Lavoratori Stagionali

Ingegneria del Software - Laurea in Informatica

A.A. 2021/2022

Sommario

Specifiche

Si vuole progettare un sistema informatico di una agenzia che fornisce servizi di supporto alla ricerca di lavoro stagionale. I lavoratori interessati possono iscriversi al servizio, rivolgendosi agli sportelli dell'agenzia. Il sistema deve permettere la gestione delle anagrafiche e la ricerca di lavoratori stagionali, nei settori dell'agricoltura e del turismo.

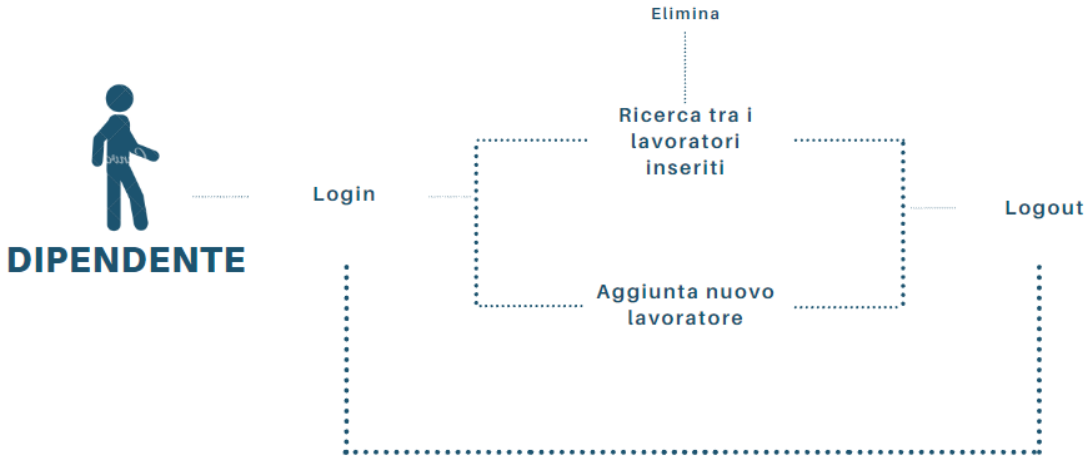
I responsabili del servizio, dipendenti dell'agenzia, inseriscono i dati dei lavoratori. Per ogni lavoratore vengono memorizzati i dati anagrafici (nome, cognome, luogo e data di nascita, nazionalità), indirizzo, recapito telefonico personale (se presente), email, le eventuali specializzazioni/esperienze precedenti (bagnino, barman, istruttore di nuoto, viticoltore, floricultore), lingue parlate, il tipo di patente di guida e se automunito. Sono inoltre memorizzati i periodi e le zone (comuni), per i quali il lavoratore è disponibile. Di ogni lavoratore si memorizzano anche le informazioni di almeno una persona da avvisare in caso di urgenza: nome, cognome, telefono, indirizzo email.

I dipendenti dell'agenzia devono autenticarsi per poter accedere al sistema e inserire i dati dei lavoratori. Il sistema permette ai dipendenti dell'agenzia di aggiornare le anagrafiche con tutti i lavori che i lavoratori stagionali hanno svolto negli ultimi 5 anni. Per ogni lavoro svolto vanno registrati: periodo, nome dell'azienda, mansioni svolte, luogo di lavoro, retribuzione lorda giornaliera. Per i dipendenti dell'agenzia si memorizzano i dati anagrafici, l'indirizzo email, il telefono e le credenziali di accesso (login e password).

Una volta registrate le informazioni sui lavoratori, il personale dell'agenzia può effettuare ricerche rispetto a possibili profili richiesti. In particolare, il sistema deve permettere ai dipendenti di effettuare ricerche per lavoratore, per lingue parlate, periodo di disponibilità, mansioni indicate, luogo di residenza, disponibilità di auto/patente di guida. Il sistema deve permettere di effettuare ricerche complesse, attraverso la specifica di differenti condizioni di ricerca (sia in AND che in OR).

Casi d'Uso

L'applicazione proposta è utilizzabile dal personale dipendente, provvisto di credenziali (fornite preventivamente dagli amministratori di sistema).
Il dipendente può autenticarsi e accedere a tutte le funzionalità dell'applicazione.

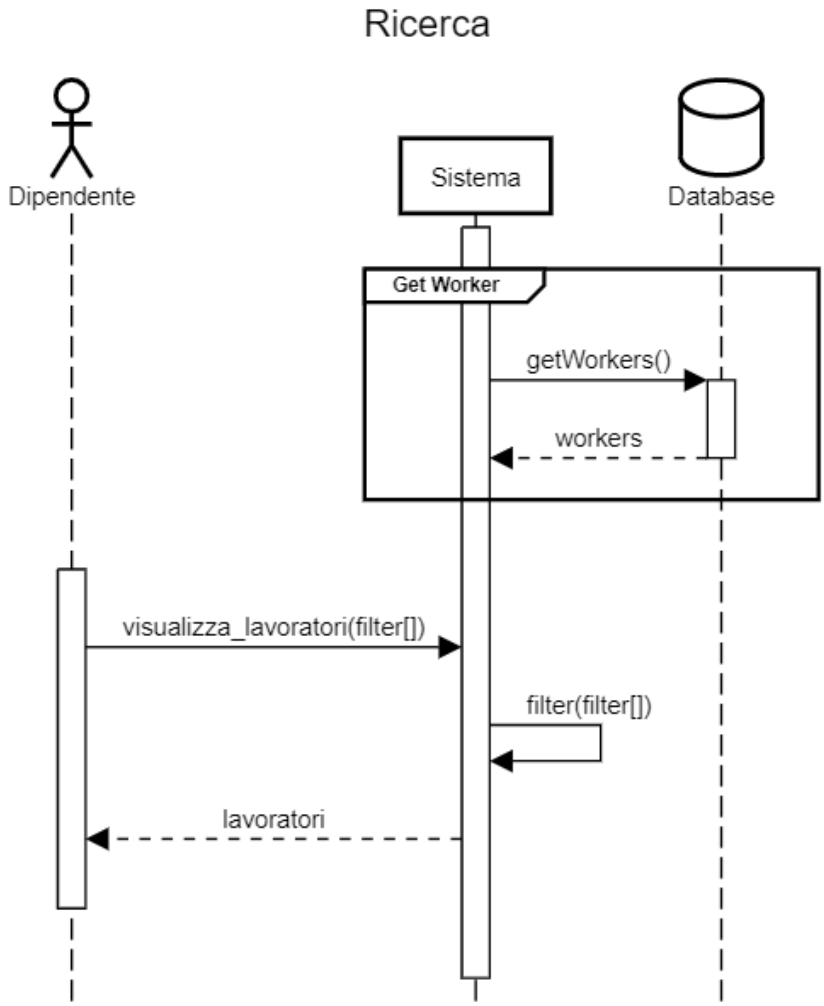


Ricerca

La ricerca può avvenire mediante una barra di testo tramite l'inserimento di parole chiave, e verrà effettuata tra nome e cognome dei lavoratori e le precedenti esperienze lavorative in modalità "AND".
In alternativa è possibile utilizzare dei filtri per ogni altro campo presente, possono inoltre essere applicati in modalità "AND" oppure "OR".

Attori	Dipendente
Precondizioni	Il dipendente deve essere autenticato
Passi	1. Il dipendente accede al sistema 2. Il dipendente è introdotto all'interfaccia per la ricerca 3. Il dipendente effettua la ricerca a. È possibile utilizzare dei filtri
Postcondizioni	

Diagramma di sequenza



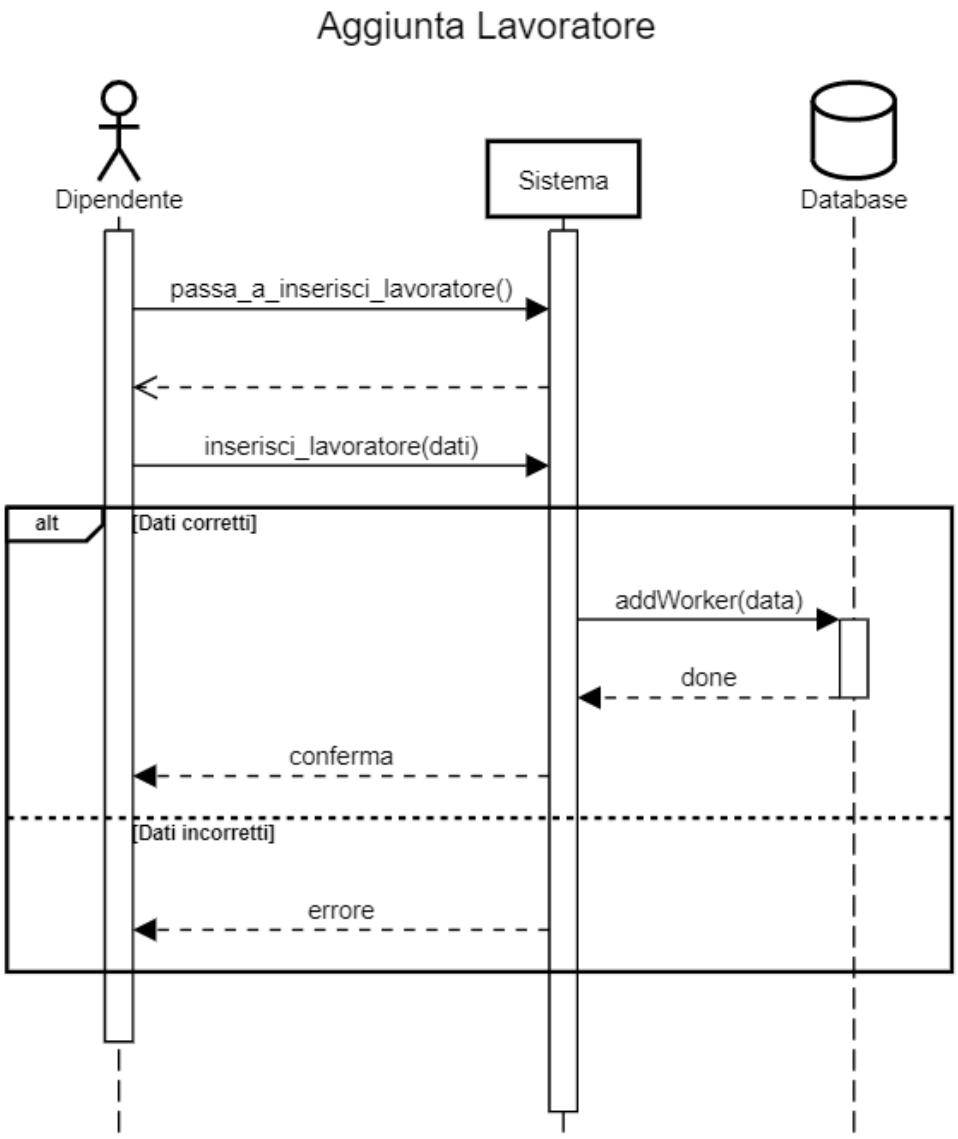
Inserimento nuovo Lavoratore

La creazione di un nuovo lavoratore al sistema avviene tramite un form.

Attori	Dipendente
Precondizioni	Il dipendente deve essere autenticato
Passi	1. Il dipendente accede al sistema 2. Il dipendente è introdotto all'interfaccia per la ricerca 3. Il dipendente accede all'interfaccia per l'aggiunta dei lavoratori 4. Il dipendente inserisce i dati relativi al lavoratore

	5. Il dipendente conferma l'aggiunta
Postcondizioni	Il nuovo lavoratore viene inserito nel database

Diagramma di sequenza



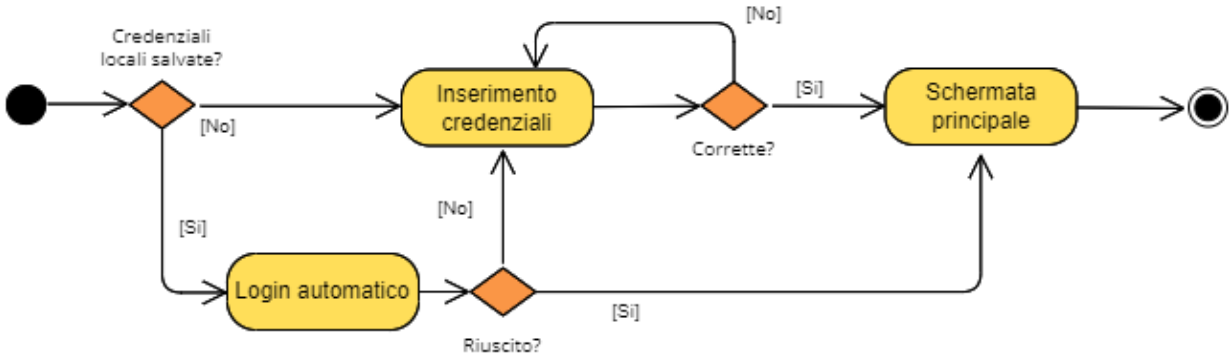
Cancellazione Lavoratore

La creazione di un nuovo lavoratore al sistema avviene tramite un form.

Attori	Dipendente
Precondizioni	Il dipendente deve essere autenticato
Passi	1. Il dipendente accede al sistema 2. Il dipendente è introdotto all'interfaccia per la ricerca 3. Il dipendente cerca il lavoratore da eliminare 4. Il dipendente elimina il lavoratore
Postcondizioni	Il lavoratore viene rimosso dal database

Diagrammi di Attività dei principali Use Cases

Autenticazione



Dipendente

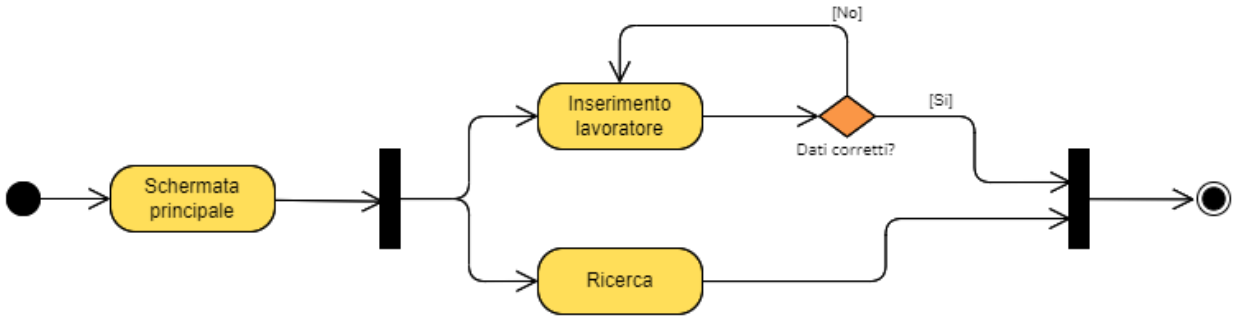


Diagramma di Sequenza

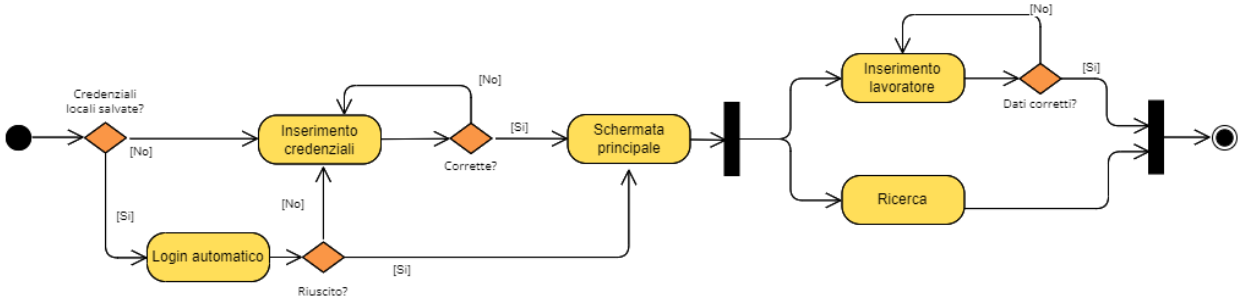


Diagramma delle Classi

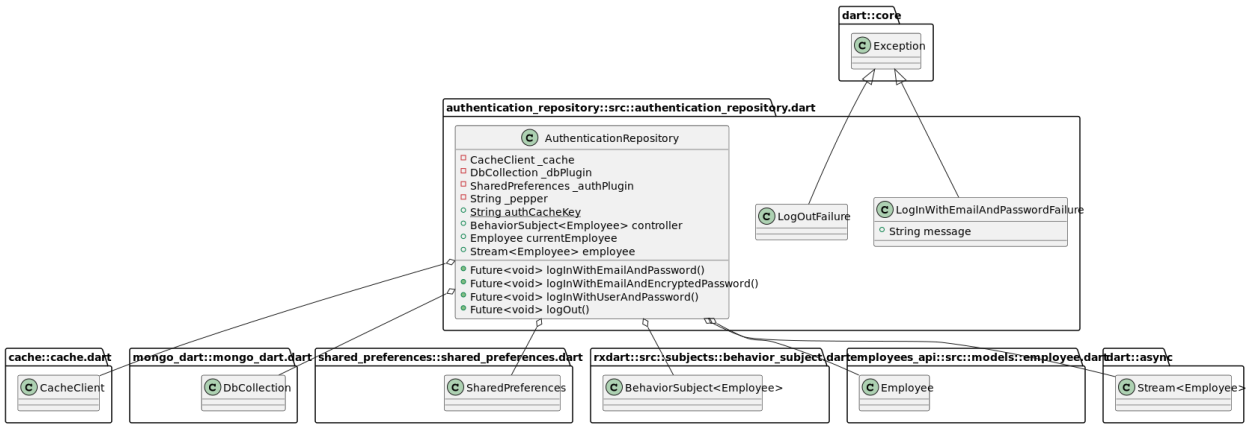


Figura 7: Package AutenticationRepository

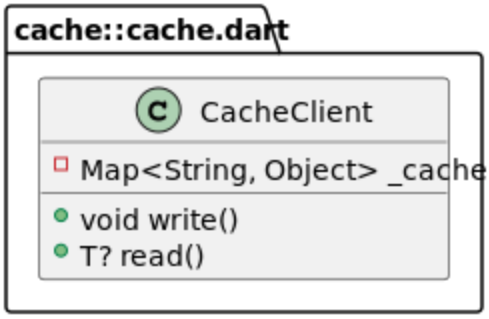


Figura 8: Package Cache

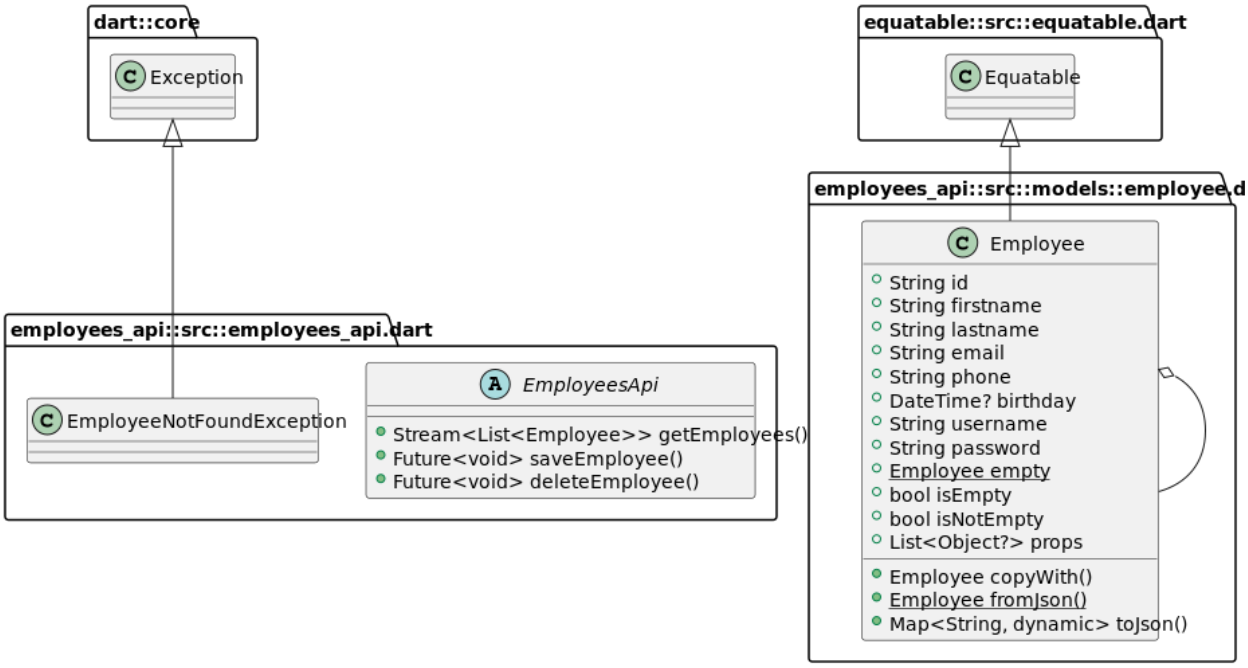


Figura 9: Package EmployeesApi

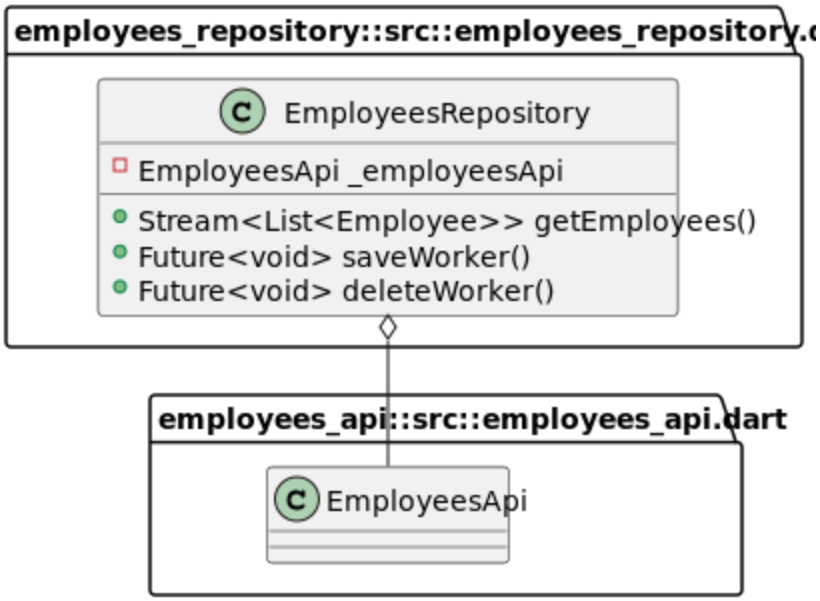


Figura 10: Package EmployeesRepository

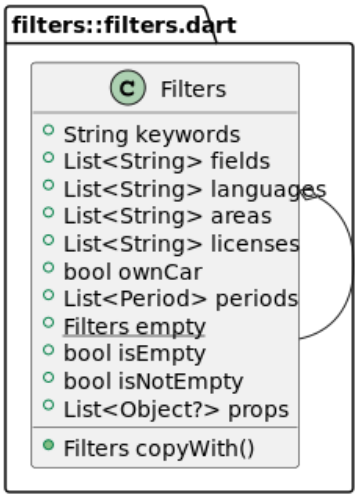


Figura 11: Package Filters

Figura 12: Package MongoDBEmployeeApi

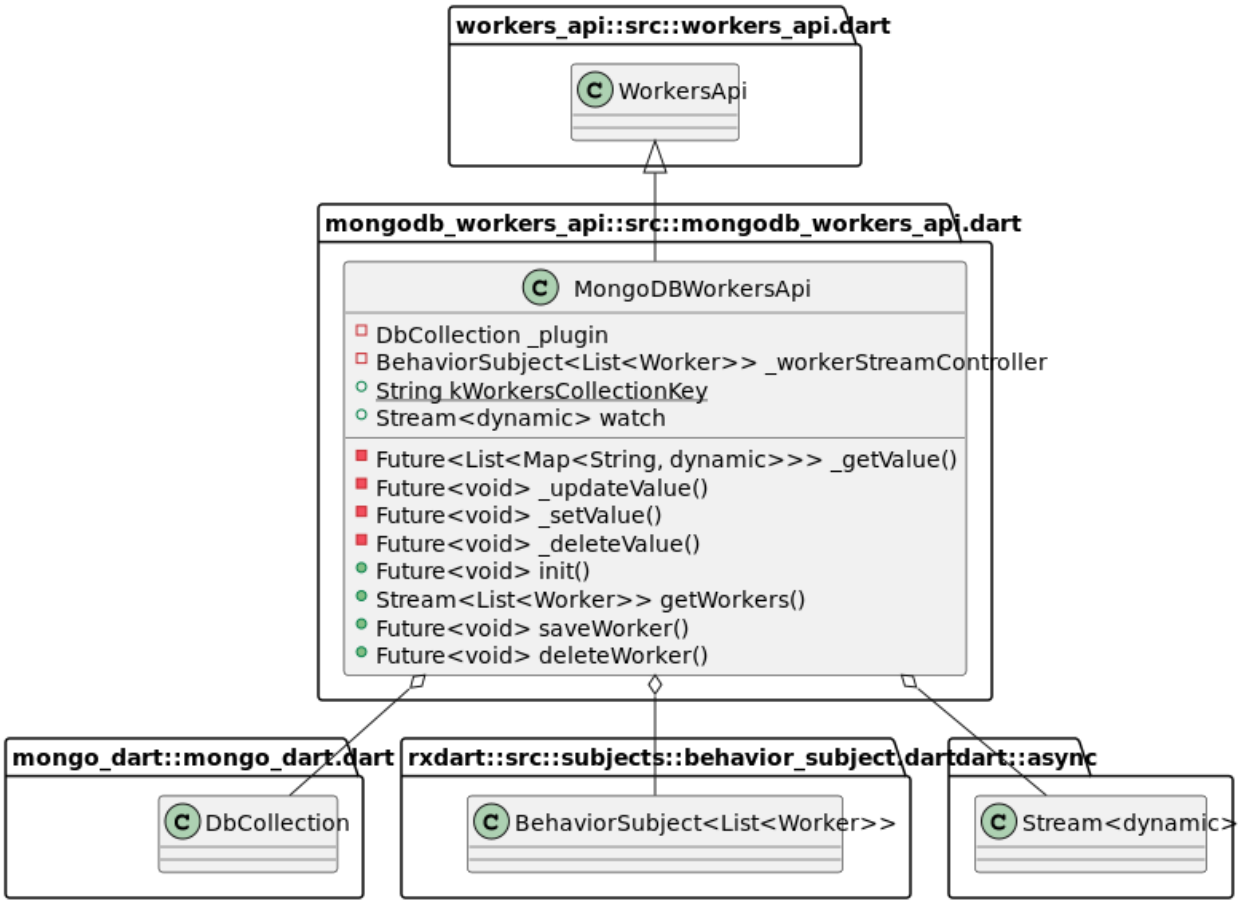


Figura 13: Package MongoDBWorkersApi

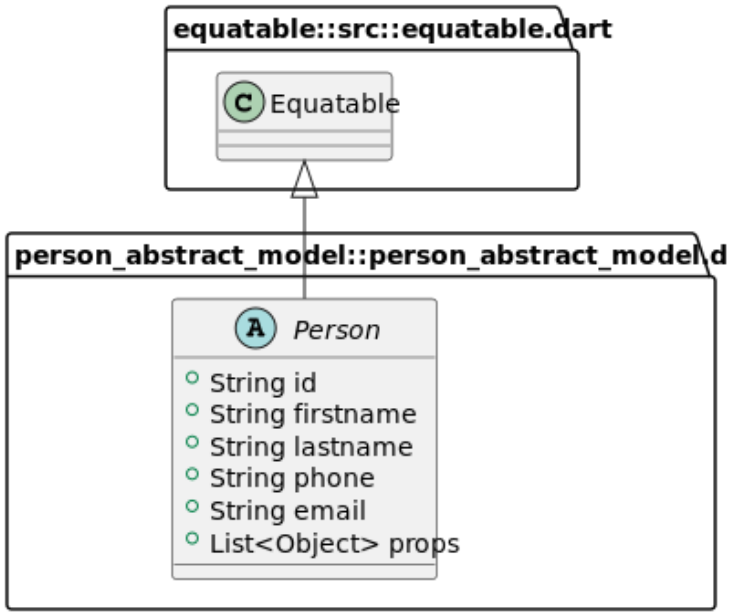


Figura 14: Package Person

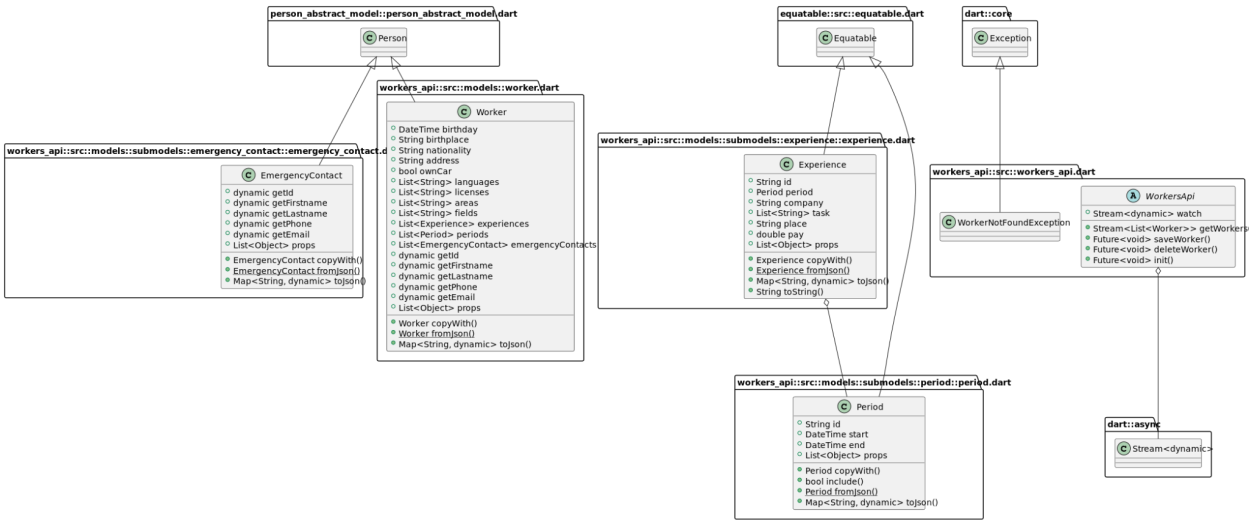


Figura 14: Package WorkersApi

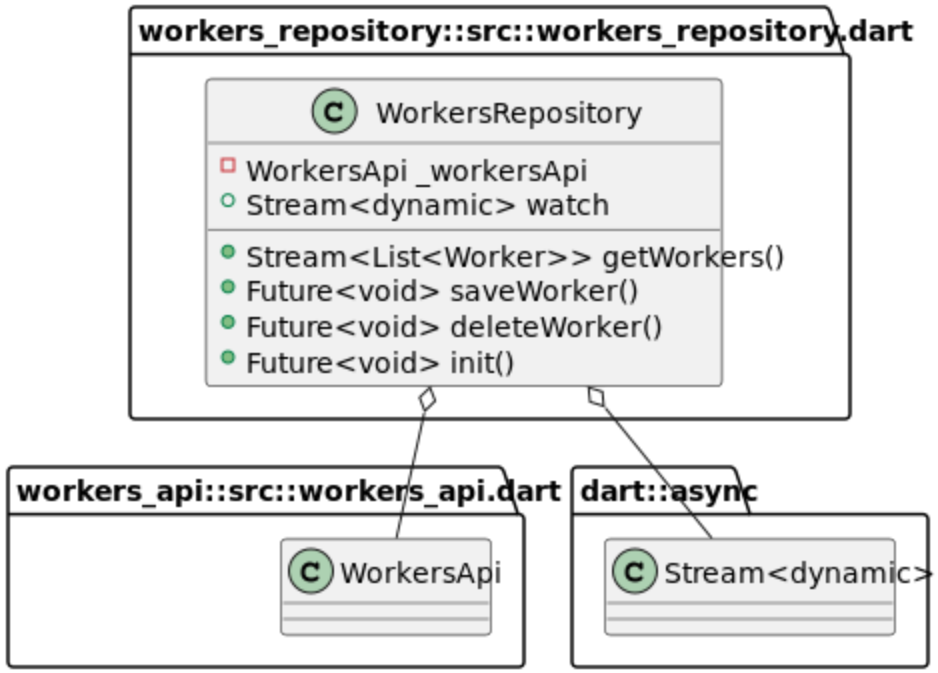


Figura 15: Package WorkersRepository

Sviluppo

Note sul processo di sviluppo

Il processo di sviluppo è stato essenzialmente di tipo *Agile* con approccio *Incrementale*. Questo è stato fatto con l'obiettivo di avere sempre un terreno solido sul quale lavorare. Dopo ogni modifica significativa è stata condotta una breve attività di test.

Prima di iniziare con lo sviluppo, si è condotta la fase di analisi dei requisiti, generando i relativi Use-Cases, i diagrammi di attività e la progettazione architetturale.

Per quanto riguarda l'implementazione, non sono state fatte grosse divisioni o piani di sviluppo programmatici. Si è seguito l'ordine prioritario di sviluppo andando ad aggiungere ciò che era ritenuto necessario..

Design Pattern

Per lo sviluppo di questa applicazione è stato utilizzato il design pattern BLoC[1]; concepito per consentire di utilizzare lo stesso codice indipendentemente dalla piattaforma.

BLoC Pattern

BLoC è stato progettato pensando a tre valori fondamentali:

- **Semplice**: facile da capire e può essere utilizzato da sviluppatori con diversi livelli di abilità;
- **Potente**: aiuta a realizzare applicazioni complesse scomponendole in componenti più piccoli;
- **Testabile**: testa facilmente ogni aspetto di un'applicazione;

BLoC Pattern si basa sul concetto di Programmazione Reattiva.

Programmazione Reattiva

"La programmazione reattiva è la programmazione con flussi di dati asincroni."

Con il concetto di programmazione reattiva, l'applicazione:

- diventa completamente asincrona;
- è strutturata attorno alla nozione di Stream e listener;
- quando succede qualcosa in qualsiasi parte (un evento, una variabile modificata...) viene inviata una notifica a uno Stream;
- se "qualcuno" ascolta quello Stream, verrà avvisato e intraprende azioni appropriate, **qualunque sia la sua posizione nell'applicazione**.

[1] BLoC: acronimo di **B**usiness **Lo**gic **C**omponent

Quindi ogni Widget[2] **non** ha più bisogno di sapere:

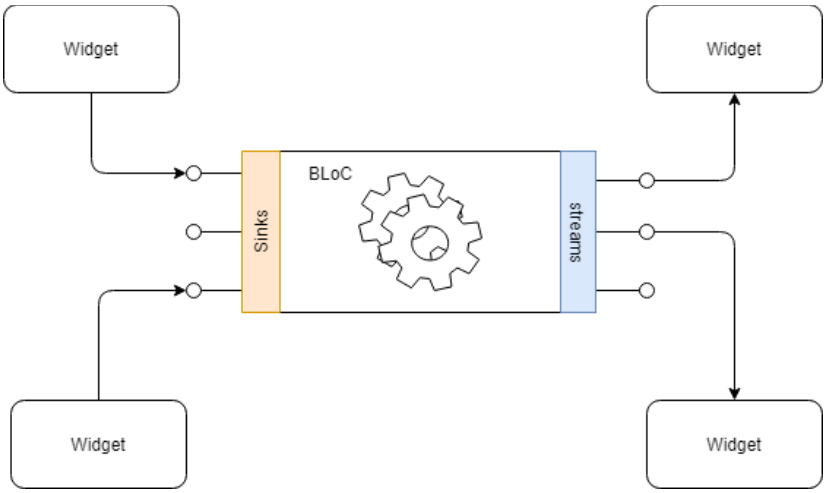
- cosa accadrà dopo aver inviato la notifica;
- chi potrebbe utilizzare le informazioni (nessuno, uno o più Widget...);
- dove le informazioni potrebbero essere utilizzate (da nessuna parte, stessa schermata, un'altra, più...);
- quando queste informazioni potrebbero essere utilizzate (subito, dopo alcuni secondi, mai...).

Regole di BLoC Pattern

Business Logic dovrebbe essere:

- essere diviso in uno o più *BLoC*;
- essere rimosso il più possibile dal livello di Presentazione;
- fare affidamento sull'uso esclusivo di Stream sia per l'input (Sinks) che per l'output (stream);
- rimanere indipendente dalla piattaforma;
- rimanere indipendente dall'ambiente;

Funzionamento di BLoC



1. Widget inviano eventi a BLoC tramite *Sinks*;
2. Widget vengono notificati dal BLoC tramite *stream*;

I widget non hanno interesse di come è implementato il Business Logic.

Vantaggi della separazione di Business Logic dalla UI

- è possibile modificare Business Logic senza impatto sull'applicazione;
- è possibile modificare l'interfaccia utente senza impatto sul Business Logic;

[2] Widget: gerarchia delle Classi centrale del framework Flutter.

- è molto più semplice testare Business Logic;
- viene aggiornata l'interfaccia grafica solo quando strettamente necessario (prestazione migliori);

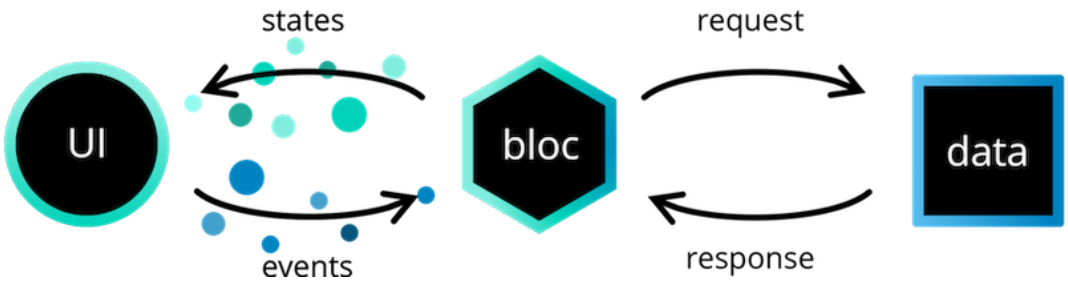
Cubit Pattern

Cubit Pattern è una versione minimale di BLoC Pattern che abbandona il concetto di Eventi e semplifica il modo di emettere stati.

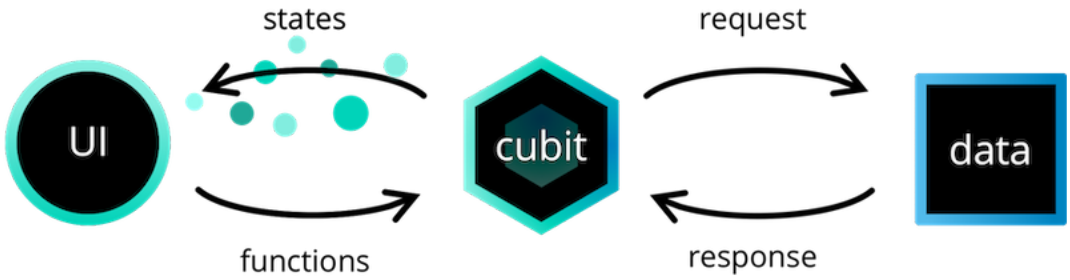
BLoC Provider

Per semplificare l'implementazione del design pattern BLoC è stata utilizzata la libreria BLoC Provider che prevedere un architettura di sviluppo che separa l'applicazione in tre livelli:

- **Presentazione** (UI): gestisce l'interfaccia grafica e gli input utente dell'applicazione;
- **Business Logic** (bloc): risponde agli input del livello di presentazione con nuovi stati. Può dipendere da uno o più *repository* per recuperare i dati necessari per creare lo stato dell'applicazione;
- **Data**: recupera/manipola i dati da una o più fonti
    - **Repository**: wrapper per uno o più Data Provider con cui comunica il livello Business Logic;
    - **Data Provider**: fornisce dati grezzi;



Con Cubit:



Database

Per lo sviluppo del sistema si è scelto un database *non relazionale*, affidando la gestione al DBMS[3] MongoDB. La struttura del database è stata creata con approccio *Plan-driven* durante la prima fase di analisi dei requisiti.

Di seguito il JSON Schema[4] della *collezione*[5] dei dipendenti:

```
{¬
  "$schema": "http://json-schema.org/draft-04/schema#",¬
  "type": "object",¬
  "properties": {¬
    "id": {¬
      "type": "string",¬
      "description": "The unique identifier for a employee"¬
    },¬
    "firstname": {¬
      "type": "string"¬
    },¬
    "lastname": {¬
      "type": "string"¬
    },¬
    "birthday": {¬
      "type": "string"¬
    },¬
    "email": {¬
      "type": "string"¬
    },¬
    "phone": {¬
      "type": "string"¬
    },¬
    "username": {¬
      "type": "string"¬
    }¬
    "password": {¬
      "type": "string"¬
    },¬
    "salt": {¬
      "type": "string",¬
      "description": "Random string used as additional input for password encryption"¬
    }¬
  },¬
  "required": [¬
    "id",¬
    "firstname",¬
    "lastname",¬
    "birthday",¬
    "email",¬
    "password",¬
    "phone",¬
    "username",¬
    "salt"¬
  ]¬
}
```

[3] DBMS: acronimo di **D**ata**b**ase **M**anagement **S**ystem
[4] JSON Schema: vocabolario che consente di annotare e validare documenti JSON
[5] Collezione: oggetti non necessariamente strutturati

Attività di Test

Development Testing

Durante lo sviluppo sono stati provati diversi input di dati, sia corretti che errati, per verificare il comportamento del software a tali input.

Alcuni test svolti:

- **Verifica dell'autenticazione**: i dati errati vengono respinti, ed a seguito dei dati corretti all'utente viene mostrata la schermata iniziale;
- **Database vuoto**
- **Verifica di input errati e vuoti**
- **Inserimento di un lavoratore**
- **Modifica di un lavoratore**
- **Eliminazione di un lavoratore**

Unit Testing

Per velocizzare le fasi di test durante lo sviluppo sono state testate le varie funzionalità dell'applicazione, senza avviare l'interfaccia grafica, grazie all'utilizzo di Mockito, uno dei framework più popolari per la realizzazione di *oggetti mock*[6], che consente di generare un *mock* a partire sia da una interfaccia che da un classe semplicemente dichiarandone il comportamento.

User Testing

Al termine dello sviluppo il software è stato sottoposto ad un test da parte di alcuni dipendenti dell'agenzia. Non è stato in alcun modo spiegato il funzionamento del software, per non influenzare i risultati. Inoltre è stato possibile individuare nuove funzionalità per migliorare il sistema tra cui la possibilità di cercare con *precedenti esperienze lavorative* come parole chiave nella barra di ricerca.

[6] Oggetti mock: oggetti simulati che riproducono il comportamento di quelli reali in modo controllato