



Manuale Sviluppatore

Gruppo VarTmp7 - Progetto Stalker vartmp7@gmail.com

Informazioni sul documento

Versione	2.0.0-R1
Approvatore	Riccardo Tassetto
Redattori	Xiaowei Wen Lorenzo Taschin Marco Ferrati Riccardo Tassetto
Verificatori	Claudia Zattara Giuseppe Zatta
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Imola Informatica S.p.A VarTmp7

Descrizione

Questo documento si occupa di descrivere come mantenere e sviluppare ulteriormente il sistema Stalker.

Registro delle modifiche

Versione	Data	Descrizione	Nominativo	Ruolo
2.0.0-R1	2020-05-14	Versione 2.0.0-R1 approvata	Riccardo Tassetto	<i>Responsabile</i>
2.0.0-INC-15	2020-05-12	Verificati §4.2 e §4.3. Documento pronto per l'approvazione.	Claudia Zattara	<i>Verificatore</i>
1.0.2-INC-15	2020-05-12	Scritte §4.2, §4.3	Riccardo Tassetto	<i>Redattore</i>
1.0.2-INC-15	2020-05-11	Verifica dettagli aggiunti	Claudia Zattara	<i>Verificatore</i>
1.0.1-INC-15	2020-05-11	Aggiunti maggiori dettagli ai diagrammi delle classi del backend	Marco Ferrati	<i>Redattore</i>
1.0.1-INC-14	2020-05-04	Verifica correzione diagrammi applicazione Android	Xiaowe Wen	<i>Verificatore</i>
1.0.0-INC-14	2020-05-04	Correzione diagrammi applicazione Android	Lorenzo Taschin	<i>Redattore</i>
1.0.0-INC-10	2020-04-12	Versione 1.0.0-INC-10 approvata	Lorenzo Taschin	<i>Responsabile</i>
1.0.0-INC-10	2020-04-11	Verifica §4.5. Documento pronto per l'approvazione	Giuseppe Zatta	<i>Verificatore</i>
0.14.0-INC-10	2020-04-11	Stesura §4.5	Riccardo Tassetto	<i>Redattore</i>
0.14.0-INC-10	2020-04-10	Verifica §4.2, §4.4	Giuseppe Zatta	<i>Verificatore</i>
0.13.0-INC-10	2020-04-10	Stesura §4.2, §4.4	Riccardo Tassetto	<i>Redattore</i>
0.13.0-INC-10	2020-04-08	Verifica §3.2, §3.4, §3.5	Giuseppe Zatta	<i>Verificatore</i>
0.12.0-INC-10	2020-04-07	Stesura §3.2, §3.4, §3.5	Marco Ferrati	<i>Redattore</i>
0.12.0-INC-10	2020-04-07	Verifica §2.4, §2.5	Claudia Zattara	<i>Verificatore</i>
0.11.0-INC-10	2020-04-07	Stesura §2.4, §2.5	Lorenzo Taschin	<i>Redattore</i>
0.11.0-INC-10	2020-04-06	Verifica §2.2	Claudia Zattara	<i>Verificatore</i>
0.10.0-INC-10	2020-04-06	Stesura §2.2	Lorenzo Taschin	<i>Redattore</i>
0.10.0-INC-9	2020-04-05	Verifica §3.3, §3.3.1, §3.3.2	Claudia Zattara	<i>Verificatore</i>
0.9.0-INC-9	2020-04-04	Modifica §3.3 e stesura §3.3.1, §3.3.2	Marco Ferrati	<i>Redattore</i>
0.9.0-INC-9	2020-04-02	Verifica §2.3.1, §2.3.2	Giuseppe Zatta	<i>Verificatore</i>
0.8.0-INC-9	2020-04-02	Stesura §2.3.1, §2.3.2	Lorenzo Taschin	<i>Redattore</i>

0.8.0-INC-9	2020-04-02	Verifica §4.3.1, §4.3.2	Claudia Zattara	<i>Verificatore</i>
0.7.0-INC-9	2020-04-01	Stesura §4.3.1, §4.3.2	Riccardo Tassetto	<i>Redattore</i>
0.7.0-INC-8	2020-03-24	Verifica §3.3	Giuseppe Zatta	<i>Verificatore</i>
0.6.0-INC-8	2020-03-24	Stesura §3.3	Marco Ferratti	<i>Redattore</i>
0.6.0-INC-8	2020-03-23	Verifica §2.3	Claudia Zattara	<i>Verificatore</i>
0.5.0-INC-8	2020-03-23	Stesura §2.3	Xiaowei Wen	<i>Redattore</i>
0.5.0-INC-8	2020-03-23	Verifica §4.3	Giuseppe Zatta	<i>Verificatore</i>
0.4.0-INC-8	2020-03-23	Stesura §4.3	Riccardo Tassetto	<i>Redattore</i>
0.4.0-INC-7	2020-03-21	Verifica §2.1	Claudia Zattara	<i>Verificatore</i>
0.3.0-INC-7	2020-03-21	Stesura §3.1	Marco Ferrati	<i>Redattore</i>
0.3.0-INC-7	2020-03-19	Verifica §2.1	Giuseppe Zatta	<i>Verificatore</i>
0.2.0-INC-7	2020-03-19	Stesura §2.1	Xiaowei Wen	<i>Redattore</i>
0.2.0-INC-7	2020-03-19	Verifica §4.1	Claudia Zattara	<i>Verificatore</i>
0.1.0-INC-7	2020-03-18	Stesura §4.1	Riccardo Tassetto	<i>Redattore</i>
0.1.0-INC-7	2020-03-17	Verifica sezione §1	Claudia Zattara	<i>Verificatore</i>
0.0.0-INC-7	2020-03-17	Stesura §1 Introduzione	Marco Ferrati	<i>Redattore</i>
0.0.0-0	2019-11-24	Creazione scheletro del documento	Claudia Zattara	<i>Redattore</i>

Indice

1	Introduzione	7
1.1	Scopo del documento	7
1.2	Glossario	7
1.3	Riferimenti	7
2	Applicazione Android	8
2.1	Tecnologie applicazione Android	8
2.1.1	Java SE 8	8
2.1.2	XML	8
2.1.3	Android Jetpack	8
2.1.4	UnboundID SDK	8
2.1.5	Retrofit	8
2.1.6	GSON	8
2.1.7	Facebook SDK	8
2.1.8	OneSignal	8
2.1.9	Project Lombok	8
2.1.10	Circle Image View	9
2.1.11	Glide	9
2.2	Setup e configurazione dell'ambiente di sviluppo per l'applicazione Android	10
2.2.1	Requisiti di sistema	10
2.2.2	Creazione e configurazione ambiente di sviluppo	10
2.2.3	Configurazione Firebase	10
2.2.4	Configurazione autenticazione con Facebook	13
2.2.5	Configurazione Firebase-database	15
2.2.6	Configurazione Sonarqube	16
2.3	Architettura applicazione Android	18
2.3.1	Model	19
2.3.2	View e ViewModel	23
2.3.3	TrackingService	28
2.3.4	Diagramma dei package	30
2.4	Estendibilità applicazione Android	31
2.4.1	Modificare l'approccio nel salvataggio dei dati persistenti	31
2.4.2	Modificare la fonte di dati remoti	31
2.5	Test applicazione Android	32
2.5.1	Aggiungere nuovi test	32
2.5.2	Eseguire i test	32
3	Backend	33
3.1	Tecnologie backend	33
3.1.1	Python	33
3.1.2	PostgreSQL	33
3.1.3	RethinkDB	33
3.1.4	Kubernetes	33
3.1.5	RabbitMQ	34
3.1.6	Flask	34
3.1.7	Flask-SQLAlchemy	34
3.1.8	Flask-RESTful	34

3.1.9	Flask-JWT	34
3.1.10	PyTest	34
3.1.11	ldap3	34
3.2	Setup e configurazione dell'ambiente di sviluppo per il backend	35
3.2.1	Requisiti di sistema	35
3.2.2	Avvio del backend	35
3.2.2.1	Make	35
3.2.2.2	Docker Compose	35
3.2.2.3	Scaffold	36
3.2.2.4	Docker	36
3.3	Architettura backend	37
3.3.1	Modelli	37
3.3.2	Risorse	39
3.3.2.1	Parser	44
3.3.2.2	Autenticazione e autorizzazione	45
3.3.2.3	Esempio creazione organizzazione	46
3.4	Estendibilità backend	48
3.4.1	Aggiungere un modello	48
3.4.2	Aggiungere una nuova API	48
3.5	Test backend	50
3.5.1	Aggiungere nuovi test	50
3.5.2	Eseguire i test	50
4	Webapp	51
4.1	Tecnologie webapp	51
4.1.1	Angular 9	51
4.1.2	Material Design	51
4.1.3	Node JS	51
4.1.4	Leaflet	51
4.1.5	Leaflet-Geoman	51
4.1.6	XML	51
4.1.7	JSON	51
4.2	Setup e configurazione dell'ambiente di sviluppo per la webapp	52
4.2.1	Requisiti di sistema	52
4.2.2	Creazione e configurazione ambiente di sviluppo	52
4.3	Architettura webapp	53
4.3.1	Model	53
4.3.1.1	Administrator	54
4.3.1.2	Organization	55
4.3.1.3	Place	55
4.3.1.4	Track	56
4.3.2	View e ViewModel	56
4.3.3	Altri component	57
4.4	Estendibilità webapp	58
4.4.1	Aggiungere un component	58
4.4.2	Aggiungere un service	58
4.5	Test webapp	59
4.5.1	Aggiungere nuovi test	59
4.5.2	Eseguire i test	59

A	Glossario	61
A.1	A	61
A.1.1	AVD	61
A.1.2	Header della richiesta	61
A.1.3	JSON	61
A.1.4	LDAP	61
A.1.5	NoSQL	61
A.2	R	61
A.2.1	REST	61
A.2.2	Real-time	61
A.3	S	61
A.3.1	SDK	61

Elenco delle figure

1	Plugin Lombok da installare	10
2	Console di Firebase	11
3	aggiungere l'applicazione al progetto di Firebase	11
4	Pagina di Facebook developers	13
5	registrazione applicazione su Facebook	14
6	i dati da inserire nella console di Firebase	15
7	link di OAuth	15
8	Struttura documento di Firebase database	16
9	https://developer.android.com/jetpack/docs/guide	18
10	Entities	19
11	Diagramma delle classi che illustra le dipendenze di OrganizationsRepository. . .	19
12	Dipendenze della classe Storage ed implementazione concreta.	20
13	Dipendenze della classe Obtainer ed implementazione concreta.	21
14	Dipendenze della classe FavoritesSource ed implementazione concreta.	22
15	Dipendenze di OrganizationsFragment e OrganizationsViewModel.	23
16	Binding dei dati tra OrganizationsFragment, OrganizationsViewModel e Organi- zationsRepository.	24
17	Dipendenze di FavoritesFragment e FavoritesViewModel.	25
18	Dipendenze di TrackingFragment e TrackingViewModel.	26
19	Dipendenze di HistoryFragment e HistoryViewModel.	27
20	Diagramma delle classi per TrackingService.	28
21	Diagramma di sequenza che illustra come avviene un tracciamento tramite il TrackingService.	29
22	Diagramma dei package dell'applicazione mobile.	30
23	un esempio di modifica sul salvataggio dei dati	31
24	Pulsante per eseguire i test.	32
25	Architettura del backend.	37
26	Diagramma delle classi dei modelli del backend.	38
27	Diagramma delle classi delle risorse in relazione a Resource.	40
28	Diagramma delle classi delle risorse in relazione a Resource.	43
29	Diagramma delle classi delle risorse in relazione ai parser.	45
30	Diagramma di sequenza per la creazione di un'organizzazione.	46
31	Estensione modelli	48
32	Estensione API	49
33	Architettura della webapp.	53
34	Diagramma delle classi dell'amministratore.	54
35	Diagramma delle classi delle organizzazioni.	54
36	Diagramma delle classi dei luoghi.	55
37	Diagramma delle classi dei tracciamenti.	56
38	Diagramma di sequenza dell'autenticazione.	56
39	Diagramma di sequenza della creazione di un'organizzazione.	57
40	Esempio di test del component AdministratorDetailsComponent	59
41	Jasmine HTML Reporter	60

1 Introduzione

1.1 Scopo del documento

L'obiettivo di questo documento è quello di guidare lo sviluppatore nella comprensione, nell'installazione, nel mantenimento e nell'estensione del progetto Stalker. Al fine di raggiungerlo, per ognuno dei tre sottosistemi:

- Applicazione smartphone;
- Backend;
- WebApp di amministrazione.

Vengono descritti il procedimento per l'installazione, l'insieme delle tecnologie utilizzate nello sviluppo e nel testing, l'architettura, i punti di possibile estensione e le procedure per il testing.

1.2 Glossario

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti è stata dedicata una sezione al Glossario in appendice al documento, contenente la definizione dei termini marcati in corsivo e con una G a pedice.

1.3 Riferimenti

- pattern strutturali: <https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E06.pdf>;
- pattern creazionali: <https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E07.pdf>;
- pattern comportamentali: <https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E08.pdf>;
- pattern architetturali: <https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E09.pdf>;

2 Applicazione Android

2.1 Tecnologie applicazione Android

2.1.1 Java SE 8

Linguaggio di programmazione orientato agli oggetti, a tipizzazione statica.

2.1.2 XML

Linguaggio di mark-up utilizzato per definire il layout e i componenti della UI.

2.1.3 Android Jetpack

Suite di librerie, strumenti e guide per aiutare gli sviluppatori a creare applicazioni di alta qualità più facilmente. Questi componenti aiutano a seguire le best practices, liberando il programmatore dal codice copia-incolla e semplificando le attività complesse. <https://developer.android.com/jetpack/>

2.1.4 UnboundID SDK

Libreria Java utilizzata per effettuare l'autenticazione presso un server *LDAP*_G
<https://github.com/pingidentity/ldapsdk>

2.1.5 Retrofit

Libreria Android e Java che semplifica la gestione delle richieste HTTP e la serializzazione e deserializzazione di oggetti in formato JSON. Utilizzata per chiamare le *REST*_G API fornite dal backend.
<https://square.github.io/retrofit/>

2.1.6 GSON

Libreria Java open source per serializzare e deserializzare oggetti Java in formato JSON. Utilizzata per la formattazione dei dati salvati su file.
<https://github.com/google/gson>

2.1.7 Facebook SDK

Software Development Kit fornito da Facebook per poter effettuare l'autenticazione con Facebook.
https://developers.facebook.com/docs/apis-and-sdks?locale=it_IT

2.1.8 OneSignal

Tecnologia utilizzata per permettere al backend di inviare le push-notification all'applicazione.
<https://onesignal.com/>

2.1.9 Project Lombok

Libreria che permette di utilizzare delle annotazioni per la creazione automatica dei metodi *set*, *get* ed *equals*.
<https://projectlombok.org/>

2.1.10 Circle Image View

È stato utilizzato per rappresentare le immagini in forma circolare.

<https://github.com/hdodenhof/CircleImageView>

2.1.11 Glide

È stato utilizzato per scaricare e salvare in cache le immagini dato un link.

<https://github.com/bumptech/glide>

2.2 Setup e configurazione dell'ambiente di sviluppo per l'applicazione Android

2.2.1 Requisiti di sistema

I requisiti di sistema richiesti per l'ambiente di sviluppo dell'applicazione android sono reperibili alla fine della pagina del sito <https://developer.android.com/studio>.

2.2.2 Creazione e configurazione ambiente di sviluppo

Per poter iniziare lo sviluppo dell'applicazione è necessario seguire i seguenti passi di configurazione:

N.B. Tutti i riferimenti numerici sono relativi alla guida di Facebook se non specificati.

1. installare Android Studio seguendo le istruzioni presenti al seguente url: <https://developer.android.com/studio/install>;
2. aprire il progetto dell'applicazione con Android Studio;
3. aprire il gestore dei plug-in di Android Studio seguendo questo percorso: File-> Settings -> Plugins;
4. digitare "Lombok" nella casella di ricerca, installare il seguente plugin:

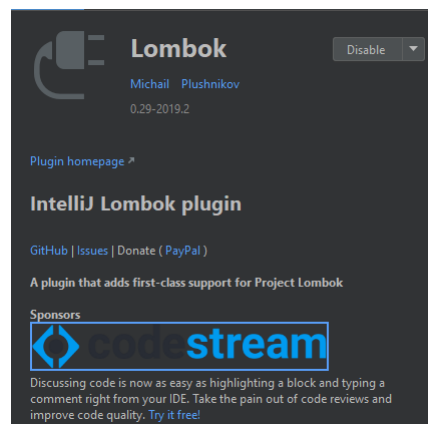


Figura 1: Plugin Lombok da installare

5. creare un AVD_G (Android virtual Device) con API level 28, ora è possibile avviare l'applicazione facendola eseguire nell'AVD e utilizzarla senza le funzionalità di login con Facebook;

2.2.3 Configurazione Firebase

1. aprire il seguente link <https://firebase.google.com/>, se necessario effettuare il login con un account Google e cliccare "vai alla console" in alto a destra;
2. cliccare su "aggiungi progetto";
3. nella pagina che si apre, inserire il nome del progetto "Stalker" e cliccare continua;
4. abilitare o disabilitare Google Analytics per il progetto;

5. alla pagina successiva selezionare "Default account for Firebase" e attendere che il progetto su Firebase venga creato, quando è pronto cliccare su continua e selezionare il progetto appena creato;

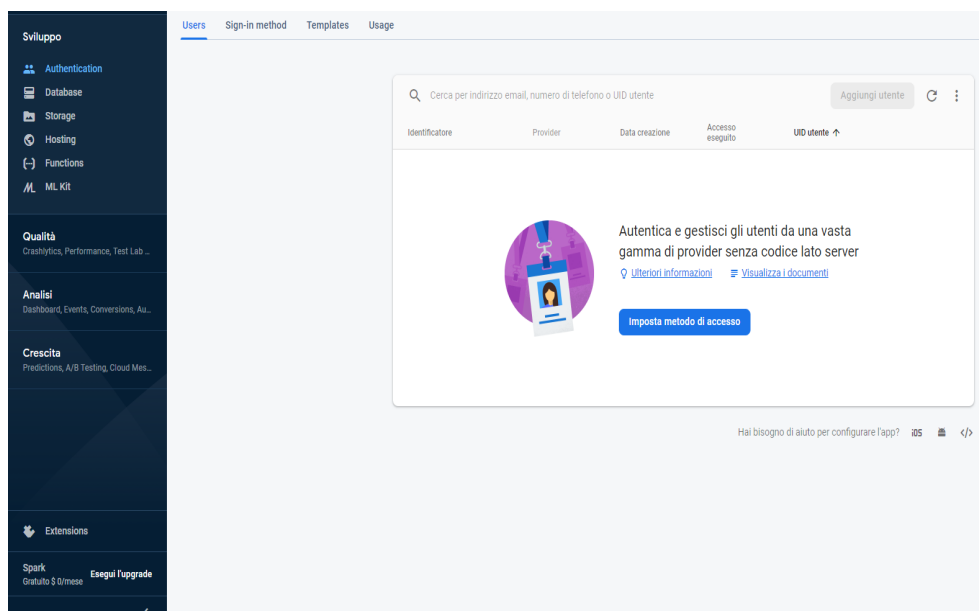


Figura 2: Console di Firebase

6. aggiungere l'applicazione nel progetto di Firebase: cliccare su panoramica del progetto, selezionando l'icona di Android;

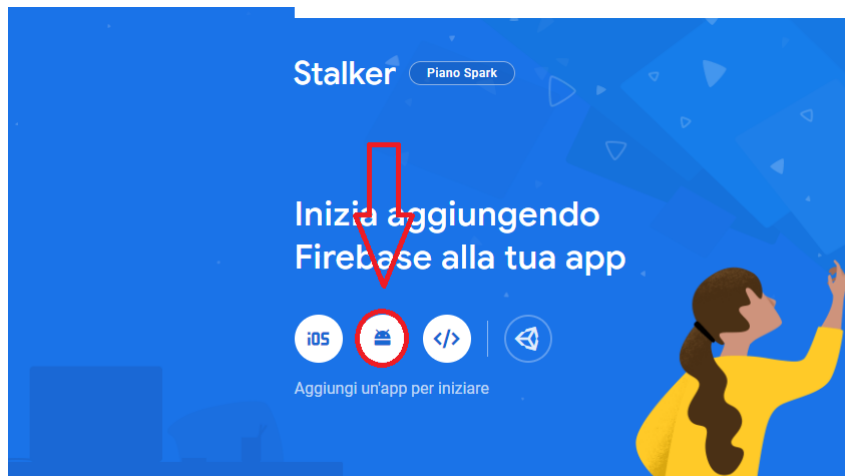


Figura 3: aggiungere l'applicazione al progetto di Firebase

7. seguire le istruzioni della pagina che si apre, inserendo anche il "certificato SHA-1" seguendo le informazioni presenti;

8. scaricare il file .json di configurazione e sostituirlo con quello presente nella cartella dell'applicazione;
9. cliccare avanti e avviare l'applicazione in modo che venga riconosciuta da Firebase;

2.2.4 Configurazione autenticazione con Facebook

Per poter utilizzare la funzionalità "accedi con Facebook" è necessario completare i seguenti passi:

1. aprire il seguente link: <https://developers.facebook.com/> ed effettuare il login se necessario;
2. cliccare su "Le mie App";
3. cliccare su "Creazione di un'app";

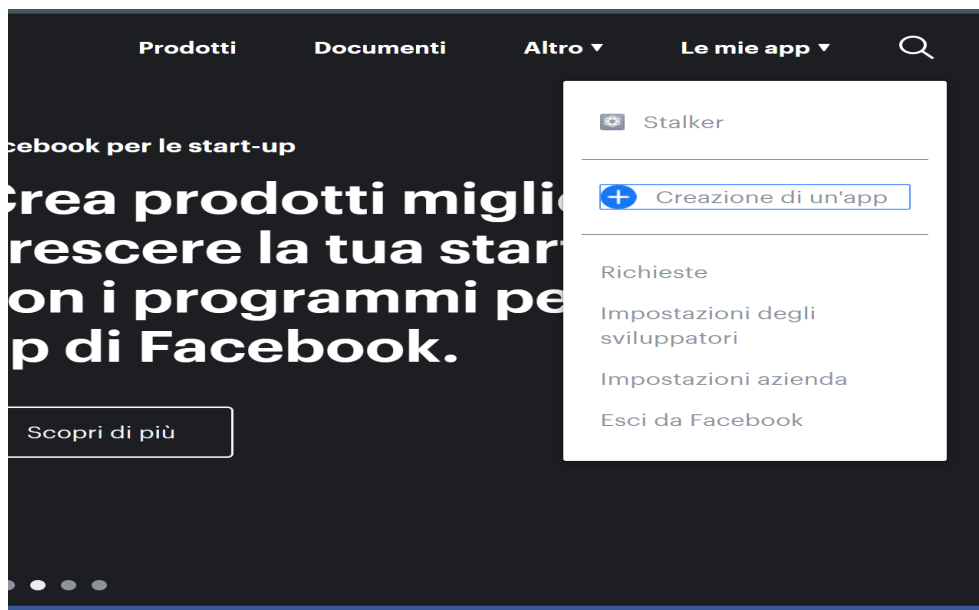


Figura 4: Pagina di Facebook developers

4. inserire il nome e l'email di contatto negli appositi spazi, quindi cliccare "Crea ID app" e completare il controllo di sicurezza; la pagina web verrà reindirizzata a quella di gestione dell'applicazione, che è la seguente:

Crea un nuovo ID app

Inizia l'integrazione di Facebook nella tua app o nel sito Web

Nome visualizzato

Il nome che desideri associare a questo ID app

Indirizzo e-mail di contatto

Si usa per le comunicazioni importanti relative alla tua app

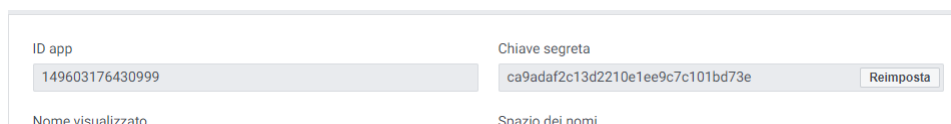
Procedendo, accetti le [Normative della Piattaforma Facebook](#)

Annulla

Crea ID app

Figura 5: registrazione applicazione su Facebook

5. per configurare il login con Facebook, cliccare su configura, quindi scegliere la piattaforma "Android";
6. saltare i primi due passi poiché sono già stati completati;
7. al terzo passo, inserire "com.vartmp7.stalker.MainActivity" nella prima e nella seconda casella di testo;
8. se l'applicazione non è stata pubblicata su Google Play Store, cliccare "Usa il nome di questo pacchetto" altrimenti verificare la correttezza dei valori inseriti;
9. seguire le istruzioni per completare il passo 4 della guida;
10. abilitare la funzionalità di accesso rapido;
11. copiare il codice del passo 6.2, aprire il file "/app/res/values/strings.xml" e sostituirlo al segnaposto "todo configurazione 6.2";
12. aprire il file "/app/manifests/manifest.xml" e sostituire il codice del passo 6.5 al segnaposto "todo configurazione 6.5", è possibile saltare il passo 6.4 perché il permesso richiesto è già presente;
13. saltare i passi 7-8-9-10 (è consigliato tenere la pagina aperta);
14. aprire la console di Firebase, selezionare la voce "Authentication", "sign-in method" quindi "Facebook", ora tornare nella pagina del passo 13 di questa guida, aprire l'impostazione di base del progetto e copiare i valori di "ID app" e "Chiave Segreta" nei campi richiesti dalla pagina di Firebase;

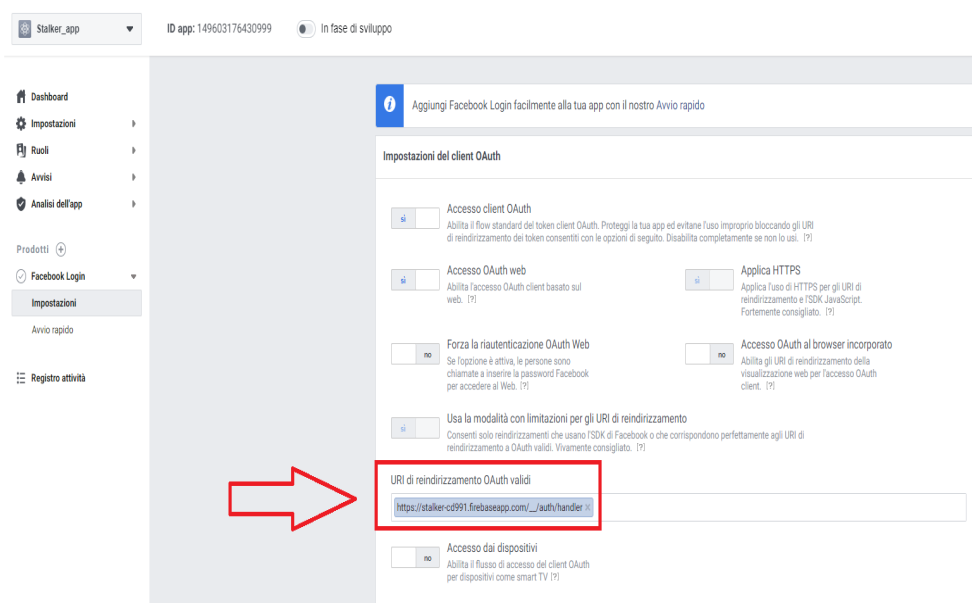


The screenshot shows the 'App configuration' page in the Firebase console. It contains the following fields:

- ID app:** 149603176430999
- Chiave segreta:** ca9adaf2c13d2210e1ee9c7c101bd73e (with a 'Reimposta' button)
- Nome visualizzato:** (empty)
- Spazio dei nomi:** (empty)

Figura 6: i dati da inserire nella console di Firebase

15. copiare il link sottostante e cliccare salva;
16. tornare nella pagina di configurazione del progetto, a sinistra, cliccare su impostazioni del prodotto "Facebook Login" e incollare il link nella casella "URI di re-indirizzamento OAuth validi" quindi salvare le modifiche;



The screenshot shows the 'Impostazioni del client OAuth' (OAuth client settings) page in the Firebase console. A red arrow points to the 'URI di reindirizzamento OAuth validi' (Valid OAuth redirect URIs) field, which contains the URL: `https://stalker-cd991.firebaseio.com/_/auth/handler`. Other settings visible include 'Accesso client OAuth', 'Accesso OAuth web', 'Applica HTTPS', 'Forza la riautenticazione OAuth Web', 'Accesso OAuth al browser incorporato', 'Usa la modalità con limitazioni per gli URI di reindirizzamento', and 'Accesso dai dispositivi'.

Figura 7: link di OAuth

17. a questo punto è possibile avviare l'applicazione ed effettuare l'accesso con l'account del passo #1 di questa guida.

La configurazione dell'accesso con Facebook è ora completata, bisogna quindi configurare il metodo di accesso con Google e quello classico con email e password, seguendo questi passaggi:

1. tornare alla pagina del passo #19 della guida precedente, selezionare la voce "Email/Password" abilitarla, e cliccare salva;
2. selezionare quindi la voce "Google", abilitarla, inserire il nome visibile al prodotto, inserire una mail dell'assistenza al prodotto, e salvare;

2.2.5 Configurazione Firebase-database

1. aprire la console di Firebase, a sinistra, selezionare la voce Database e creare un database;
2. selezionare di avviare in modalità di prova e cliccare avanti;

3. selezionare il server di Firebase sul quale si vuole che vengano salvati i dati e premere "fine";
4. a configurazione avvenuta, si apre il database di Firebase, a questo punto cliccare su "Avvia raccolta" inserendo "utenti" come id raccolta, "utenti" come id documento, e aggiungere i seguenti campi:
 - un campo di nome "nome" di tipo String;
 - un campo di nome "cognome" di tipo String;
 - un campo di nome "organizzazioni" di tipo array che abbia gli elementi di tipo numerico, il documento ottenuto dovrebbe avere questa struttura:

The screenshot shows the Firebase console interface for adding a new document. At the top, a blue header bar contains the text "Avvia una raccolta" and two steps: "1 Assegna un ID alla raccolta" (completed) and "2 Aggiungi il primo documento" (active). Below the header, the "Percorso principale documento" is set to "/utenti". The "ID documento" field is empty, with a blue button labeled "ID automatico". The main area displays the document structure with three fields: "nome" (string), "cognome" (string), and "organizzazioni" (array). Each field has a "Campo" input, a "Tipo" dropdown, and a "Valore" input. The "organizzazioni" field is expanded, showing an array of elements, with the first element having a "Tipo" of "number" and a "Valore" of "0". There are plus and minus icons to add or remove elements. At the bottom right, there are "Annulla" and "Salva" buttons.

Figura 8: Struttura documento di Firebase database

5. la creazione del Firebase Database è stata completata.

2.2.6 Configurazione Sonarqube

Per utilizzare Sonarqube per il tracciamento della qualità del codice sorgente, bisogna seguire i seguenti passi:

- seguire la guida di installazione di Docker reperibile al seguente link: <https://docs.docker.com/get-docker/>;
- dopo aver installato Docker, eseguire il seguente comando: *docker pull sonarqube*;
- quindi aprire il browser e inserire nella barra degli indirizzi: *http://localhost:9000/*;
- per effettuare il login utilizzare come username: admin, e come password: admin;
- quindi si deve eseguire il seguente comando nel Terminal di Android Studio:
`gradlew sonarqube -Dsonar.projectKey=Stalker_App`
`-Dsonar.host.url=http://localhost:9000`
`-Dsonar.login=11ca1a408698561b7b343325b831cb3ad4b279bf.`

Quindi dovresti vedere il progetto nella sezione Projects di della pagina web locale di Sonarqube.

2.3 Architettura applicazione Android

Per lo sviluppo dell'applicazione mobile è stato adottato il pattern architetturale Model-View-ViewModel (MVVM). Esso consente la separazione tra la logica dell'applicazione e la presentazione dei dati. Il seguente diagramma indica la struttura del pattern MVVM indicato da Google per i sviluppatori Android.

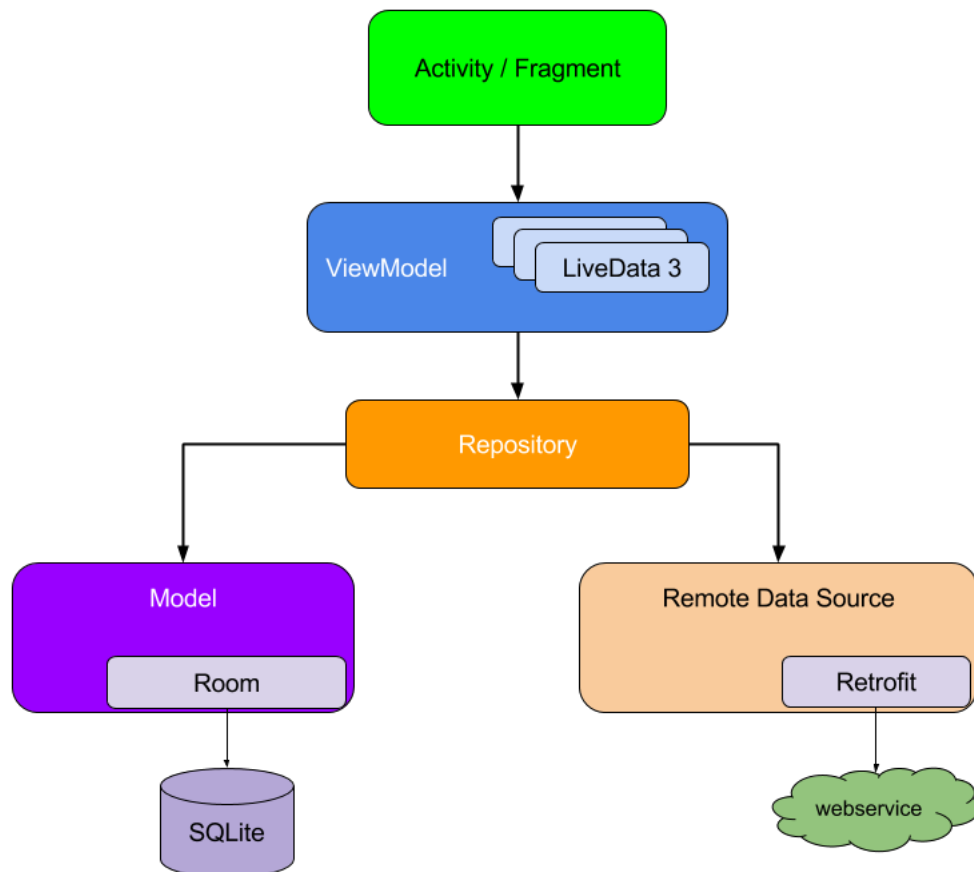


Figura 9: <https://developer.android.com/jetpack/docs/guide>

```

classDiagram
    class Organization {
        +address: String
        +city: String
        +email: String
        +id: long
        +name: String
        +nation: String
        +phone_number: String
        +postal_code: String
        +region: String
        +site: String
        +zip_code_main_name: String
        +zip_dwnn_comparent: String
        +zip_ext: String
        +zip_int: String
        +zip_main: String
        +zip_nation: boolean +rate
        +isTracking: boolean +false
        +isClosed: boolean +false
        +isAnonymous: boolean +false
        +isTrackingKnown: boolean +false
        +parentMatch: String
        +parentSource: String
        +hash(Coordinate): int
        +Organization()
        +equals(): Object: boolean
        +get(Name): String
    }

    class AbstractPlace {
        +id: long
        +name: String
        +hash: int: long
    }

    class RayCasting {
        +RayCasting(longitude: long, latitude: long, Coordinate)
        +isForbidden(): boolean
        +get(IsForbidden): boolean
        +isForbidden(longitude: long, latitude: long, Coordinate): boolean
    }

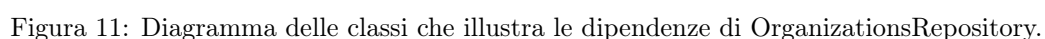
    class PolygonPlace {
        +PolygonPlace()
        +PolygonPlace(long name: String, coordinate: List<Coordinate>)
        +PolygonPlace(long name: String, num_max_people: long, coordinate: List<Coordinate>)
        +isClosed(): boolean
        +isClosed(): Coordinate: boolean
    }

    class Coordinate {
        +Coordinate()
        +Coordinate(longitude: long, latitude: long)
        +Coordinate(longitude: long, latitude: long, longitude: double)
        +isClosed(): String
        +equals(): Object: boolean
        +hash(Coordinate): int
        +radius: double: double
        +getDistanceToOtherCoordinate(): Coordinate: double
    }

    class Line {
        +id: double
        +id: double
        +get(id): double
        +get(id): double
        +Line(int, Coordinate, 2 Coordinate)
        +Line(int, longitude: double, s: double)
        +calculateArea(longitude: double) double
        +intersection(): Line: Coordinate
        +equals(): Object: boolean
        +hash(Coordinate): int
        +isClosed(): String
    }

    Organization --> AbstractPlace
    Organization --> RayCasting
    Organization --> PolygonPlace
    Organization --> Coordinate
    Organization --> Line
    AbstractPlace --> RayCasting
    AbstractPlace --> PolygonPlace
    AbstractPlace --> Coordinate
    AbstractPlace --> Line
    RayCasting --> Coordinate
    PolygonPlace --> Coordinate
    Coordinate --> Line
    
```

In questo diagramma delle classi sono illustrate le entità del mondo reale modellate dall'applicazione. La classe `Organization` rappresenta un'organizzazione presso la quale un utente può farsi tracciare. Essa possiede dei riferimenti ad `AbstractPlace`, i luoghi dell'organizzazione. `AbstractPlace` è una classe astratta, la cui interfaccia pubblica fornisce il metodo `isInside(Coordinate)` che data una coordinata verifica se essa si trova all'interno del luogo o meno. La classe concreta `PolygonPlace` eredita da `AbstractPlace` e rappresenta un luogo a forma poligonale, definito quindi da una lista ordinata di coordinate. Essa utilizza un'istanza della classe `RayCasting`, per calcolare la funzione `isInside(Coordinate)`.



La classe `OrganizationsRepository` si occupa di fornire i dati relativi alle Organizzazioni a qualunque client le richieda (tipicamente i client sono i `ViewModel`), nascondendone la fonte, e lo fa attraverso il metodo `getOrganizations()` che restituisce un `LiveData<List<Organization>>`. Possiede un riferimento ad un'interfaccia `Obtainer`, da dove attinge a nuovi dati nel caso venga richiesto un refresh, uno ad un'interfaccia `FavoritesSource`, che usa per recuperare gli id delle organizzazioni salvate come "preferite", e infine uno ad un'interfaccia `Storage`, che è la sorgente dei dati salvati in locale, la quale viene utilizzata per prelevare i dati da fornire ai client, e per salvare i nuovi dati ottenuti dall'`Obtainer` corrispondente alla sorgente dei dati remoti. Offre inoltre dei metodi per modificare i dati delle organizzazioni, le cui chiamate si riflettono su `Storage`. L'interfaccia `Obtainer` rappresenta una fonte dalla quale si possono ottenere organizzazioni. Fornisce infatti un metodo `getOrganizations()` che restituisce un `LiveData<List<Organization>>`. L'interfaccia `Storage` è una specializzazione di `Obtainer` che aggiunge funzionalità di modifica dei dati tramite opportuni metodi, come ad esempio `updateOrganizations(List<Organization>)`. L'interfaccia `FavoritesSource` invece rappresenta una fonte dalla quale si possono ottenere gli id delle organizzazioni preferite, tramite il metodo `getFavoriteOrganizationsID()`, che restituisce un `LiveData<List<Long>>`.

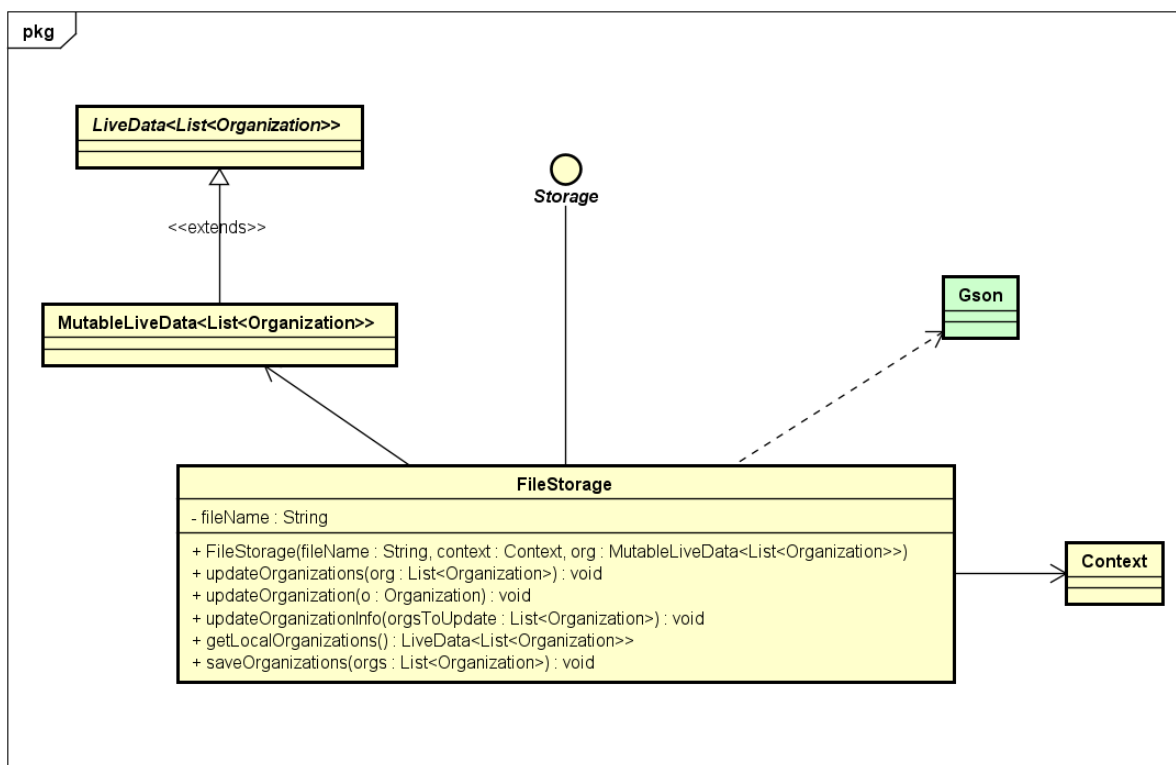


Figura 12: Dipendenze della classe `Storage` ed implementazione concreta.

`FileStorage` è un'implementazione concreta di `Storage` che mantiene le organizzazioni salvate su file, in formato JSON. Utilizza un'istanza della classe `Context`, di Android, per poter gestire lettura e scrittura su di un file di proprietà esclusiva dell'applicazione, e della classe `Gson`, della libreria GSON, per serializzare/deserializzare oggetti in formato JSON.

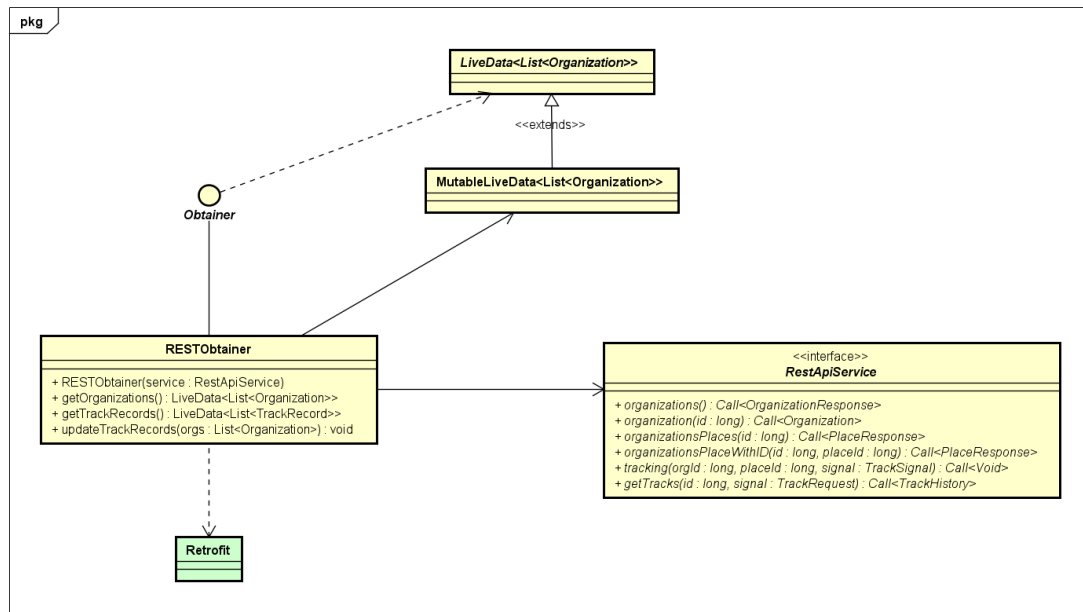


Figura 13: Dipendenze della classe Obtainer ed implementazione concreta.

RESTObtainer invece è un'implementazione concreta di **Obtainer** che recupera i dati delle organizzazioni tramite una chiamata alle REST API. Per farlo utilizza l'interfaccia **RestApiService** la quale espone un metodo per ogni richiesta. La libreria **Retrofit** genera automaticamente l'implementazione concreta di **RestApiService** in modo del tutto trasparente alle classi che dipendono da quest'interfaccia.

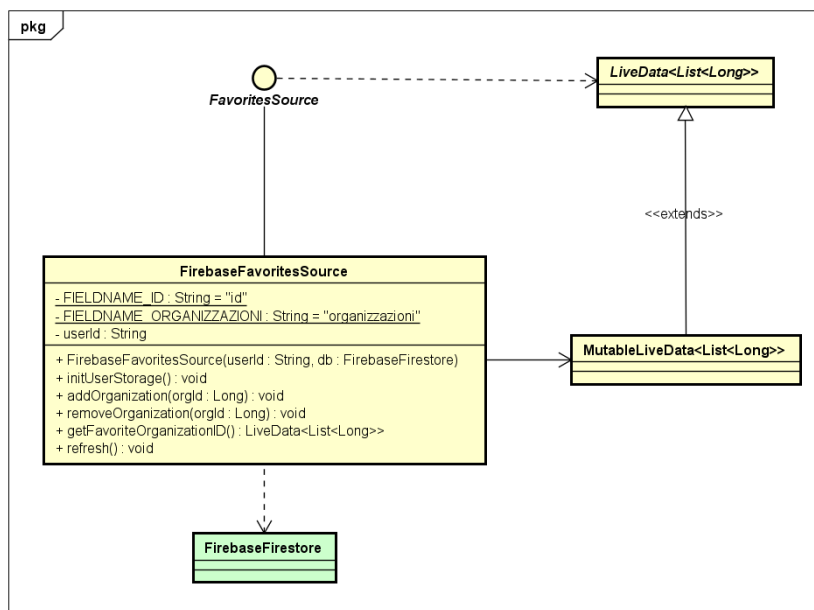


Figura 14: Dipendenze della classe FavoritesSource ed implementazione concreta.

La classe `FirebaseFavoritesSource` implementa l'interfaccia `FavouritesSource`. La sua responsabilità è quella di recuperare da Firebase, servizio cloud di google, gli id delle organizzazioni preferite dell'utente, e di fornirli ai client. Per recuperare gli id delle organizzazioni preferite dal cloud utilizza un'istanza della classe `FirebaseFirestore`, della libreria di Firebase, che gestisce le query al database Cloud Firestore di Firebase.

2.3.2 View e ViewModel

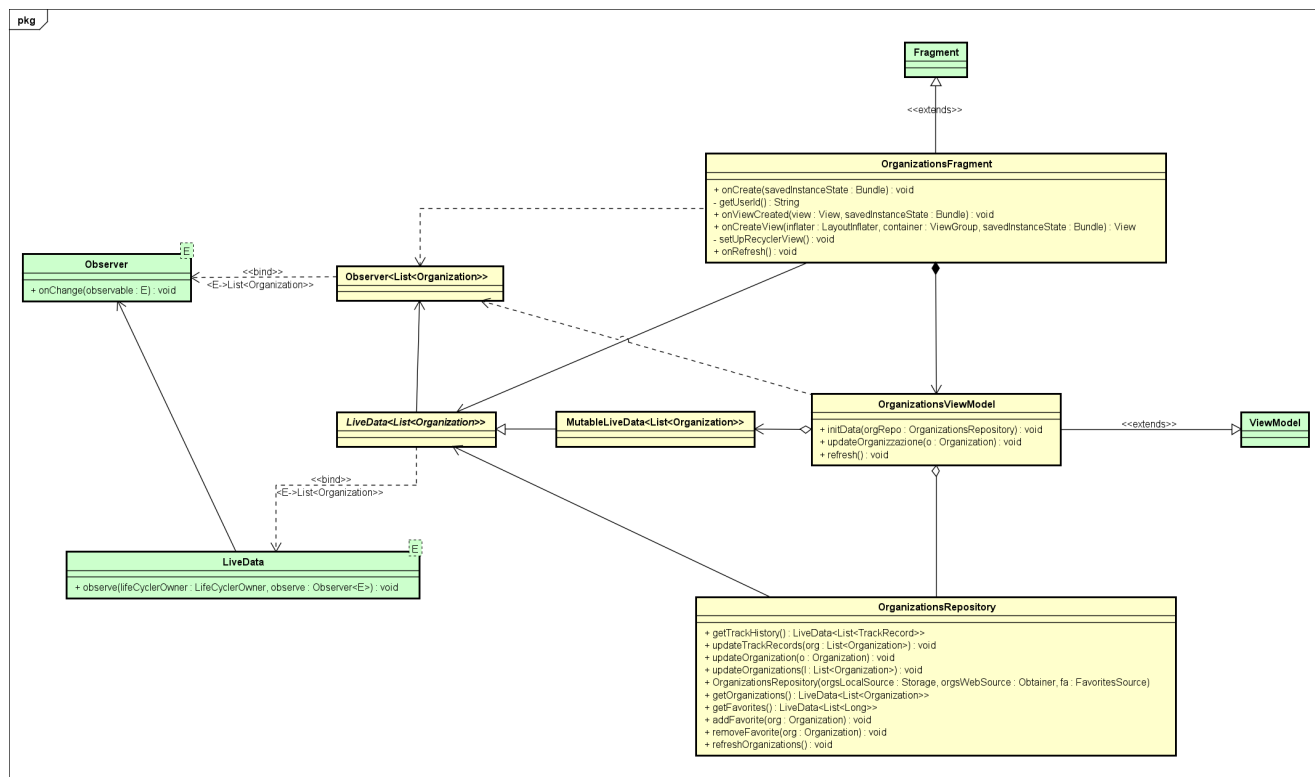


Figura 15: Dipendenze di OrganizationsFragment e OrganizationsViewModel.

La classe `OrganizationsViewModel` fornisce un modello dei dati per la View, interpretando quindi il ruolo di `ViewModel`, ed estende una classe Android che si chiama appunto così. Essa mantiene un riferimento ad un'istanza di `MutableLiveData<List<Organization>>`, che rappresenta un contenitore osservabile per i dati relativi alle organizzazioni, e possiede un riferimento ad `OrganizationsRepository`, che le serve per recuperare i dati sulle organizzazioni. Utilizza un `Observer<List<Organization>>` per osservare i dati restituiti dall'istanza di `OrganizationsRepository` e aggiornare quindi i suoi dati per mantenerne la consistenza. Fornisce il metodo `getOrganizations()` che restituisce il `LiveData` contenente la lista delle organizzazioni, `updateOrganization()` che permette di modificare dati di un'organizzazione riflettendo tale modifica su `OrganizationsRepository`, e `refresh()` che richiede un aggiornamento dell'`OrganizationsRepository` con i dati provenienti da remoto, che si riflette quindi sui dati di `OrganizationsViewModel`, grazie all'`Observer` Pattern utilizzato. La classe `OrganizationsFragment`, che estende `Fragment`, rappresenta una porzione di UI dell'applicazione, più precisamente, quella che mostra le organizzazioni presenti, nel sistema stalker, e che permette di aggiornarne la lista ottenendo nuovi dati. Essa funge da View, presentando i dati relativi alle varie organizzazioni e fornendo dei metodi che gestiscono l'interazione dell'utente con la UI. Utilizza un `Observer<List<Organization>>` per osservare i dati restituiti dall'istanza di `OrganizationsViewModel`, e aggiornare l'interfaccia di conseguenza. Fornisce vari metodi per la reazione agli eventi dati dall'interazione con l'utente, come ad esempio `onRefresh()`, che provoca la chiamata di `refresh()` sull'`OrganizationsViewModel`.

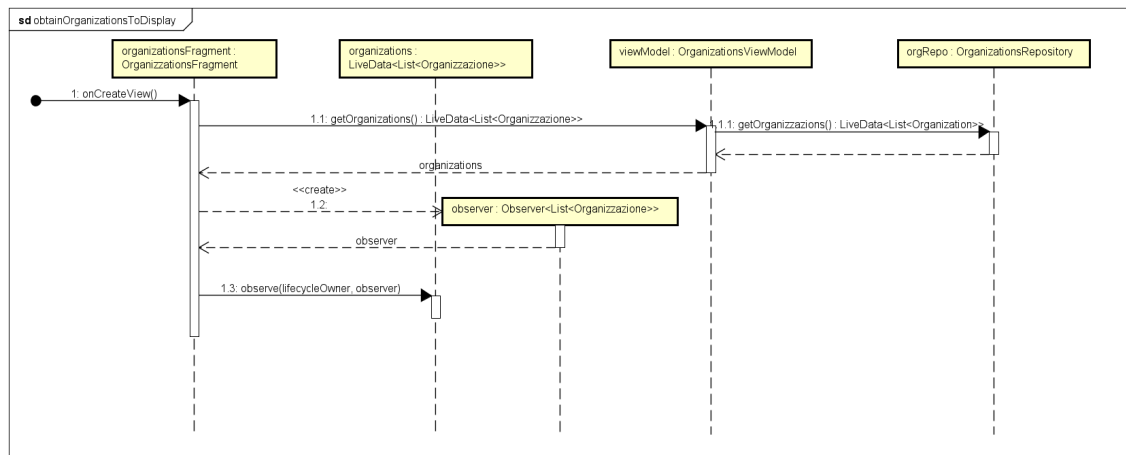


Figura 16: Binding dei dati tra OrganizationsFragment, OrganizationsViewModel e OrganizationsRepository.

Il binding dei dati tra view, viewModel e Model avviene nel modo seguente: l'oggetto organizationsFragment(istanza di OrganizationsFragment) recupera un riferimento ad un *LiveData<List<Organization>>* tramite la chiamata 1.1 su viewModel(istanza dell'Organization-sViewModel di cui organizationsFragment ha un riferimento), che a sua volta recupera un riferimento dello stesso tipo, tramite la chiamata 1.1.1 a orgRepo(istanza dell'Organization-sRepository di cui viewModel ha un riferimento), e lo restituisce al chiamante. organization-sFragment, una volta recuperato l'oggetto, ci registra un'implementazione dell'interfaccia *Observer<List<Organization>>*, tramite la chiamata 1.3, in modo tale da reagire a qualsiasi aggiornamento dei dati.

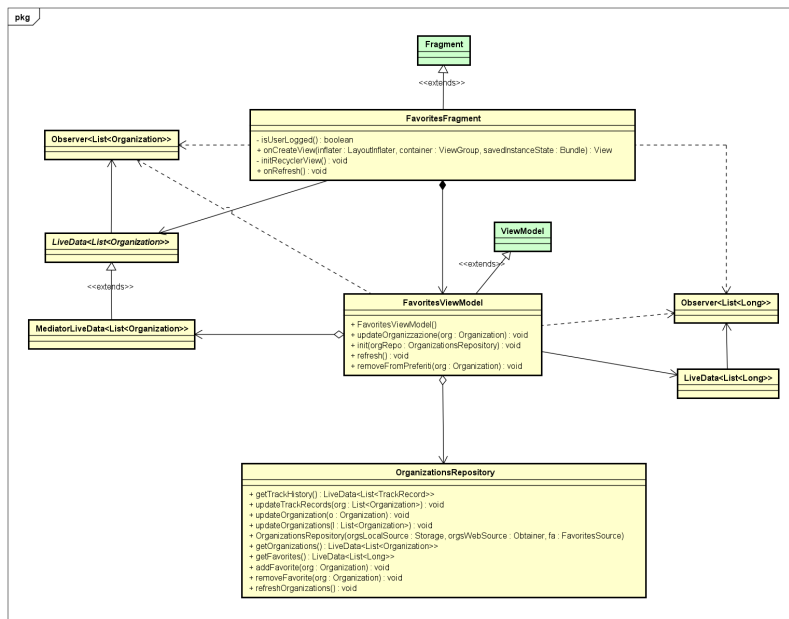


Figura 17: Dipendenze di FavoritesFragment e FavoritesViewModel.

La classe FavoritesViewModel fornisce un modello dei dati per la View, interpretando quindi il ruolo di ViewModel, ed estende una classe Android che si chiama appunto così. Essa mantiene un riferimento ad un'istanza di *MediatorLiveData<List<Organization>>*, che rappresenta un contenitore osservabile per i dati relativi alle organizzazioni preferite dell'utente, un riferimento ad *OrganizationsRepository*, che le serve per recuperare i dati sulle organizzazioni, e un riferimento a un *LiveData<List<Long>>*, che rappresenta un contenitore osservabile circa gli id delle organizzazioni preferite. Utilizza un *Observer<List<Organization>>* per osservare i dati circa le organizzazioni, e un *Observer<List<Long>>* per osservare i dati circa gli id delle organizzazioni preferite, restituiti dall'istanza di *OrganizationsRepository*. Fornisce il metodo *getOrganizations()* che restituisce il *LiveData* contenente la lista delle organizzazioni preferite, *addToFavorites()* che permette di aggiungere un'organizzazione ai preferiti, e *removeFromFavorites()*, che invece permette di toglierne. Infine il metodo *refresh()* richiede un aggiornamento dell' *OrganizationsRepository* con i dati sui preferiti e sulle organizzazioni, provenienti dalle fonti remote aggiornate, che si riflette quindi sui dati di FavoritesViewModel, grazie all'Observer Pattern utilizzato. La classe FavoritesFragment, che estende *Fragment*, rappresenta una porzione di UI dell'applicazione, più precisamente, quella che permette la gestione delle organizzazioni preferite per l'utente loggato. Essa funge da View, presentando i dati relativi alle varie organizzazioni preferite e fornendo dei metodi che gestiscono l'interazione dell'utente con la UI. Utilizza un *Observer<List<Organization>>* per osservare i dati restituiti dall'istanza di *OrganizationViewModel*, e aggiornare l'interfaccia di conseguenza. Fornisce vari metodi per la reazione agli eventi dati dall'interazione con l'utente, come ad esempio *onRefresh()*, che provoca la chiamata di *refresh()* sul FavoritesViewModel.

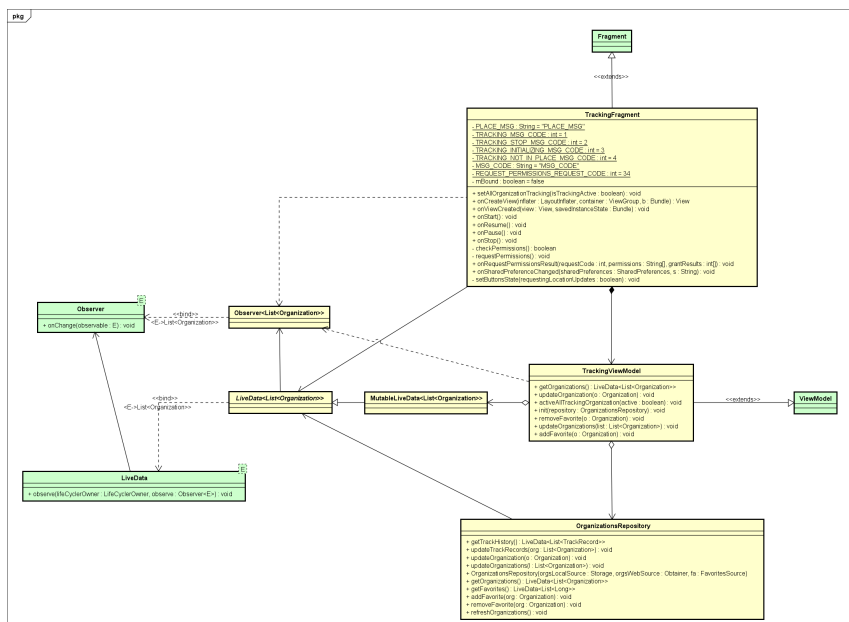


Figura 18: Dipendenze di TrackingFragment e TrackingViewModel.

Nel figura 19 sono illustrate le classi che riguardano l'interfaccia utente relativa alla gestione dei tracciamenti presso le organizzazioni. La struttura del diagramma delle classi è sostanzialmente la stessa vista nella figura 15. La classe TrackingFragment estende da Fragment e funge da View, mentre TrackingViewModel estende da ViewModel. Quest'ultima ha un riferimento ad OrganizationsRepository e lo utilizza per ottenere le organizzazioni implementando il pattern Observer tramite l'ausilio dei LiveData. TrackingFragment utilizza un *Observer<List<Organization>>* per osservare i dati restituiti dall'istanza di TrackingViewModel, e aggiornare l'interfaccia di conseguenza. Fornisce vari metodi per la reazione agli eventi dati dall'interazione con l'utente, come ad esempio il metodo *setAllOrganizationsTracking()*, che permette di attivare/disattivare tutti i tracciamenti disponibili nella schermata.

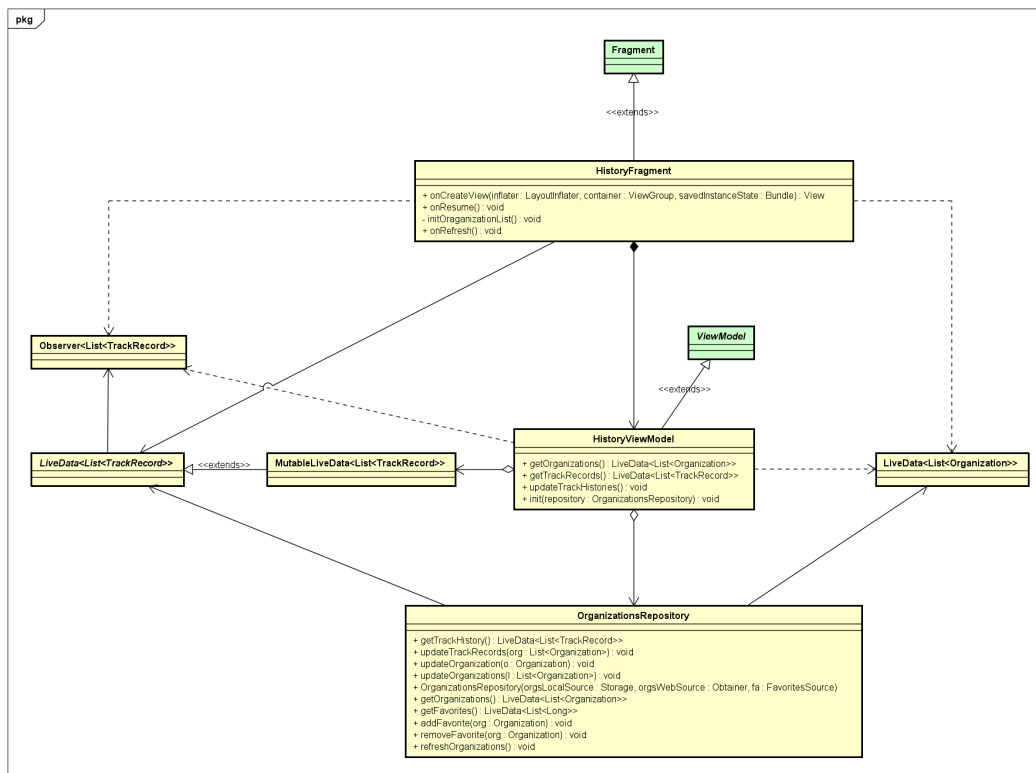


Figura 19: Dipendenze di HistoryFragment e HistoryViewModel.

La classe HistoryViewModel fornisce un modello dei dati per la View, interpretando quindi il ruolo di ViewModel, ed estende una classe Android che si chiama appunto così. Essa mantiene un riferimento ad un'istanza di *MutableLiveData<List<TrackRecords>>*, che rappresenta un contenitore osservabile per i dati relativi alle organizzazioni, e possiede un riferimento ad *OrganizationsRepository*, che le serve per recuperare i dati sulle organizzazioni. Utilizza un *Observer<List<TrackRecords>>* per osservare i dati relativi ai record dei tracciamenti, restituiti dall'istanza di *OrganizationsRepository* e aggiornare quindi i suoi dati per mantenerne la consistenza. Fornisce il metodo *getOrganizations()* che restituisce il *LiveData* contenente la lista delle organizzazioni, *getTrackRecords()* che restituisce il *LiveData* contenente la lista dei record relativi ai tracciamenti, *updateTrackHistory()* che richiede, all'*OrganizationsRepository*, un aggiornamento dei record relativi ai tracciamenti. La classe HistoryFragment, che estende *Fragment*, rappresenta una porzione di UI dell'applicazione, più precisamente, quella che permette di visualizzare i tracciamenti degli utenti, loggati tramite LDAP, registrati nel sistema stalker. Essa funge da View, presentando i dati relativi ai tracciamenti e fornendo dei metodi che gestiscono l'interazione dell'utente con la UI. Utilizza un *Observer<List<TrackRecord>>* per osservare i dati restituiti dall'istanza di *HstoryViewModel*, e aggiornare l'interfaccia di conseguenza. Fornisce vari metodi per la reazione agli eventi dati dall'interazione con l'utente, come ad esempio *onRefresh()*, che provoca la chiamata di *updateTrackHistory()* sull'istanza di *HistoryViewModel*.

2.3.3 TrackingService

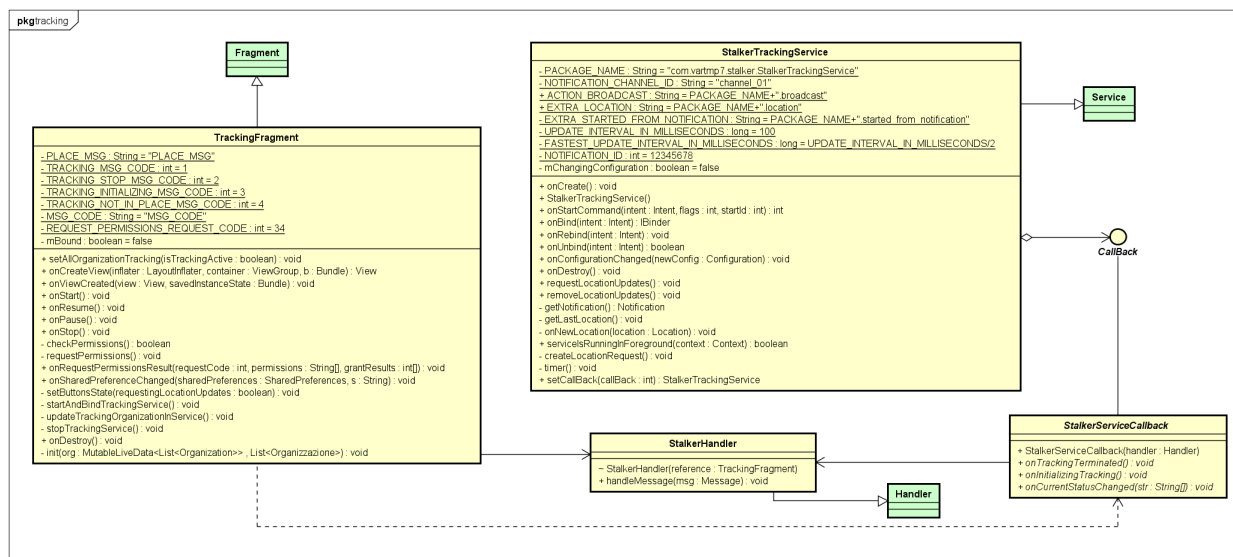


Figura 20: Diagramma delle classi per TrackingService.

La classe `StalkerTrackingService` eredita da `Service`, classe di Android necessaria ad avviare un servizio in background, cioè che non necessita di un Activity attiva. Essa si occupa di effettuare, in background, i tracciamenti presso le organizzazioni. Per implementare la comunicazione tra `StalkerTrackingService` e `TrackingFragment` è stato utilizzato il Command Pattern. `StalkerTrackingService` ha il ruolo dell'invoker. Essa ha un riferimento all'interfaccia `Callback`, che interpreta quindi il command. `StalkerServiceCallback` implementa `Callback` e ha una dipendenza verso `StalkerHandler`, che eredita dalla classe Android `Handler`, e gioca il ruolo di receiver. Il `TrackingFragment` funge da client e ha un riferimento alla classe `StalkerHandler`.

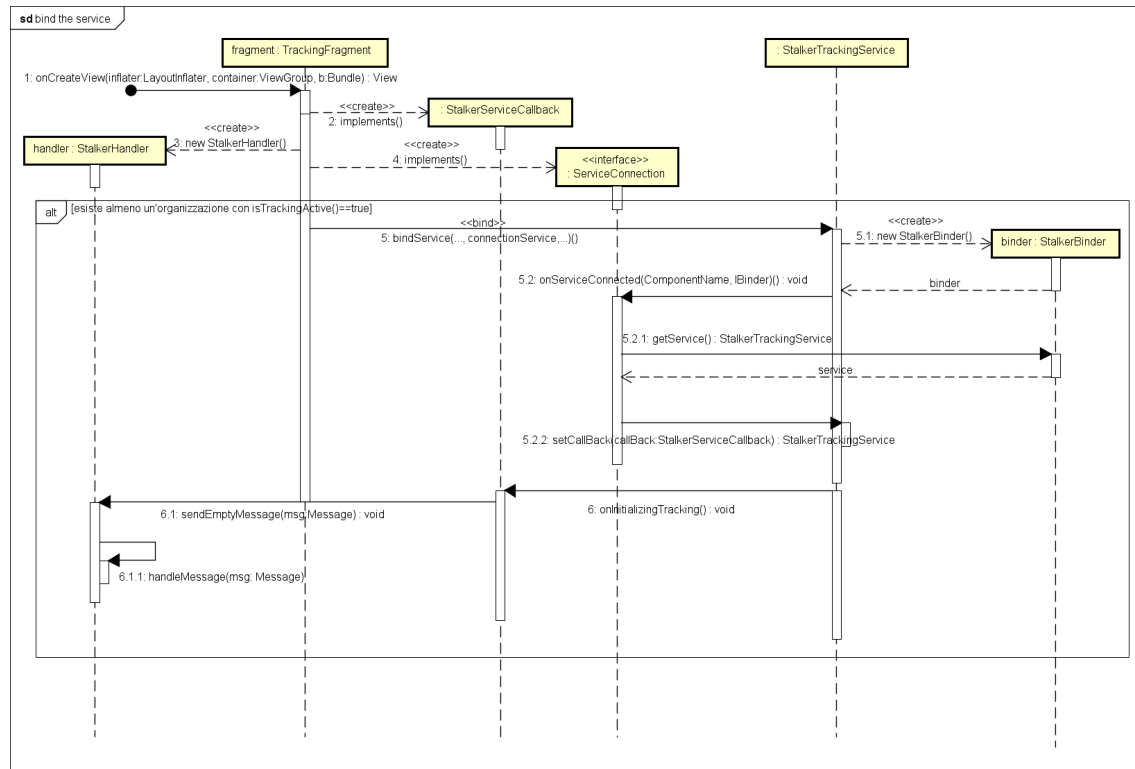


Figura 21: Diagramma di sequenza che illustra come avviene un tracciamento tramite il TrackingService.

La figura 21 mostra come avviene il binding tra StalkerTrackingService e il TrackingFragment.

1. onCreateView(...): quando viene avviata l'applicazione, viene chiamato questo metodo per inizializzare la vista;
2. implements di StalkerServiceCallback: implementazione dell'interfaccia StalkerServiceBack che successivamente verrà utilizzata per la comunicazione tra il Service e il fragment.
3. new StalkerHandler(fragment): istanza dell'Handler che quando viene chiamato qualche metodo del callback dal StalkerTrackingService sa come portare a termine tali metodi.
4. implements di ServiceConnection: l'interfaccia che serve per il binding del StalkerTrackingService.
5. se esiste almeno un'organizzazione con il campo dati isTrackingActive == true, allora viene chiamato il metodo bindService(..., serviceConnection, ...).
 - (a) in questo modo viene fatto il binding del StalkerTrackingService, e viene chiamato il metodo onBind(), che costruisce un oggetto di tipo StalkerBinder e lo restituisce.
 - (b) quindi il sistema Android chiama il metodo onServiceConnected(..., IBinder) dell'interfaccia ServiceConnection, passandogli l'oggetto restituito dal metodo onBind().

- i. all'oggetto di tipo IBinder viene fatto il downcasting a StalkerBinder, quindi posso utilizzare il metodo getService di tale classe per ottenere il riferimento dello StalkerTrackingService.
 - ii. quindi sul riferimento dello service viene chiamato il metodo setCallback passandogli l'interfaccia implementata al passo 1.
6. in seguito viene chiamato il metodo onInitializingTracking() dell'interfaccia StalkerServiceCallback;
- (a) che a sua volta chiama il metodo sendMessage() dell'oggetto handler.
 - i. handler chiama il metodo handleMessage() per portare al termine l'operazione.

2.3.4 Diagramma dei package

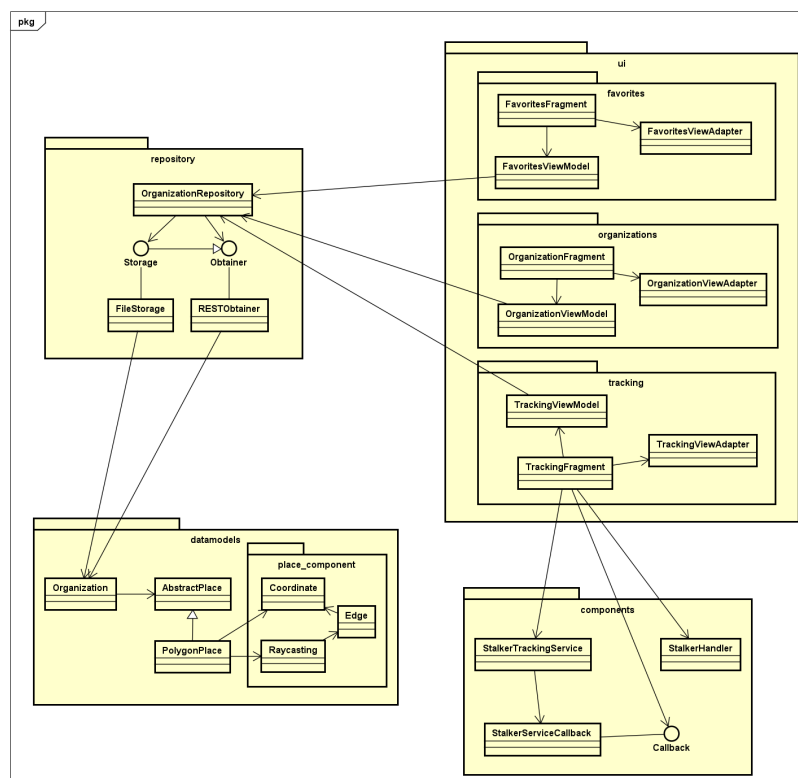


Figura 22: Diagramma dei package dell'applicazione mobile.

2.4 Estendibilità applicazione Android

2.4.1 Modificare l'approccio nel salvataggio dei dati persistenti

Allo stato attuale l'applicazione salva i dati relativi alle organizzazioni su file, in formato JSON. Questo perchè al costruttore di OrganizationsRepository viene passata un'istanza di FileStorage come implementazione di Storage. Tuttavia è possibile modificare il modo di gestire i dati in locale, ad esempio sfruttando le caratteristiche offerte da un database relazionale. Per farlo è sufficiente definire una nuova implementazione dell'interfaccia Storage che si occupa appunto del salvataggio e recupero delle informazioni sulle organizzazioni, e fornirla al costruttore di OrganizationsRepository, al momento dell'istanziatura di quest'ultima. La figura 23 mostra appunto un esempio dato dalla definizione della classe RelationalDBStorage.

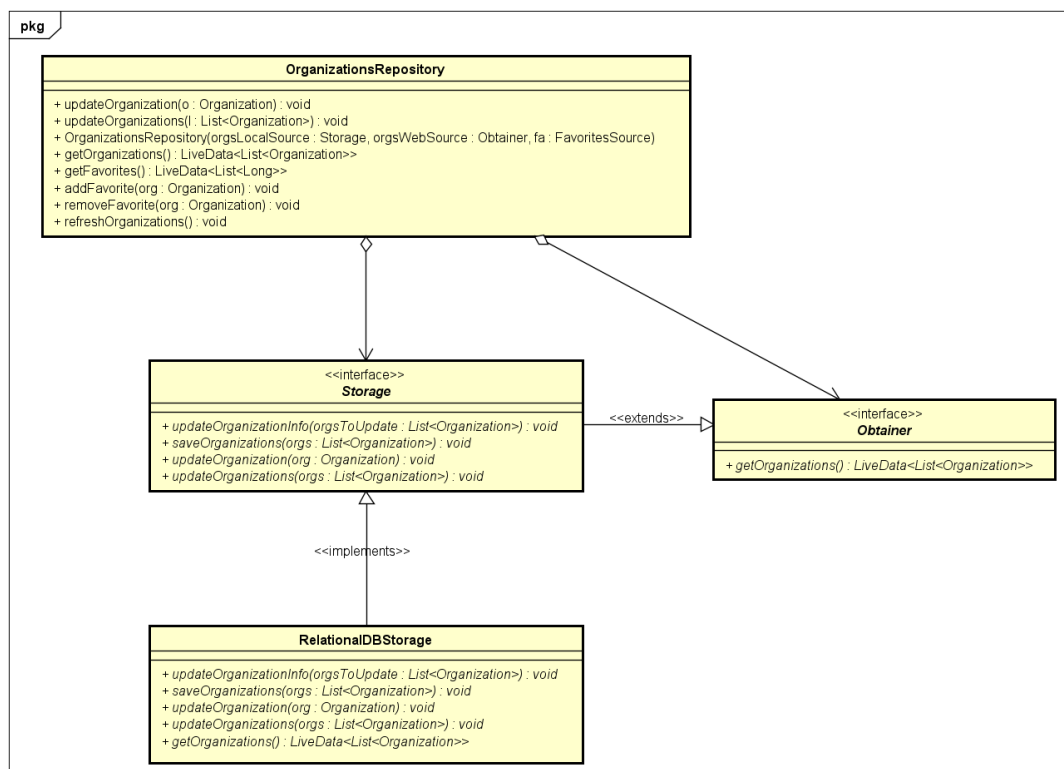


Figura 23: un esempio di modifica sul salvataggio dei dati

2.4.2 Modificare la fonte di dati remoti

L'applicazione allo stato attuale ottiene i dati aggiornati sulle organizzazioni dal backend, attraverso le REST API, tramite richieste HTTP. Viene infatti costruita l'istanza di OrganizationsRepository passando un'istanza di RESTObtainer, come implementazione di Obtainer, al costruttore. È possibile modificare la fonte, o comunque il modo di recuperare i dati da remoto, semplicemente definendo una nuova classe che implementa l'interfaccia Obtainer, e passandone un'istanza al costruttore di OrganizationsRepository.

2.5 Test applicazione Android

2.5.1 Aggiungere nuovi test

Se si desidera aggiungere nuovi test, è opportuno innanzitutto valutare se per il test sia necessario utilizzare le caratteristiche del sistema operativo Android, trattandosi quindi un instrumented test, oppure se è sufficiente un semplice unit test. Per l'instrumented test è necessario ad esempio nel caso sia necessario testare una classe che estende Activity o Fragment. Per aggiungere nuovi unit test è opportuno creare una nuova classe nella cartella *test* e aggiungere la annotazione `@RunWith`, seguita dalla classe che si vuole utilizzare per il test, racchiusa dalle parentesi.

Per aggiungere nuovi instrumented test, cioè test che eseguono su un dispositivo Android (fisico o emulato) è opportuno creare una nuova classe nella cartella *androidTest* e aggiungere la annotazione `@RunWith`, seguita dalla classe che si vuole utilizzare per il test, racchiusa tra le parentesi.

In entrambi i casi il nome di queste classi Java devono terminare con Test. Un esempio corretto è:

`TrackingFragmentTest.java`

2.5.2 Eseguire i test

Per eseguire i test bisogna eseguire il comando:

`gradlew clean test jacocoTestDebugUnitTestReport sonarqube`

dopo aver avviato il container di sonarqube, come specificato in fase di setup.

In questo modo vengono eseguiti i test, fatto analisi statica del codice e generato il report di code coverage. In alternativa è possibile l'esecuzione direttamente da Android Studio tramite GUI aprendo la classe di test d'interesse, o la test suite d'interesse, e nell'editor di codice, vicino al nome della classe, c'è un pulsante verde, è sufficiente cliccare tale pulsante e selezionare una delle voci disponibili.

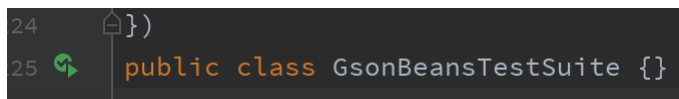


Figura 24: Pulsante per eseguire i test.

3 Backend

3.1 Tecnologie backend

3.1.1 Python

I linguaggi proposti dal proponente per la realizzazione del backend erano:

- Python;
- Java;
- Javascript (NodeJS).

Dopo un'attenta analisi sui vantaggi e svantaggi dei diversi linguaggi abbiamo scelto Python per le seguenti motivazioni:

- è un linguaggio di alto livello, molto semplice da imparare e che permette di sviluppare software complessi in poco tempo;
- è portatile (in quanto interpretato) e quindi il software prodotto può essere eseguito su qualsiasi piattaforma abbia l'interprete Python installato;
- è un linguaggio molto diffuso e ricco di librerie e framework anch'essi molto diffusi;
- è un linguaggio multi-paradigma ed è quindi possibile usare sia la programmazione ad oggetti che la programmazione procedurale e funzionale.

<https://www.python.org>

3.1.2 PostgreSQL

PostgreSQL è un database relazionale noto per la sua affidabilità e ampiamente supportato da librerie di terze parti.

<https://www.postgresql.org>

3.1.3 RethinkDB

RethinkDB è un database *NoSQL_G realtime_G* open-source, è stato scelto in particolar modo per la caratteristica di essere real-time, grazie alla quale è possibile sapere in tempo reale il numero di persone all'interno di un luogo. Inoltre non è un prodotto AaaS ed è quindi possibile usarlo senza alcuna restrizione.

<https://rethinkdb.com>

3.1.4 Kubernetes

Kubernetes è un sistema open-source che permette di automatizzare il deployment, la scalabilità e la gestione di applicazioni containerizzate. È stato scelto sotto consiglio del proponente per soddisfare il vincolo della scalabilità orizzontale, inoltre è disponibile una vasta documentazione sulla quale fare affidamento.

<https://kubernetes.io>

3.1.5 RabbitMQ

RabbitMQ è un message broker open-source che si adatta alla richiesta del proponente di avere scalabilità orizzontale grazie alla gestione della coda di messaggi.

<https://www.rabbitmq.com>

3.1.6 Flask

Flask è un web-framework leggero scritto in Python e facilmente estendibile.

<https://flask.palletsprojects.com/en/1.1.x/>

3.1.7 Flask-SQLAlchemy

Estensione per Flask che aggiunge il supporto per SQLAlchemy. Il suo scopo è quello di semplificare l'interazione con il database sfruttando la tecnica dell'object-relational mapping (ORM).

<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>

3.1.8 Flask-RESTful

Estensione per Flask che aggiunge il supporto per la realizzazione di REST API.

<https://flask-restful.readthedocs.io/en/latest/>

3.1.9 Flask-JWT

Estensione per Flask che aggiunge il supporto ai JSON Web Token (JWT).

<https://flask-jwt-extended.readthedocs.io/en/stable/>

3.1.10 PyTest

PyTest è uno dei framework di test più utilizzati per Python. Ha inoltre la possibilità di essere esteso tramite plugin.

<https://docs.pytest.org/en/latest/>

3.1.11 ldap3

Libreria di Python che permette di interfacciarsi con un server LDAP utilizzato per l'autenticazione dei dipendenti di un'organizzazione.

<https://ldap3.readthedocs.io/en/latest/>

3.2 Setup e configurazione dell'ambiente di sviluppo per il backend

3.2.1 Requisiti di sistema

I requisiti di sistema richiesti per poter installare il backend sono i seguenti:

- Docker 19.03.8.

È inoltre consigliato avere installato

- Python 3.7.7;
- PyCharm 2019.3.4;
- Docker Compose 1.25.4;
- Kubernetes 1.15.9.
- Scaffold 1.4.0;
- Make 3.81.

3.2.2 Avvio del backend

3.2.2.1 Make Per avviare il backend viene fornito un Makefile con il quale è possibile avviare il backend con il comando:

```
make start-backend
```

oppure, in caso sia necessario avviare il backend utilizzando https si può usare il comando:

```
make start-backend-https
```

Questi due comandi usano Docker Compose per avviare il backend.

3.2.2.2 Docker Compose Per avviare il backend è possibile usare Docker Compose tramite il comando:

```
docker-compose up --build
```

Questo si occupa di avviare:

- un container postgres;
- un container rethinkdb;
- un container leggendo il Dockerfile (il backend);
- un container ldap;
- un container phpldapadmin.

Si occupa inoltre di creare (in caso non esistano):

- un volume per i dati di postgres;
- un volume per i dati di rethinkdb;
- una rete per far comunicare i vari servizi.

3.2.2.3 Skaffold In caso si voglia eseguire il backend tramite Kubernetes è possibile utilizzare Skaffold per avviare tutti i deployment, secret, service e volume necessari con il comando:

```
skaffold dev --port-forward
```

3.2.2.4 Docker In caso non si voglia usare né Docker Compose né Kubernetes viene fornito il Dockerfile con il quale poter compilare l'immagine del backend tramite il comando:

```
docker build -t stalker-backend
```

Prima di avviare l'immagine appena compilata è necessario settare alcune variabili d'ambiente necessarie al corretto funzionamento del backend. È anche possibile passarle al container al momento dell'avvio dello stesso tramite l'opzione `-env`.

- `FLASK_APP=stalker_backend`;
- `FLASK_ENV`;
- `RETHINK_URL`;
- `DATABASE_TYPE`;
- `BASE_POSTGRES_URL` (necessaria solo in caso `DATABASE_TYPE` sia `postgres://`);
- `POSTGRES_USER` (necessaria solo in caso `DATABASE_TYPE` sia `postgres://`);
- `POSTGRES_PASSWORD` (necessaria solo in caso `DATABASE_TYPE` sia `postgres://`);
- `TESTING`;
- `FLASK_DEBUG`;
- `EMAIL_PASSWORD`;
- `MAIL_SUPPRESS_SEND`;

È inoltre necessario avviare RethinkDB (raggiungibile all'url indicato nella variabile d'ambiente `RETHINK_URL`) e PostgreSQL in caso la variabile d'ambiente `DATABASE_TYPE` sia `postgres://`.

Una volta compiuti tutti questi passaggi è possibile avviare il container con l'immagine del backend tramite il seguente comando:

```
docker run -p 5000:5000 stalker-backend
```

3.3 Architettura backend

Per lo sviluppo del backend è stata adottata l'architettura a micro-servizi.

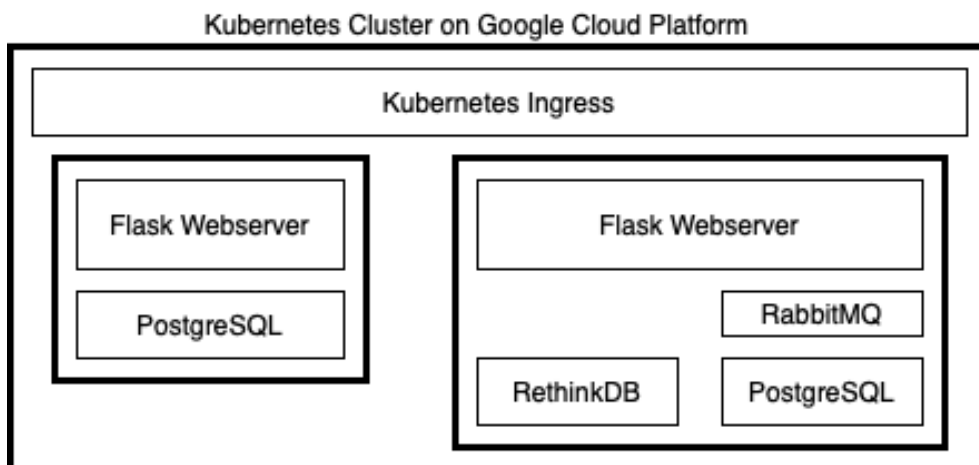


Figura 25: Architettura del backend.

I micro-servizi individuati sono:

- uno per la gestione delle organizzazioni e dei luoghi, tutti dati che vengono salvati all'interno del database relazionale, usato principalmente per ottenere informazioni e gestire questi elementi;
- uno per la gestione dei tracciamenti e il conteggio del numero di persone all'interno dei luoghi delle organizzazioni che, utilizzando RethinkDB, può offrire in tempo reale questi dati. Inoltre, sfruttando una coda RabbitMQ, permette di distribuire nel tempo la scrittura sul database PostgreSQL, evitando così di creare colli di bottiglia. È questo il micro-servizio che beneficerà maggiormente della scalabilità offerta da Kubernetes.

La topologia scelta è quella REST API.

Ogni micro-servizio è sviluppato adottando il pattern N-tier.

3.3.1 Modelli

I modelli sono le entità gestite all'interno del database relazionale PostgreSQL. Sono delle classi che derivano dalla classe base Model della libreria Flask-SQLAlchemy che permette di applicare la tecnica dell'object-relational mapping (ORM).

Ogni modello rappresenta una tabella e ogni attributo del modello rappresenta una colonna della tabella.

Ogni istanza della classe rappresenta un record della tabella.

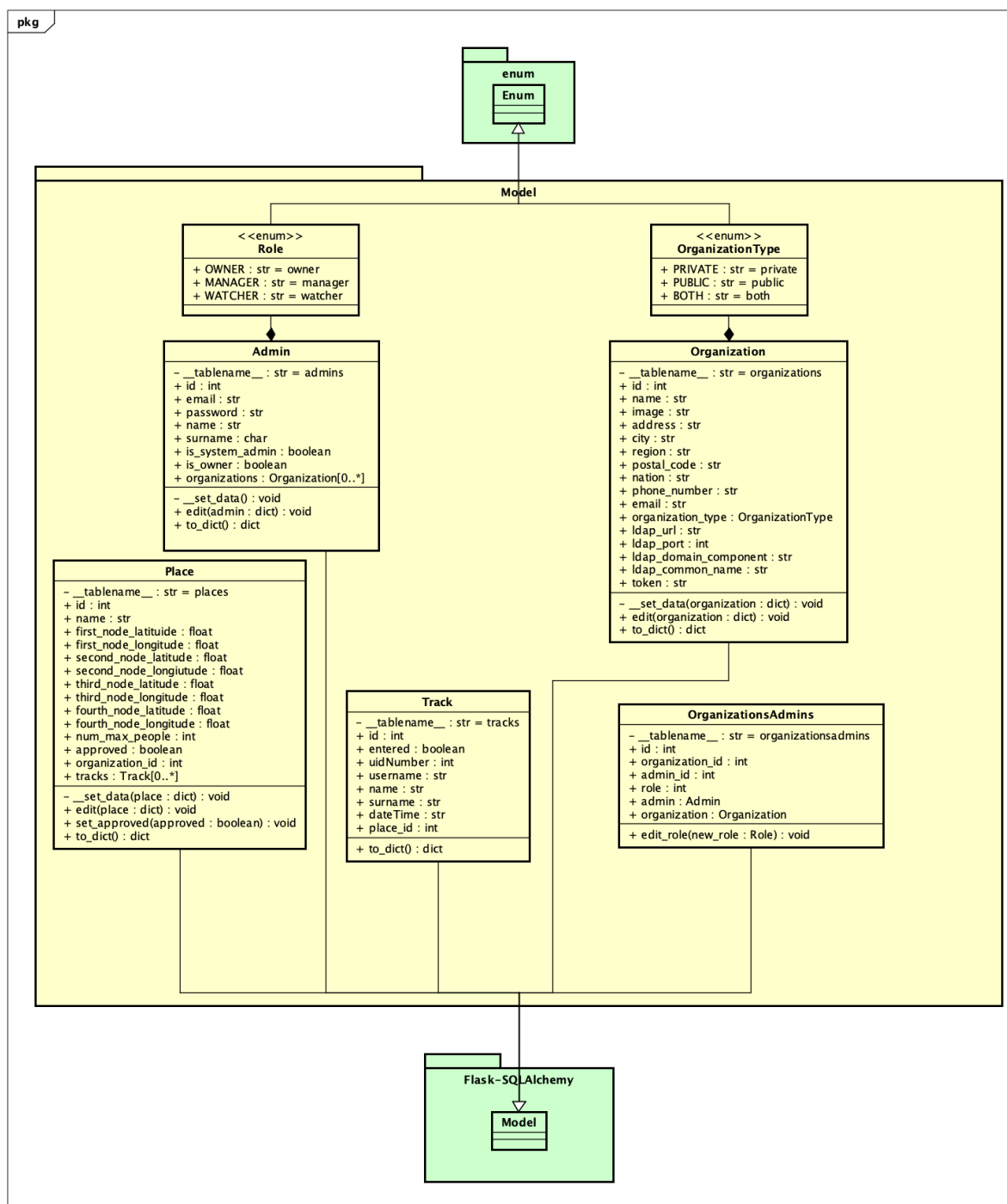


Figura 26: Diagramma delle classi dei modelli del backend.

- **Organization**: rappresenta un'organizzazione all'interno del sistema Stalker;

- Place: rappresenta un luogo di un'organizzazione;
- Track: rappresenta un tracciamento autenticato per un luogo;
- Admin: rappresenta un amministratore;
- OrganizationsAdmins: rappresenta la relazione tra amministratore e organizzazioni; siccome questa relazione è di tipo molti a molti è stato necessario creare un'entità apposita.

Il meta-attributo `__tablename__` viene usato da Flask-SQLAlchemy come il nome che assumerà la tabella quando verrà creata.

Gli altri attributi sono segnati come pubblici in quanto si è fatto uso dei property decorator, una funzionalità di Python che permette di definire un getter e un setter per degli attributi che vengono applicati in automatico quando si utilizza un attributo della classe. Siccome lo standard UML 2.5 non prevede un modo per definire questo comportamento e vengono effettivamente usati come attributi, nel diagramma delle classi sono segnati come attributi pubblici.

Di seguito è riportato un esempio di come viene scritto l'attributo di una classe:

```
@property
def attributo(self):
    return self._attributo

@attributo.setter
def attributo(self, new_value):
    self._attributo = new_value
```

3.3.2 Risorse

Le REST API espongono delle risorse, rappresentate tramite classi che ereditano dalla classe base `Resource` della libreria `Flask-RESTful`, la quale si occupa di chiamare il metodo della classe corrispondente al metodo della richiesta HTTP (GET, POST, PUT, DELETE).

Ogni classe implementa quindi i soli metodi che vuole soddisfare, la classe base `Resource` si occuperà di restituire l'errore 405 Method Not Allowed in caso venga fatta una richiesta non gestita.

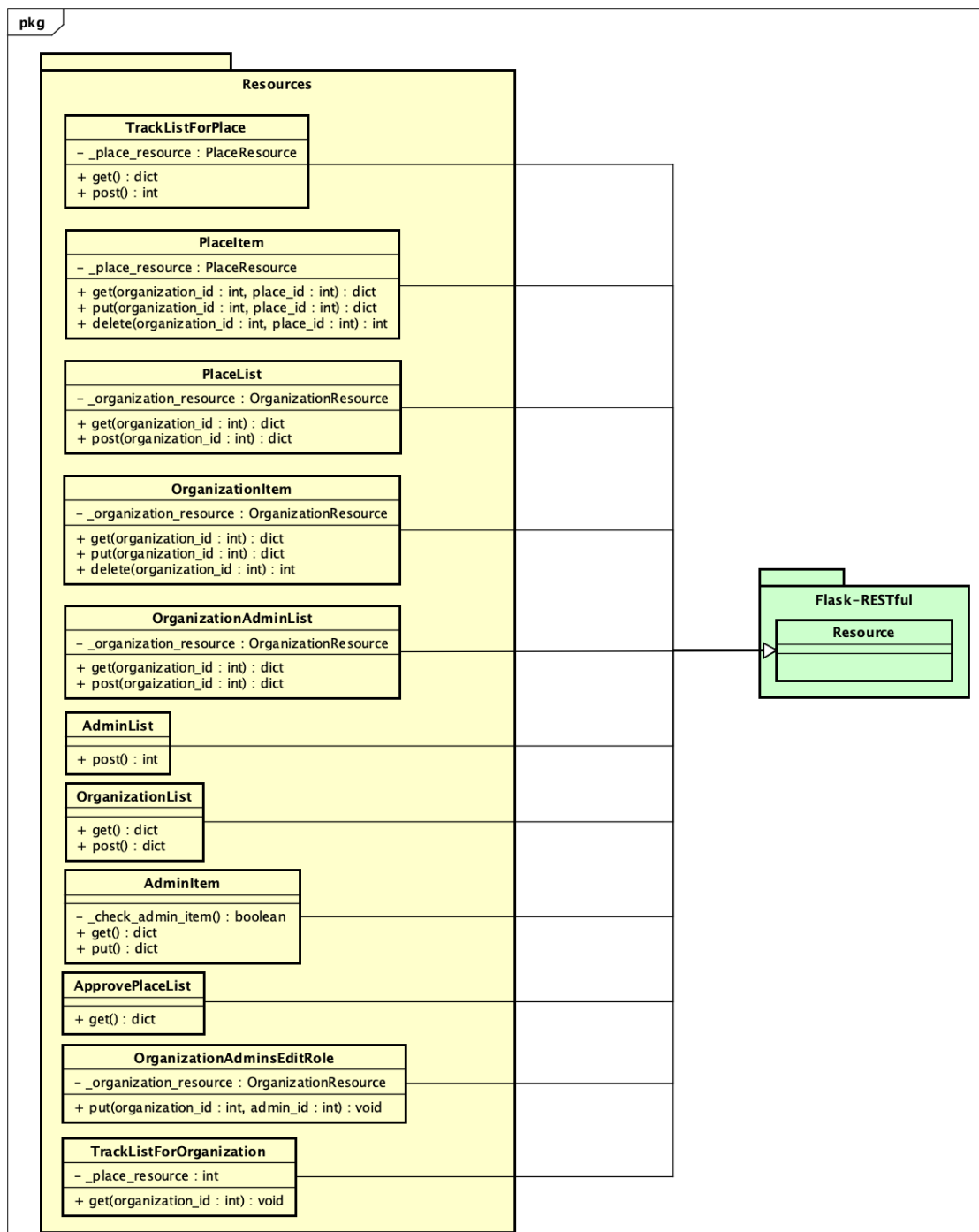


Figura 27: Diagramma delle classi delle risorse in relazione a Resource.

- TrackListForPlace: classe che si occupa di gestire le risorse *tracciamenti di un luogo*. Mette a disposizione i metodi:

- get: permette di ottenere la lista dei tracciamenti del luogo indicato;
 - post: permette di creare un nuovo tracciamento.
- PlaceItem: classe che si occupa di gestire la risorsa *luogo*. Mette a disposizione i metodi:
 - get: permette di ottenere le informazioni del luogo indicato;
 - put: permette di modificare le informazioni del luogo indicato;
 - delete: permette di eliminare il luogo indicato.
- PlaceList: classe che si occupa di gestire le risorse *luoghi*. Mette a disposizione i metodi:
 - get: permette di ottenere la lista dei luoghi dell'organizzazione indicata;
 - post: permette di creare un nuovo luogo.
- OrganizationItem: classe che si occupa di gestire la risorse *organizzazione*. Mette a disposizione i metodi:
 - get: permette di ottenere le informazioni dell'organizzazione indicata;
 - put: permette di modificare le informazioni dell'organizzazione indicata;
 - delete: permette di eliminare l'organizzazione indicata.
- OrganizationAdminList: classe che si occupa di gestire le risorse *amministratore di un'organizzazione*. Mette a disposizione i metodi:
 - get: permette di ottenere la lista degli amministratori che possono operare sull'organizzazione indicata;
 - post: permette di associare un nuovo amministratore all'organizzazione indicata, in caso l'amministratore non esista ancora allora verrà creato.
- AdminList: classe che si occupa di gestire le risorse *amministratori owner e di sistema*. Mette a disposizione i metodi:
 - get: permette di ottenere una lista con tutti gli amministratori presenti nel sistema;
 - post: permette di creare un nuovo amministratore di sistema oppure owner.
- OrganizationList: classe che si occupa di gestire le risorse *organizzazioni*. Mette a disposizione i metodi:
 - get: permette di ottenere la lista di tutte le organizzazioni presenti nel sistema;
 - post: permette di creare una nuova organizzazione.
- AdminItem: classe che si occupa di gestire la risorsa *amministratore*. Mette a disposizione i metodi:
 - get: permette di ottenere le informazioni dell'amministratore indicato;
 - put: permette di modificare le informazioni dell'amministratore indicato.
- ApprovePlaceList: classe che si occupa di gestire le risorse *luoghi da approvare*. Mette a disposizione il metodo:
 - get: permette di ottenere la lista di luoghi da approvare.

- `TrackListForOrganization`: classe che si occupa di gestire le risorse *tracciamenti dell'organizzazione*. Mette a disposizione il metodo:
 - `get`: permette di ottenere tutti i tracciamenti di un'organizzazione.
- `OrganizationAdminEditRole`: classe che si occupa di gestire la risorsa *relazione amministratore-organizzazione* in particolare il ruolo tra queste due entità. Mette a disposizione il metodo:
 - `put`: permette di modificare il ruolo dell'amministratore rispetto all'organizzazione indicata.
- `OrganizationsAdminList`: classe che si occupa di gestire le risorse *organizzazioni di un amministratore*. Mette a disposizione il metodo:
 - `get`: permette di ottenere la lista di organizzazioni sulle quali l'amministratore può operare.

Alcune classi hanno un attributo privato che viene passato al costruttore tramite il meccanismo di dependency injection (constructor injection).

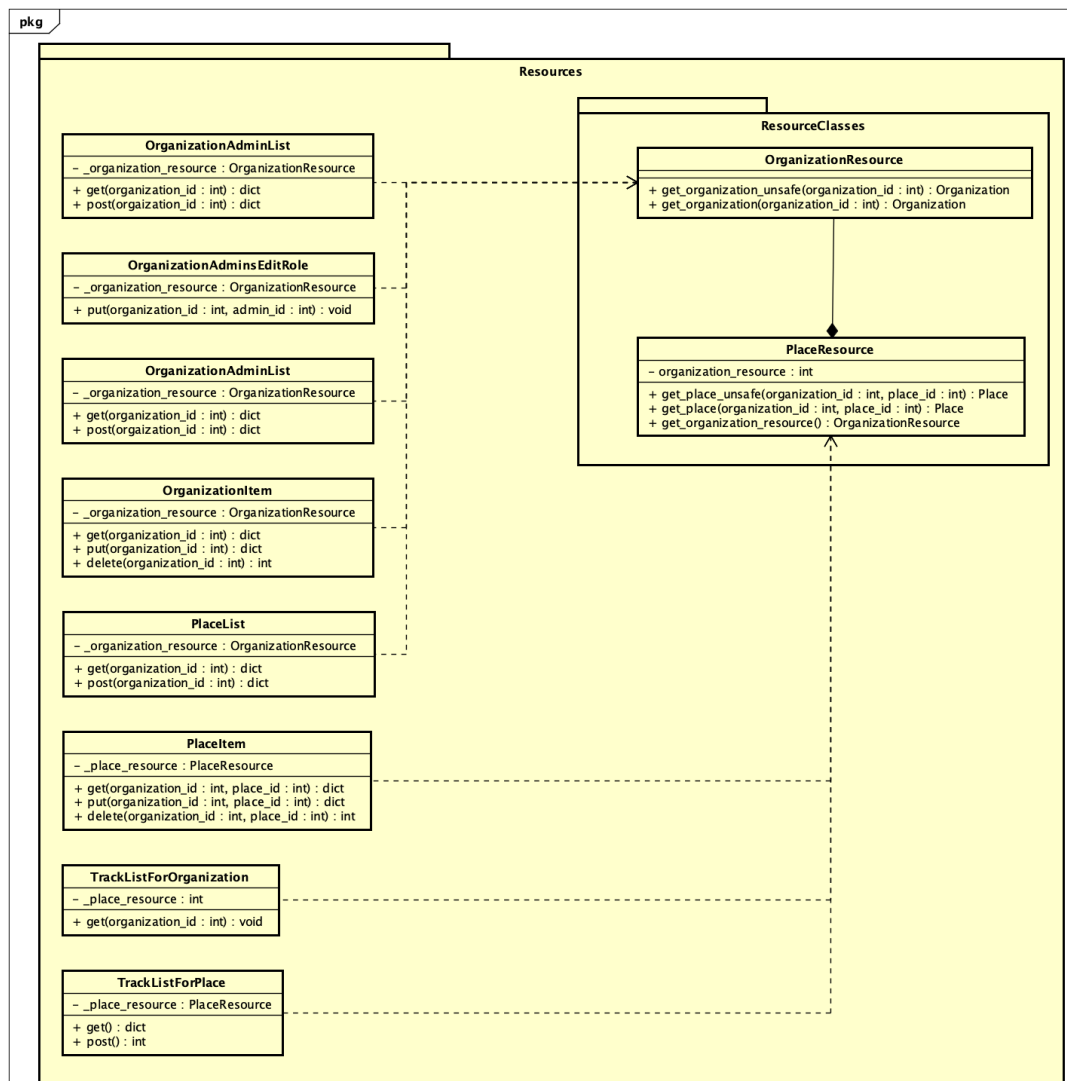


Figura 28: Diagramma delle classi delle risorse in relazione a Resource.

Le classi OrganizationResource e PlaceResource mettono a disposizione dei metodi utilizzati per ottenere organizzazioni e luoghi. I metodi messi a disposizione sono:

- OrganizationResource:
 - get_organization_unsafe(organization_id): che permette di ottenere i dettagli di un'organizzazione senza poter operare modifiche su di essa;
 - get_organization(organization_id): che, dopo aver verificato i permessi dell'amministratore, restituisce un'organizzazione sulla quale si possono effettuare modifiche che verranno salvate nel database.
- PlaceResource:

- `get_place_unsafe(organization_id, place_id)`: che permette di ottenere di dettagli di un luogo senza poter operare modifiche su di esso;
- `get_place(organization_id, place_id)`: che, dopo aver verificato i permessi dell'amministratore, restituisce un luogo sul quale si possono effettuare modifiche che verranno salvate nel database;
- `get_organization_resource()`: restituisce l'istanza di `organization_resource`.

3.3.2.1 Parser Per semplificare la lettura dei JSON ricevuti sono stati usati dei parser per i vari tipi di JSON ricevibili.

Sono tutti oggetti di tipo `RequestParser` della libreria `Flask-RESTful` ai quali sono stati aggiunti gli argomenti di cui devono fare il parsing.

Sono usati dalle risorse che ricevono un JSON.

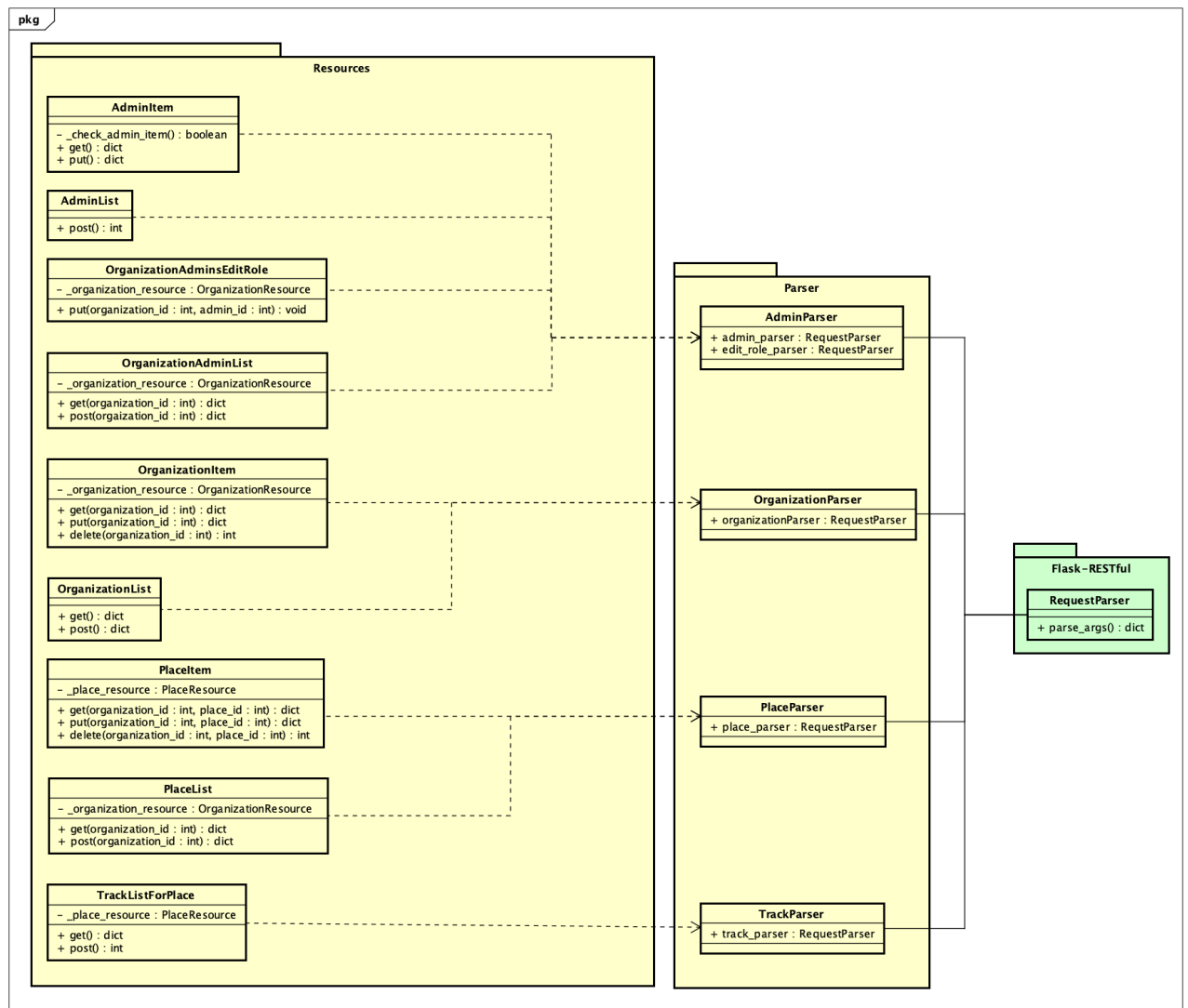


Figura 29: Diagramma delle classi delle risorse in relazione ai parser.

Quando viene invocato il metodo `parse_args()` viene restituito un dizionario con i valori presenti nel JSON. In caso il JSON sia mal formato sarà la classe `RequestParser` a restituire l'errore 400 Bad Request comunicando al client quale parametro è assente o errato.

3.3.2.2 Autenticazione e autorizzazione Per poter accedere alle risorse è necessario essere autorizzati. Esistono 5 tipi di autorizzazioni:

1. Organizzazioni: ogni organizzazione ha un token univoco utilizzabile per poter accedere alla lista delle organizzazioni e dei luoghi. Questo è il tipo di autorizzazione usato dall'applicazione Android;

2. Watcher-Admin: autorizzazione che permette l'accesso alle risorse degli amministratori visualizzatori;
3. Manager-Admin: autorizzazione che permette l'accesso alle risorse degli amministratori visualizzatori e gestori;
4. Owner-Admin: autorizzazione che permette l'accesso alle risorse degli amministratori visualizzatori, gestori e owner;
5. System-Admin: autorizzazione degli amministratori di sistema.

Per quanto riguarda il sistema di autenticazione degli amministratori vengono usati i JSON Web Token (JWT). Un amministratore quando si autentica presso il sistema riceve un JWT che deve essere incluso nell' *header delle richieste* nel formato `Authorization: Bearer JWT`.

Per controllare l'autenticità e la generazione dei token viene usata la libreria Flask-JWT la quale mette a disposizione il decorator `@jwt_required` da applicare alle funzioni che si vogliono proteggere dietro il meccanismo di autenticazione. Inoltre sono stati scritti altri 4 decorator (`@watcher_admin_required`, `@manager_admin_required`, `@owner_admin_required`, `@system_admin_required`) che si occupano di estendere le funzionalità della libreria e di gestire l'accesso alle risorse da parte degli amministratori.

Per esempio per creare una nuova organizzazione è necessario essere amministratori owner; questo concetto viene espresso nel modo seguente:

```
@jwt_required
@owner_admin_required
def post(self):
    ...logica per la creazione di una nuova organizzazione...
```

In questo modo è possibile garantire la precondizione che se il flusso di controllo entra dentro il metodo `post(self)` allora nell'header della richiesta vi era un JWT valido e che quel JWT appartiene ad un amministratore owner.

Tali decorator sono definiti nel file `AuthUtils.py`.

3.3.2.3 Esempio creazione organizzazione Di seguito è presente il diagramma di sequenza e la sua descrizione dettagliata per quando riguarda la creazione di un'organizzazione. Questo è un esempio completo che comprende autorizzazione di un amministratore owner e utilizzo di un parser; le altre richieste vengono gestite in maniera simile.

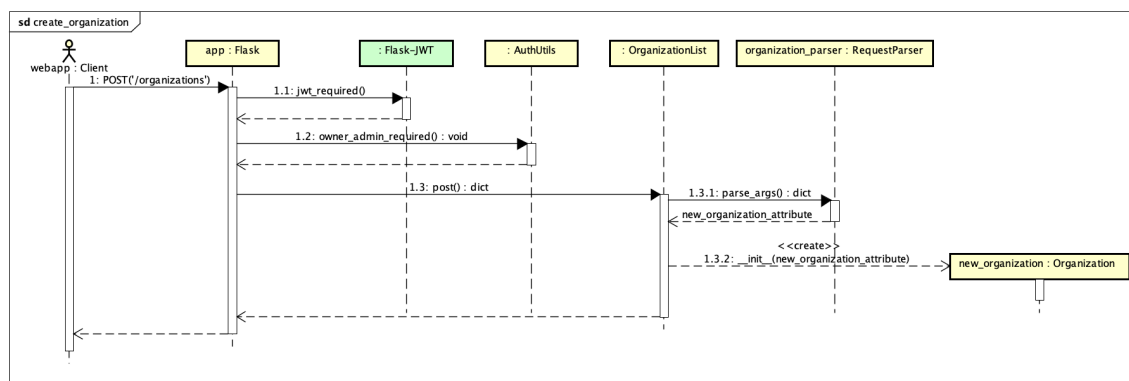


Figura 30: Diagramma di sequenza per la creazione di un'organizzazione.

La webapp fa una richiesta HTTP di tipo POST all'url /organizations che è stato registrato alla risorsa OrganizationUtils, la quale ha implementato il metodo post. Tale metodo ha due decorator: @jwt_required che si assicura che nell'header della richiesta ci sia il JWT e che questo sia valido, e @owner_admin_required che si assicura che il JWT appartenga ad un amministratore owner.

Il metodo post() utilizza il parser organization_parser per controllare che il JSON ricevuto sia nel ben formato e, in caso tutto sia andato a buon fine, crea un nuovo oggetto di tipo Organization che viene aggiunto al database.

Infine viene restituita una risposta al client contenente lo status code 200 e un JSON rappresentante l'organizzazione appena creata.

3.4 Estendibilità backend

3.4.1 Aggiungere un modello

Per aggiungere un nuovo modello (e quindi una nuova tabella) al backend è sufficiente creare una nuova classe che estende dalla classe base `Model` resa disponibile dalla libreria `Flask-SQLAlchemy`. Se si è creato un nuovo file per la classe, questo va importato nel metodo `_initialize_database()` presente nel file `__init__.py` presente nella directory `stalker_backend`.

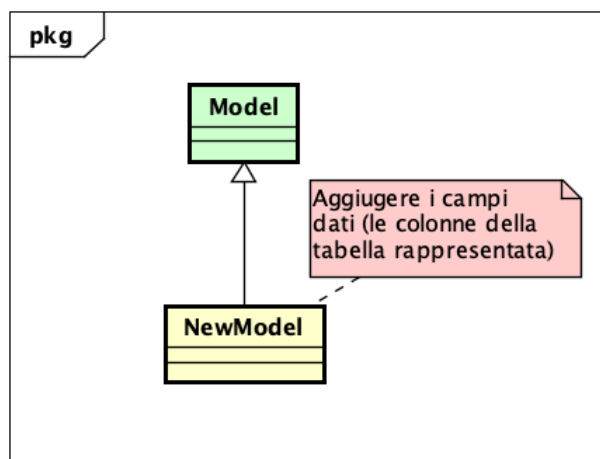


Figura 31: Estensione modelli

3.4.2 Aggiungere una nuova API

Per aggiungere la gestione di una nuova API al backend è sufficiente creare una nuova classe che estende dalla classe base `Resource` resa disponibile dalla libreria `Flask-RESTful` e implementare i metodi che devono rispondere alle richieste HTTP. I nomi dei metodi della classe devono corrispondere ai nomi dei metodi delle chiamate HTTP che si vogliono supportare per la nuova API (`get()`, `post()`, `put()`, `delete()`).

In caso all'interno dell'URL al quale la nuova API dovrà rispondere vi sia una parte dinamica, un identificativo univoco ad esempio, si deve aggiungere ai parametri del metodo che dovrà rispondere a quella API il nome del placeholder associato alla parte dinamica dell'URL definito al momento della registrazione della nuova API.

Per poter rendere accessibile la nuova API è sufficiente invocare il metodo `add_resource()` sull'oggetto API presente nel metodo `_register_api()` del file `__init__.py` nella directory `stalker_backend`, passandogli come argomento la classe creata, l'url e gli argomenti da passare al costruttore in caso ce ne siano.

In caso l'URL contenga delle parti dinamiche è possibile specificarle usando il formato `/static_section/<dynamic_section>`. I metodi che risponderanno a questo URL dovranno avere il parametro `dynamic_section`.

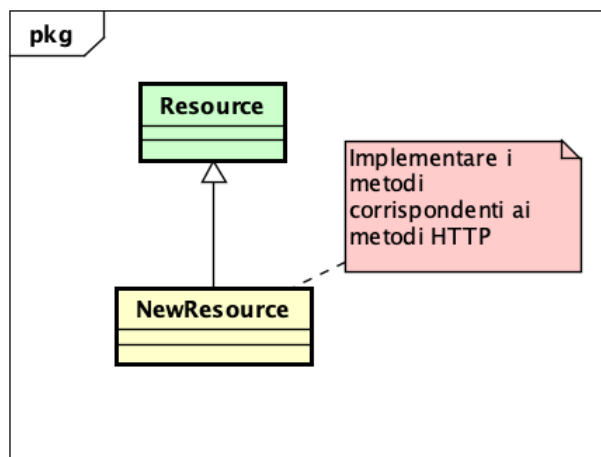


Figura 32: Estensione API

3.5 Test backend

3.5.1 Aggiungere nuovi test

Per creare nuovi test è necessario creare un nuovo file (nel formato `test_*.py`) nella cartella `tests` oppure aggiungere una funzione ad uno dei file già esistenti.

3.5.2 Eseguire i test

Per poter eseguire i test è necessario aver installato nell'ambiente di esecuzione la libreria `pytest` e alcuni plugin. Ciò può essere fatto tramite il comando

```
pip install pytest pytest-cov pytest-flask-sqlalchemy
```

Per eseguire la suite di test disponibile è sufficiente eseguire il comando

```
make run-tests
```

In caso si voglia ottenere anche la code-coverage in formato XML è possibile eseguire il comando

```
make run-tests-with-coverage
```

Se non si ha a disposizione il comando `make` è comunque possibile eseguire la suite di test con il comando

```
python3 -m pytest
```

4 Webapp

4.1 Tecnologie webapp

4.1.1 Angular 9

Framework open source scelto per lo sviluppo della webapp. La versione utilizzata è Angular 9.0.7.

<https://angular.io/>

4.1.2 Material Design

Design creato da Google, utilizzato per sviluppare l'interfaccia grafica. La versione utilizzata è Material Design 9.2.0.

<https://material.io/design/>

4.1.3 Node JS

Runtime di JavaScript open source multiplatforma orientato agli eventi per l'esecuzione di codice JavaScript, utilizzato per avviare l'applicazione in locale.

<https://nodejs.org/it/>

4.1.4 Leaflet

Leaflet è una libreria JavaScript open source per sviluppare mappe geografiche interattive. La versione utilizzata è Leaflet 1.6.0.

<https://leafletjs.com/>

4.1.5 Leaflet-Geoman

Geoman è un plugin open source che permette di modificare le mappe interattive sviluppate con Leaflet. <https://geoman.io/leaflet-geoman>

4.1.6 XML

Linguaggio di mark-up utilizzato per definire il layout e i componenti della UI.

4.1.7 JSON

Formato di file utilizzato per lo scambio di dati con il backend.

4.2 Setup e configurazione dell'ambiente di sviluppo per la webapp

4.2.1 Requisiti di sistema

I requisiti di sistema necessari per l'ambiente di sviluppo della webapp sono:

- Node.js, versione utilizzata v12.16.1;
- npm package manager, versione utilizzata v6.13.4.

Per quanto riguarda Node.js, lo si può scaricare al link <https://nodejs.org/> e il pacchetto npm è compreso di default.

4.2.2 Creazione e configurazione ambiente di sviluppo

Per configurare l'ambiente di sviluppo è necessario installare Angular CLI. Una breve guida è disponibile al link <https://angular.io/guide/setup-local>. Dopo aver aperto il terminale utilizzare il comando seguente:

```
npm install -g @angular/cli
```

A questo punto bisogna aprire il workspace del progetto (esempio "my-app") utilizzando i seguenti comandi:

```
cd my-app  
ng serve --open
```

Con il primo comando ci si sposta nella cartella del progetto, con il secondo invece si avvia il server in locale all'indirizzo <http://localhost:4200/>.

4.3 Architettura webapp

Per lo sviluppo della webapp è stato adottato il pattern architetturale Model-View-ViewModel (MVVM), in modo da consentire la separazione tra la logica dell'applicazione e la presentazione dei dati.

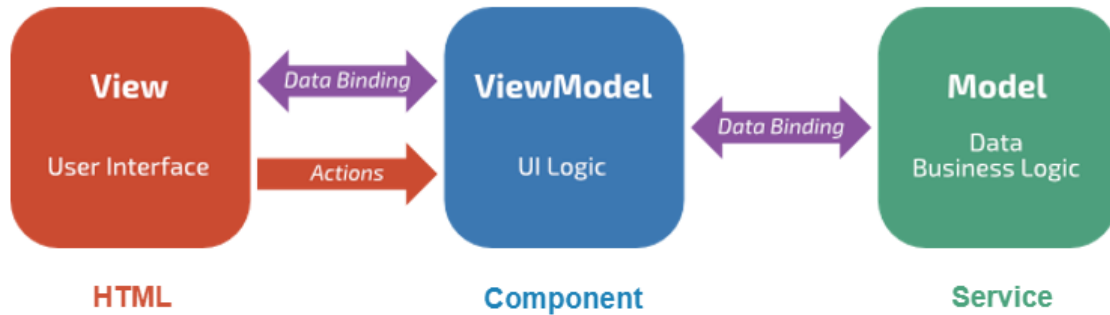


Figura 33: Architettura della webapp.

4.3.1 Model

Le classi che fanno parte del Model sono:

- Place, che modella i luoghi;
- Organization, che modella le organizzazioni;
- Administrator, che modella gli amministratori;
- Track, che modella i tracciamenti.

Essenziali sono i service, che contengono tutti i metodi utili per interfacciare il modello con i dati del backend. Sono stati implementati i seguenti service:

- PlacesService, che contiene i metodi di Place;
- OrganizationService, che contiene i metodi di Organization;
- AdminService, che contiene i metodi di Administrator;
- TrackService, che contiene i metodi di Track;
- ImageService, che permette l'upload del logo dell'organizzazione;
- AutheticationService, che contiene i metodi per gestire l'autenticazione.

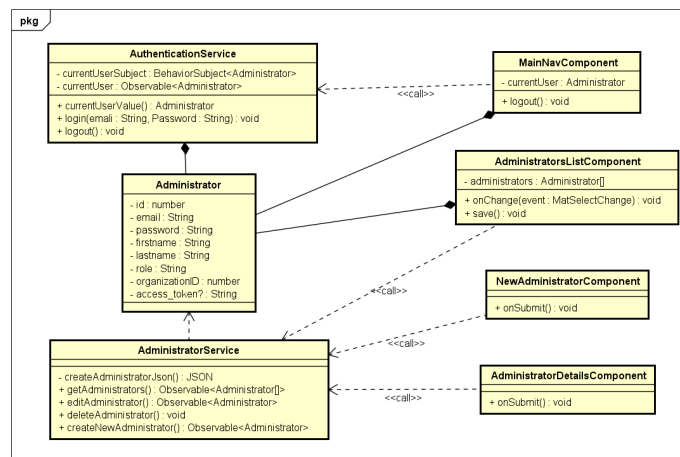


Figura 34: Diagramma delle classi dell'amministratore.

4.3.1.1 Administrator Per quanto riguarda la gestione dell'amministratore abbiamo una classe base Administrator che contiene gli attributi dell'amministratore. MainNavComponent è il componente che sviluppa il menù di navigazione della webapp. AuthenticationService è invece il service che gestisce l'autenticazione degli amministratori, attraverso i metodi login() e logout(). AdministratorService contiene i metodi per l'inserimento, la modifica e l'eliminazione degli amministratori e si interfaccia con il backend tramite l'invio di file JSON. AdministratorListComponent è un component che permette la visualizzazione della lista di amministratori presenti all'interno dell'organizzazione precedentemente selezionata. NewAdministratorComponent è il component che gestisce il form di inserimento dei dati di un nuovo amministratore: una sua istanza viene generata quando all'interno della lista amministratori viene cliccato il pulsante "New administrator". AdministratorDetailsComponent è il component che gestisce la visualizzazione dei dati relativi ad un amministratore selezionato e ne permette la modifica.

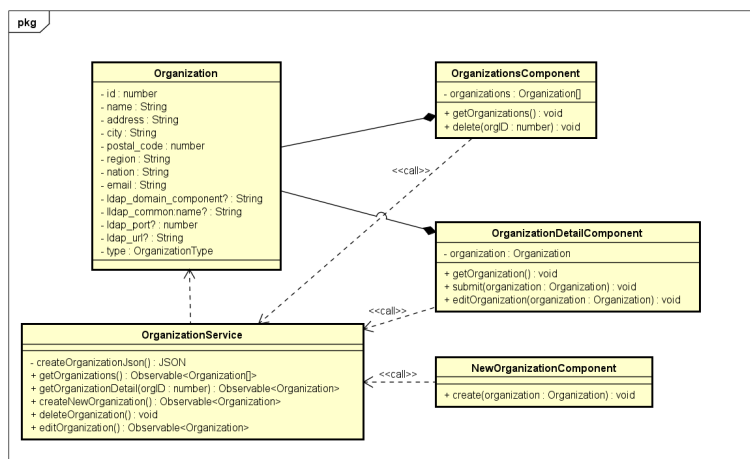


Figura 35: Diagramma delle classi delle organizzazioni.

4.3.1.2 Organization La classe base Organization modella un'organizzazione e contiene tutti i suoi attributi. OrganizationService è il service che contiene i metodi per l'inserimento, la modifica e l'eliminazione delle organizzazioni e si interfaccia con il backend tramite l'invio di file JSON. OrganizationsComponent si occupa di gestire la visualizzazione delle organizzazioni registrate. NewOrganizationComponent è il component che gestisce il form di inserimento dei dati di un nuovo amministratore: una sua istanza viene generata quando all'interno della lista di organizzazioni viene cliccato il pulsante "New organization". OrganizationDetailsComponent è il component che gestisce la visualizzazione dei dati relativi ad una organizzazione selezionata, permettendone la modifica.

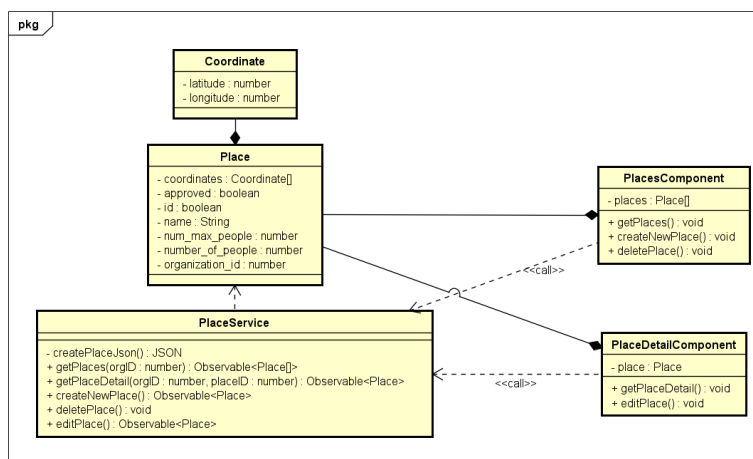


Figura 36: Diagramma delle classi dei luoghi.

4.3.1.3 Place Per ciò che riguarda la gestione dei luoghi, abbiamo una classe base Place che contiene un'array di coordinate che lo identificano attraverso latitudine e longitudine. PlaceService è il service che contiene tutti i metodi relativi alla gestione dei luoghi, come la creazione, la modifica e l'eliminazione, interfacciandosi con il backend tramite l'invio di file JSON. PlacesComponent è un component che gestisce la visualizzazione dei luoghi di un'organizzazione precedentemente selezionata, permettendo le azioni di creazione ed eliminazione, chiamando i metodi di PlaceService. PlaceDetailComponent è il component che gestisce la visualizzazione dei dati relativi ad un amministratore selezionato e ne permette la modifica.

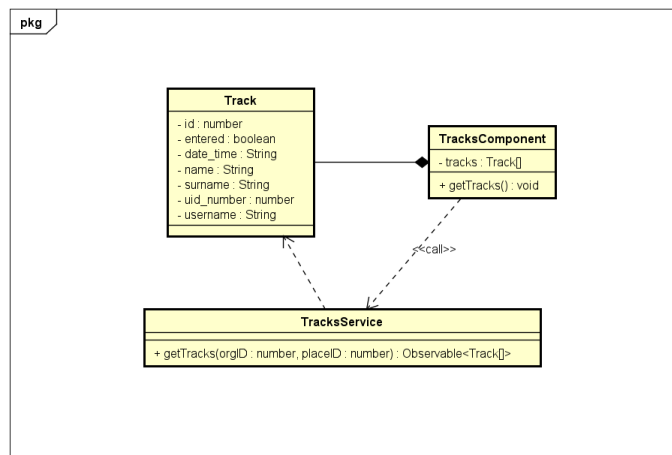


Figura 37: Diagramma delle classi dei tracciamenti.

4.3.1.4 Track La classe base *Track* modella un tracciamento con i relativi attributi. *TracksComponent* è il componente che permette la visualizzazione dei tracciamenti. I tracciamenti sono disponibili avendo prima selezionato una organizzazione e un luogo appartenente alla stessa. *TrackService* è il service che si interfaccia con il backend tramite la ricezione di un file JSON dei tracciati.

4.3.2 View e ViewModel

I component di Angular permettono la gestione della View e ViewModel. Un component combina un template HTML e una classe TypeScript. Un component, nella sua classe TypeScript, contiene i metodi per gestire la classe a cui è associato e controlla una relativa View (che corrisponde al template HTML). Vediamo un diagramma di sequenza che mostra come viene gestita l'autenticazione di un amministratore.

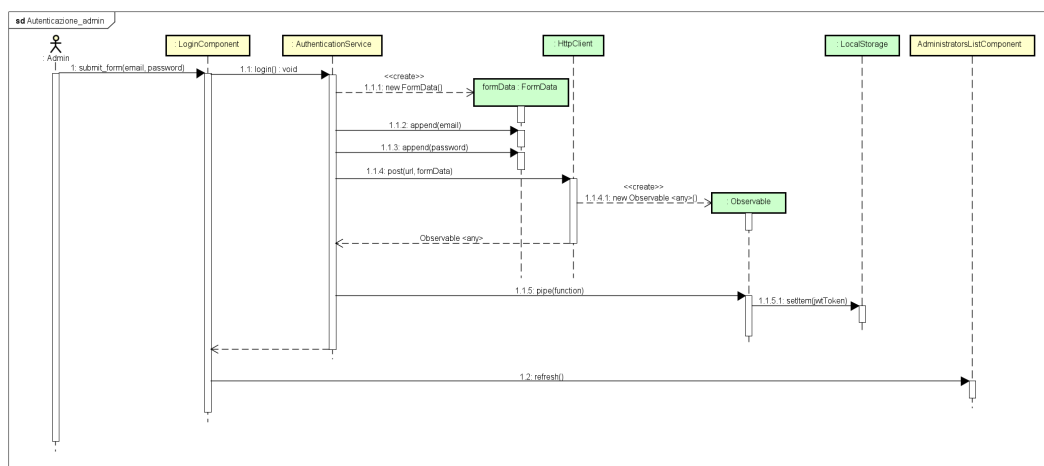


Figura 38: Diagramma di sequenza dell'autenticazione.

L'amministratore visualizza la pagina di login e ne riempie il form inserendo email e password. LoginComponent chiama la funzione login() di AuthenticationService, che crea un FormData. Al FormData viene fatto l'append di email e password. Il FormData viene inviato tramite il metodo post() ad un'istanza di HttpClient, classe interna di Angular che gestisce le richieste http, che crea un oggetto Observable <any>. Questo Observable viene passato dall'AuthenticationService tramite il metodo pipe(), successivamente viene chiamato il metodo setItem() della classe Observable, passando come parametro un token (jwtToken) e memorizzandolo in LocalStorage, permettendo quindi di mantenere attiva la sessione dell'amministratore.

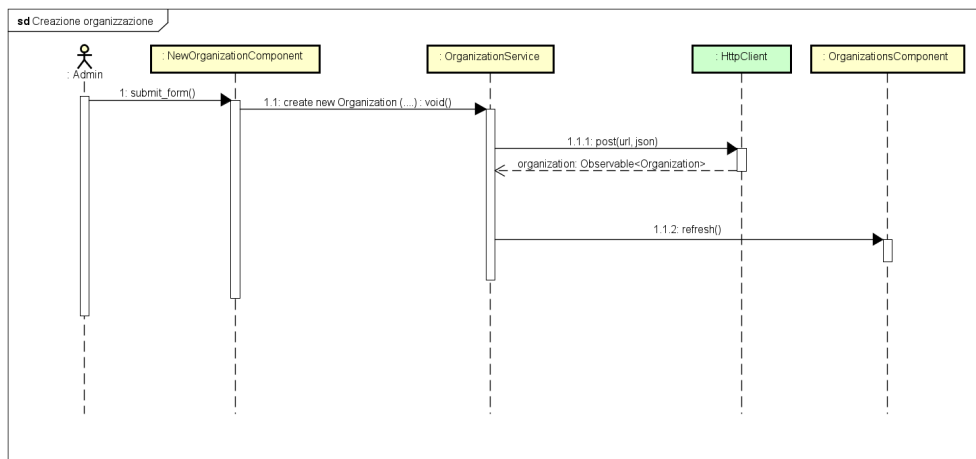


Figura 39: Diagramma di sequenza della creazione di un'organizzazione.

L'amministratore, dalla lista di organizzazioni, clicca il pulsante "Crea organizzazione", viene quindi creata un'istanza di NewOrganizationComponent che gestisce il form di inserimento. Una volta effettuato confermata la creazione tramite il metodo submit(), OrganizationService() manda tramite il metodo post() il JSON con i dati della nuova organizzazione all'HttpClient che gestisce la richiesta http e si interfaccia con il backend. A questo punto l'HttpClient ritorna un Observable<Organization> all'OrganizationService, che tramite il metodo refresh() ricarica la lista delle organizzazioni che ora contiene anche l'ultima aggiunta.

4.3.3 Altri component

Altri component creati e utilizzati per sviluppare la webapp sono:

- LoginComponent, utilizzato per creare l'interfaccia di login e la gestione dell'autenticazione;
- MapComponent, utilizzato per aggiungere una mappa interattiva sfruttando la libreria JS Leaflet;
- ReportComponent, utilizzato per gestire i report dei tracciamenti;
- Password-ResetComponent, utilizzato per creare l'interfaccia di reset password;
- Approve-PlacesComponent, utilizzato per gestire l'interfaccia di approvazione luoghi da parte degli amministratori.

4.4 Estendibilità webapp

4.4.1 Aggiungere un component

Per aggiungere un component è sufficiente utilizzare nel terminale il comando:

```
ng generate component nome
```

In questo modo verrà generata all'interno del workspace una cartella denominata "nomeComponent" e al suo interno saranno presenti:

- **nomeComponent.component.css;**
- **nomeComponent.component.html;**
- **nomeComponent.component.spec.ts;**
- **nomeComponent.component.ts.**

Per legare il componente a una classe (ad esempio Organization) del nostro modello sarà necessario aggiungere nel file *nomeComponent.component.ts* il seguente codice:

```
import {Organization, OrganizationType} from '../models/Organization';
```

4.4.2 Aggiungere un service

Per aggiungere un service è sufficiente utilizzare nel terminale il comando:

```
ng generate service nome
```

In questo modo verranno generati all'interno del workspace due file denominati:

- **nomeService.service.spec.ts;**
- **nomeService.service.ts;**

Sarà possibile successivamente, modificando il file *nomeService.service.ts*, aggiungere i metodi necessari al servizio.

4.5 Test webapp

Per creare nuovi test è necessario utilizzare il framework di testing Jasmine. Questo framework è compreso di default nell'Angular CLI. La guida ufficiale è disponibile al link <https://angular.io/guide/testing>.

4.5.1 Aggiungere nuovi test

Per l'inserimento di nuovi test relativi a un component o a un service è sufficiente inserire le funzioni di test all'interno dei file con estensione "spec.ts". I file con questa estensione vengono creati di default alla creazione del component (all'interno della directory dello stesso), o nella directory radice per quanto riguarda un service. Segue un esempio di test per il component `AdministratorDetailsComponent`:

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { AdministratorDetailsComponent } from './administrator-details.component';

describe('AdministratorDetailsComponent', () => {
  let component: AdministratorDetailsComponent;
  let fixture: ComponentFixture<AdministratorDetailsComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ AdministratorDetailsComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(AdministratorDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Figura 40: Esempio di test del component `AdministratorDetailsComponent`

4.5.2 Eseguire i test

Aprire la console all'interno del workspace del progetto e utilizzare il comando:

ng test

In questo modo verrà compilata l'applicazione e lanciato il Karma test runner. Segue un esempio di log di output:

```
...INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876/
...INFO [launcher]: Launching browser Chrome ...
...INFO [launcher]: Starting browser Chrome
...INFO [Chrome ...]: Connected on socket ...
Chrome ...: Executed 3 of 3 SUCCESS (0.135 secs / 0.205 secs)
```

Successivamente nel browser apparirà una finestra che mostra i test superati e il log:



Figura 41: Jasmine HTML Reporter

A Glossario

A.1 A

A.1.1 AVD

L'acronimo di Android Virtual Device. È un software che permette di creare delle macchine virtuali con il sistema operativo Android, viene utilizzato per simulare il dispositivo reale, quindi di fare il debug su questi dispositivi.

A.1.2 Header della richiesta

Ogni richiesta HTTP è composta da uno o più header contenenti informazioni utili a gestire la richiesta da parte del backend.

A.1.3 JSON

Formato usato per scambiare informazioni tra le componenti del sistema.

A.1.4 LDAP

Lightweight Directory Access Protocol (LDAP) è un protocollo standard usato per accedere alle informazioni conservate centralmente in un sistema di directory, cioè un sistema che organizza le informazioni in un insieme di record gerarchici. Esso viene usato per permettere l'autenticazione di un dipendente presso un'organizzazione.

A.1.5 NoSQL

Database caratterizzato dal fatto di non utilizzare il modello relazionale; l'espressione "NoSQL" indica che non viene fatto uso del linguaggio SQL per interagire con il database come avviene invece per i database relazionali.

A.2 R

A.2.1 REST

Acronimo per REpresentational State Transfer, stile architetturale ibrido che fa uso di richieste HTTP per far comunicare un client con il server.

A.2.2 Real-time

Un database real-time esegue l'invio di aggiornamenti ai client in modo da poter mostrare sempre gli ultimi dati a differenza dei database non real-time dove deve essere il client a fare il polling dei dati.

A.3 S

A.3.1 SDK

Un software development kit (SDK, traducibile in italiano come "pacchetto di sviluppo per applicazioni") indica genericamente un insieme di strumenti per lo sviluppo e la documentazione di software.