Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software* - Prof C. Gravino

Object **D**esign **D**ocument

Outfit**M**aker

Riferimento	
Versione	1.0
Data	1/12/2023
Destinatario	Studenti di Ingegneria del Software 2023/24
Presentato da	Avallone Francesca Buongiorno Rocco Pio Letterese Michele
Approvato da	

Sommario

DATA	VERSIONE	DESCRIZIONE	AUTORI
15/12/2023	0.1	Prima stesura	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
15/12/2023	0.2	Introduzione e Glossario	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
16/12/2023	0.3	Stesura sezione Package	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
21/12/2023	0.4	Stesura Class Interfaces, Class diagram ristrutturato	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
23/12/2023	0.5	Componenti terzi	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
10/01/2024	0.6	Design Pattern	Francesca Avallone Rocco Pio Buongiorno Michele Letterese
17/01/2024	1.0	Revisione finale documento	Francesca Avallone Rocco Pio Buongiorno Michele Letterese

1. Introduzione

Lo scopo principale del sistema OutfitMaker è quello di fornire agli utenti dei consigli di abbigliamento personalizzati in base alle loro preferenze e al contesto. Adattare le raccomandazioni di outfit in base al gusto personale dell'utente considerando tipologie preferiti, colori preferiti, e altre preferenze individuali.

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 vengono delineate le regole da seguire durante l'implementazione del sistema, con particolare attenzione alle convenzioni di codifica. Di seguito è riportato un elenco delle convenzioni utilizzate nella scrittura del codice, con link ufficiali inclusi per la documentazione di riferimento.

Documentazione ufficiale:

- **Java:** https://dev.java/learn/
- **Android Studio (utilizzabile con Java):** https://developer.android.com/docs
- **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;

• **Interface Layer:** nel pattern Three-layer rappresenta la parte che si occupa della logica di visualizzazione dell'applicazione;

• **Application Logic Layer:** nel pattern Three-layer rappresenta la parte che si occupa della logica di business dell'applicazione;

• **Storage 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:

- Requirements Analysis Document **(RAD)**
- System Design Document **(SDD)**
- Test Plan **(TP)**
- Test Case Specification **(TCS)**
- Matrice di tracciabilità

2. Packages

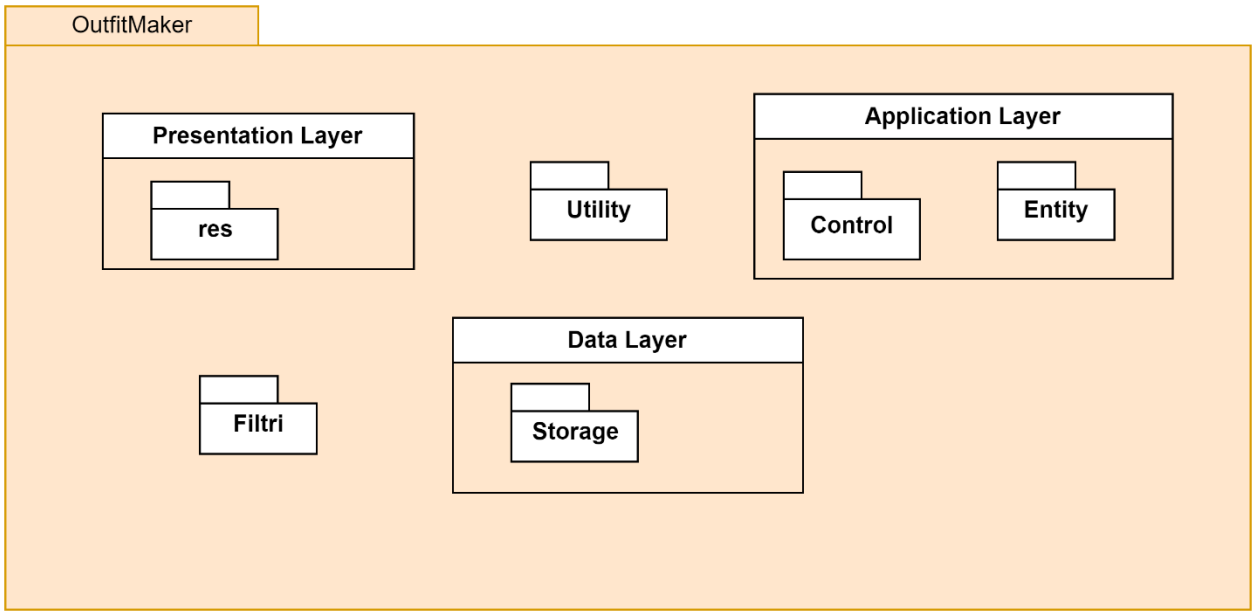
Questa sezione del documento espone la divisione del progetto in package sulla base dell'architettura Three Layer definita nel System Design.

- .idea
- .gradle, che contiene i file di configurazione per gradle
- app, contiene tutti i file sorgente del sistema.
    - Build
    - Src
        - Main
            - java contiene le classi Java relative alle componenti
                - Control, contiene le classi relative ai controller per collegare l'Application e il Data Layer
                - Entity, contiene le classi relative alle entità del sistema.
                - Storage, contiene i DAO delle entità del sistema e le relative classi Service.
            - res contiene le classi relative al Presentation Layer
                - Layout, contiene tutte le pagine di activity e i frammenti visibili all'utente.
                - Drawable, contiene tutte le immagini inserite all'interno del sistema.
                - Menu, contiene il menu a tendina dell'applicazione.
    - Test, contiene le classi relative al testing.

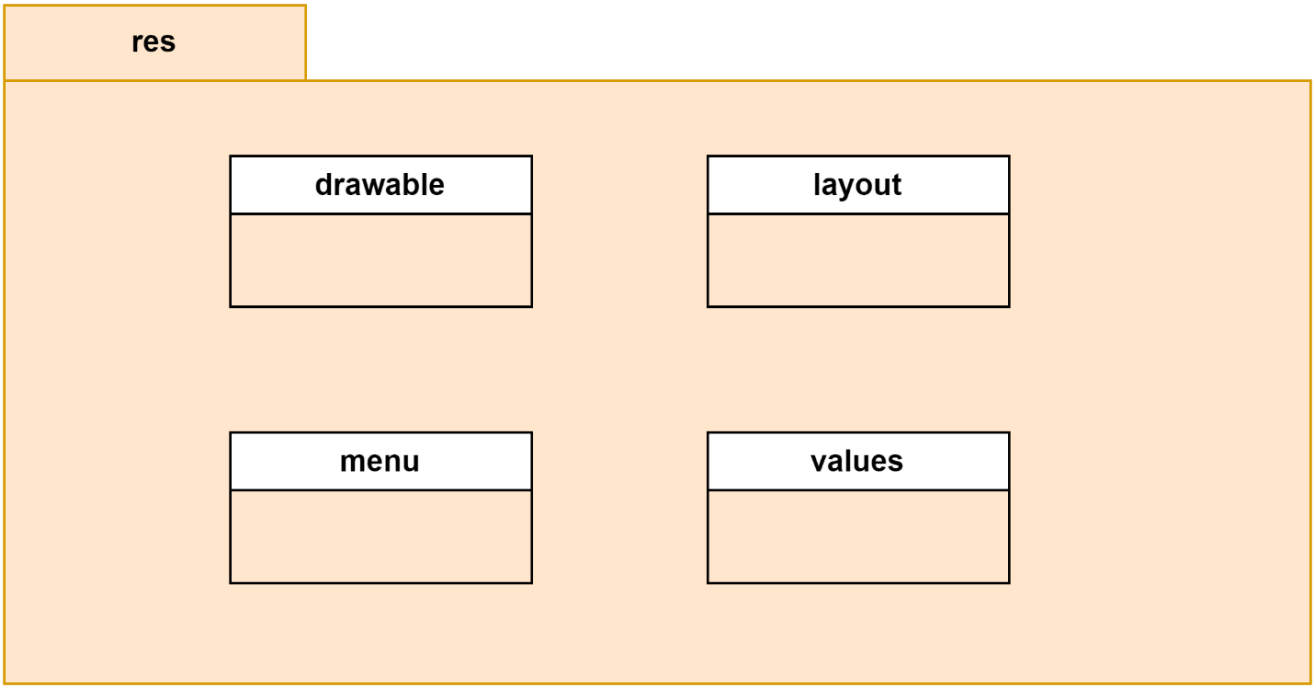
Package OutfitMaker

In questa sezione viene mostrata la struttura dei package del sistema, la struttura generale è stata creata in base alla divisione seguendo l'architettura scelta nel System Design in modo da avere un package separato per ogni sottosistema: Entity, Repository e Storage.

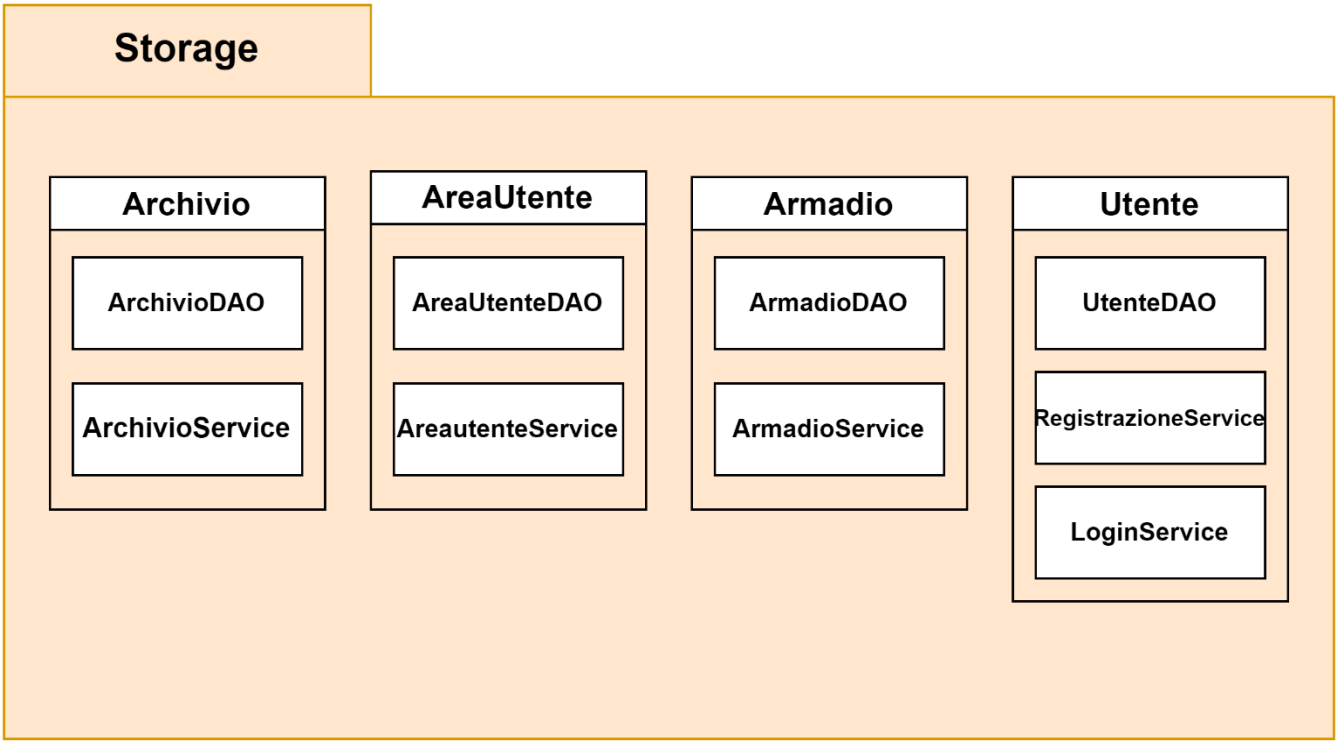
È inoltre stato aggiunto un package separato utility che contiene la classe utile per tutti i sottosistemi, e un package filtri per la gestione della ricerca degli oggetti tramite un filtro inserito dall'utente.



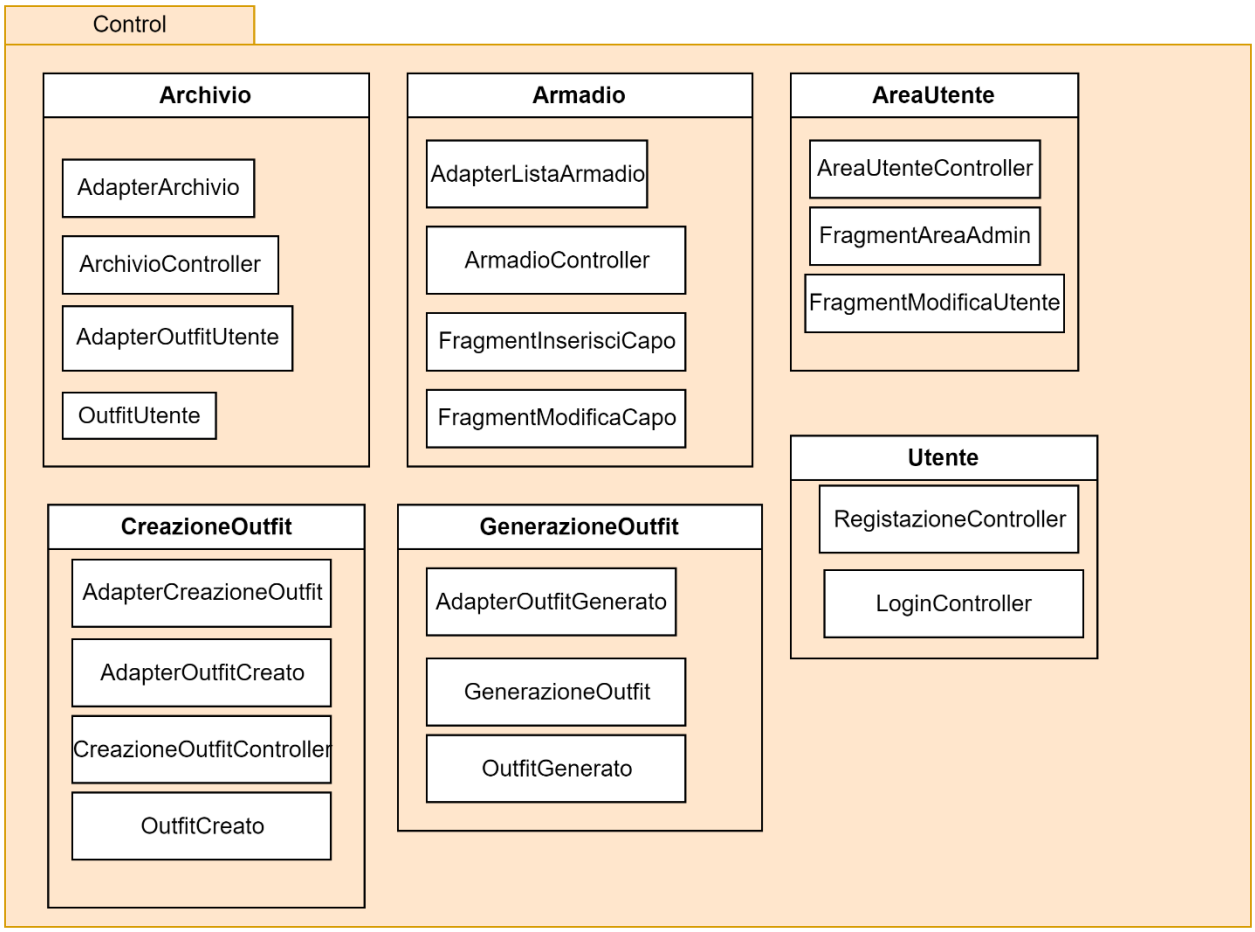
Package res



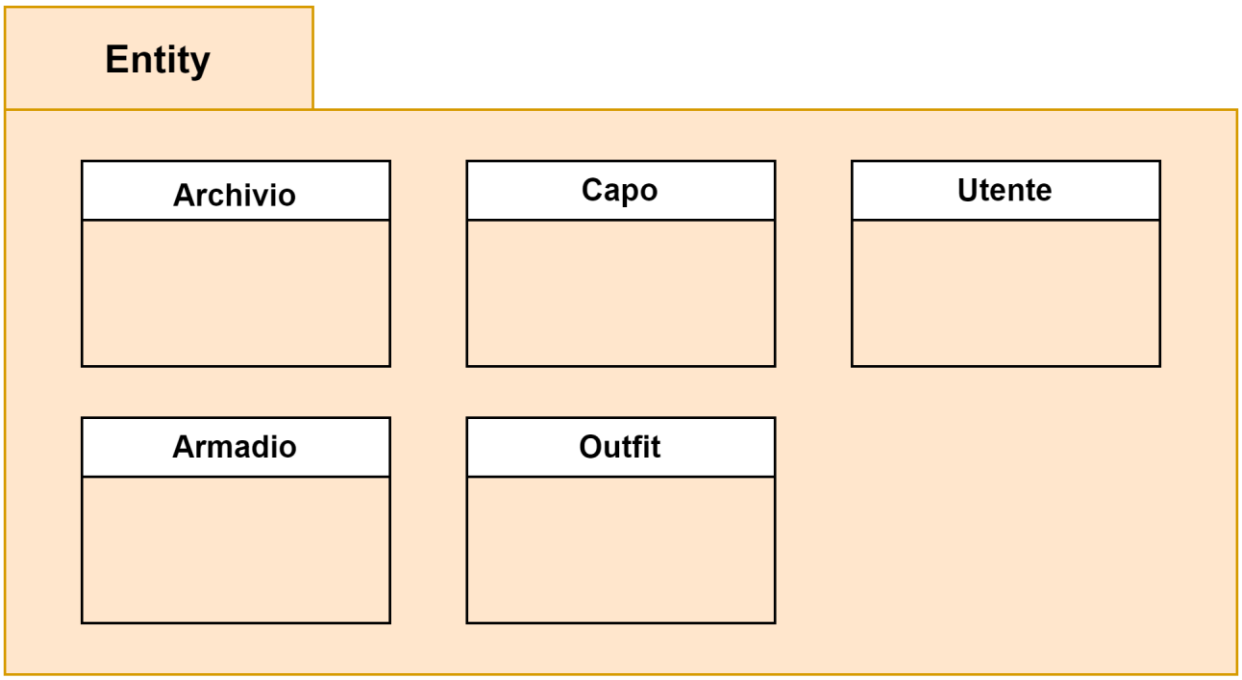
Package Storage



Package Control


Control
Archivio
AdapterArchivio
ArchivioController
AdapterOutfitUtente
OutfitUtente
Armadio
AdapterListaArmadio
ArmadioController
FragmentInserisciCapo
FragmentModificaCapo
AreaUtente
AreaUtenteController
FragmentAreaAdmin
FragmentModificaUtente
Utente
RegistazioneController
LoginController
CreazioneOutfit
AdapterCreazioneOutfit
AdapterOutfitCreato
CreazioneOutfitController
OutfitCreato
GenerazioneOutfit
AdapterOutfitGenerato
GenerazioneOutfit
OutfitGenerato

Package Entity



3. Package Class

Per quanto riguarda il package Control, verrà mostrato solo una classe adapter specifica in quanto vengono utilizzati gli stessi metodi statici anche in altre classi.

Non saranno descritte le classi degli oggetti Entity, essendo classi formate da soli costruttori, metodi getter e setter.

Package Control

Package Archivio

Nome Classe	AdapterArchivio
Descrizione	Questa classe estende RecyclerView.Adapter e viene utilizzata per adattare e visualizzare una lista di elementi di tipo Outfit all'interno di un RecyclerView. Questa classe è progettata per gestire la visualizzazione degli outfit nell'archivio dell'utente.
Metodi	+ onCreateViewHolder(ViewGroup parent, int viewType): SupportA, + onBindViewHolder(SupportA Holder, int position), + getItemCount(): int outfitArrayList
Invariante di classe	/

Nome Metodo	+ onCreateViewHolder(ViewGroup parent, int viewType) : SupportA
Descrizione	Questo metodo viene utilizzato per gestire la creazione di nuovi elementi della vista all'interno di un RecyclerView.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ onBindViewHolder(SupportA Holder, int position)
Descrizione	Questo metodo è chiamato dalla RecyclerView per associare i dati di un elemento specifico della lista al suo ViewHolder e per gestire gli eventi relativi a quell'elemento.

Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ getItemCount(): int outfitArrayList**
Descrizione	Questo metodo restituisce il numero totale di elementi nella lista di outfit.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Classe	**Archivio**
Descrizione	La classe Archivio rappresenta un'activity all'interno di un'applicazione Android, ed è responsabile di visualizzare l'archivio degli outfit dell'utente.
Metodi	+ onCreate(Bundle savedInstanceState), - eventChangeListener(), + onCreateOptionsMenu(Menu menu): boolean, + onOptionsItemsSelected(MenuItem item): boolean
Invariante di classe	/

Nome Metodo	**+ onCreate(Bundle savedIstanceState)**
Descrizione	Metodo chiamato quando l'activity viene creata. Inizializza i componenti principali dell'activity, come la RecyclerView, il database Firestore, il gestore di autenticazione, e l'AdapterArchivio.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	- eventChangeListener()
Descrizione	Un metodo privato che utilizza il meccanismo di ascolto di Firestore per rilevare le modifiche agli outfit nell'archivio.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ onCreateOptionsMenu(Menu menu): boolean
Descrizione	Metodo che gestisce la creazione del menù nell'ActionBar
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ onOptionsItemsSelected(MenuItem item): boolean
Descrizione	Metodo che gestisce le singole azioni associate agli elementi del menù
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package AreaUtente

Nome Classe	AreaUtenteController
Descrizione	La classe AreaUtenteController rappresenta un'activity all'interno di un'applicazione Android, focalizzata sull'area utente.
Metodi	+ onCreate(Bundle savedIstanceState), + onStart(), + ottieniDatiUtente(), + settaDatiUtente(), + inserimentoModificaDati(View v), + rimuoviFrammentoModifica(View v), + disconnessioneClicked(View v)
Invariante di classe	/

Nome Metodo	+ onStart()
Descrizione	Metodo chiamato quando l'activity diventa visibile. Verifica se l'utente è autenticato. In caso affermativo, recupera l'UID dell'utente.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ ottieniDatiUtente()
Descrizione	Metodo che utilizza il campo Uid per cercare e recuperare i dati dell'utente dal database Firestore.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ settaDatiUtente()
Descrizione	Metodo che aggiorna le TextView con i dati dell'utente ottenuti dal database.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ inserimentoModificaDati(View v)
Descrizione	Metodo chiamato quando l'utente desidera inserire la modifica dei dati.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ disconnessioneClicked(View v)
Descrizione	Metodo chiamato quando l'utente desidera disconnettersi.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package Armadio

Nome Classe	ArmadioController
Descrizione	La classe ArmadioController è un'activity all'interno di un'applicazione Android, focalizzata sulla gestione dell'armadio dell'utente.
Metodi	+ inserisciCapoClicked(View v), + rimuoviFrammentoClicked(View v), + inserisciFrammentoModificaCapo(View v), + inserisciFrammentoFiltri(View v), + rimuoviFrammentoFiltri(View v)
Invariante di classe	/

Nome Metodo	+ inserisciCapoClicked(View v)
Descrizione	Metodo chiamato quando l'utente desidera inserire un nuovo capo nell'armadio.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ rimuoviFrammentoClicked(View v)
Descrizione	Metodo chiamato quando l'utente desidera rimuovere il frammento per l'inserimento di un nuovo capo.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ inserisciFrammentoModificaCapo(View v)
Descrizione	Metodo chiamato quando l'utente desidera inserire il frammento di modifica capo.
Pre-Condizione	CurrentUser.getUid() != null && capo != null
Post-Condizione	/

Nome Metodo	+ inserisciFrammentoFiltri(View v)
Descrizione	Metodo chiamato quando l'utente desidera inserire il frammento di filtri.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package CreazioneOutfit

Nome Classe	CreazioneOutfitController
Descrizione	La classe CreazioneOutfitController è un'activity all'interno di un'applicazione Android, focalizzata sulla gestione della logica di business correlata alla creazione di outfit all'interno dell'applicazione.
Metodi	+ onCreate(Bundle savedInstanceState), - EventChangeListener()
Invariante di classe	/

Nome Metodo	+ onCreate(Bundle savedInstanceState)
Descrizione	Metodo chiamato quando l'activity viene creata. Inizializza i componenti principali dell'activity, come la RecyclerView, il database Firestore, il gestore di autenticazione e AdapterCreazioneOutfit.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ EventChangeListener()
Descrizione	Un metodo privato che utilizza il meccanismo di ascolto di Firestore per rilevare le modifiche ai capi selezionati.

Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package GenerazioneOutfit

Nome Classe	**GenerazioneOutfitController**
Descrizione	La classe GenerazioneOutfitController è un'activity all'interno di un'applicazione Android, focalizzata sulla gestione della logica correlata alla generazione di outfit all'interno dell'applicazione.
Metodi	+ onCreate(Bundle savedInstanceState)
Invariante di classe	/

Nome Metodo	**+ onCreate(Bundle savedInstanceState)**
Descrizione	Metodo chiamato quando l'activity viene creata. Inizializza i componenti principali dell'activity, come la RecyclerView, il database Firestore, il gestore di autenticazione e AdapterOutfitgenerato.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package Utente

Nome Classe	LoginController
Descrizione	La classe LoginController è un controller associato a un'attività (Activity). Il suo scopo principale è gestire l'autenticazione degli utenti, consentendo loro di effettuare il login.
Metodi	+ inserisciDatiLog(View v), + vaiRegistrazione(View v)
Invariante di classe	/

Nome Metodo	+ inserisciDatiLog(View v)
Descrizione	Metodo richiamato quando l'utente preme il pulsante per effettuare il login. Recuperando l'email e la password inseriti dall'utente.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ vaiRegistrazione(View v)
Descrizione	Metodo richiamato quando l'utente preme il pulsante per passare alla schermata di registrazione.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Classe	RegistrazioneController
Descrizione	La classe RegistrazioneController è un controller associato a un'attività (Activity). Il suo scopo principale è gestire l'autenticazione degli utenti, consentendo loro di effettuare la registrazione.
Metodi	+ inserisciDati(View v), + vaiLogin(View v)
Invariante di classe	/

Nome Metodo	+ inserisciDati(View v)
Descrizione	Metodo richiamato quando l'utente preme il pulsante per effettuare la registrazione.
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	+ vaiLogin(View v)
Descrizione	Metodo richiamato quando l'utente preme il pulsante per passare alla schermata di login.
Pre-Condizione	/
Post-Condizione	/

Package Storage

Package Archivio

Nome Classe	ArchivioDAO
Descrizione	La classe ArchivioDAO è responsabile per l'interazione con il database Firestore per la gestione dell'archivio degli outfit dell'utente.
Metodi	+ creaArchivioFirestore(String idArchivio, String uid), + generateUniqueArchivioId(): String UUID
Invariante di classe	/

Nome Metodo	+ creaArchivioFirestore(String idArchivio, String uid)
Descrizione	Metodo che crea un archivio in cloud nell'archivio Firestore per un utente specifico.
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	+ generateUniqueArchivioId(): String UUID
Descrizione	Metodo che genera un identificativo univoco per un archivio.
Pre-Condizione	/
Post-Condizione	/

Package AreaUtente

Nome Classe	AreaUtenteDAO
Descrizione	La classe AreaUtenteDAO è responsabile per l'interazione con il database Firestore per la gestione dell'area utente dell'utente.
Metodi	+ ottieniDatiUtente(String uid, String nome, String cognome, String email, String telefono): Task<Boolean>, + modificaDati(String nome_nuovo, String cognome_nuovo, String telefono_nuovo): Task<Boolean>
Invariante di classe	/

Nome Metodo	+ ottieniDatiUtente(String uid, String nome, String cognome, String email, String telefono): Task<Boolean>
Descrizione	Metodo chiamato per ottenere i dati dell'utente corrente.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	+ modificaDati(String nome_nuovo, String cognome_nuovo, String telefono_nuovo): Task<Boolean>
Descrizione	Metodo chiamato per modificare i dati utente nel database Firestore.
Pre-Condizione	CurrentUser.getUid() != null

Post-Condizione	/

Package Armadio

Nome Classe	ArmadioDAO
Descrizione	La classe ArmadioDAO è responsabile per l'interazione con il database Firestore per la gestione dell'armadio virtuale e dei capi dell'utente.
Metodi	+ creaArmadioFirestore(String idArmadio), + aggiungiCapo(String nomeBrand, List<String> colori, String tipologia, String stagionalita, String occasione): Task<Boolean>, + modificaCapo(String nomeBrand, List<String> colori, String tipologia, String stagionalita, String occasione): Task<Boolean>, + ricercaFiltri(ArrayList<String> colori, String stagionalita, String tipologia): Task<Boolean>, + resettaSceltaCapi(): Task<Boolean>, + creaOutfit(ArrayList<Capo> lista_capi):Task<Boolean> + capiMinimiTopCenterBottom(String uid): Task<Boolean>, + generaOutfit(String stagionalità): Task<ArrayList<Capo>>
Invariante di classe	/

Nome Metodo	+ creaArmadioFirestore(String idArmadio)
Descrizione	Metodo che crea un armadio in cloud nell'archivio Firestore per un utente specifico.

Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ aggiungiCapo(String nomeBrand, List<String> colori, String tipologia, String stagionalita, String occasione): Task<Boolean>**
Descrizione	Metodo chiamato quando l'utente vuole aggiungere un nuovo capo.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ modificaCapo(String nomeBrand, List<String> colori, String tipologia, String stagionalita, String occasione): Task<Boolean>**
Descrizione	Metodo chiamato quando l'utente vuole modificare un capo.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ ricercaFiltri(ArrayList<String> colori, String stagionalita, String tipologia): Task<Boolean>**
Descrizione	Metodo chiamato per cercare capi basati su filtri specifici come colori, stagionalità e tipologia.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ resettaSceltaCapi(): Task<Boolean>**
Descrizione	Metodo chiamato per settare la scelta dei capi.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ creaOutfit(ArrayList<Capo> lista_capi): Task<Boolean>**

Descrizione	Metodo chiamato per creare un nuovo outfit in base alla lista dei capi fornita.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ capiMinimiTopCenterBottom(String uid): Task<Boolean>**
Descrizione	Metodo che serve per verificare se l'utente ha almeno un capo per ogni determinata tipologia.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ generaOutfit(String stagionalità): Task<ArrayList<Capo>>**
Descrizione	Metodo chiamato per generare un nuovo outfit in base alle preferenze dell'utente.
Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Package Utente

Nome Classe	**UtenteDAO**
Descrizione	La classe UtenteDAO è responsabile per l'interazione con il database Firestore per la gestione e l'autenticazione dell'utente.
Metodi	+ creaUtente(String nome, String cognome, String email, String password, String telefono): Task<Boolean>, + creaUtenteFirestore(String uid, String nome, String cognome, String email, String password, String telefono, String idArmadio, String idArchivio), + effettuaLogin(String email, String password): Task<Boolean>
Invariante di classe	/

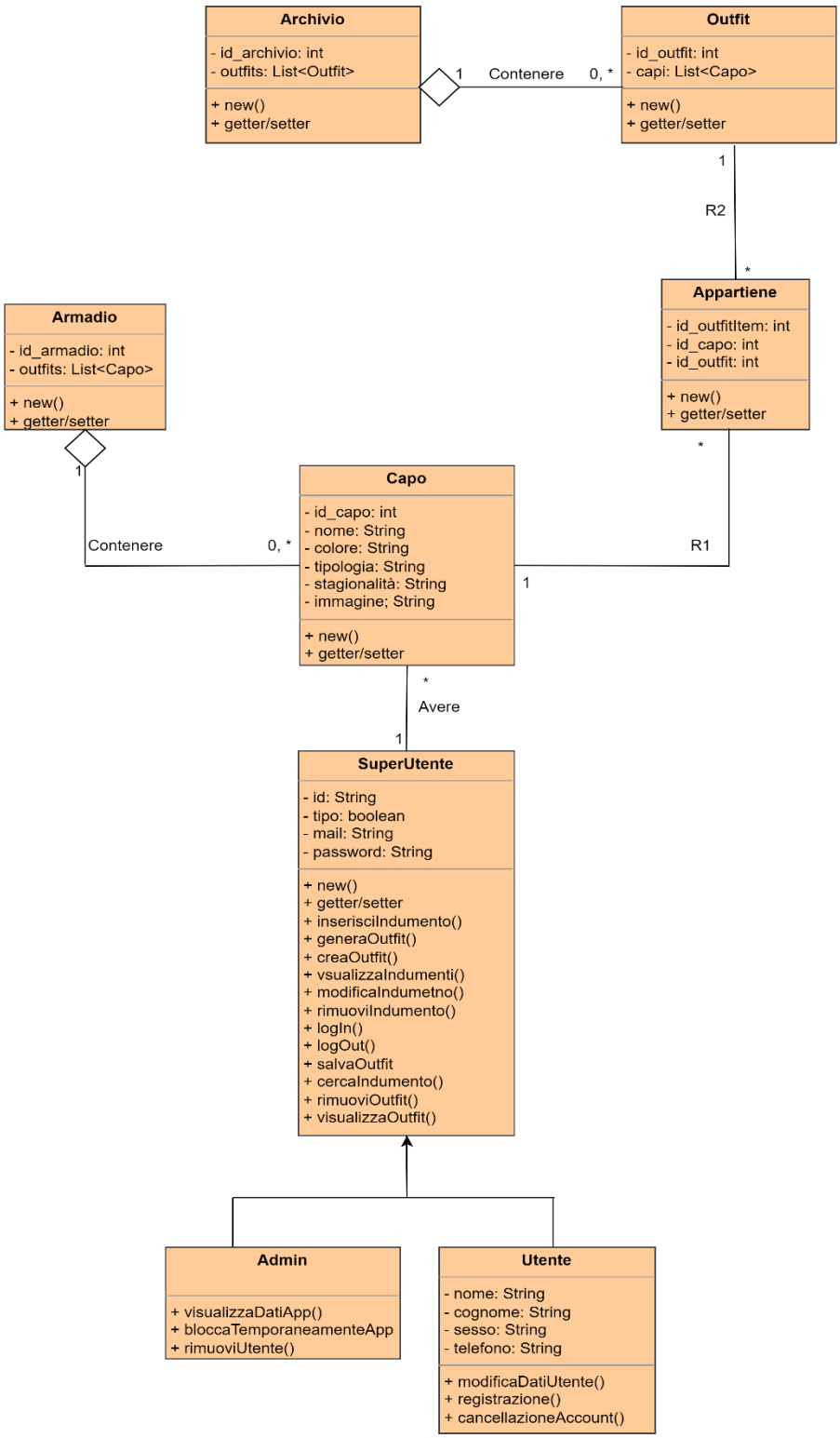
Nome Metodo	**+ creaUtente(String nome, String cognome, String email, String password, String telefono): Task<Boolean>**
Descrizione	Metodo che crea un nuovo utente nel sistema utilizzando l'autenticazione di Firebase.

Pre-Condizione	CurrentUser.getUid() != null
Post-Condizione	/

Nome Metodo	**+ creaUtenteFirestore(String uid, String nome, String cognome, String email, String password, String telefono, String idArmadio, String idArchivio)**
Descrizione	Metodo che aggiungere un documento utente a Firestore, inizializzando anche l'armadio e l'archivio associati.
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	**+ effettuaLogin(String email, String password): Task<Boolean>**
Descrizione	Effettua il login di un utente nel sistema utilizzando l'autenticazione di Firebase.
Pre-Condizione	/
Post-Condizione	/

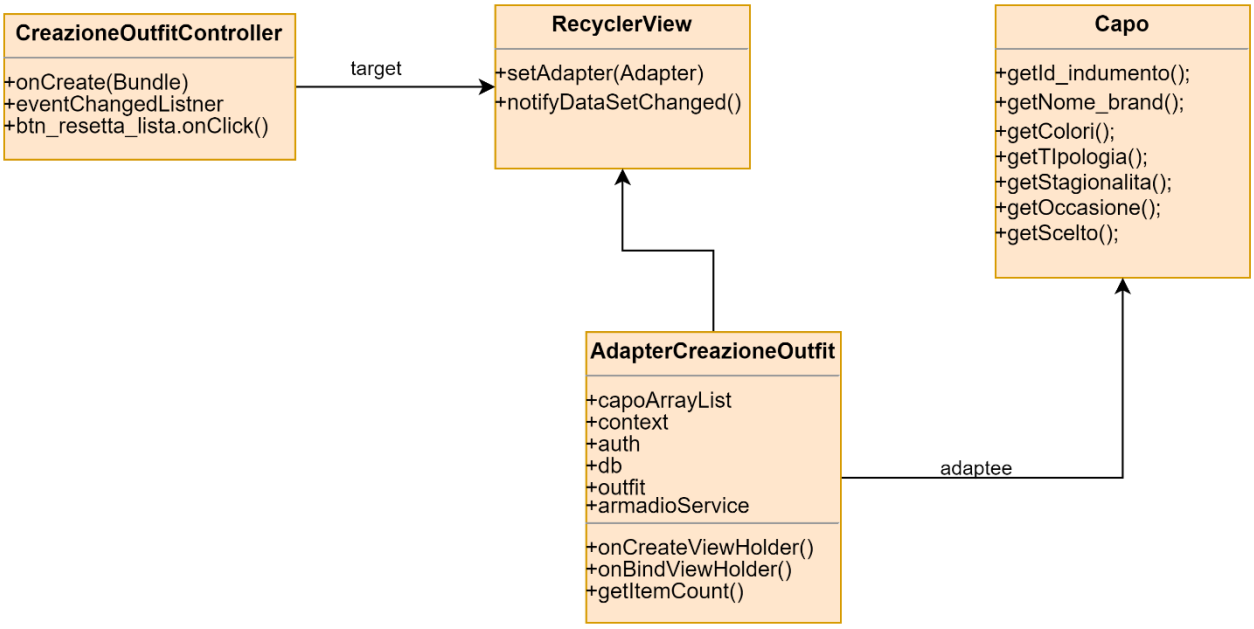
4. Class Diagram



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.



La classe **AdapterCreazioneOutfit** segue il design pattern Adapter per adattare la lista di capi alla RecyclerView, classe di Android. In particolare, distinguiamo:

- **Target:** La RecyclerView è il "target" in questo contesto, rappresenta l'interfaccia desiderata per la visualizzazione dei dati.
- **Client:** L'attività che utilizza AdapterCreazioneOutfit per visualizzare la lista di capi, nel nostro caso CreazioneOutfitController.
- **Adapter:** La classe **AdapterCreazioneOutfit** funge da "adapter" che collega la lista di capi all'interfaccia desiderata della RecyclerView.

- **Adapter**: La classe Capo rappresenta l'oggetto esistente che deve essere adattato per interagire con l'AdapterCreazioneOutfit.

In questo modo, **AdapterCreazioneOutfit** si adatta all'interfaccia della RecyclerView, consentendo la corretta visualizzazione e interazione con la lista di capi nel contesto dell'applicazione Android.

5.2 Componenti terzi

All'interno del sistema è stato usato un componente COTS:

- 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.

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.
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.