Realizzato dall'Università degli Studi di Cagliari

simple

StrumentI e Modelli Per La mobilità sostenibilE

PoolBus - Repository dell'applicazione server

Progetto finanziato con fondi *POR FESR 2014/2020 - ASSE PRIORITARIO I "RICERCA SCIENTIFICA, SVILUPPO TECNOLOGICO E INNOVAZIONE".*

Informazioni sul progetto

Numero del progetto	N/A	**Acronimo**	SIMPLE
Titolo completo	StrumentI e Modelli Per La mobilità sostenibilE		
Soggetto	Progetto CLUSTER ICT		
Data inizio	01/02/2018		
Durata in mesi	30		
Coordinatore	UniCA – Università degli Studi di Cagliari		
URL del progetto	http://www.simple-cluster.it		

Informazioni sul documento

Numero del Deliverable	2.3.2.2	**Titolo**	PoolBus - Repository dell'applicazione server
Numero del Workpackage	WP3	**Titolo**	Sperimentazione
Data di inizio redazione del documento	28/09/2020		
Autore/i responsabile/i	Lucia Pintor		
Livello di diffusione	Non applicabile		

Modifiche del documento

Data	Autore	Modifiche	Versione
28/09/2020	Lucia Pintor	Organizzazione della struttura e prima stesura	v0.0
30/09/2020	Lucia Pintor	Descrizione dei moduli Corridor Services e Hailing Services	v0.1
02/10/2020	Lucia Pintor	Descrizione dei moduli mancanti	v0.2
06/10/2020	Lucia Pintor	Tabelle del db	v0.3

Tavola dei contenuti

1 Introduzione

In questo documento viene descritto il funzionamento dell'applicazione server che fa parte del prototipo PoolBus: l'applicazione è stata sviluppata dal team SIMPLE ed è implementata nel linguaggio Python.

L'applicazione ha lo scopo di generare piani di viaggio personalizzati e gestire le prenotazioni dei servizi di navetta.

Il documento prosegue con la descrizione dei moduli che compongono l'applicazione, la descrizione delle principali API e un capitolo in cui sono spiegati alcuni concetti dello sviluppo Python con i framework Django.

1.1 Django Framework

Django[1] è un framework Web Python di alto livello che agevola notevolmente lo sviluppo Web, fornendo diversi pacchetti e moduli adattabili a qualunque applicazione. Utilizza la filosofia DRY (Don't Repeat Yourself): in questo modo si realizzano funzioni e comportamenti generici che consentono di evitare ripetizioni, rendendo il codice più snello. Il codice può essere sviluppato velocemente e in maniera modulare, in modo che ciascuna funzione possa essere utilizzata e aggiornata indipendentemente dalle altre. Inoltre dispone del toolkit Django Rest Framework, che semplifica la creazione di API (Application Programming Interface) e servizi web complessi e altamente personalizzabili.

1.2 Open Trip Planner

OpenTripPlanner[2] (OTP) è una famiglia di progetti software open source che forniscono servizi di analisi delle reti di trasporto e dei passeggeri. Il componente Java core sul lato server trova gli itinerari che combinano i segmenti di transito, pedone, bicicletta e auto attraverso reti basate su dati OpenStreetMap e GTFS standard aperti. È possibile accedere a questo servizio direttamente tramite la sua API Web o utilizzando una vasta gamma di librerie client.

[1] https://www.djangoproject.com/
[2] https://www.opentripplanner.org/

2 Moduli

In questa sezione sono descritti i moduli che compongono l'applicazione: Core App, Corridor Services, Hailing Services, Pooling Services, GTFS manager, Registration mobile API, Password reset, Statistics, Common e due moduli incompleti (Agency site, Driver Tracker).

In questa sezione sono descritti inoltre i modelli, ovvero delle strutture dati (classi) che mappano le tabelle e i rispettivi record del database.

2.1 Common

Questo modulo contiene porzioni di codice che vengono riutilizzate in vari contesti. In particolare sono presenti mixin, funzioni, serializzatori e valori costanti.

I mixin sono classi astratte, da cui derivano per specializzazione dei modelli o altre classi concrete. I principali mixin di questo modulo sono Contactable, Geomixinable, WorkingDaysCheckable, Operator e APIable. Questi mixin aggiungono dei campi ai modelli per gli operatori.

- Il mixin Contactable aggiunge i campi name, address, email, phone, url_website e type.
- Il mixin Geomixinable aggiunge il campo geografico area.
- Il mixin WorkingDaysCheckable aggiunge il campo json working_days, che raccoglie gli orari lavorativi di un servizio, distinti per giorno della settimana.
- Il mixin Operator unisce i mixin Contactable e Geomixinable e fa derivare i modelli HailingServices, PoolingServices e CorridorServices.
- Il mixin APIableaggiunge i campi API_url, API_type e API_key.

Le funzioni presenti in questo modulo riguardano principalmente la conversione di dati da un tipo ad un altro, le query al server con l'istanza di Open Trip Planner e dei filtri di ricerca per i modelli dei moduli HailingServices, PoolingServices e CorridorServices.

I serializzatori di questo modulo sono componenti utilizzati da più moduli per analizzare e modificare il piano di viaggio.

I valori costanti di questo modulo riguardano l'affidabilità del servizio (MAX_RELIABILITY, MID_RELIABILITY, NO_RELIABILITY) e la velocità dei veicoli (car_speed).

2.2 Core App

Il modulo Core App interagisce con l'applicazione mobile e con un'istanza di Open Trip Planner.

Un tipico Use Case è:

- Un utente richiede un percorso tramite l'applicazione mobile.
- L'applicazione mobile inoltra ad una specifica API del server la richiesta.
- Il modulo Core App adatta la richiesta e la inoltra all'istanza Open Trip Planner.

- Il modulo Core App analizza il piano di viaggio ricevuto e verifica se è possibile aggiungere nuove opzioni con servizi di hailing, pooling e navette.
- Il modulo Core App invia il nuovo piano di viaggio all'applicazione mobile.
- L'utente visualizza il piano di viaggio sul proprio smartphone e seleziona un itinerario.
- Lo smartphone utilizza un'altra API per comunicare la selezione al server.
- Il modulo Core App finalizza la richiesta, prenotando le opzioni di viaggio che necessitano la prenotazione.

Questo modulo ha un solo modello, Plan, che serve per memorizzare i piani di viaggio e altre informazioni come l'utente e l'itinerario selezionato. Esistono inoltre altri oggetti che servono per analizzare le soluzioni di viaggio delle navette (VehicleSolution, VehiclePlanSegment, PassengerListElement e ReferenceRow) e i rispettivi serializer (PlanSerializer, VehicleSolutionSerializer, VehiclePlanSegmentSerializer, PassengerListElementSerializer).

Questo modulo ha diversi file views, che sono riassunti nella seguente tabella.

Nome del file	Descrizione
views.py	Contiene due API: OperatorList e OTPManager. - OperatorList mostra la lista degli operatori di HailingServices, PoolingServices e CorridorServices. - OTPManager interagisce con l'istanza OTP in modo da costruire dei piani di viaggio.
views_additional_functions.py	Questo file contiene diverse funzioni che interfacciano i diversi tipi di servizio in modo da interrogarli allo stesso modo e gestire le loro risposte in modo uniforme.
views_clean_db.py	Contiene due API: CleanDB e CleanEmptyTrips. - CleanDB cancella tutti i record del database di tipo Plan, CorridorReservation e CorridorTrip. Si tratta di una funzione utile ai fini sperimentali, che verrà rimossa per le sperimentazioni fisiche. - CleanEmptyTrips cancella tutti i CorridorTrip obsoleti.
views_close_trips.py	Contiene l'API CloseCorridorTrips, che blocca le modifiche e avvia la procedura per confermare i viaggi delle navette.
views_get_travels.py	Contiene l'API GetUserTravels: questa API richiede l'autenticazione e restituisce la lista di itinerari selezionati.
views_select_itinerary.py	Contiene l'API SelectItinerary: questa API di tipo POST richiede l'autenticazione e permette di modificare un piano di viaggio del db, selezionando un itinerario.

views_select_itinerary_create_new_trip.py	Questo file contiene le funzioni che vengono chiamate dall'API SelectItinerary nel caso sia necessario creare un nuovo viaggio delle navette (primo utente che prenota quel viaggio).
views_select_itinerary_find_best_match.py	Questo file contiene le funzioni che vengono chiamate dall'API SelectItinerary nel caso sia necessario aggiungere una prenotazione ad un viaggio delle navette (utenti successivi al primo che prenotano quel viaggio).
views_select_itinerary_utils.py	Questo file contiene delle funzioni che si interfacciano con il pacchetto OR-Tools per individuare il percorso ottimale di un viaggio di una navetta.
views_update_fares.py	Contiene l'API FareUpdater, che scarica il file della matrice tariffaria per il trasporto pubblico dal sito sardegnamobilità.
views_utilities.py	Contiene delle funzioni per analizzare il piano di viaggio.

Ove non specificato, le API sono di tipo GET e non richiedono l'autenticazione.

2.3 Corridor Services

Il modulo Corridor Services gestisce i servizi navetta (o servizi di corridoio).

Un tipico Use Case è la ricerca di servizi attivi:

- Il modulo Core App (nello specifico l'API OTPManager) richiede quali servizi di corridoio operano tra il punto A e il punto B durante una fascia oraria X del giorno Y.
- Il modulo Corridor Services analizza i dati nel database e prepara una lista con le varie opzioni di viaggio.
- Le opzioni di viaggio sono poi inviate alla Core App per costruire opzioni di itinerario all'interno di un piano di viaggio.

I modelli di questo modulo sono CorridorService, CorridorVehicle, CorridorTrip, CorridorStop, MatrixElement, CorridorReservation.

- CorridorService raccoglie le informazioni riguardo un'azienda che fornisce servizi di navetta. Questa struttura dati eredita dal mixin Operator i campi i campi name, address, email, phone, url_website e type. In aggiunta possiede i campi kilometre_fare_euro e regional_contribute_percentage.
- CorridorVehicle raccoglie le informazioni riguardo un veicolo (navetta) e ha i campi vehicle_model, total_seats, empty_seats, corridor_service e is_available.

- CorridorTrip raccoglie le informazioni di un viaggio navetta e ha i campi pickups_deliveries, time_window, vehicle_plan, is_confirmed, corridor_service, starting_service_hr, ending_service_hr, max_service_duration, does_accept_changes, max_passengers_simultaneously, total_fare_euro, total_distance_meters, vehicle, driver, creation_timestamp e schedule.

- CorridorStop raccoglie le informazioni di una fermata e ha i campi name, point, location_type, corridor_service e is_active.

- MatrixElement è un elemento della matrice origine destinazione e si riferisce ad una coppia di fermate. Questa struttura dati ha i campi distance_meters, duration_seconds, shape_linestring, shape_encoded, origin e destination.

- CorridorReservation raccoglie le informazioni di una prenotazione e ha i campi user, plan, selected_itinerary, selected_leg, previous_leg_arrival, next_leg_departure, fare_euro, max_fare_euro, time_window, arrive_by, passengers, corridor_service, min_trip_duration_seconds, trip_duration_seconds, min_distance_meters, distance_meters, start_stop, start_stop_confirmed_hr, end_stop, end_stop_confirmed_hr, creation_timestamp, did_pay, is_confirmed, corridor_trip.

Per ciascun modello esiste il corrispondente serializer.

Questo modulo ha diversi file views, che sono riassunti nella seguente tabella.

Nome del file	Descrizione
views.py	Contiene: - Alcune API che restituiscono la lista di oggetti che fanno parte di un certo modello. - L'API CorridorServicesMatcher, che verifica se è possibile collegare la richiesta di viaggio di un utente con un viaggio navetta pre-esistente o nuovo.
views_create_fake_route.py	Contiene il codice per generare un dataset di prova per questo modello.
views_utilities.py	Contiene alcune funzioni utilizzate dal modulo Core App per modificare i modelli di Corridor Services.

Tutte le le API sono di tipo GET e non richiedono l'autenticazione.

2.4 Hailing Services

Il modulo Hailing Services gestisce i servizi taxi (o servizi di hailing).

Un tipico Use Case è la ricerca di servizi attivi:

- Il modulo Core App (nello specifico l'API OTPManager) richiede quali servizi di taxi operano tra il punto A e il punto B durante una fascia oraria X del giorno Y.
- Il modulo Hailing Services analizza i dati nel database e prepara una lista con le varie opzioni di viaggio.
- Le opzioni di viaggio sono poi inviate alla Core App per costruire opzioni di itinerario all'interno di un piano di viaggio.

Questo modulo contiene solo il modello HailingService, che raccoglie le informazioni riguardo un'azienda che fornisce servizi di taxi. Questa struttura dati eredita dal mixin Operator i campi i campi name, address, email, phone, url_website e type. Esiste un serializer specifico per questo modello, HailingServiceSerializer.

Questo modulo ha un unico file view, che contiene:

- Alcune API che restituiscono la lista di oggetti che fanno parte di un certo modello.
- L'API HailingMatcher, che verifica se è possibile collegare la richiesta di viaggio di un utente con un'offerta di tipo hailing.

Tutte le le API sono di tipo GET e non richiedono l'autenticazione.

2.5 Pooling Services

Il modulo Pooling Services gestisce i servizi di carpooling, in particolare permette di vedere l'offerta del servizio Clacsoon[3].

Un tipico Use Case è la ricerca di servizi attivi:

- Il modulo Core App (nello specifico l'API OTPManager) richiede quali servizi di pooling operano tra il punto A e il punto B durante una fascia oraria X del giorno Y.
- Il modulo Pooling Services interroga tramite API i servizi associati (al momento solo Clacsoon) e formatta la loro risposta per preparare una lista con le varie opzioni di viaggio.
- Le opzioni di viaggio sono poi inviate alla Core App per costruire opzioni di itinerario all'interno di un piano di viaggio.

Questo modulo contiene solo il modello PoolingService, che raccoglie le informazioni riguardo un'azienda che fornisce servizi di pooling. Questa struttura dati eredita dal mixin Operator i campi i campi name,

[3] http://www.clacsoon.com/

address, email, phone, url_website e type. Inoltre la classe PoolingService eredita i campi API_url, API_type e API_key dal mixin APIable. Esiste un serializer specifico per questo modello, PoolingServiceSerializer.

Questo modulo ha un unico file view, che contiene:

- Alcune API che restituiscono la lista di oggetti che fanno parte di un certo modello.
- L'API PoolingMatcher, che verifica se è possibile collegare la richiesta di viaggio di un utente con un'offerta di tipo pooling.

Tutte le le API sono di tipo GET e non richiedono l'autenticazione.

2.6 GTFS manager

Questo modulo si occupa di fornire all'istanza OTP i link per scaricare i file GTFS.

Il suo Use Case è quindi legato all'istanza OTP:

- L'istanza OTP interroga più volte al giorno il server PoolBus per sapere quando sono stati aggiornati l'ultima volta i GTFS.
- Il server PoolBus risponde alla richiesta con un json in cui è riportata la lista dei GTFS con la data di ultimo aggiornamento e l'url di download.
- Per ciascun GTFS, se la data di aggiornamento fornita da PoolBus è diversa da quella memorizzata nell'istanza di OTP l'istanza OTP scarica il nuovo file.

Questo modulo contiene solo il modello PublicTransitImport, che eredita da diversi mixin e ha i campi url, installed, installed_at, uninstalled_at, status, primary_agency_id, is_map e logs. Questo modello serve per memorizzare le informazioni riguardo un file GTFS. Il modello PublicTransitImport ha un serializer corrispondente PublicTransitImportSerializer.

Questo modulo ha un unico file view, che contiene una API di tipo GET FeedImportAPI, che restituisce la lista di PublicTransitImport.

2.7 Registration mobile API

Questo modulo si occupa di creare un nuovo account utente, effettuare il login e reimpostare la password.

Use Case per la creazione di un nuovo account:

- L'utente utilizza la schermata di Registrazione dell'applicazione mobile per inserire le credenziali (email e password) legate al nuovo account.
- Lo smartphone interroga l'API di Registrazione del server.
- Il server aggiunge il nuovo utente nel database e invia una mail di benvenuto.

- Dopo il server crea un nuovo token e lo invia allo smartphone.
- Lo smartphone registra il token e lo utilizza finché l'utente non esegue il logout.

Login Use Case:

- L'utente utilizza la schermata di Login dell'applicazione mobile per inserire le proprie credenziali (email e password).
- Lo smartphone interroga l'API di Login del server.
- Il server verifica che le credenziali corrispondano a quelle memorizzate nel database e, in caso affermativo, invia un token per le successive comunicazioni.
- Lo smartphone registra il token e lo utilizza finché l'utente non esegue il logout.

Questo modulo utilizza il modello user presente nel framework Django e alcune views di default.

L'API per creare il nuovo utente e quella per reimpostare la password sono state personalizzate in modo da interfacciarsi con l'applicazione mobile.

2.8 Password reset

Si tratta di un modulo per reimpostare la password degli utenti che utilizzano l'applicazione web. Esso utilizza le views della libreria django.contrib.auth.

La personalizzazione in questo caso riguarda solo i template.

2.9 Statistics

Questo modulo serve per generare delle statistiche su piani di viaggio, prenotazioni e viaggi delle navette: esso utilizza serializer specifici per questi tre modelli esportati dai moduli Core App e Corridor Services.

Nel file views è presente l'API Statistics API che analizza le informazioni del e restituisce un json con le statistiche generate. Questa API è di tipo GET e non necessita autenticazione.

2.10 Moduli incompleti (Agency site, Driver Tracker)

Lo sviluppo di PoolBus non è stato terminato, tuttavia è possibile utilizzare l'applicazione per effettuare gli esperimenti. I due moduli incompleti sono Agency Site e Driver Tracker.

Agency Site è il modulo che avrebbe dovuto implementare l'applicazione web per gli operatori.

Driver Tracker invece è il modulo che avrebbe dovuto gestire il lato server dell'applicazione per autisti tramite un'API per collezionare le posizioni e un'API per inviare i percorsi delle navette. Quest'ultima funzionalità è attualmente implementata tramite l'invio di email.

2.11 Integrazione tra i moduli

Nel seguente schema sono indicate le connessioni tra i diversi moduli e tra i moduli e i sistemi esterni. Il modulo Common è omesso, perché contiene porzioni di codice utilizzate da quasi tutti i componenti.

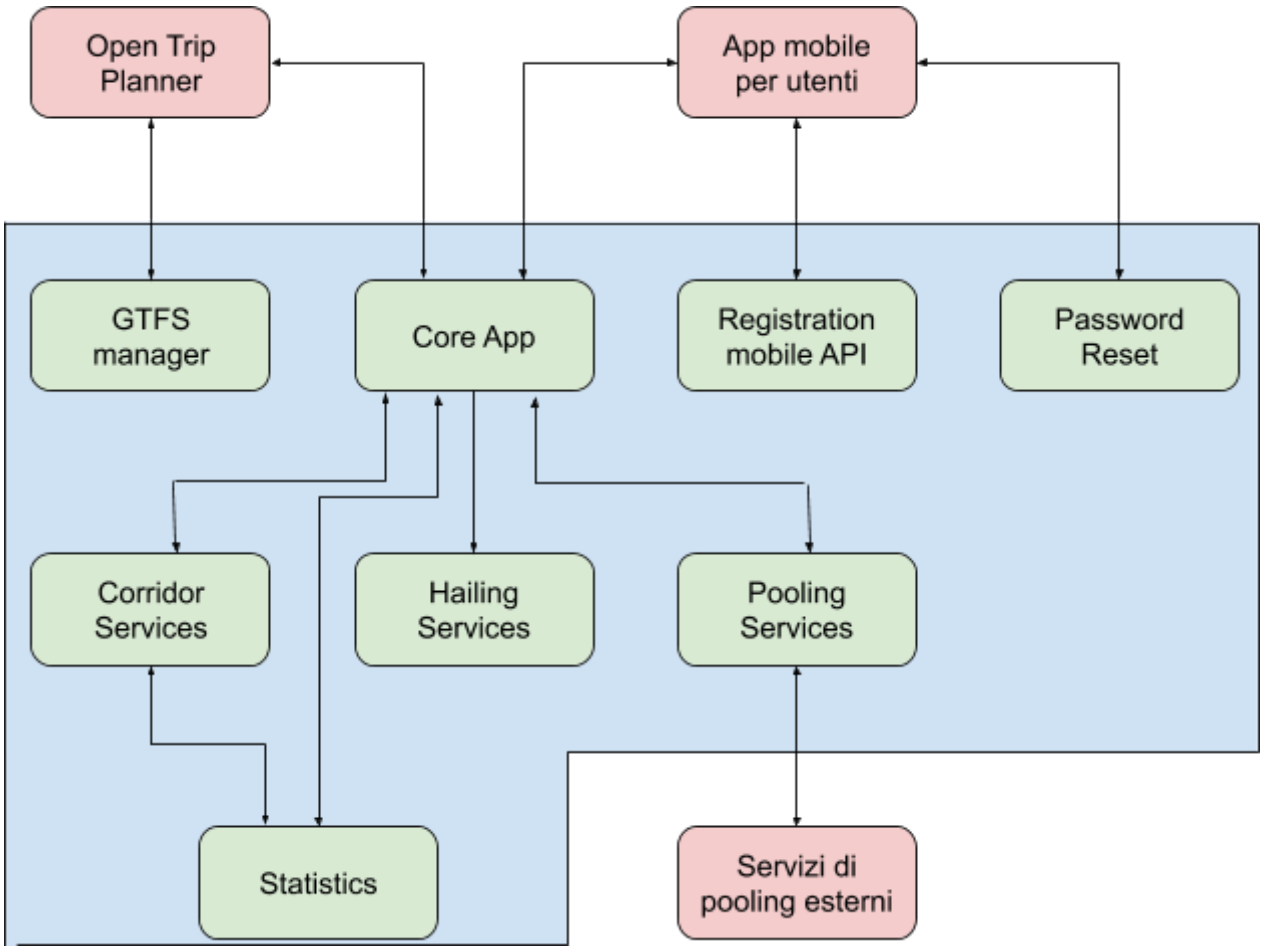

Fig. 1: Integrazione tra i moduli

L'applicazione mobile per utenti si interfaccia con diversi moduli:

- Utilizza il modulo Registration mobile API per creare un nuovo utente e delle nuove credenziali e per i login successivi.
- Utilizza il modulo Password Reset per reimpostare la password.
- Utilizza il modulo Core App per richiedere dei nuovi piani di viaggio e visionare le prenotazioni.

L'istanza di Open Trip Planner interagisce con il modulo GTFS Manager per aggiornare i propri file GTFS e con il modulo Core App per fornire dei piani di viaggio con il trasporto pubblico.

Il modulo Core App, dopo aver ricevuto una richiesta di piano di viaggio dall'applicazione mobile per utenti e ricevuto un piano di viaggio con il trasporto pubblico, analizza quest'ultimo per inserire tratte con servizi di tipo corridor, hailing e pooling, interrogando i relativi moduli.

Il modulo Pooling Services interagisce con API di servizi di Pooling esterni, mentre gli altri due moduli di servizi di trasporto gestiscono internamente le proposte di viaggio.

Infine il modulo Statistics genera delle statistiche dai modelli dei moduli Core App e Corridor Services.

2.12 Schema del database

Di seguito sono riportate le tabelle del database con i rispettivi campi. I mixin sono evidenziati in colori diversi.

Core App:

Plan
user (User) plan (JSON Field) selected_itinerary (Integer Field) updated_selected_itinerary (JSON Field) extra (JSON Field) did_book (Boolean Field) creation_timestamp (Datetime) request_timestamp (Datetime)

Corridor Services:

Corridor Service **(mixins: Operator, WorkingDaysCheckable)**	**Corridor Vehicle**
name (CharField) address (CharField) email (EmailField) phone (CharField) url_website (URLField) type (CharField) area (PolygonField) working_days (JSONField) kilometre_fare_euro (Float) regional_contribute_percentage (Float)	vehicle_model (CharField) total_seats (IntegerField) empty_seats (IntegerField) corridor_service (corridor_service)

Corridor stops	**Matrix element**

name (CharField) point (PointField) corridor_service (corridor_service) location_type (CharField) is_active (BooleanField)	distance_meters (FloatField) duration_seconds (FloatField) shape (linestring) origin (corridor stop) destination (corridor stop)

Corridor Trip	**Corridor Reservation**
time_window (JsonField) pickups_deliveries (JsonField) vehicle_plan (JsonField) is_confirmed (BooleanField) corridor_service (corridor_service) starting_service_hr (DateTimeField) ending_service_hr (DateTimeField) max_service_duration (IntegerField) does_accept_changes (BooleanField) max_passengers_simultaneously (IntegerField) total_fare_euro(FloatField) total_distance_meters(FloatField) vehicle (Corridor Vehicle) driver (User) creation_timestamp (TimestampField) schedule (schedule)	user (User) plan (plan) selected_itinerary (IntegerField) selected_leg (IntegerField) previous_leg_arrival (DateTimeField) next_leg_departure (DateTimeField) fare_euro (FloatField) max_fare_euro (FloatField) time_window (Array field) arrive_by (BooleanField) passengers (IntegerField) corridor_service (corridor_service) min_trip_duration_seconds (FloatField) trip_duration_seconds (FloatField) min_distance_meters (FloatField) distance_meters (FloatField) start_stop (Stop) start_stop_confirmed_hr(DateTimeField) end_stop (Stop) end_stop_confirmed_hr(DateTimeField) creation_timestamp (TimestampField) did_pay (BooleanField) is_confirmed (BooleanField) corridor_trip (Corridor Trip)

Hailing Services:

HailingService **(Operator, WorkingDaysCheckable)**
name (CharField) address (CharField) email (EmailField) phone (CharField) url_website (URLField) type (CharField) area (PolygonField) working_days (JSONField)

Pooling Services :

Pooling Service (Operator, APIable)
name (CharField) address (CharField) email (EmailField) phone (CharField) url_website (URLField) type (CharField) area (PolygonField) API_url (URLField) API_type (CharField) API_key (CharField)

GTFS manager:

Public Transit Import
logs (JSON Field) code (CharField) installed (BooleanField) install_date(DateTimeField) uninstalled_date(DateTimeField) status (CharField) primary_agency_id (CharField) is_map (BooleanField) url (CharField)

3 API principali

3.1 Registration mobile API

3.1.1 API per la creazione di un nuovo utente (applicazione mobile per utenti)

Url: host/registration_mobile_api/api_create_user/

Classe: UserCreate

Parametri della query (POST): username e password

Esempio di risposta:


```
{
   "is_invalid_username": false,
   "is_invalid_password": false,
   "is_user_correctly_created": true,
   "is_user_already_in_db": false
}
```


3.1.2 API per il login (applicazione mobile per utenti)

Url: host/registration_mobile_api/api_token_auth/

Classe: CustomAuthToken

Parametri della query (POST): username e password

Esempio di risposta:


```
{
   "token": "039126...c39446ce"
}
```


3.2 Core App

3.2.1 API per la richiesta di un piano di viaggio

Url: host/otp/

Classe: OTPManager

Esempio di query (GET):

```
host/otp?destination=39.7001853,8.8354817&origin=39.683214,8.7756495&time=11:00&date=29-09-2020&maxWalkDistance=700&passengerNumber=1&format=json
```

Esempio di risposta:


```
{
   "requestParameters": {
       "date": "09-29-2020",
       "requestedModes": "corridorservices,poolingservices,hailingservices",
       "arriveBy": false,
       "fromPlace": "39.683214,8.7756495",
       "toPlace": "39.7001853,8.8354817",
       "time": "11:00AM",
       "maxWalkDistance": 300,
       "passengers": 1,
       "isOtpServerOnline": true
   },
   "plan": {
       "from": {...},
       "to": {...},
       "itineraries": [
           {
               "startTime": 1601370000000.0,
               "endTime": 1601370078319.619,
               "duration": 78319.61889648438,
               "legs": [
                   {
                       "startTime": 1601370000000.0,
                       "endTime": 1601370159363.8516,
                       "duration": 159.36385161007746,
                       "distance": 221.33868279177426,
                       "mode": "WALK",
                       "transitLeg": false,
                       "from": {
                           "arrival": 1601370000000.0,
                           "departure": 1601370000000.0,
                           "lat": 39.683214,
                           "lon": 8.7756495,
                       },
                       "to": {
                           "arrival": 1601370159363.8516,
```

```
                "departure": 1601370159363.8516,
                "lat": 39.683361,
                "lon": 8.777858,
                "vertexType": "TRANSIT"
            },
        },
        {
            "startTime": 1601370000000.0,
            "endTime": 1601370782000.0,
            "duration": 782.0,
            "distance": 6202.053,
            "mode": "SHUTTLE",
            "departureDelay": 0.0,
            "arrivalDelay": 0.0,
            "transitLeg": false,
            "agencyId": 7,
            "agencyUrl": null,
            "agencyName": "Shuttles",
            "realTime": false,
            "from": {
                "arrival": 1601370000000.0,
                "departure": 1601370000000.0,
                "name": "Mogoro",
                "lat": 39.683361,
                "lon": 8.777858,
                "stopId": 11,
                "vertexType": "TRANSIT"
            },
            "to": {
                "arrival": 1601370782000.0,
                "departure": 1601370782000.0,
                "name": "Gonnoscodina",
                "lat": 39.700267,
                "lon": 8.834397,
                "stopId": 8,
                "vertexType": "TRANSIT"
            },
            "reliability": 1,
            "offer": {
```

```
                        "maxPrice": 3.256077825,
                        "minDistance": 6202.053
                    },
                    "pathway": null,
                    "arriveBy": false,
                    "email": null,
                    "phone": null,
                    "address": null,
                    "passengers": 1,
                    "timeWindow": [
                        1601370000000.0,
                        1601371800000.0
                    ],
                    "maxPrice": 3.256077825,
                    "otherPassengers": 0
                },
                {...}
            ]
        },
        {...}
            ...
            ]
        }
    ]
  },
  "error": "",
  "planId": 693,
  "isAnonymous": false
}
```


3.2.2 API per la prenotazione di un itinerario

Url: host/select-itinerary/

Classe: SelectItinerary

Esempio di query (POST): host/select-itinerary?planId=693&itineraryId=1

Esempio di risposta:

```
{
   "passengers": 1,
   "arriveBy": false,
   "planId": 693,
   "itineraryId": 1,
   "reservationIds": [
       669
   ],
   "tripIds": [],
   "isPlanIdValid": true,
   "isItineraryIdValid": true,
   "doesUserOwnThisPlan": true,
   "doesPlanExists": true,
   "isAnonymous": false,
   "isAlreadyBooked": false,
   "error": null
}
```


3.3 Statistics

3.3.1 API per la generazione di statistiche

Url: host/statistics/

Classe: StatisticsAPI

Questa API GET non richiede parametri di input.

Esempio di risposta:


```
{
   "sel_itinerary": {
       "shuttle_duration": {
           "max": 3781.0,
           "min": 394.0,
           "mean": 1879.7843137254902
       },
       "walk_duration": {
           "max": 237.68347055919037,
           "min": 0,
           "mean": 4.753669411183807
       },
```

```
        "shuttle_distance": {
            "max": 59279.43,
            "min": 3289.797,
            "mean": 23144.20700000001
        },
        "walk_distance": {
            "max": 330.11593133220885,
            "min": 0,
            "mean": 6.602318626644177
        },
        "shuttle_min_duration": {
            "max": 3781.0,
            "min": 394.0,
            "mean": 1879.7843137254902
        },
        "shuttle_min_distance": {
            "max": 59279.43,
            "min": 3289.797,
            "mean": 23144.20700000001
        },
        "duration": {
            "max": 3781000.0,
            "min": 78319.61889648438,
            "mean": 1976342.2116360134
        },
        "distance": {
            "max": 57090.31,
            "min": 5372.055,
            "mean": 24883.367914759227
        },
    },
    "tot_plans": 52,
    "tot_pt_plans": 0,
    "tot_shuttle_plans": 51,
    "tot_multimodal_plans": 0,
    "tot_unselected_plans": 1,
    "tot_unsatisfied_plans": 0,
    "tot_shuttle_trips": 20,
    "duration_shuttle_trip": {
```

```
        "max": 9979.0,
        "min": 2340.0,
        "mean": 4568.3
    },
    "distance_shuttle_trip": {
        "max": 116332.555,
        "min": 10764.66,
        "mean": 53411.505900000004
    },
    "occupation_percentage_shuttle_trip": {
        "max": 63.118019293147334,
        "min": 25.0,
        "mean": 31.072567631324027
    }
}
```


4 Concetti Fondamentali dello sviluppo Python con Django

Questa sezione introduce alcuni concetti di programmazione Python riferiti al framework Django.

4.1 Struttura di un'applicazione Django

Un'applicazione django è solitamente contenuta all'interno di una cartella con il nome del progetto.

All'interno di questa cartella esistono sottocartelle riferite ai vari moduli, salvate con lo stesso nome del modulo a cui si riferiscono.

In aggiunta, nella cartella principale sono presenti:

- una cartella con lo stesso nome del progetto (essa contiene i file di impostazioni e gli url dei moduli)
- il file manage.py (che serve ad avviare il programma)

All'interno del file di impostazioni sono elencati tutti i moduli utilizzati dal programma.

4.2 Struttura di un modulo Django

Un modulo Django (detto anche applicazione) è costituito da un insieme di file:

- una cartella di template html
- un file admin (per gestire le tabelle del database da un'applicazione grafica tramite il link host/admin)
- un file apps (che contiene alcune configurazioni del modulo, come ad esempio il nome)
- un file models (che contiene i modelli, ovvero delle classi Python che mappano gli oggetti del db)
- un file serializers (che serve a serializzare i modelli)
- un file urls (che serve per associare gli url a delle funzioni)

- uno o più file views (che contengono le funzioni)

4.3 Modelli, mixins e serializers

Un modello[4] è una classe che contiene i campi e i comportamenti essenziali dei dati del database. In genere, ogni modello viene mappato da una singola tabella di database.

Un Mixin[5] è un tipo speciale di eredità in Python e consente alle classi di condividere metodi tra qualsiasi classe che eredita da quel Mixin.

I serializers[6] consentono di convertire dati complessi in tipi di dati Python nativi che possono quindi essere facilmente renderizzati in JSON, XML o altri tipi di contenuto. I serializzatori forniscono anche la deserializzazione, consentendo ai dati analizzati di essere riconvertiti in tipi complessi, dopo aver prima convalidato i dati in arrivo.

4.4 Views

Una view[7] è una funzione Python che accetta una richiesta Web e restituisce una risposta Web. Questa risposta può essere il contenuto HTML di una pagina Web, un reindirizzamento o un errore 404, un documento XML o qualsiasi altra cosa. La view stessa contiene la logica necessaria per restituire quella risposta. La convenzione è di mettere le views in un file chiamato views.py, inserito nel progetto o nella directory dell'applicazione.

4.5 API

Un'API[8] (Application Programming Interface) è un'interfaccia di elaborazione che definisce le interazioni tra più intermediari software. Definisce i tipi di chiamate o richieste che possono essere effettuate, come effettuarle, i formati dei dati da utilizzare e le convenzioni da seguire.

Un'API può essere completamente personalizzata, specifica per un componente oppure può essere progettata in base a uno standard di settore per garantire l'interoperabilità. Attraverso l'occultamento delle informazioni, le API consentono la programmazione modulare, che consente agli utenti di utilizzare l'interfaccia indipendentemente dall'implementazione.

Nello sviluppo Django è solitamente utilizzato Django REST framework[9], un utili strumento per lo sviluppo Web.

[4] https://docs.djangoproject.com/en/3.1/topics/db/models/
[5] https://docs.djangoproject.com/en/3.1/topics/class-based-views/mixins/
[6] https://www.django-rest-framework.org/api-guide/serializers/
[7] https://docs.djangoproject.com/en/3.1/topics/http/views/
[8] https://en.wikipedia.org/wiki/API
[9] https://www.django-rest-framework.org/