Relazione progetto Reega

Benazzi Daniel
Pola Manuele
Salomone Marco

25/04/2021

Indice

Capitolo 1

Analisi

Il progetto REEGA vuole essere un portale per la visualizzazione e la gestione dei **servizi** offerti da un'ipotetica azienda. In particolare questi servizi sono di 4 tipologie: **Rifiuti**, **Energia Elettrica**, **Gas** e **Acqua**; i rifiuti sono divisi a loro volta in 4 categorie: carta, plastica, vetro ed indifferenziata.

Accedendo come *amministratore* sarà possibile gestire gli utenti ed i loro contratti aggiungendone o rimuovendone ad utenti già registrati. Sarà inoltre possibile verificare le informazioni degli utenti e l'ammontare dei servizi loro forniti. Accedendo alla piattaforma come *utente finale* sarà possibile visualizzare i propri contratti ed i propri consumi divisi per tipologia. Sarà in oltre possibile avere brevi **report** dei mesi passati oltre a **grafici** riepilogativi.

Non essendo gli utenti reali dovrà anche essere presente una **generazione** verosimile dei dati sull'utilizzo dei servizi per ogni contratto di ogni utente.

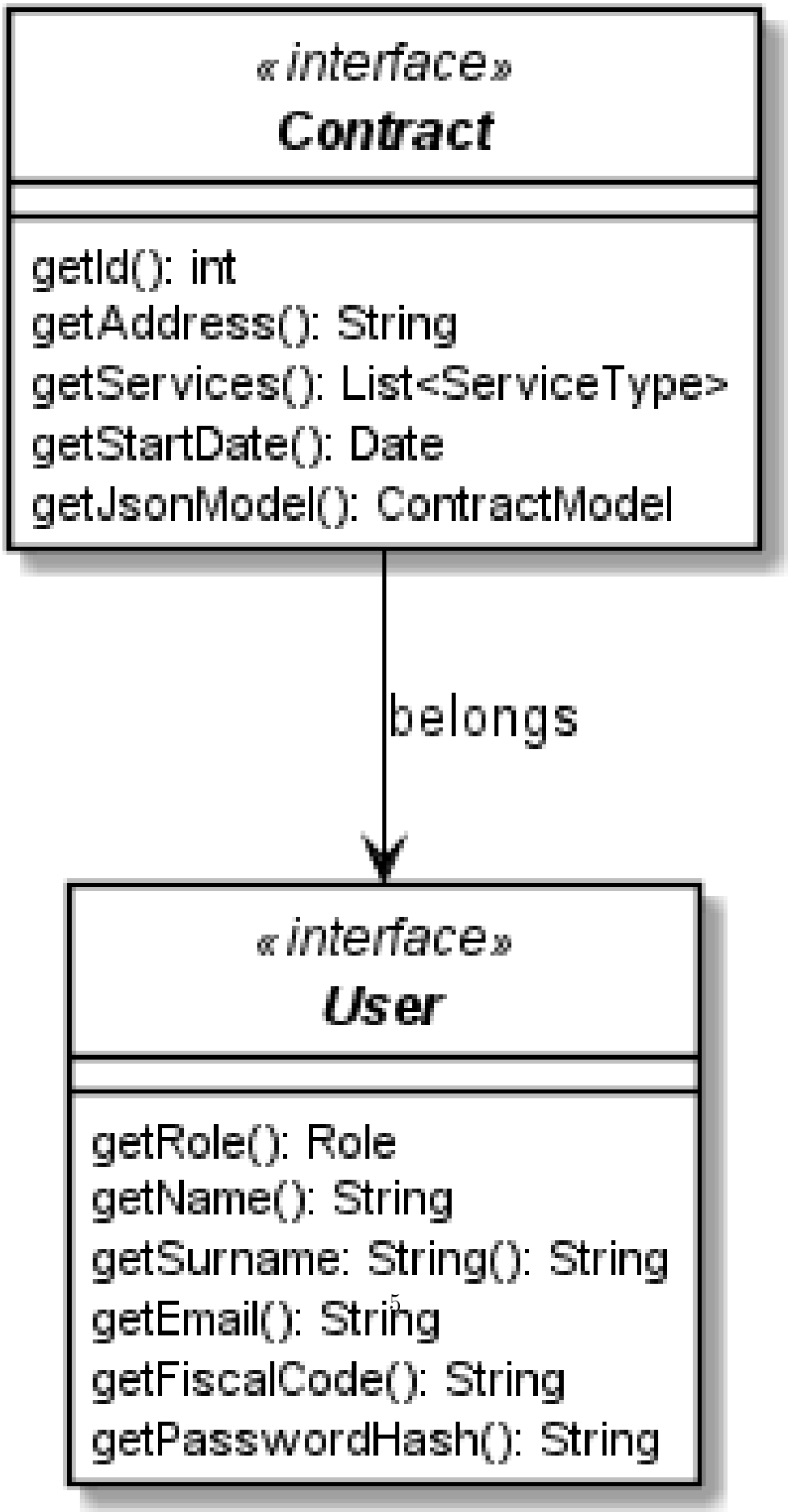
1.1 Requisiti

1.1.1 Requisiti funzionali

- Visualizzazione consumi nel mese corrente, e relativi grafici;
- L'utente può registrarsi autonomamente, ma deve essere un admin a stipulare un nuovo contratto;
- Un contratto comprende un sottoinsieme di tutti i servizi che l'azienda può fornire di cui l'utente ha deciso di usufruire;
- Un utente può avere più di un contratto;
- Un utente può vedere il report delle bollette dei mesi scorsi;
- È data la possibilità di esportare i dati su file.

1.2 Analisi e modello del dominio

Il dominio applicativo si basa sull'utente utilizzatore, sia esso utente semplice o amministratore, che può eseguire tutte le operazioni appropriate relative al suo ruolo. L'applicativo dovrà essere in grado di mostrare i dati corretti relativi al richiedente. Un utente finale avrà associato uno o più contratti, che definiscono gli estremi dei servizi forniti da Reega. In questa fase iniziale in mancanza di fonti di dati reali, questi verranno generati una volta al giorno in maniera artificiale, e saranno salvati tramite un servizio remoto.


«interface»
Contract
getId(): int
getAddress(): String
getServices(): List<ServiceType>
getStartDate(): Date
getJsonModel(): ContractModel
belongs
«interface»
User
getRole(): Role
getName(): String
getSurname: String(): String
getEmail(): String
getFiscalCode(): String
getPasswordHash(): String

Capitolo 2

Design

2.1 Architettura

L'architettura di Reega segue il pattern architetturale MVVM(Model View ViewModel). Questo pattern è stato utilizzato per scindere completamente la View dal ViewModel, infatti la View è a conoscenza dell'interfaccia esposta dal ViewModel, ma il ViewModel è ignaro della View che lo utilizza; il ViewModel quindi espone solamente la strutturazione dei dati che poi vengono manipolati dalla View. Il ViewModel scambia informazioni con la View attraverso esposizione di metodi e/o utilizzo di handler che permettono di monitorare lo stato di alcuni elementi del ViewModel. Il ViewModel rappresenta la diretta rappresentazione del Model nella View, quindi ViewModel e Model sono tightly-coupled. Non vi è alcuna interazione tra View e Model, quindi la View ha visione sul Model in sola lettura in modo tale da poterlo rappresentare. Per evitare il tight-coupling della View con il ViewModel è stato deciso di utilizzare il meno possibile costrutti proprietari di una specifica libreria in modo tale da non impattare pesantemente il cambio della libreria utilizzata per la View; in questo modo si è però escluso uno degli elementi fondamentali dell'MVVM quale il Data Binding. Il ViewModel utilizza dei Controller per gestire/caricare i dati da mostrare poi alla View. Nella figura Figura 2.1 è esemplificato il diagramma UML architetturale.

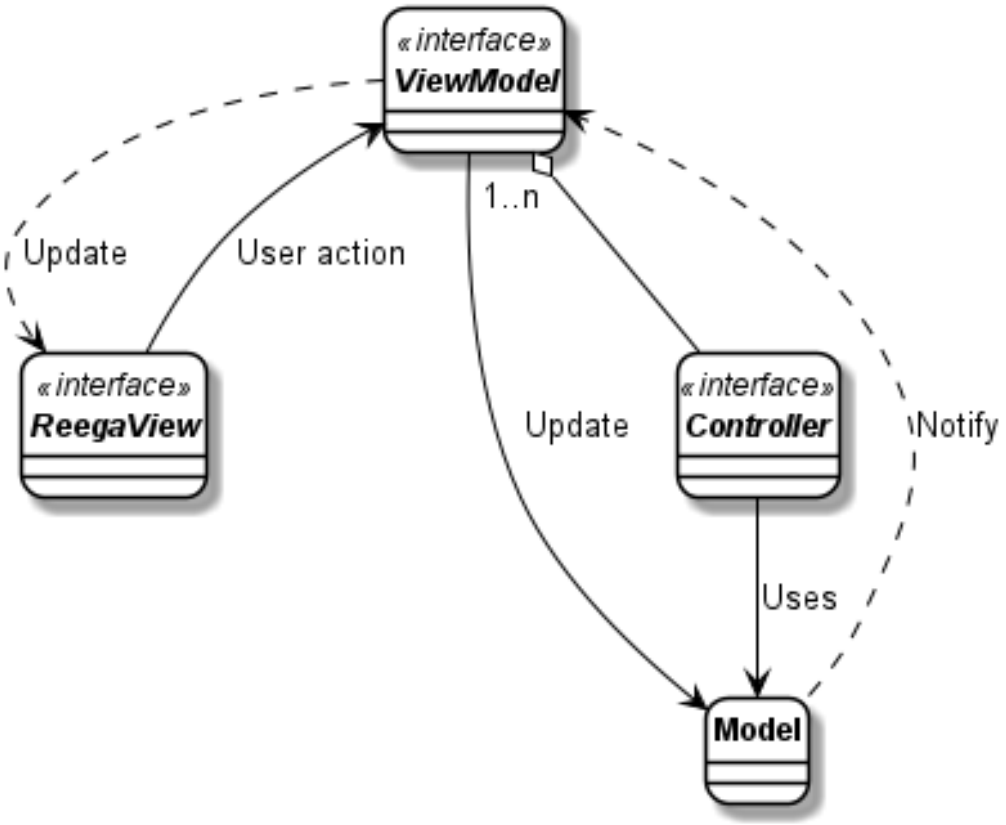


Figura 2.1: Schema UML dell'architettura di Reega. Model e Controller sono solo interfacce rappresentative rispettivamente degli oggetti di dominio e dei servizi utilizzati dai ViewModel

2.2 Design dettagliato

2.2.1 Marco Salomone

Dependency Injection

Per modellare un meccanismo di **Dependency Injection** che permettesse di evitare la costruzione diretta di oggetti ho creato una ServiceCollection che racchiude in sè tutte le dipendenze necessarie a Reega per funzionare. Queste dipendenze vengono "registrate" al boot dell'applicazione e vengono suddivise in Singleton e Transient; l'unica differenza è l'istante di tempo in cui essi vengono risolti(creato l'oggetto e caricate le sue dipendenze): se i Singleton vengono risolti quando registrati, i Transient invece vengono risolti ad ogni richiesta eseguita. Dopo aver finito di inizializzare una ServiceCollection

è possibile costruire un ServiceProvider che permette di richiedere i servizi registrati. In figura Figura 2.2 è esemplificato il diagramma del meccanismo di Dependency Injection

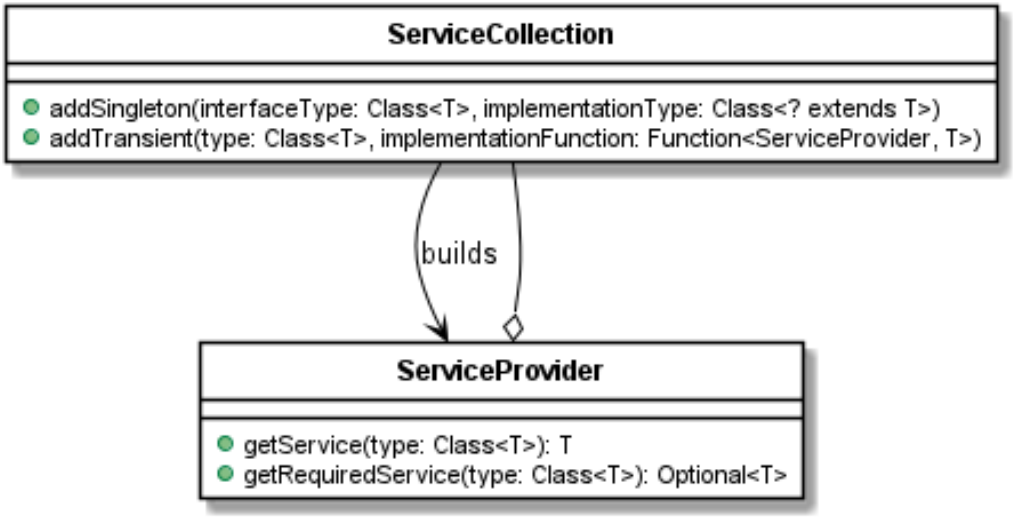


Figura 2.2: Schema UML del meccanismo di **Dependency Injection**

Managers

Per evitare l'enorme utilizzo di boilerplate code nei ViewModel ho deciso di utilizzare degli **Adapter** che potessero incapsulare le chiamate al server ospitante la piattaforma Reega. Ne sono un esempio il RemindableAuthManager e ContractManagerImpl, essi infatti incapsulano al loro interno dei Controller utilizzati per chiamate al server ed implementano un'interfaccia che sarà poi visibile e utilizzabile dai ViewModels. In figura Figura 2.3 è esemplificato il diagramma di gestione dei Managers.

Events

Data la necessità di fornire un strumento per la navigazione tra le pagine, è stato resa necessaria la creazione di un EventHandler generico che gestisce degli argomenti(della classe EventArgs, anch'essa generica). In questo modo è stato possibile utilizzare gli EventHandler sia per la comunicazione ViewModel-ViewModel, sia per la comunicazione tra ViewModel e Navigator utile per la navigazione tra le diverse pagine. Data la modularità dell'EventHandler per come è stato concepito, è stato possibile implementare quest'ultima comunicazione semplicemente attraverso l'estensione della classe EventArgs.

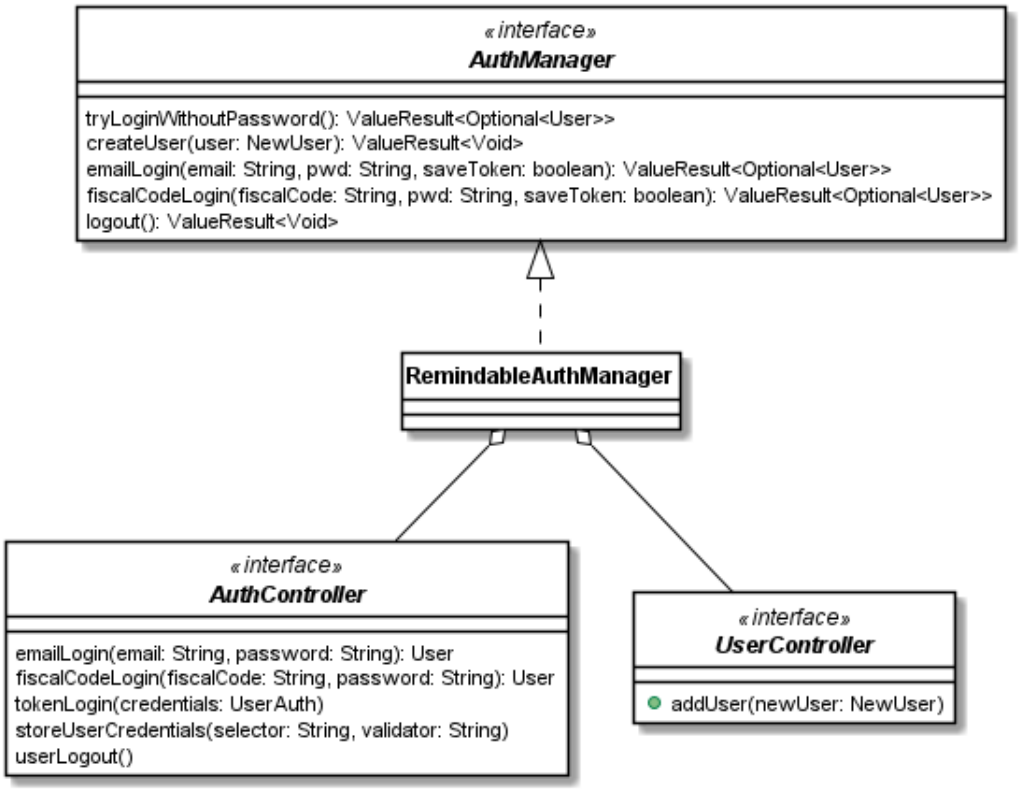


Figura 2.3: Schema UML per la gestione del RemindableAuthManager come esempio di utilizzo del pattern **Adapter**

2.2.2 Manuele Pola

Overview

L'applicazione può potenzialmente lavorare (in mutua esclusione) con dati provenienti da molteplici fonti, come ad esempio un database in locale, un servizio di API esterno, un software aziendale, etc. In questa fase di sviluppo, si è deciso di utilizzare un servizio esterno, per poter fornire una demo completa e significativa dell'applicazione.

Comunicazione

Il servizio che si occupa di memorizzare e fornire dati espone un'interfaccia API via HTTP. Si mapperanno quindi tutte le possibili operazioni in un'interfaccia con dei decoratori appositi, tali da poter sfruttare la libreria Retrofit2 per gestire il protocollo. Il servizio creato verrà wrappato dalla classe RemoteConnection, che sarà condivisa tra tutti i controller, per poter gestire in maniera trasparente

l'autenticazione delle chiamate. Impostando un token all'accesso, sarà possibile riconoscere l'utente che richiede le risorse, consentendo così di poter accedere ai dati corretti. Vedi figura fig. 2.5.

Export

È possibile esportare i dati del mese corrente su file nei formati JSON e CSV. Questi dati arrivano come aggregati tramite una chiamata apposita al servizio remoto.Vengono poi trasformati secondo quanto necessario a renderli fruibili e salvati su file scelto dall'utente. Lo sviluppo sarà modulare, consentendo quindi, su necessità, di implementare ulteriori formati di export, semplicemente aggiungendo un **Exporter** dedicato, e renderlo disponibile tramite l'apposita **Factory**. Vedi figura fig. 2.6.

2.2.3 Daniel Benazzi

Data generation

Data la natura simulativa del progetto vi era la necessità di avere una forma di generazione dei dati. Questo sistema sfrutta una **static factory** in modo da lasciare la possibilità, in futuro, di implementare nuovi algoritmi per generazione dei dati. In particolare *UsageSimulator* usa *GaussianGeneratorFactory* per ottenere i generatori dei dati in base al tipo specificato di *DataType*. fig. 2.7.

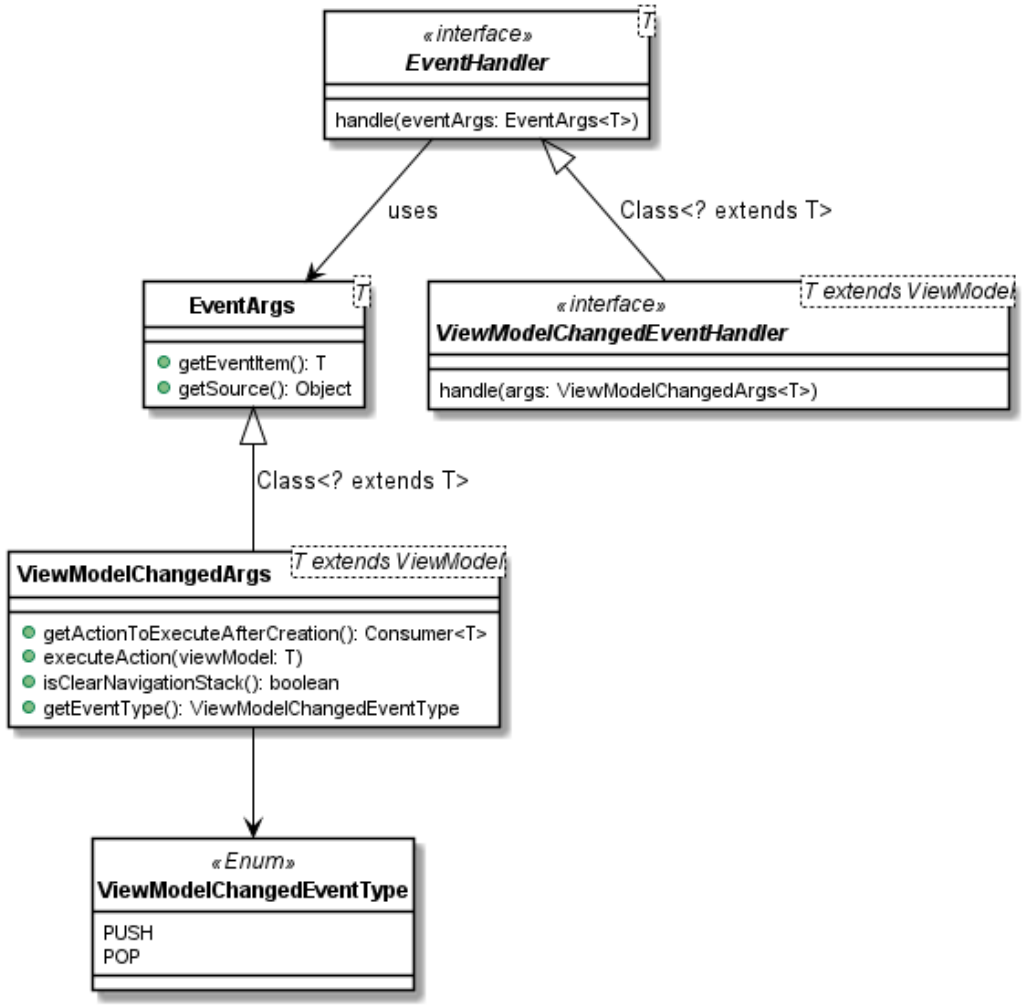


Figura 2.4: Schema UML per la gestione degli eventi, con la specializzazione per la gestione di eventi di "cambio ViewModel"

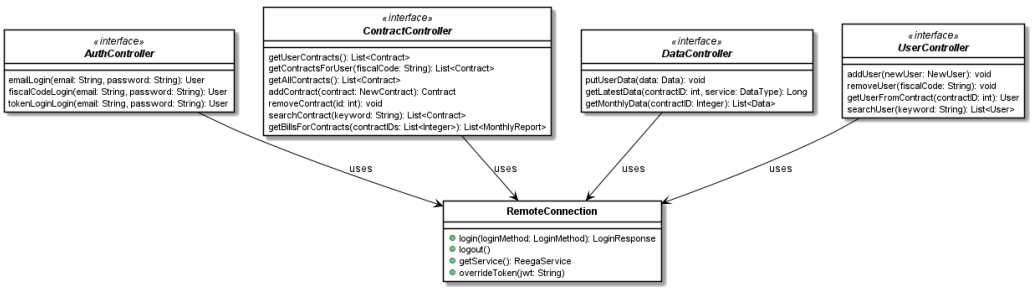


Figura 2.5: Schema UML dei controller per l'interfacciamento con i dati

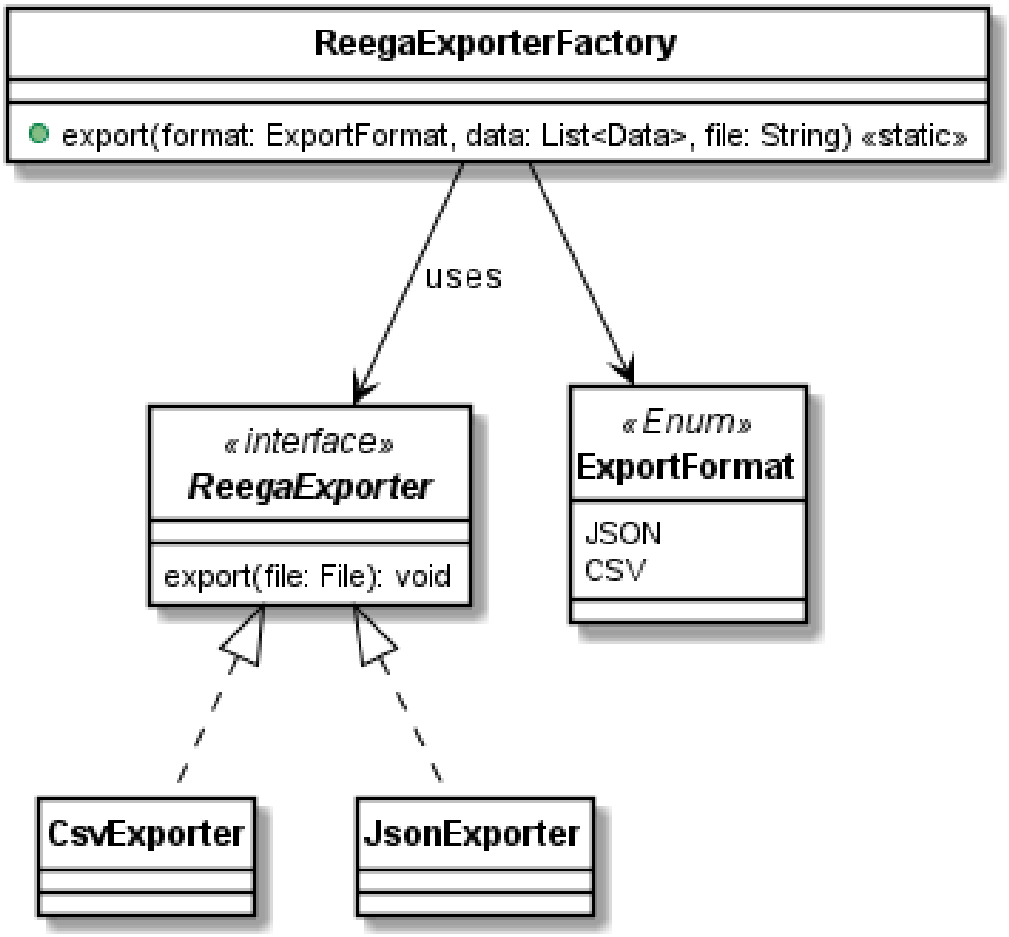


Figura 2.6: Schema UML per l'esportazione dei dati

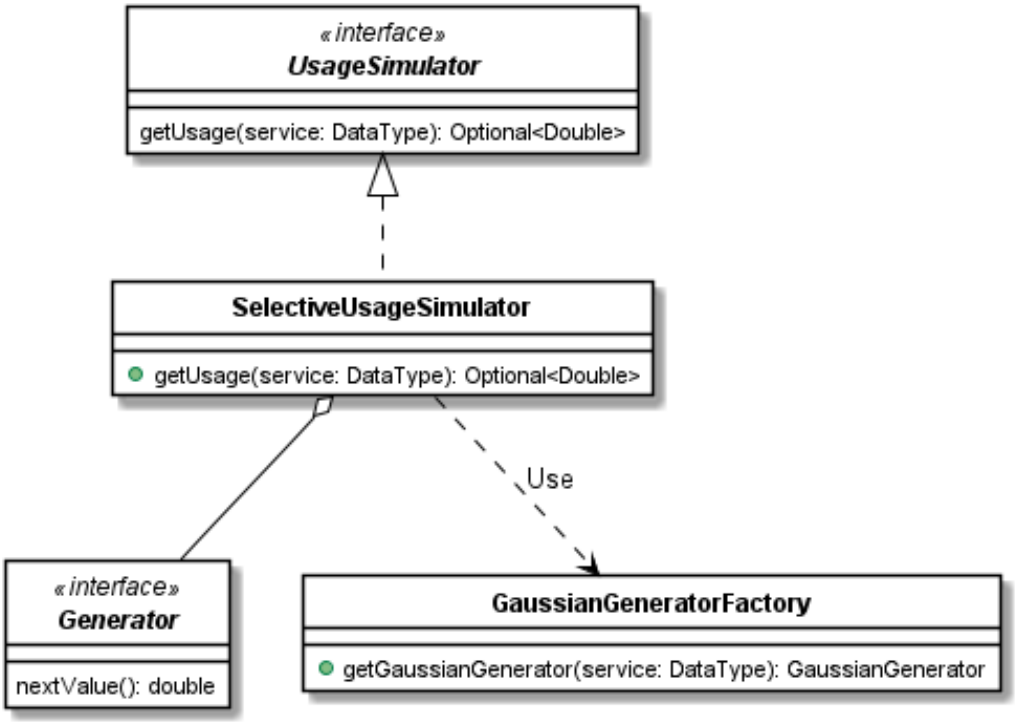


Figura 2.7: Schema UML del meccanismo di creazione dei generatori di dati

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il progetto è corredato da diversi test automatici, che mirano a testare i punti più sensibili dell'applicazione: **i dati**. Questi test utilizzano JUnit 5, e sono eseguiti automaticamente prima di assemblare il jar vero e proprio. Su github è presente una **Action** che ad ogni push sul branch master, provvede ad eseguire i test, assemblare il jar e caricarlo nella release **latest**. Si ha quindi la certezza che il jar che si scarica è stato testato con successo. Ogni classe di test utilizza un'istanza di TestConnection che provvede a stabilire una connessione con il servizio di gestione dei dati, creare un'istanza isolata ed in uno stato ben definito, e fornire una connessione privilegiata a questa istanza, in maniera da poter testare ogni aspetto della comunicazione e dell'interazione con le API. Inoltre, dopo l'esecuzione dei test, è generato in automatico un report con jacoco per verificare la code coverage, che può essere consultato con /jacoco/test/html/index.html.

I test includono ma non sono limitati a:

- Aggiunta, ricerca e rimozione di un utente;
- Login con i vari metodi di accesso (remind-me, codice fiscale, email);
- Aggiunta e richiesta di dati generati;
- Aggiunta, ricerca ed eliminazione di contratti;
- Correttezza report delle bollette;
- Export dati nei formati CSV e JSON;
- Validazione codice fiscale.

3.2 Metodologia di lavoro

Abbiamo affrontato il progetto sviluppandolo in maniera incrementale, dividendo i compiti per "feature", ed espandendo gradualmente le funzionalità. Allo stesso modo abbiamo utilizzato il DVCS, in questo caso GitHub, creando branch, su cui lavoravamo per lo più singolarmente, per nuove funzionalità da implementare. Si è fatto uso di *merge*, *rebase*, *cherry-pick* e *tags*.

3.2.1 Marco Salomone

Creazione di un meccanismo, seppur semplice, di Dependency Injection presente nel package reega.util con le classi ServiceCollection che rappresenta una collezione di servizi divisibili tra transienti(ad ogni nuova creazione ne viene generato uno nuovo) e singleton(viene generato un solo oggetto per tutta la durata dell'applicazione). Creazione di un meccanismo di Navigation che non coinvolge la View ma solamente i ViewModel in modo da avere una View che dipende dall'altra. Creazione di un DataTemplateManager che si avvale di appositi DataTemplate per collegare i Controller alle loro implementazioni come View di JavaFX.

3.2.2 Manuele Pola

Le principali funzionalità sviluppate sono sono:

- Interfacciamento con API;
- Encryption password;
- Autenticazione chiamate con JWT e middleware;
- Logica di base per la funzione "remind-me";
- Export dati;
- Report bollette e relativa UI;
- Metodi e meccanismi di base per eseguire test sui dati;
- Configurazione logger slf4j.

3.2.3 Daniel Benazzi

Io mi sono dedicato a compiti di diverse tipologie: generazione dati, view per ricerca e view per creazione contratti, con i rispettivi view model, e creazione grafico con il proprio fornitore di dati (DataPlotter).

Per quanto riguarda il grafico mi sono trovato a creare una sotto-sezione della view principale del programma, sviluppata dal collega Salomone, allo scopo di non aumentare le dipendenze della view, a causa del grafico, abbiamo cooperato per inglobare il DataPlotter in StatisticController così da avere i dati sincronizzati ed evitare di richiedere i dati di volta in volta. Per le parti di ricerca utenti/contratti e di creazione contratti abbiamo deciso insieme gli agganci necessari, e quindi le interfaccie, per poi dedicarci ai nostri rispettivi ruoli, in particolare mentre io mi dedicavo alla view (UserSearchView e ContractCreationView) ed al viewModel (SearchUserViewModel e ContractCreationViewModel), il collega Salomone si occupava dell'integrazione con il sistema di navigazione e il collega Pola dell'implementazione delle classi di interfacciamento al database remoto. In fine la generazione dati mi ha permesso di lavorare maggiormente in autonomia esponendo all'esterno del package solo DataFiller, la classe che effettivamente registra i dati generati per ogni utente, le cui dipendenze, per gestire dati in lettura e scrittura, sono state soddisfatte dalla cooperazione con il collega Pola; sempre definendo i metodi delle interfacce per poi dedicarsi autonomamente alle implementazioni.

3.3 Note di sviluppo

3.3.1 Marco Salomone

- Progettazione con generici per la gestione di DataTemplate e per la gestione della Navigation;
- Utilizzo di *Stream* per la manipolazione dei dati nella classe StatisticsController;
- Utilizzo di *lambda expressions* per la creazione di comandi e spostamento tra i diversi ViewModel;
- Utilizzo della reflection per la creazione del meccanismo di Dependency Injection;
- Utilizzo di JavaFX e conseguente creazione di un controllo necessario alla Navigation.

3.3.2 Manuele Pola

- Utilizzo di lambda in diversi punti, sia per manipolazione di stream di dati, sia per fornire istruzioni specifiche con medesimi risultati a dei metodi (come ad esempio il login fatto nelle varie forme;
- Configurazione dei processi di build, test e report con gradle;

Le librerie utilizzate sono:

- **Gson** per la conversione a/da json;
- **Retrofit2** come libreria per la composizione di chiamate HTTP e **OkHttp** come http client;
- **Spring Security Crypto Module** per l'encryption della password con BCrypt;
- **slf4j** per il logging degli eventi, ed opzionalmente reindirizzabile su file di log;
- **jacoco** per il code coverage automatico in fase di build con gradle;
- **ShadowJar** per l'assemblaggio finale del jar;
- **JavaFX** per la UI;
- vari moduli **Apache Commons** come utility.

3.3.3 Benazzi Daniel

- Utilizzo di JavaFX ed FXML per creare le view e i grafici;
- Utilizzo di CSS per lo styling della view;
- Favorimento di Stream e lambda al codice imperativo. Ad esempio per la manipolazione dei dati da mostare nel grafico.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Marco Salomone

Sono molto soddisfatto del lavoro svolto sia da me che dal mio team di sviluppo; c'è stata una continua comunicazione per discutere di dubbi progettuali o implementativi permettendoci di essere sempre al corrente dei progressi di ognuno dei componenti del gruppo. Fin dall'inizio mi sono preoccupato di dare basi solide ai miei compagni di team su cui potevano costruire tutto ciò che desideravano rispettando opportuni constraint settati da me. In questo modo abbiamo potuto creare un progetto come Reega che ha una struttura consistente per ogni parte della sua architettura, sia dal punto di vista delle classi che dal punto di vista dei package. Fondamentale è stato l'utilizzo di Git, che usavo già da tempo ma di cui ho imparato nuove feature durante lo sviluppo del progetto. Ho anche pensato di portare avanti il progetto, non in quanto tale, ma riutilizzando elementi pensati e costruiti durante il suo sviluppo data l'estrema modularità con cui sono state creati per sviluppare un mini-framework per la creazione di un'applicazione MVVM simile a WPF, con anche eventuali meccanismi di Dependency Injection.

4.1.2 Manuele Pola

Complessivamente l'applicazione rispecchia le funzionalità che ci eravamo prefissati di raggiungere. Il mio operato si è concentrato sullo sviluppo dei metodi di gestione dei dati, interfacciamento con il servizio di storage remoto, e più in generale sul fornire delle funzionalità di base allo sviluppo dell'applicativo, sia dal punto di vista del codice che da quello dell'infrastruttura. L'utilizzo costante di Slite, git e sistemi di videoconferenza hanno permesso

al gruppo di rimanere aggiornati sugli sviluppi, e tenere sotto controllo in maniera chiara ed inequivocabile le cose fatte e quelle ancora da fare. Credo che per trovare una reale applicazione a questo progetto ci siano ancora dei punti su cui lavorare, come ad esempio un approvvigionamento di dati reali. Si potrebbe migliorare anche dal punto di vista tecnico l'interfacciamento con il backend, sviluppandone uno su misura, e rimpiazzare le chiamate HTTP con un rpc (come ad esempio GRPC), ed utilizzare un algoritmo di serializzazione dei dati, per ridurre sensibilmente la banda occupata per trasferire notevoli quantità di dati (come ad esempio Protobuf). Sono soddisfatto del risultato, di come abbiamo organizzato il lavoro e del gruppo, che nonostante alcune serie incombenze, è riuscito a portare a termine il lavoro.

4.1.3 Daniel Benazzi

Considero il risultato di Reega nel complesso soddisfacente, ma in retrospettiva avrei dovuto approcciare diversamente il progetto. Innanzitutto avrei studiato gli strumenti per tempo, dandomi modo di non usare il progetto come "banco di prova" ma di arrivarvi già preparato. Inoltre avrei prestato più attenzione all'applicazione di pattern, essendo giunto, attraverso la pianificazione, a soluzioni vicine ma non classificabili come veri e propri pattern. Fortunatamente i miei colleghi, con molta più esperienza di me, sono stati più che disponibili ad aiutare, anche se temo che questo li abbia un po' frenati al raggiungere una dimensione del progetto rimasta solo in potenza. Nonostante ciò abbiamo lavorato in modo coeso per raggiungere il risultato finale, spartendo i compiti e consultandoci spesso per le imminenti scelte implementative. Senza dubbio un progetto di questo tipo da la possibilità agli studenti di sviluppare un discreto livello di competenza nella programmazione e fornisce anche un primo approccio ad una programmazione diversa dal semplice esercizio di laboratorio.

Appendice A

Guida utente

Il software parte con una schermata di Login, qui si può eseguire il login attraverso due credenziali:

- User
  - Email: user@reega.it
  - Password: BC_PASSWORD
- Administrator
  - Email: admin@reega.it
  - Password: BC_PASSWORD

In basso è possibile passare alla schermata di Registrazione, qui è possibile creare un Utente con credenziali valide, è quindi necessario fornire un codice fiscale valido. Dopo aver creato un utente nella fase di registrazione è possibile fare il login con le credenziali appena create. Nel login è possibile specificare la volontà di salvare un token che permette di accedere all'applicazione la volta successiva senza utilizzare Email/Codice Fiscale e Password.

A.0.1 Utente

Se l'utente è appena stato creato vedrà solamente due bottoni in alto a sinistra per esportare i dati, che in questo caso risulteranno in file "nulli". Se invece l'utente è *user@reega.it* allora vedrà i dati dei suoi contratti a partire dal primo giorno del mese corrente; inizialmente tutti i contratti sono selezionati ma si possono deselezionare a piacere. Se si desiderano vedere i grafici di questi dati è necessario cliccare su uno dei servizi. Quando si è conclusa la visualizzazione è possibile uscire dall'applicazione attraverso la

chiusura della finestra oppure, nel caso in cui si voglia effettuare un altro login oppure anche eliminare il token per l'applicazione, premere il tasto di Logout posizionato in alto a destra.

A.0.2 Amministratore

L'amministratore, appena effettuato il login, vedrà subito i dati generali della piattaforma Reega dal primo giorno del mese ad oggi, anch'essi esportabili in formato CSV e JSON. L'amministratore però può anche ricercare un utente attraverso il suo nome, cognome, Codice Fiscale oppure può anche cercare uno specifico contratto attraverso l'indirizzo(anche parziale) oppure anche tutti i contratti di uno specifico utente. Effettuando una ricerca senza parola chiave, verranno restituiti tutti gli elementi disponibili. Dopo aver selezionato un utente è possibile vedere i suoi dati dal primo giorno del mese ad oggi, vedere i suoi dati anagrafici ed eventualmente aggiungere o rimuovere uno o più contratti. Eliminando la selezione corrente è possibile tornare a vedere i dati generali della piattaforma e, come dal lato utente, è possibile vedere i grafici di utilizzo per ogni servizio.

Appendice B

Esercitazioni di laboratorio

B.1 Marco Salomone

- Laboratorio 04: https://virtuale.unibo.it/mod/forum/discuss.php?d=62685#p101097
- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=62684#p101100
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p100921
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p100923
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p102765
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p103973
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=66753#p106755
- Laboratorio 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=66463#p106757

B.2 Daniel Benazzi

- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p101486