Laurea Triennale in Informatica - Università degli Studi di Salerno

Corso di *Ingegneria del Software* - Prof.ssa F. Ferrucci, Prof. F. Palomba

REPORT.IT

ITALIAN DISCRIMINATION COMMUNITY

Object Design Document

Report.it

Riferimento	Report.it_ODD_V1.0
Versione	1.0
Data	15/01/2023
Destinatario	F. Ferrucci
Presentato da	Riccardo Kevin Ferraris, Nicola Frugieri, Alberto Genovese, Arturo Gentile, Marisa La Sorda, Leonardo Schiavo
Approvato da	Simona Grieco, Maria Concetta Schiavone

Team composition

Ruolo	Nome	Acronimo	Contatti
Top Management	Filomena Ferrucci	FF	f.ferrucci@unisa.it
Project Manager	Simona Grieco	SG	s.grieco13@studenti.unisa.it
Project Manager	Maria Concetta Schiavone	MCS	m.schiavone29@studenti.unisa.it
Team Member	Riccardo Kevin Ferraris	RKF	r.ferraris1@studenti.unisa.it
Team Member	Nicola Frugieri	NF	n.frugieri@studenti.unisa.it
Team Member	Alberto Genovese	ALG	a.genovese42@studenti.unisa.it
Team Member	Arturo Gentile	ARG	a.gentile38@studenti.unisa.it
Team Member	Marisa La Sorda	MLS	m.lasorda@studenti.unisa.it
Team Member	Leonardo Schiavo	LS	l.schiavo15@studenti.unisa.it

Sommario

Revision History

Data	Versione	Descrizione	Autori
14/12/2022	0.1	Prima stesura del documento	[TEAM MEMBER]
15/12/2022	0.2	Introduzione e Glossario	RKF, MLS
15/12/2022	0.3	Stesura sezione Package	ALG, ARG, NF
21/12/2022	0.4	Stesura Class Interfaces, Class Diagram ristrutturato	ALG, ARG, NF
21/12/2022	0.5	Componenti terzi	ALG
23/12/2022	0.6	Design Pattern	ALG
15/01/2023	1.0	Revisione finale documento	[TEAM MEMBER]

1. Introduzione

Lo scopo del sistema Report.it è quello di fornire un accesso rapido e sicuro ai cittadini vittime di discriminazione ai sistemi di aiuto, snellendo la procedura burocratica attualmente esistente per l'inoltro di una denuncia di questo tipo e proponendosi come ponte comunicativo tra cittadino, forze dell'ordine e centralino CUP.
In questa sezione del presente documento verranno illustrate le linee guida per la scrittura del codice, elencate le definizioni, gli acronimi e le abbreviazioni che verranno usate in tutto il documento ed infine i riferimenti e link utili.

1.1 Linee Guida per la Scrittura del Codice

In questa sezione del documento verranno illustrate di seguito le regole da rispettare durante l'implementazione del sistema. È di seguito riportata una lista delle convenzioni utilizzate nella stesura del codice, con link ufficiali inclusi.

Documentazione ufficiale:

- **Dart:** https://dart.dev/guides
- **Flutter:** https://docs.flutter.dev/
- **Firebase:** https://firebase.google.com/docs

1.2 Definizioni, acronimi e abbreviazioni

Elenco delle definizioni, acronimi ed abbreviazioni utilizzate all'interno del documento:

- Package: raggruppamento di classi, interfacce o file correlati;
- Design Pattern: template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità;
- UpperCamelCase: pratica che prevede la scrittura di parole composte o frasi unendole tra loro, lasciando le loro iniziali maiuscole;
- lowerCamelCase: pratica che prevede la scrittura di parole composte o frasi unendole tra loro, in modo tale che le parole in mezzo alla frase abbiano l'iniziale maiuscola;
- Application Layer: nel pattern Three-layer rappresenta la parte che si occupa della logica di business dell'applicazione

- Presentation Layer: nel pattern Three-layer rappresenta la parte che si occupa della logica di visualizzazione dell'applicazione
- Data Layer: nel pattern Three-layer rappresenta la parte che si occupa dell'interazione con il database

1.3 Riferimenti e Link Utili

Elenco dei riferimenti e link utili ad altri documenti, utili durante la lettura:

- Statement of Work (SOW)
- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Test Plan (TP)
- Test Case Specification (TCS)
- Matrice di tracciabilità

2. Packages

Questa sezione del documento espone la divisione del progetto in package sulla base dell'architettura Three Layer definita nel System Design. La directory del progetto è stata scelta seguendo quella standard del framework Flutter:

- .dart_tool, il package repository generato da Flutter
- .vscode, contiene i setting di VSCode
- fonts, contiene i fonts usati nel sistema
- Android, contiene i file necessari per far funzionare l'app su Android
- Assets, contiene i vari assets dell'app, come immagini o icone
- Build, contiene i file necessari per far funzionare l'app su vari dispositivi
- iOS, contiene i file necessari per far funzionare l'app su iOS
- lib, contiene i file sorgente
    - ***Presentation***, contiene le classi dart relative al Presentation Layer
        - Pages, contiene le classi relative alle pagine dell'applicazione
        - Widgets, contiene le classi relative ai widgets usati nelle pagine
    - ***Application***, contiene le classi relative all'Application Layer

- Repository, contiene le classi relative ai controller per collegare l'Application e il Data Layer
- Entities, contiene le classi relative alle Entità del sistema

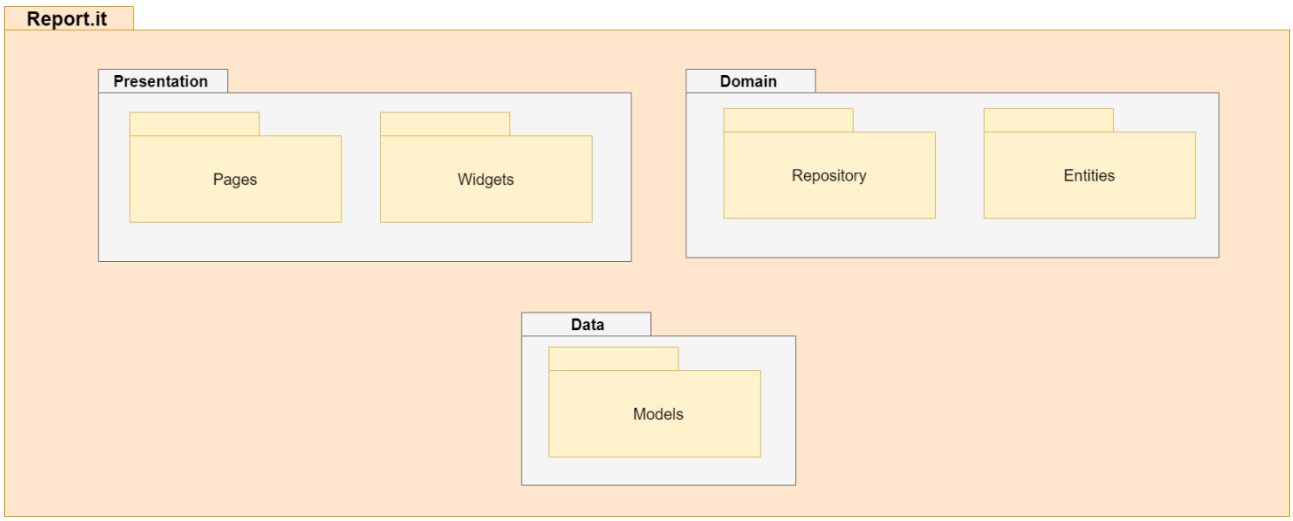
- ***Data***, contiene le classi relative al Data Layer
  - Models, contiene i DAO delle entità del Sistema

- Test, contiene le classi relative al testing
- Web, contiene i file necessari per far funzionare l'app sul browser

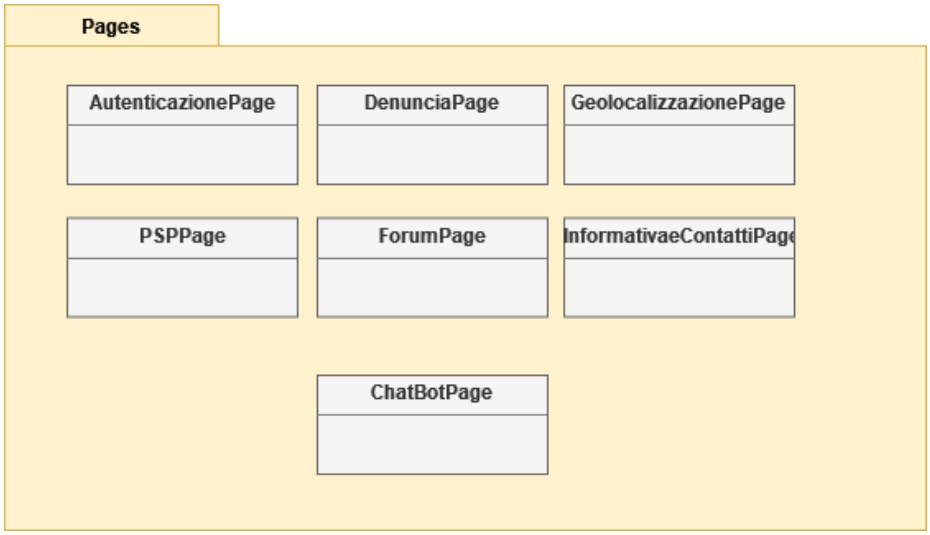
Package Report.it

Qui mostreremo la struttura dei package del sistema, questa divisione è stata scelta per 2 motivi principalmente:

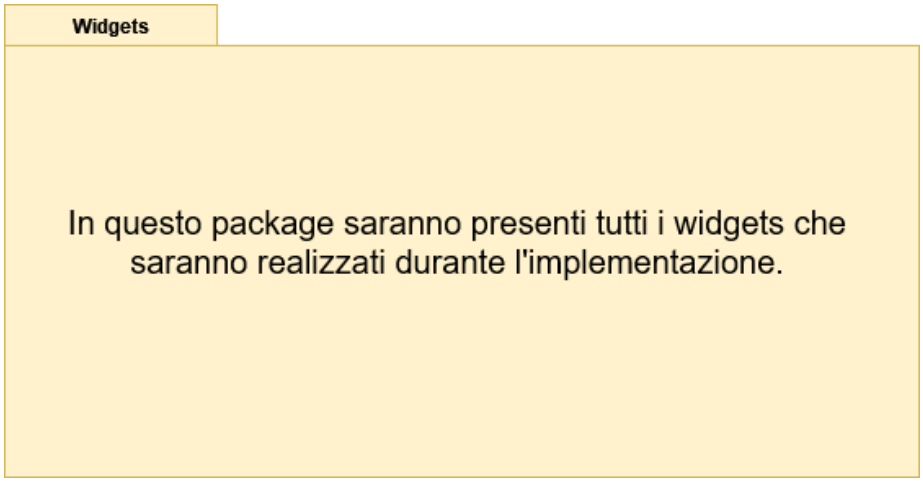
1. dividere le classi seguendo l'architettura scelta nel System Design in modo da avere le classi con responsabilità simili nella stessa posizione;
2. dividere il collegamento al database con le effettive query realizzate su quest'ultimo, in modo da rendere il codice non direttamente dipendente dal tipo di driver usato per il collegamento.



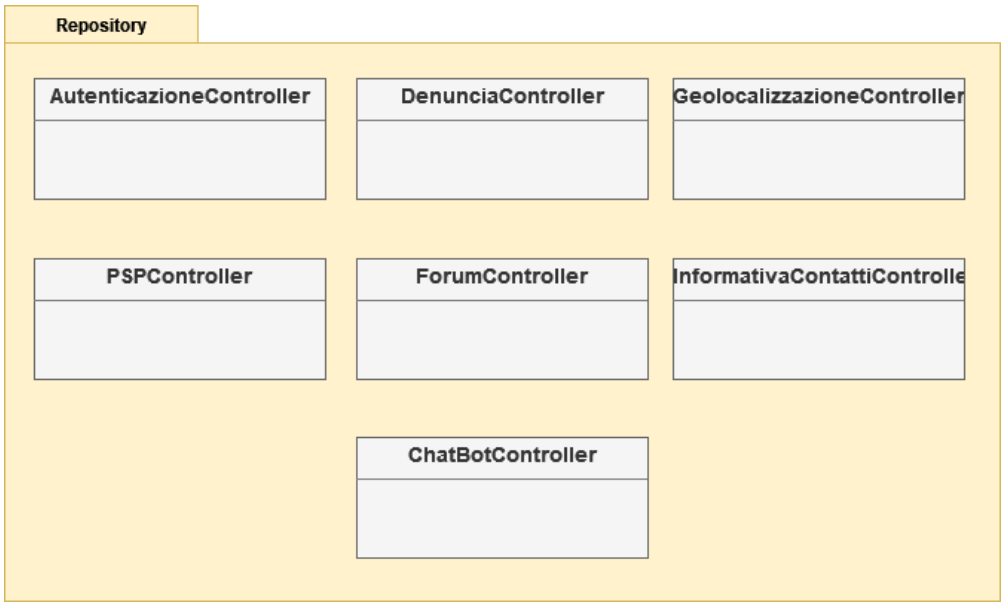
Package Pages



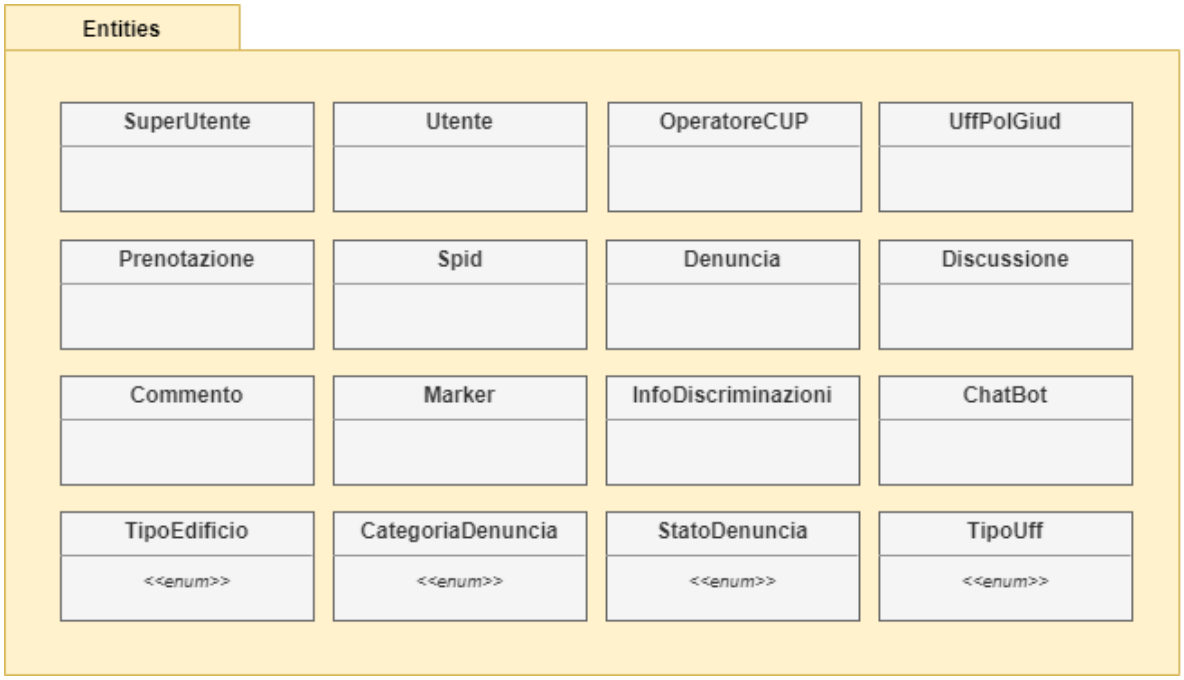
Package Widgets



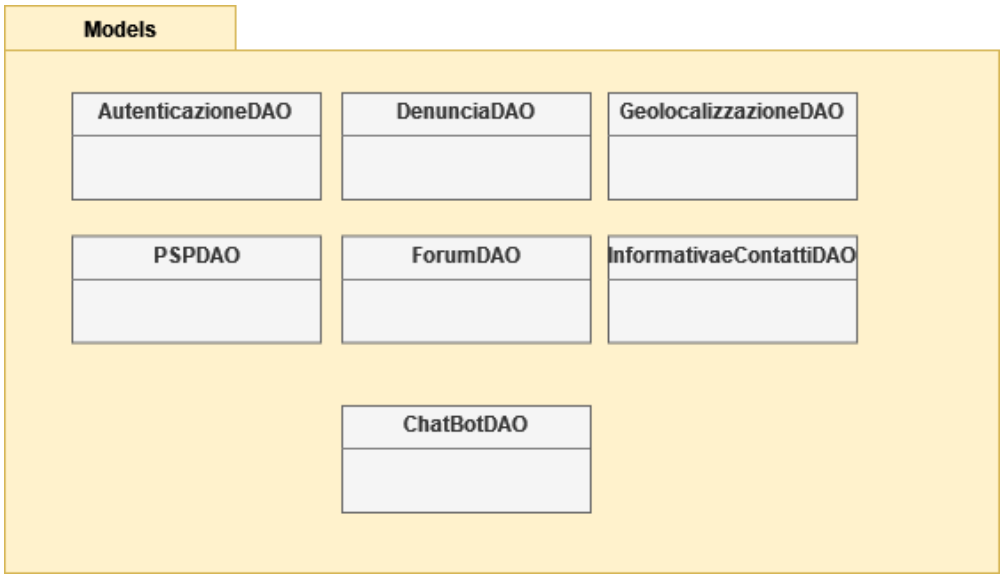
Package Repository



Package Entities



Package Models



3. Class Interfaces

In questa sezione vengono mostrare alcune interfacce del sistema, una per ogni package sopra definito.

Package Pages

lib.Presentation.Pages.DenunciaPage

Nome Classe	**DenunciaPage**
Descrizione	Questa classe permette di visualizzare la pagina relativa al modulo di denuncia con tutte le sue funzionalità
Metodi	+ createState(): State<StatefulWidget>
Invariante di classe	/

Nome Metodo	**+ createState() : State<StatefulWidget>**
Descrizione	Questo metodo permette di creare la pagina relativa al modulo gestionale della denuncia
Pre-condizione	/
Post-condizione	/

Package Repository

lib.Application.Repository.DenunciaController

Nome Classe	**DenunciaController**
Descrizione	Questa classe gestisce le varie funzionalità della denuncia e permette il collegamento tra il back-end e il front-end
Metodi	+inoltraDenuncia(Denuncia d): Boolean, +accettaDenuncia(String idDenuncia, String idUfficiale):Boolean,

	+cambiaStatoDenuncia(String idDenuncia, StatoDenuncia nuovoStato) +visualizzaDenunceUtente(): List<Denuncia>, +visualizzaDenunceUff(): List<Denuncia> +visualizzaDenunceByStato(StatoDenuncia stato):List<Denuncia>
Invariante di classe	/

Nome Metodo	**+inoltraDenuncia (Denuncia d): Boolean**
Descrizione	Questo metodo permetto all'utente di inoltrare una nuova denuncia,
Pre-condizione	UtenteLoggato == Utente
Post-condizione	/

Nome Metodo	**+accettaDenuncia (String idDenuncia, String idUfficiale): Boolean**
Descrizione	Questo metodo permette di far accettare ad un ufficiale la denuncia di un utente
Pre-condizione	UtenteLoggato == UffPolGIud
Post-condizione	/

Nome Metodo	**+cambiaStatoDenuncia (String idDenuncia, StatoDenuncia nuovoStato): Boolean**
Descrizione	Questo metodo permette di cambiare lo stato di una denuncia.
Pre-condizione	UtenteLoggato == UffPolGIud
Post-condizione	/

Nome Metodo	**+visualizzaDenunceUtente(): List<Denuncia>**
Descrizione	Questo metodo ritorna una lista con tutte le denunce inoltrate dall'utente loggato.
Pre-condizione	UtenteLoggato !=null
Post-condizione	/

Nome Metodo	**+visualizzaDenunceUff(): List<Denuncia>**
Descrizione	Questo metodo ritorna una lista con tutte le denunce accettate dall'UffPolGiud
Pre-condizione	UtenteLoggato == UffPolGlud
Post-condizione	/

Nome Metodo	**+visualizzaDenunceByStato (StatoDenuncia stato): List<Denuncia>**
Descrizione	Questo metodo ritorna una lista con tutte le denunce in base allo stato
Pre-condizione	UtenteLoggato !=null
Post-condizione	/

Package Entities

lib.Application.Entity.Denuncia

Nome Classe	**Denuncia**
Descrizione	Questa classe rappresenta l'entità relativa alla denuncia
Metodi	+getters +setters +fromMap(map) : Denuncia

	+fromJson(Map<String,dynamic>json): Denuncia +toMap() : Map<String, dynamic>
Invariante di classe	/

Nome Metodo	**getters**
Descrizione	Questi metodi restituiscono l'attributo corrispondente della denuncia
Pre-condizione	/
Post-condizione	/

Nome Metodo	**setters**
Descrizione	Questi metodi permettono di modificare il corrispondente attributo della denuncia
Pre-condizione	/
Post-condizione	denuncia.attributo = newValue

Nome Metodo	**+toMap(): Map<String, dynamic>**
Descrizione	Questo metodo restituisce l'entity sottoforma di mappa json
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+fromJson(Map<String,Dynamic>json): Denuncia**
Descrizione	Questo metodo restituisce l'oggetto relativo al json passato come argomento
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+fromMap(map): Denuncia**
Descrizione	Questo metodo restituisce l'oggetto relativo alla Mappa<String, dynamic> passata come parametro
Pre-condizione	/
Post-condizione	/

Package Models

lib.Data.Models.DenunciaDAO

Nome Classe	**DenunciaDAO**
Descrizione	Questa classe è il metodo di accesso al database relativo al modulo gestionale della denuncia
Metodi	+retrieveByUtente (String id): List<Denuncia> +retrieveAll() : List<Denuncia> +retrieveByID(String id):Denuncia +retrieveByUff(String idUff): List<Denuncia> +retriveByStato(StatoDenuncia stato):List<Denuncia>
Invariante di classe	/

Nome Metodo	**+retrieveByUtente(String idUtente) : List<Denuncia>**
Descrizione	Questo metodo restituisce la lista delle Denuncie relative ad un singolo utente
Pre-condizione	/
Post-condizione	/

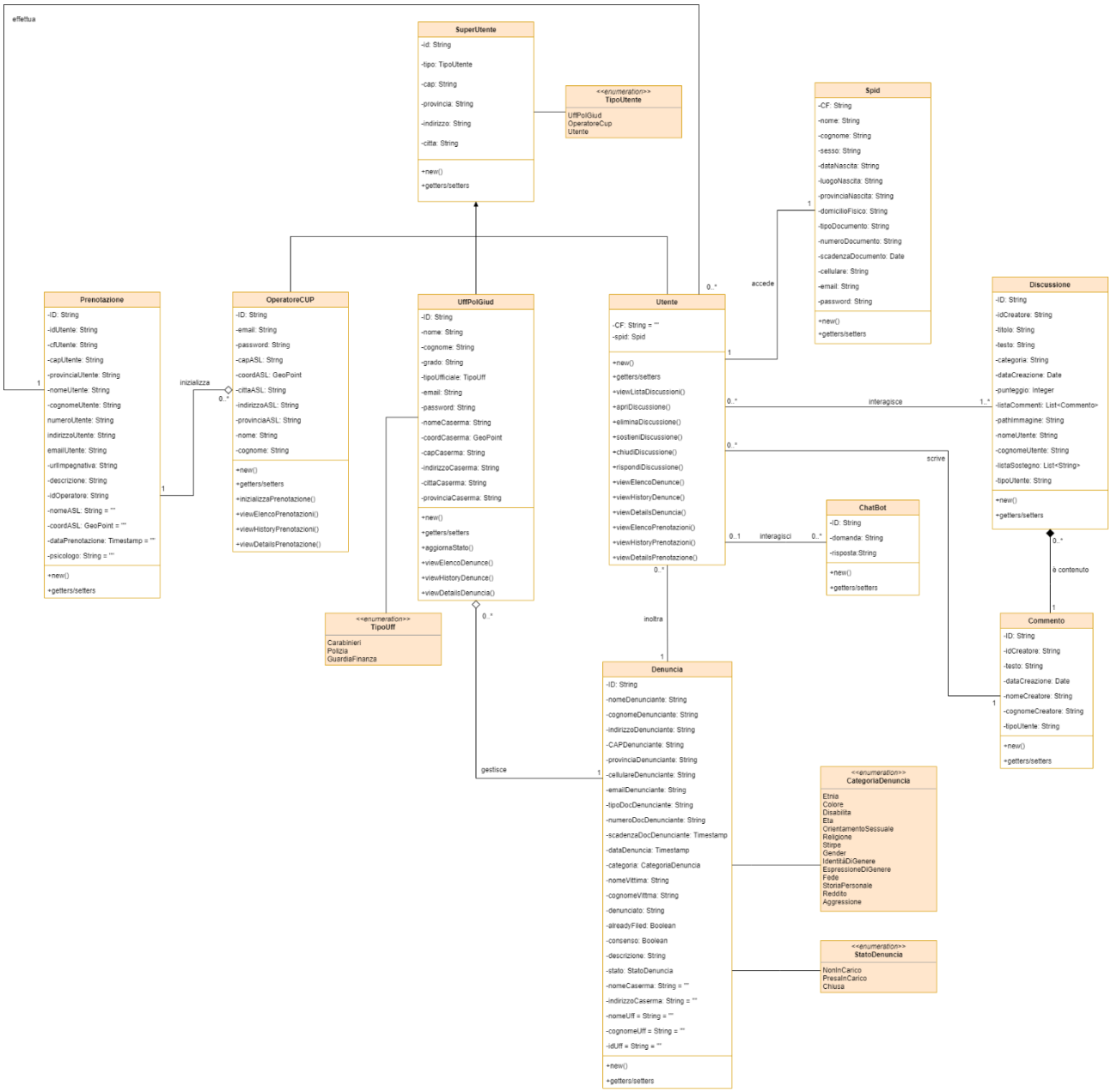
Nome Metodo	**+retrieveAll () : List<Denuncia>**
Descrizione	Questo metodo restituisce una lista con tutte le denunce presenti sul database
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+retrieveByID(String id): Denuncia**
Descrizione	Questo metodo restituisce l'oggetto Denuncia, col campo "ID" uguale all'id passato come stringa
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+retrieveByUff (String idUff): List<Denuncia>**
Descrizione	Questo metodo restituisce una lista contenente tutti gli oggetti Denuncia relativi ad un particolare UffPolGiud.
Pre-condizione	/
Post-condizione	/

Nome Metodo	**+retrieveByStato(StatoDenuncia): List<Denuncia>**
Descrizione	Questo metodo restituisce l'oggetto Denuncia, col campo "Stato" uguale allo stato passato come stringa
Pre-condizione	/
Post-condizione	/

4. Class Diagram Ristrutturato



5. Elementi di Riuso

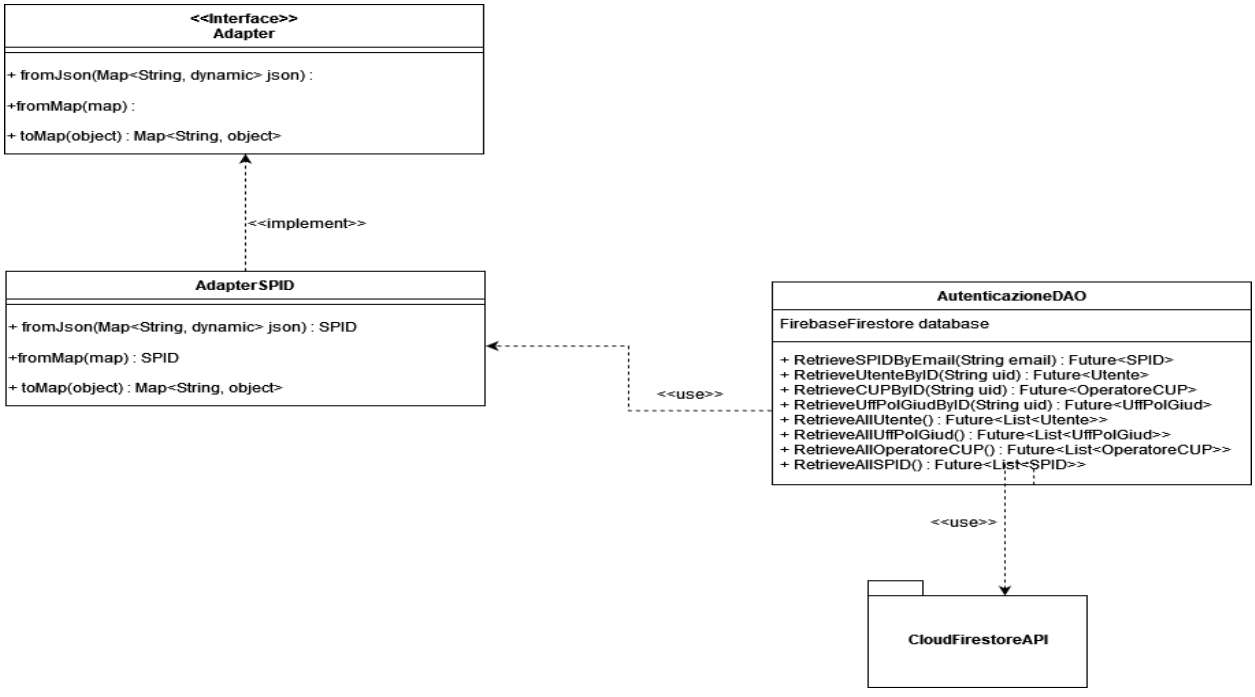
5.1 Design Pattern usati

All'interno del sistema sono stati usati dei pattern per risolvere alcuni problemi di programmazione e per rendere il codice più comprensibile.

Adapter

Il sistema usa Firestore per la gestione dei dati persistenti, un database NoSQL che salva i dati in formato Json.
Il nostro sistema, essendo realizzato in Dart, usa un modello ad oggetti. Per risolvere questa discrepanza, è stato usato il design Pattern Adapter che permette l'accesso ai dati del database come se fossero degli oggetti Dart.

Il design pattern è realizzato tramite l'uso di 3 metodi definiti in un'interfaccia comune che vengono poi implementati in relative classi per ogni entity. Questi metodi sono fromJson, fromMap, toJson; il primo permette di trasformare un json Dart, quindi una Map<String, dynamic>, in un oggetto entity, il secondo permette di trasformare qualsiasi mappa in un oggetto entity, mentre il terzo permette di trasformare un oggetto entity in un json Dart.



5.2 Componenti terzi

All'interno del sistema sono usati alcuni componenti COTS necessarie al corretto funzionamento dei sottosistemi, queste sono:

1. **Firebase**: Piattaforma per la realizzazione di app creata da Google con integrato il firestore database, quest'ultimo utilizzato nel sistema per la gestione dei dati persistenti
2. **FlutterFire**: Un plugin che permette di usare i sistemi di firebase in app realizzate in Flutter
3. **API Google Maps**: Queste API sono utilizzate nel sistema per le funzionalità di geolocalizzazione del relativo sottosistema

6. Glossario

Sigla/termine	Definizione
Package	Raggruppamento di classi ed interfacce.
DAO	Data Access Object. Pattern architetturale per la gestione della persistenza.
Controller	Classe che gestisce le richieste del client.
Model	Insieme dei metodi che permettono l'accesso ai dati.
Three-layer	Modello di organizzazione del codice applicativo basato sulla separazione delle funzionalità logiche
Adapter	Design Pattern di tipo strutturale che permette di collegare tra loro librerie con interfacce non compatibili
Componenti COTS	Con il termine componenti COTS ci si riferisce a componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti (https://www.wikiwand.com/it/COTS).