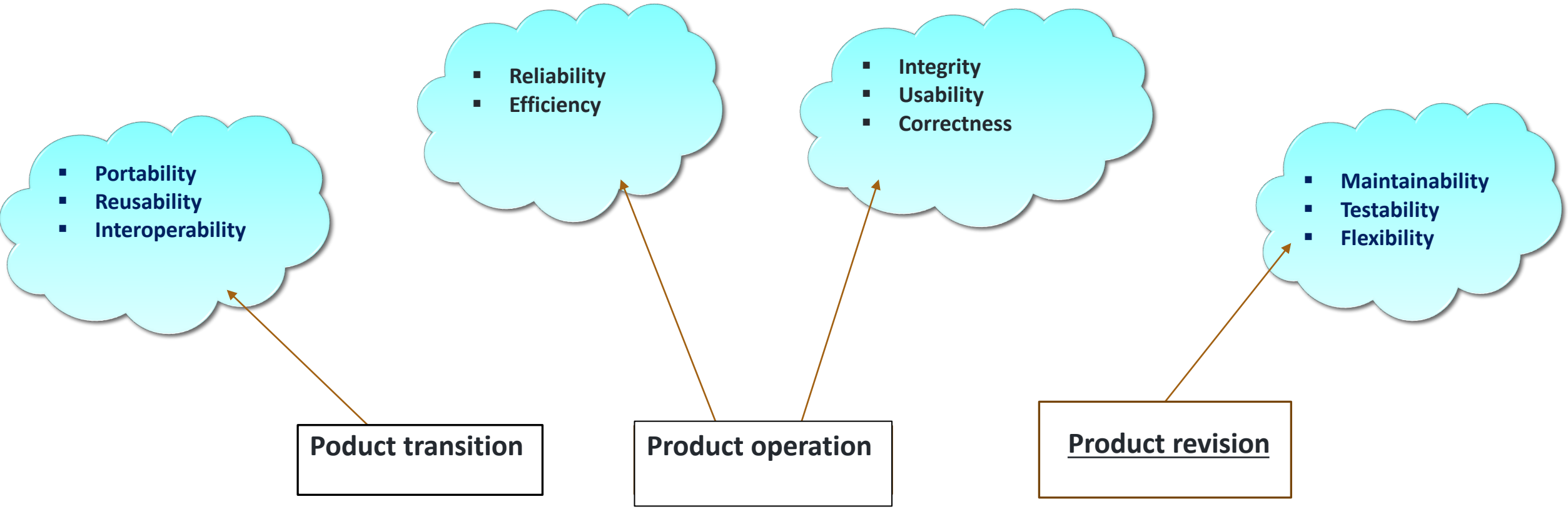
SCRUM Life Cycle



2 SPRINTS
OF 4 WEEKS

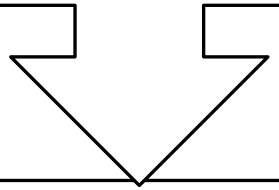
Quality of software

Parametri per la valutazione della qualità del software:



Requisiti

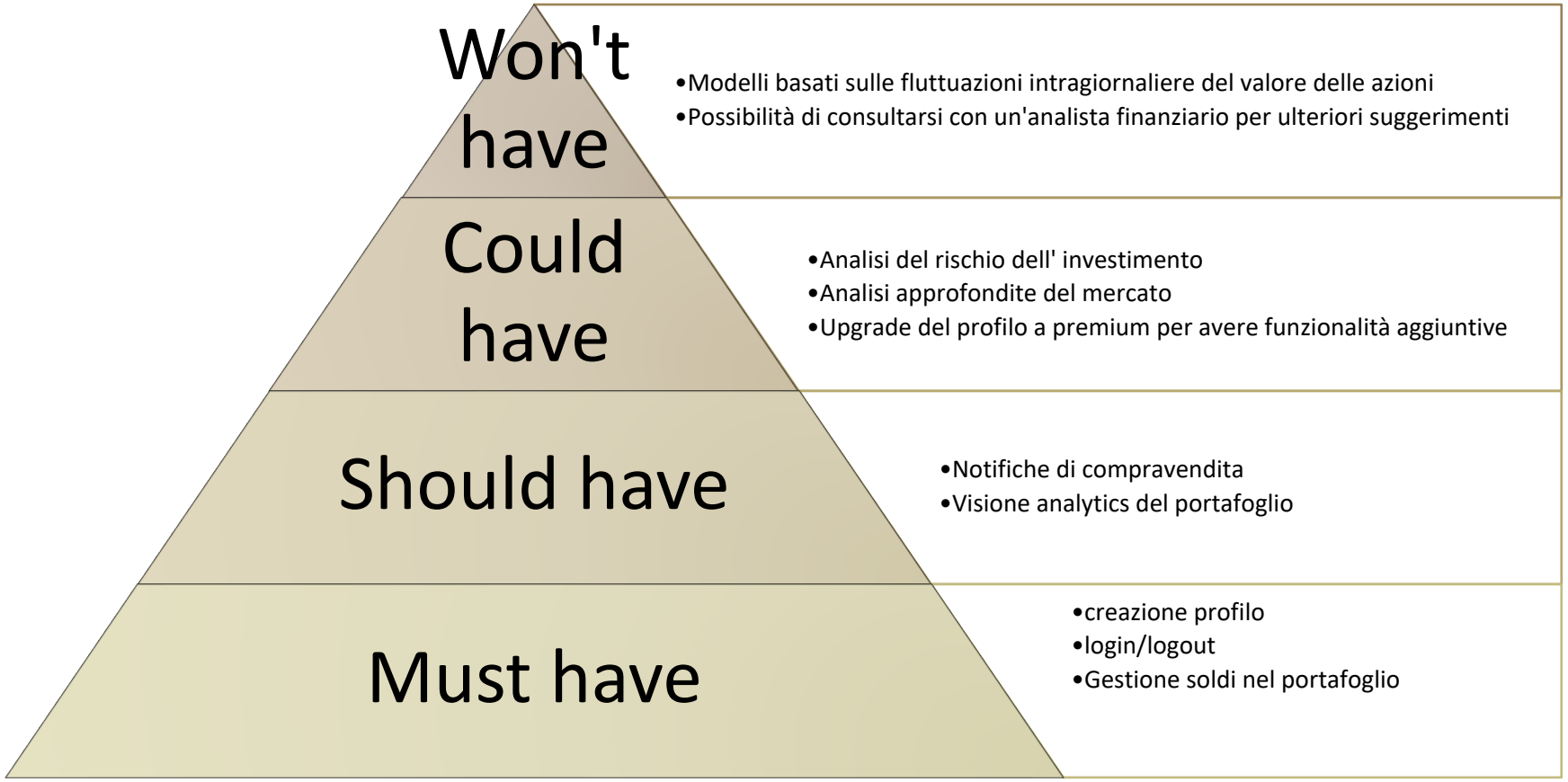
I requisiti del progetto sono stati descritti nella specifica dei requisiti



Per la specifica ci siamo basati sia sul modello MOSCOW che il modello KANO

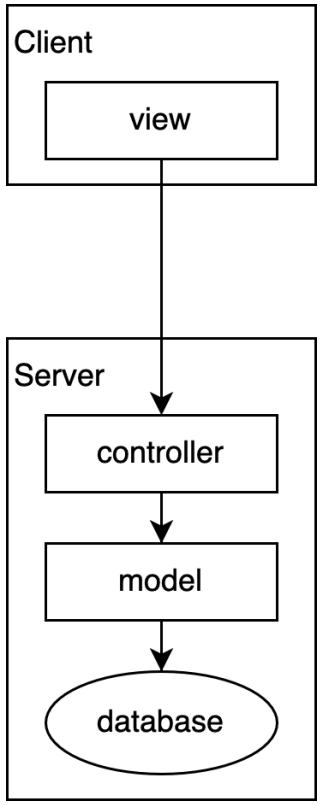
Requisiti funzionali (riguardanti le funzionalità del sistema)	Requisiti non funzionali (standard da rispettare, per esempio qualità, accessibilità e sicurezza)

MOSCOW requisiti funzionali

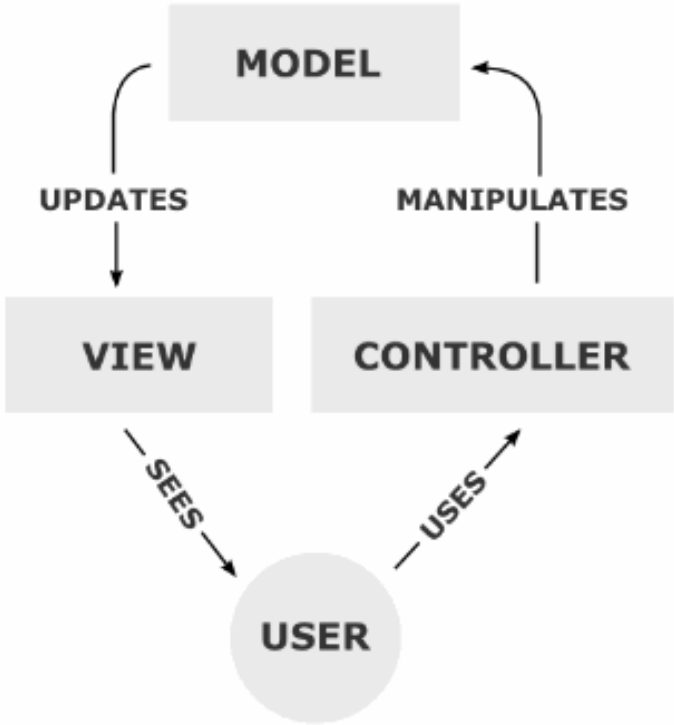


Architettura

Architettura client- server



Stile architetturale MVC (Model View Controller)

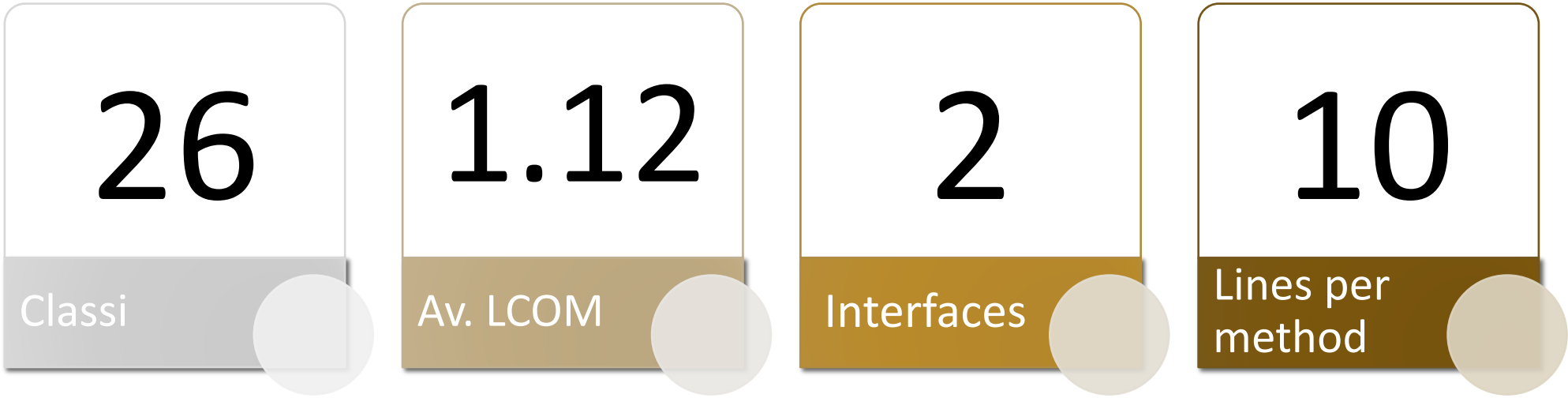


Design pattern

All'interno del framework Laravel sono presenti molteplici design pattern:

- **Builder pattern**
- **Factory pattern**: utilizzato per la generazione di dati (per utenti fittizi)
- **Strategy pattern**: utilizzato per la scelta di stockes differenti in base al profilio di rischio dell' utente
- **Provider pattern**
- **Repository pattern**
- **Facade pattern**: utilizzato per l' autenticazione. Interfaccia di facciata per utente base e utente admin

OO metrics

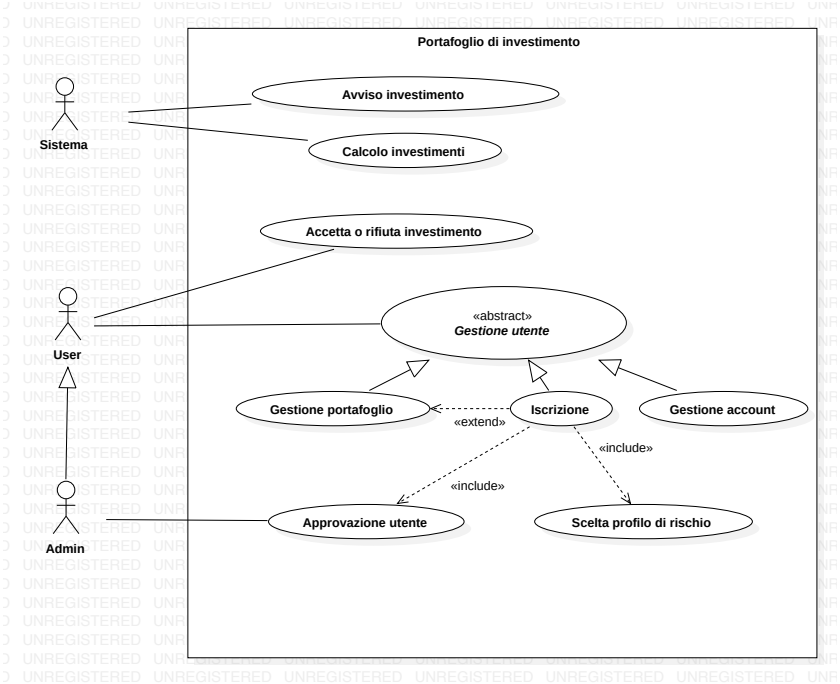


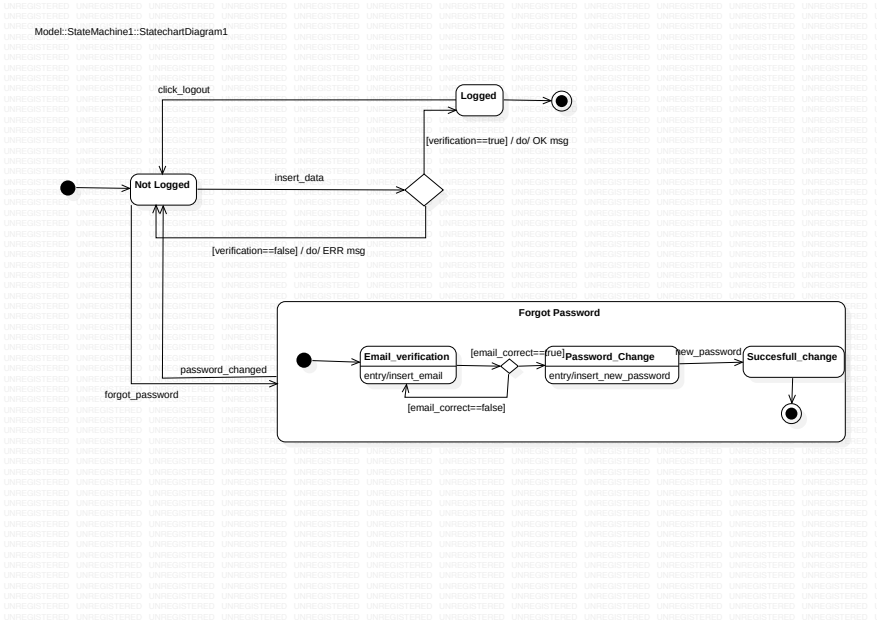
Modellazione

1	Diagramma delle classi
1	Diagramma dei casi d’uso
2	Diagrammi di sequenza
2	Diagrammi di stato
2	Diagrammi delle attività

Diagrammi

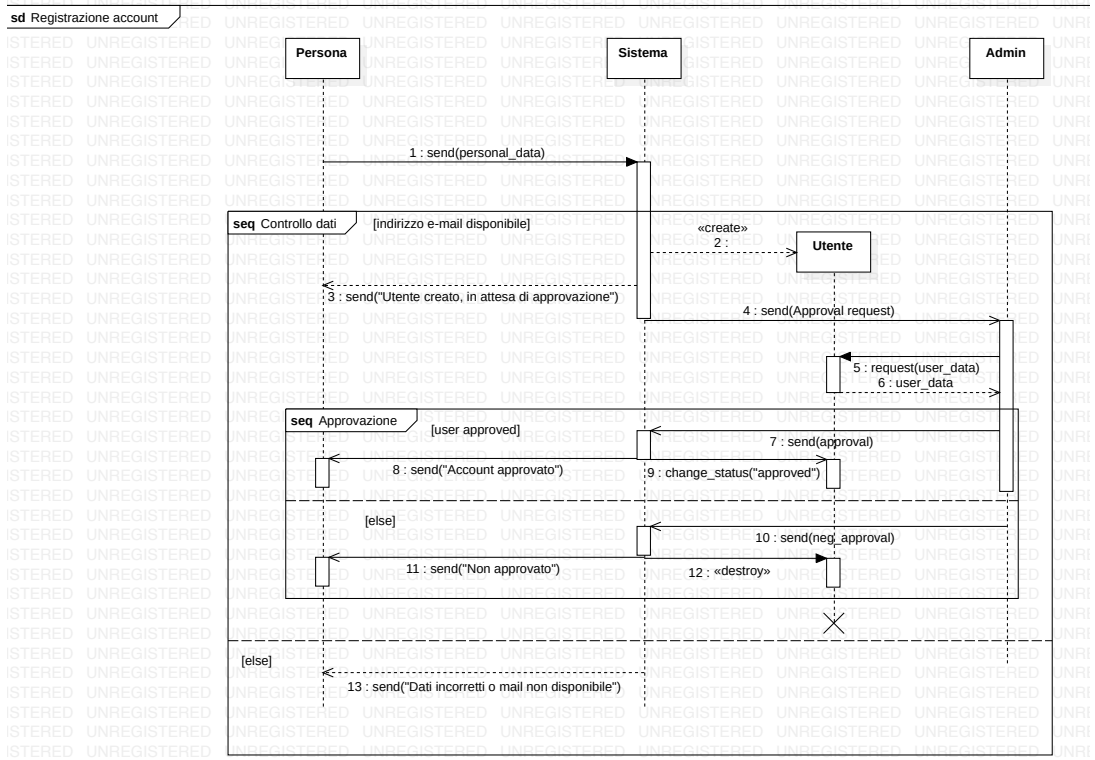
ESEMPI DI MODELLI USATI:

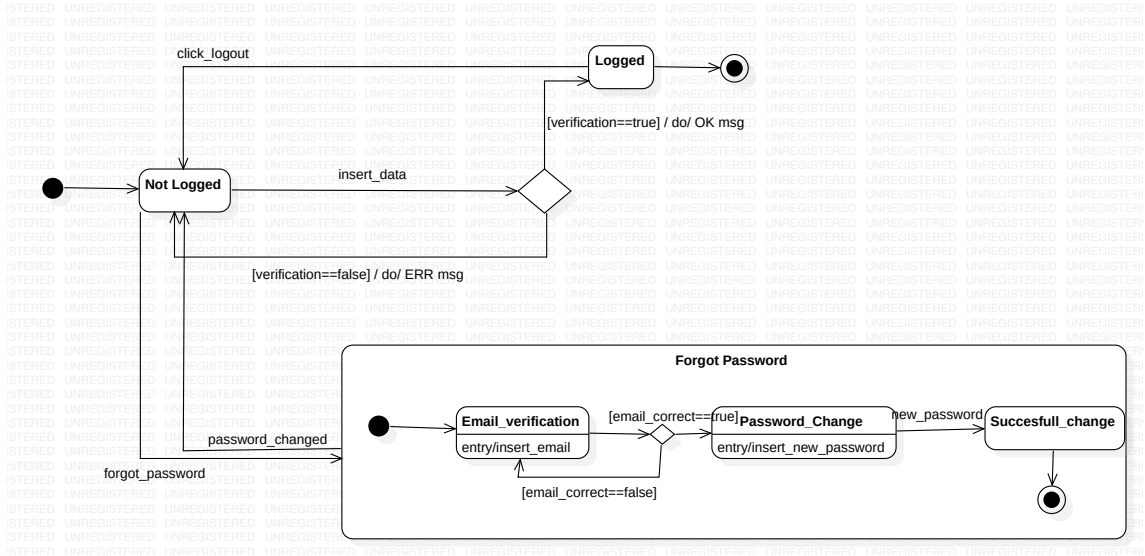




Diagrammi

ESEMPI DI MODELLI USATI:

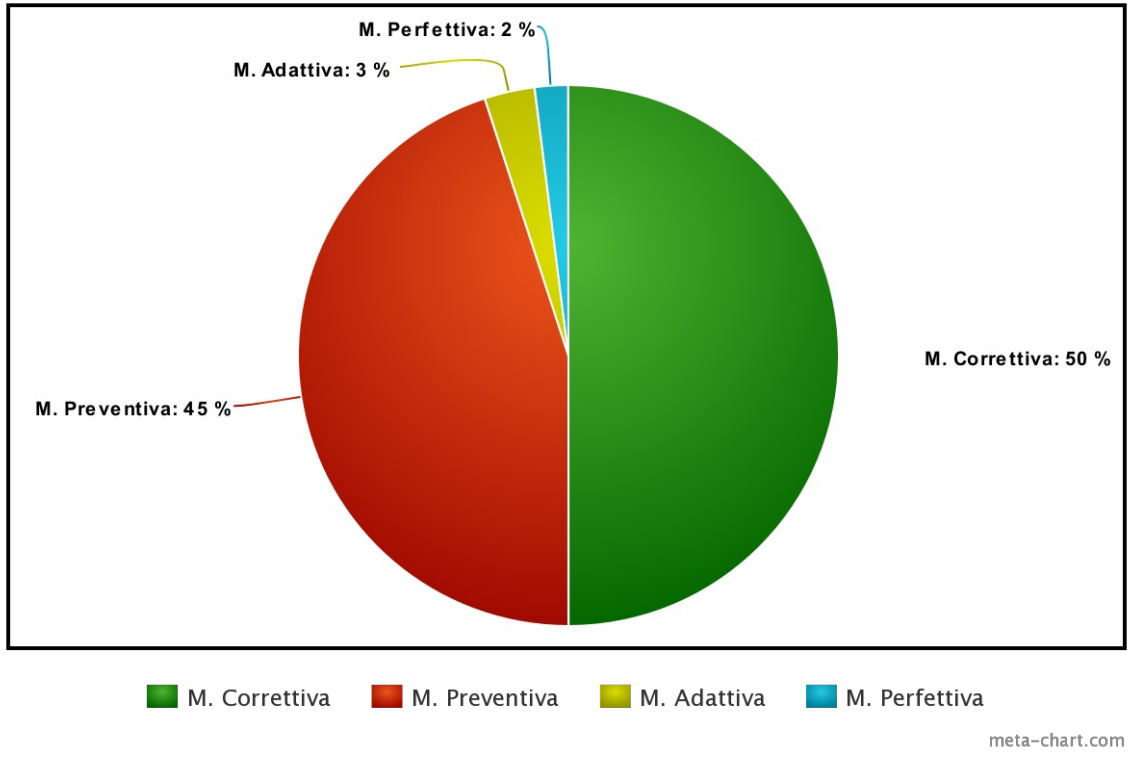




Implementazione

- ✓Registrazione
- ✓Autenticazione
- ✓Login/logout
- ✓Pagina degli Analytics
- ✓Impostazioni
- ✓Script di analisi stocks in R
- ✓Caricamento e prelevamento fondi
- ❑Notifiche

Manutenzione



- La fase di manutenzione è cominciata immediatamente dopo la consegna dell'applicativo e durerà per tutta la vita utile dello stesso.

Testing

Test effettuati con la libreria **PHPUnit**, dati generati con la libreria **Faker**

3 test

- AuthTest: testare sistema di autenticazione
- UserAPITest: test dell'API riguardante utenti di livello base
- AdminAPITest: test dell'API riguardante utenti di livello admin

```
PASS  Tests\Unit\ExampleTest
✓ that true is true

PASS  Tests\Feature\AdminApiTest
✓ get users
✓ get stocks
✓ approved by administrator

PASS  Tests\Feature\AuthTest
✓ registration
✓ login
✓ logout

PASS  Tests\Feature\UserApiTest
✓ get stocks
✓ increase or decrease wallet

Tests:  9 passed
Time   2.40s
```

69,9 % Coverage

GRAZIE PER L'ATTENZIONE