Bushaj Aldo 847091

Bushaj Antonino 847013

Frisullo Stefano Silvestro 832813

**Relazione Progetto: B-Next**

****

[**Tema dell’app e deviazioni rispetto a quanto specificato nella scheda**](#_jr3a2cfw5bil) **3**

[**Schema del moqups dell’applicazione:**](#_bpc8dht5479s) **4**

[**Metodologia di sviluppo**](#_t10s344hx7t5) **5**

[**Funzionalità e Struttura dell’applicazione**](#_lbu9b4ucrvqd) **5**

[Struttura del Server:](#_x19o1vgzs6f) 5

[User](#_n3s8h3r4qn3t) 6

[Car](#_gr2k2prc7d4a) 7

[Struttura dell’applicazione:](#_lwnkt7cvwoya) 9

[Struttura del Database:](#_5cmolvb7zczf) 9

[Dettagli sulle schermate dell’app](#_1ze5ogui38w) 10

[**Scelte Progettuali e Librerie Utilizzate**](#_ovtxbrzc74g0) **16**

[Librerie più significative:](#_6j4rk4qlh67q) 16

[**Test**](#_8xlyj9kkwmj0) **17**

## Tema dell’app e deviazioni rispetto a quanto specificato nella scheda

B-Next si propone di offrire un servizio di ride-hailing, in cui l’utente tramite l’utilizzo di un’app può richiedere, dal proprio smartphone, di essere accompagnato in una determinata destinazione da una delle auto disponibili (che non sono state già prenotate da altri utenti nello stesso arco temporale).

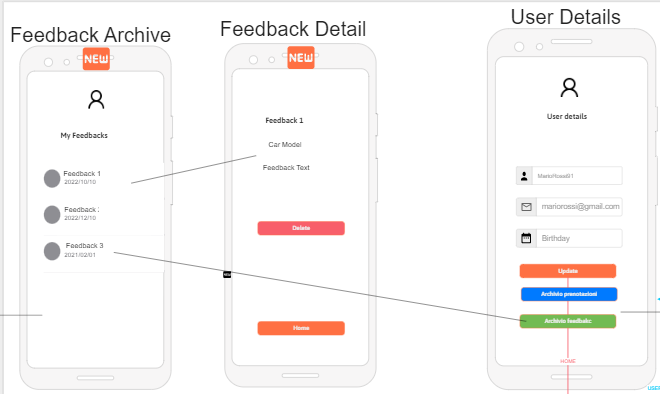
La peculiarità del nostro progetto sta nel fatto che le auto utilizzate per l’erogazione del servizio sono a guida autonoma in cui appunto non serve un conducente fisico, infatti nel momento in cui il veicolo lascia il cliente alla propria destinazione è immediatamente disponibile per la corsa successiva.

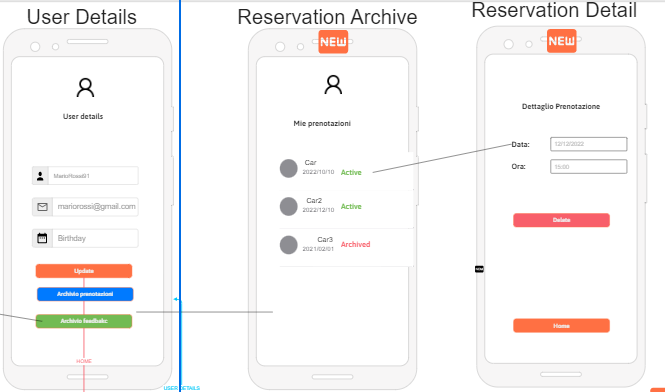
L’app B-Next addebita poi una commissione per il servizio erogato. Dal punto di vista di chi offre il servizio abbiamo i proprietari dei veicoli che possono essere chiunque abbia un veicolo da mettere a disposizione.

Il servizio di ride-hailing è visto come un’alternativa ai normali servizi di taxi. In questo caso il vantaggio è dovuto al fatto che le auto saranno a guida autonoma ed il proprietario del veicolo potrebbe anche essere a lavoro e lasciare la propria auto a disposizione solo durante le ore a lui più comode, quindi andiamo ad ammortizzare dei costi di base per il mantenimento del servizio.

Inizialmente l’applicazione si proponeva di offrire un servizio di car-pooling in cui la corsa veniva condivisa con altri utenti che potevano usufruire del servizio anche a corsa avviata. Il tutto è stato modificato in quanto la prenotazione adesso viene effettuata nella fase iniziale e non possono salire altri passeggeri una volta effettuata la prenotazione, il tutto quindi come già accennato si presta ad offrire un servizio di ride-hailing.

## Schema del moqups dell’applicazione:





## 

## Metodologia di sviluppo

Per quanto riguarda la metodologia di sviluppo ci si è basati su un approccio Agile, quindi un approccio che enfatizzasse la flessibilità, la collaborazione tra i vari componenti del gruppo e la capacità di adattarsi rapidamente a requisiti o circostanze mutevoli. Abbiamo optato per un approccio strutturato in brevi cicli di sviluppo tipico dell’extreme programming, in modo da dare rilievo ad una scrittura del codice di qualità.

## Funzionalità e Struttura dell’applicazione

L’applicazione è stata sviluppata per dispositivi Android, la versione minima supportata è la versione 10.0 Q API level 29, mentre per lo sviluppo è stato utilizzato il linguaggio Kotlin.

E’ stato previsto un supporto per diversi cellulari come il Pixel 5 e Pixel 6 su cui è stato testato opportunamente il funzionamento tramite un emulatore. L’applicazione è stata pensata con un supporto a lingue diverse, al momento è presente il supporto anche per la lingua Inglese oltre alla lingua Italiana, tuttavia l’integrazione di altre lingue è facilmente realizzabile grazie alla suddivisione in moduli separati delle stringhe utilizzate nel progetto.

### 

### Struttura del Server:

Il Server è stato implementato utilizzando il Framework [Spring Boot](https://spring.io/projects/spring-boot) (Java), che gestisce tutta la logica dell’applicazione lato backend quindi:

* registrazione utenti
* aggiunta di nuovi veicoli al servizio
* prenotazioni delle auto
* aggiunta di feedback da parte dell’utente
* recupero di prenotazioni e feedback
* ricerca di veicoli disponibili in un arco temporale
* gestione di dati generici di tutti i componenti dell’applicazione

Nella fase di login registriamo le credenziali inserite, queste vengono inviate al backend, il quale verifica che nel db vi siano effettivamente tali credenziali salvate e se l'esito è positivo allora viene generato un token dalla durata limitata associata al’utente che effettua il login, altrimenti si lancia un errore.

Il server nella gestione della logica del programma espone diverse API che vengono poi chiamate dalla nostra applicazione di cui ne mostriamo qualcuna di seguito:

#### **User**

***getAllUsers****@GetMapping("allUsers")*

*@NotNull*

* *public @NotNull* [*List*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/List.html)*<User>* ***getAllUsers****()*

***getUserById****@GetMapping("id={userId}")*

*@Nullable*

* *public @Nullable User* ***getUserById****(@PathVariable @NotNull, @NotNull* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html) *userId)*

***getUserByName****@GetMapping("name={userName}")*

*@Nullable*

* *public @Nullable User* ***getUserByName****(@PathVariable* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html) *userName)*

***deleteUser****@DeleteMapping("del={userId}")*

*@NotNull*

* *public @NotNull ResponseEntity<*[*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html)*>* ***deleteUser****(@PathVariable @NotNull, @NotNull* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html) *userId)*

***generateJwtToken****@PostMapping("/signin")*

* *public ResponseEntity<Response>* ***generateJwtToken****(@RequestBody User user)*

***signup****@PostMapping("/signup")*

* *public ResponseEntity<*[*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html)*>* ***signup****(@RequestBody @NotNull User user)*

***updateUser****@PutMapping("/update")*

* *public* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html)***updateUser****(@RequestBody @NotNull User user)*

#### **Car**

***getAllCar****@GetMapping("/cars")@NotNull*

* *public @NotNull* [*List*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/List.html)*<Car>* ***gatAllCar****()*

***getCarById****@GetMapping("/cars/{carID}")*

* *public Car* ***getCarById****(@PathVariable @NotNull, @NotNull* [*UUID*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/UUID.html) *carID)*

***addCar****@RequestMapping(method=POST, value="/cars")@NotNull*

* *public @NotNull ResponseEntity<Car> addCar(@RequestBody @NotNull, @NotNull Car car)*

***updateCar****@PutMapping("/cars")*

*@NotNull*

* *public @NotNull <*[*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html)*>* ***updateCar****(@RequestBody @NotNull, @NotNull Car car)*

***deleteCar****@RequestMapping(method=DELETE, value="/cars/{carId}")*

*@NotNull*

* *public @NotNull* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html)***deleteCar****(@PathVariable @NotNull* [*UUID*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/UUID.html) *carId)*

***getOwnedCars****@GetMapping("/cars/user={ownerId}")*

* *public @Nullable* [*List*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/List.html)*<Car>* ***getOwnedCars****(@PathVariable* [*String*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html) *ownerId)*

***getCarsWithLowBattery****@GetMapping("/cars/battery/user={userId}")*

*@Nullable*

* *public @Nullable* [*List*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/List.html)*<Car>* ***getCarsWithLowBattery****(@PathVariable @NotNull @NotNull* [*UUID*](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/UUID.html) *userId)*

### Struttura dell’applicazione:

**Model:** contiene tutte le classi che fanno da modello quindi ad esempio l’utente con i dati personali (nome, cognome, data di nascita ecc..)

**Bnext:** contiene tutte le principali classi che si occupano di gestire il funzionamento dell’app, quindi tutto ciò concerne la logica del programma.

**Bnext (androidTest):** cartella che contiene i test strumentali sull’app, esempio corretta visualizzazione di bottoni, view, ecc.

**Bnext (test):** cartella che contiene il codice degli unit test

**Layout:** contiene tutte le varie activity del programma

**Drawable:** contiene tutte le varie immagini e icone mostrate nel programma

### 

### Struttura del Database:

Per quanto riguarda la struttura del database, lato server, è stato utilizzato H2, scelta fatta in quanto è un database relazionale molto rapido all’avvio e leggero in memoria. Esso è stato reso persistente facendo si che i dati venissero scritti su un file e mantenendo il DB aperto anche quando la connessione al database viene chiusa in modo da garantire che i dati nel DB non vengano persi nel momento in cui il server viene spenta.

Gli elementi contenuti nel database sono elencati di seguito:

**Car**

*car\_id* UUID NOT NULL,

*name* VARCHAR(255),

*car\_model* VARCHAR(255),

*plate\_number* VARCHAR(255) NOT NULL,

*battery* INT,

*price\_hour* INT,

*price\_km* INT,

*availability\_present* BOOLEAN NOT NULL,

*user\_id* UUID,

*position\_id* UUID,

*CONSTRAINT* pk\_car PRIMARY KEY (*car\_id*)

**Feedback**

*id\_feedback* UUID NOT NULL,

*comment* VARCHAR(255) NOT NULL,

*user\_id* UUID,

*car\_id* UUID,

CONSTRAINT pk\_feedback PRIMARY KEY (*id\_feedback*)

**Reservation**

*reservation\_id* UUID NOT NULL,

*start\_of\_book* TIMESTAMP,

*end\_of\_book* TIMESTAMP,

*user\_id* UUID

*car\_car\_id* UUID,

*destination\_id* UUID,

*start\_position\_id* UUID,

CONSTRAINT pk\_reservation PRIMARY KEY (*reservation\_id*)

**User**

*user\_id* UUID NOT NULL,

*name* VARCHAR(255),

*surname* VARCHAR(255),

*birth\_date* TIMESTAMP,

*username* VARCHAR(255) NOT NULL,

*password* VARCHAR(255) NOT NULL,

*active* INT NOT NULL,

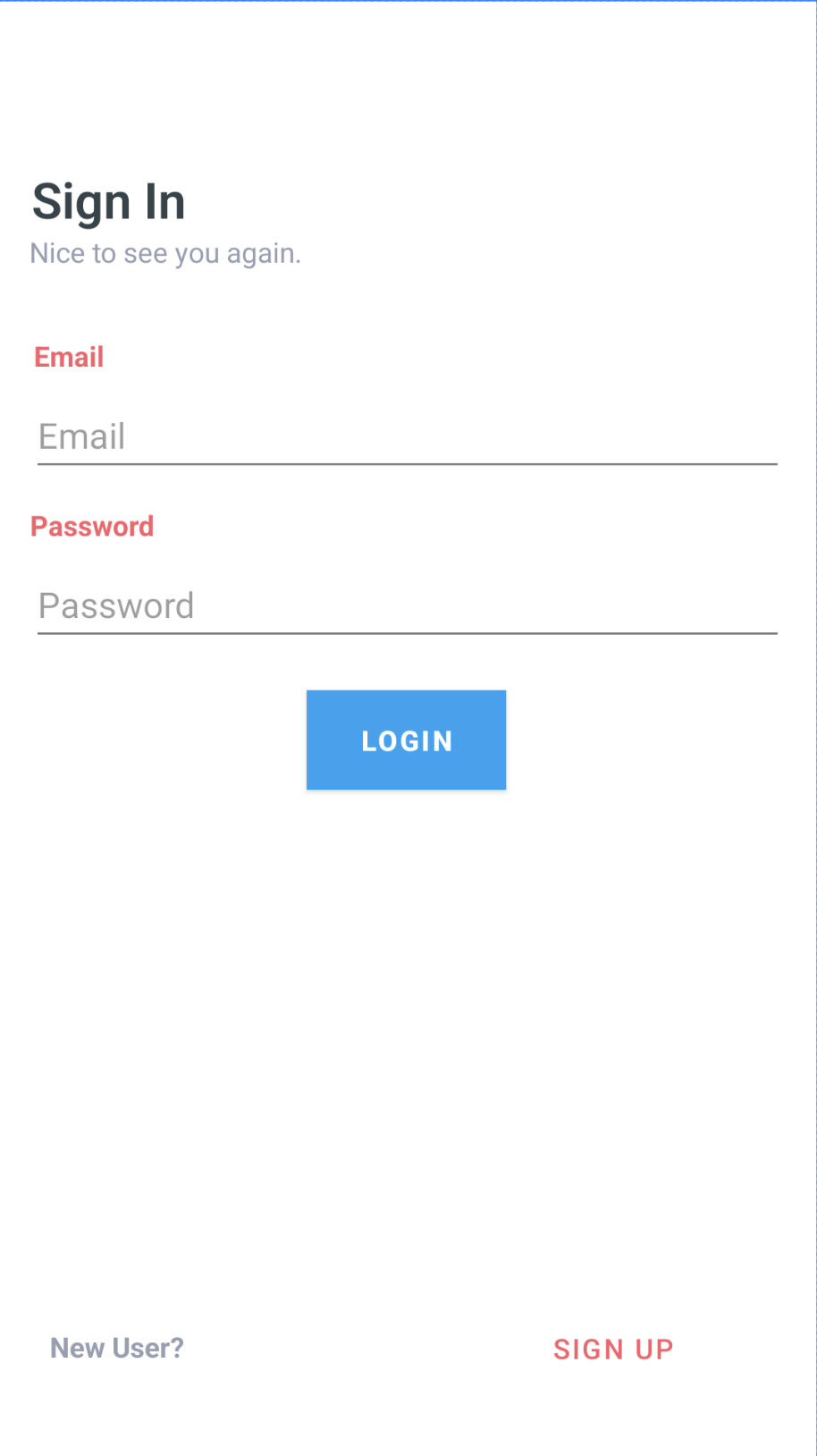
*permissions* VARCHAR(255),

*roles* VARCHAR(255),

CONSTRAINT pk\_user PRIMARY KEY (*user\_id*)

Qui di seguito spieghiamo nello specifico il comportamento delle varie activity che compongono l’applicazione

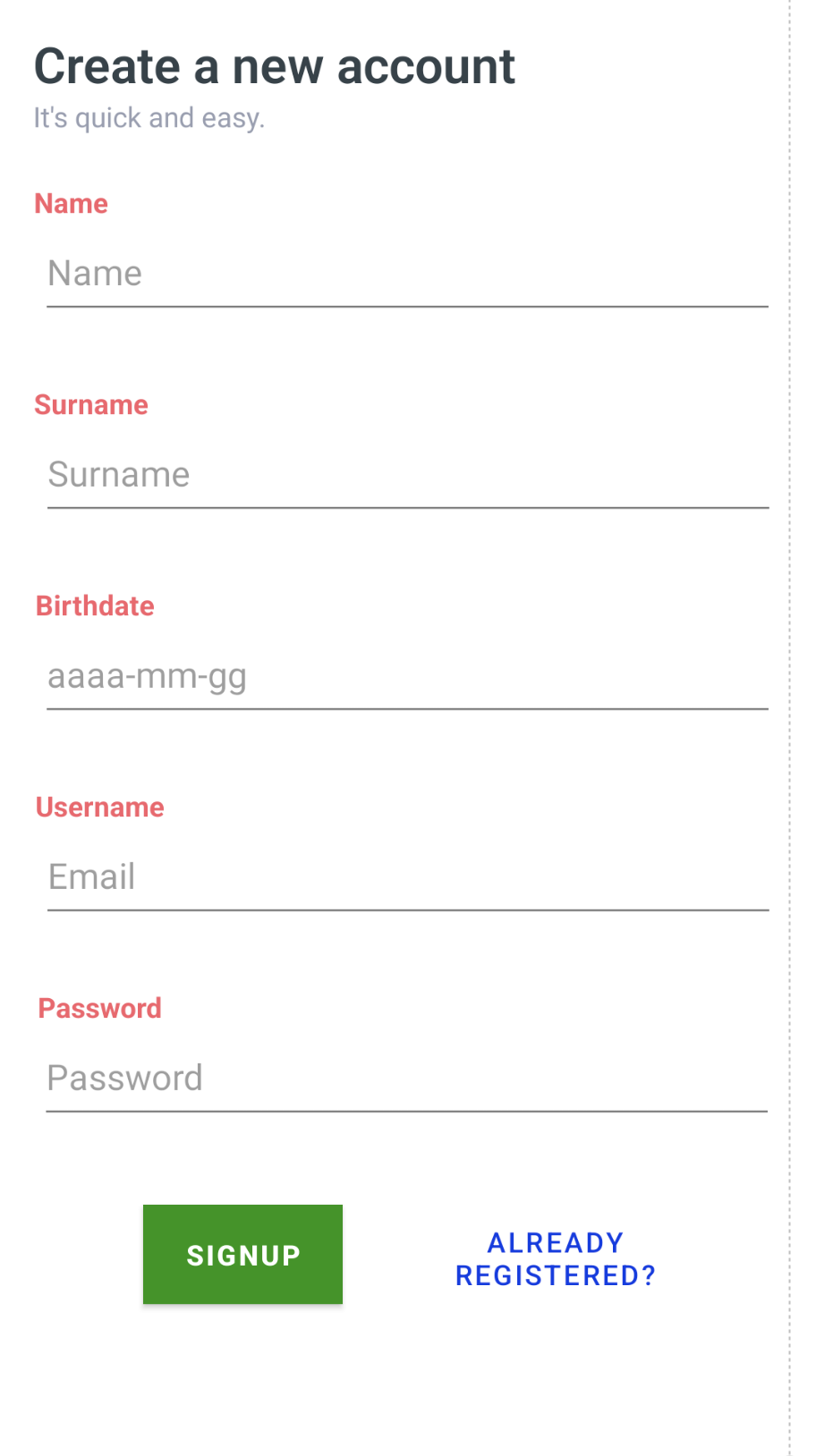
### Dettagli sulle schermate dell’app



**Activity\_main:**

La prima activity che viene mostrata all’utente è la activity\_main che ha la funzione di indirizzare l’utente verso l’attività di login tramite email e password o alla fase di registrazione tramite il tasto SIGN UP.

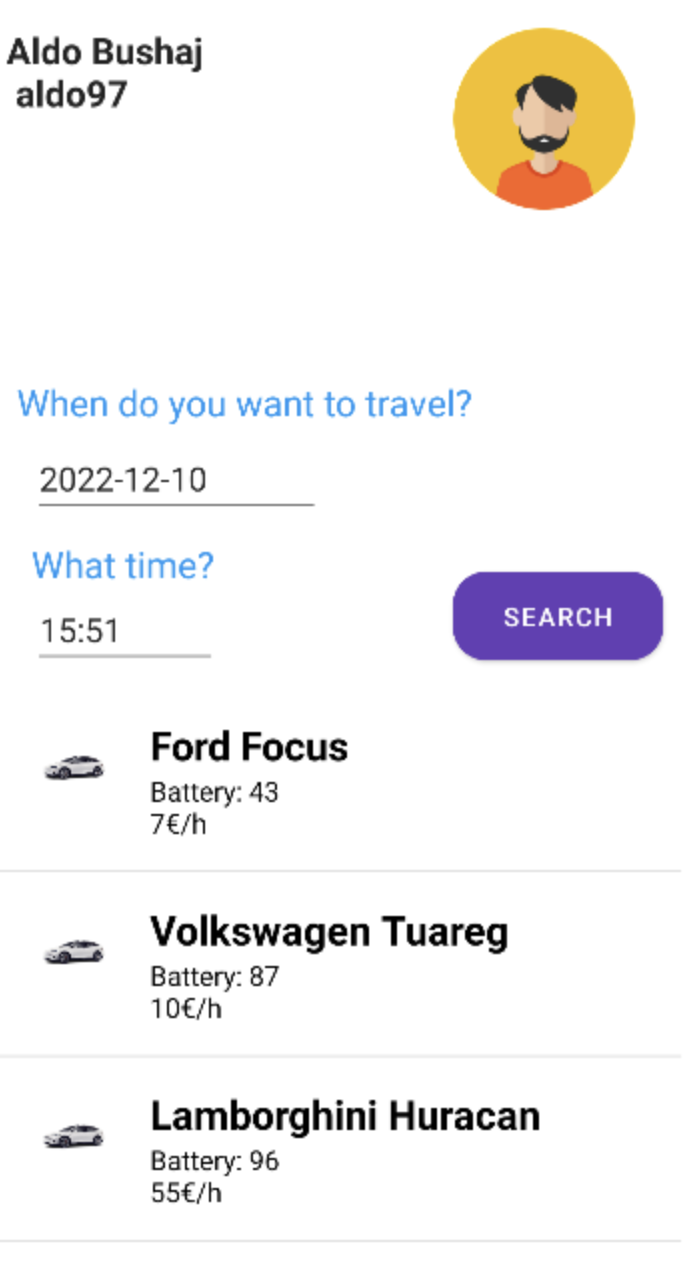
All’avvio dell’applicazione viene sempre mostrata questa schermata che poi una volta effettuato il login porta alla fase di ricerca di un veicolo disponibile.



**Activity SignUp:**

Questa è l’activity che viene mostrata ai nuovi clienti che devono ancora effettuare la registrazione, vengono richiesti i dati dell’utente tra cui username e password che saranno poi le credenziali utilizzate nella fase di login. Attraverso questa activity viene creata un’istanza di accesso relativa all’utente nel database al quale viene associato un token con una durata limitata ed assicura che l’utente sia connesso. Nel momento della creazione di un nuovo utente, ad esso viene assegnato anche un UID univoco generato casualmente.

Se nella fase di registrazione manca qualche dato oppure il formato non è quello corretto questo viene notificato all’utente tramite un Toast.

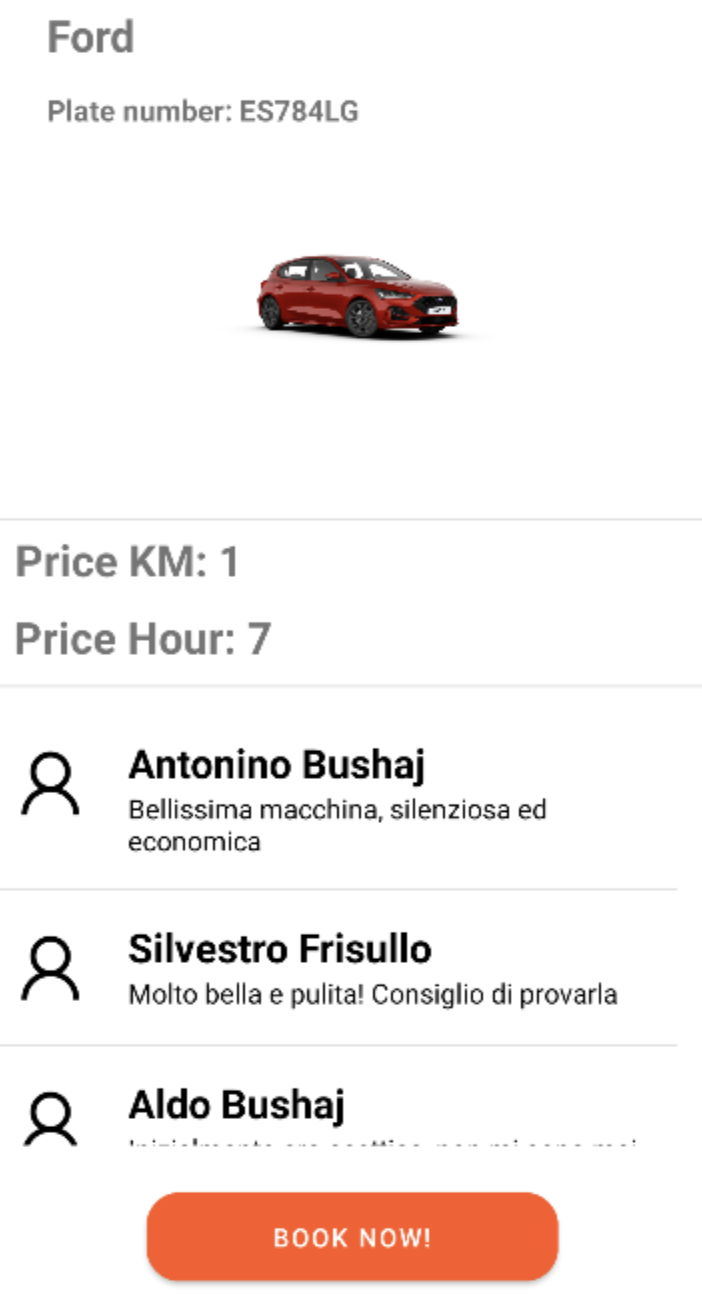


**Activity\_book\_ride:**

Questa è l’activity a cui si viene reindirizzati nel momento in cui viene effettuato il login, in questa fase è possibile fare una ricerca dei veicoli disponibili .

Una volta scelti il giorno e l’ora ci vengono mostrate sotto tutte le auto disponibili in quel momento con indicato anche il livello di batteria(ricordiamo che le auto sono elettriche e a guida autonoma) e il prezzo all’ora dei rispettivi veicoli.

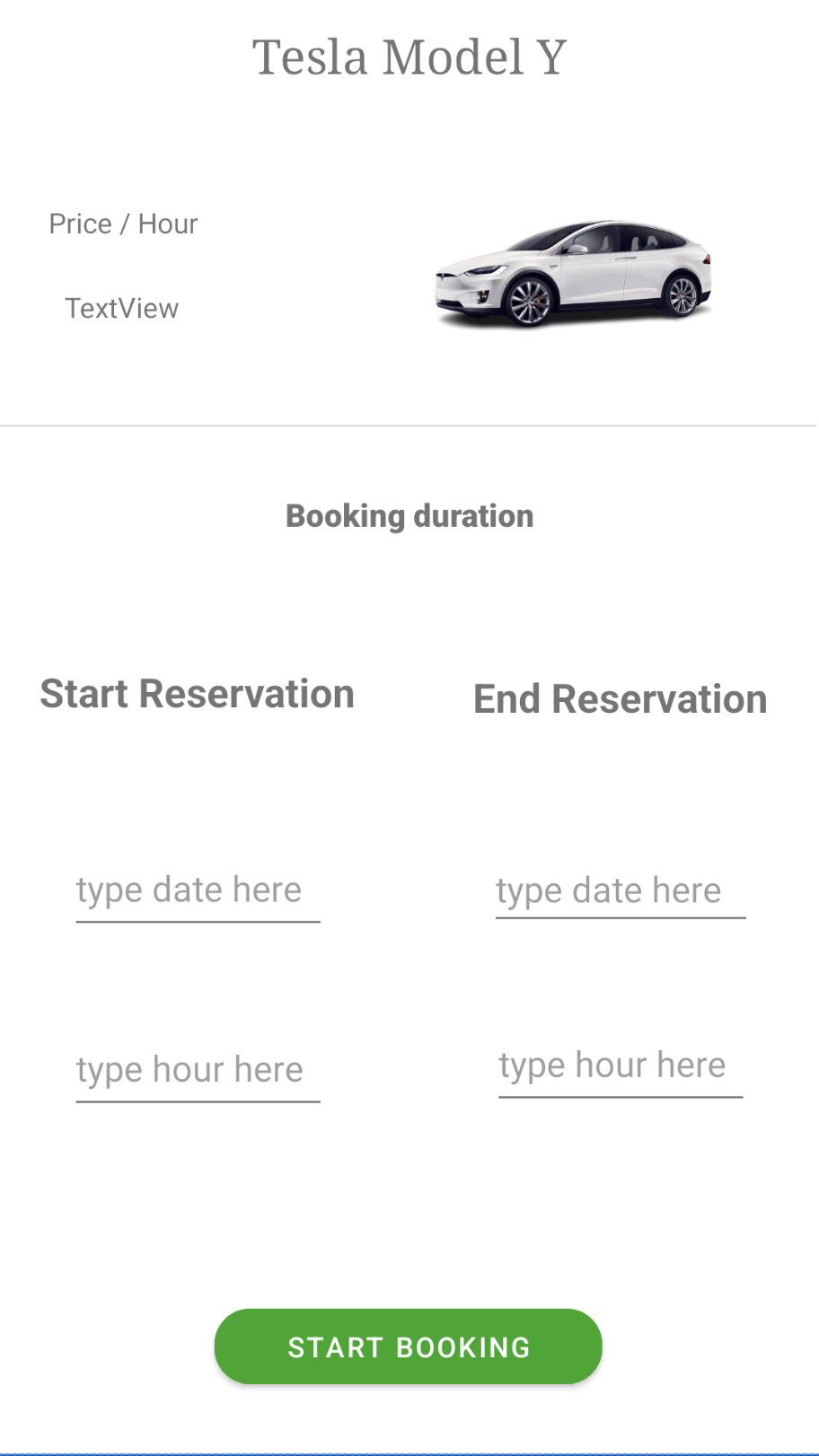
Una volta scelta la macchina che si vuole cliccandola andiamo alla fase successiva spiegata di seguito.



**Activity\_stats\_car:**

Una volta selezionato il veicolo che è disponibile nelle fasce orarie da noi indicate passiamo a questa schermata che ci mostra i dettagli dell’auto. In alto abbiamo il nome della macchina, il numero di targa e dettagli quali il prezzo al km, il prezzo all’ora e tutti i vari commenti lasciati dagli utenti a seconda della loro esperienza, inoltre abbiamo una foto che cambia a seconda del veicolo scelto per mostrare graficamente l’aspetto della vettura.

Tramite il tasto *BOOK NOW* possiamo infine prenotare il nostro veicolo che ci porta alla pagina successiva.



**Activity\_book:**

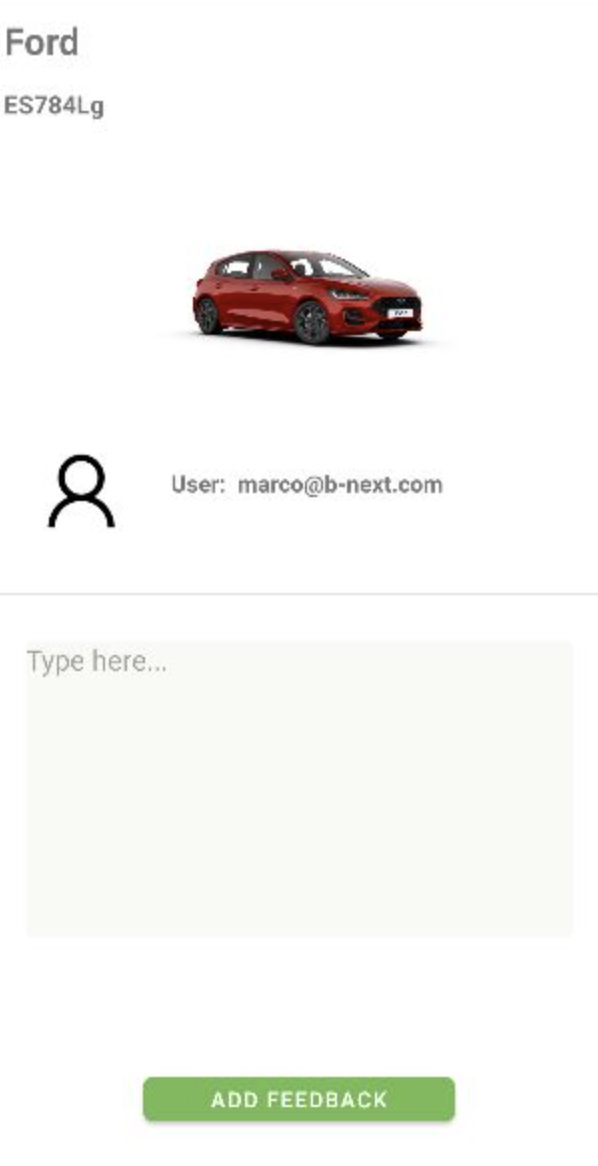
In questa fase andiamo ad effettuare la prenotazione vera e propria del veicolo che abbiamo scelto. Si inseriscono la data e l’ora di inizio e di fine prenotazione e tramite il tasto *start booking* si conferma l’operazione.



**Activity Payment**

Questa activity ci mostra i dettagli della nostra prenotazione con inizio e fine della corsa che abbiamo selezionato precedentemente e il prezzo totale da pagare.

Tramite il tasto “Leave a feedback” abbiamo la possibilità di lasciare un commento riguardo la nostra esperienza con il servizio fruito. Questi commenti verranno visualizzati durante le future prenotazioni.

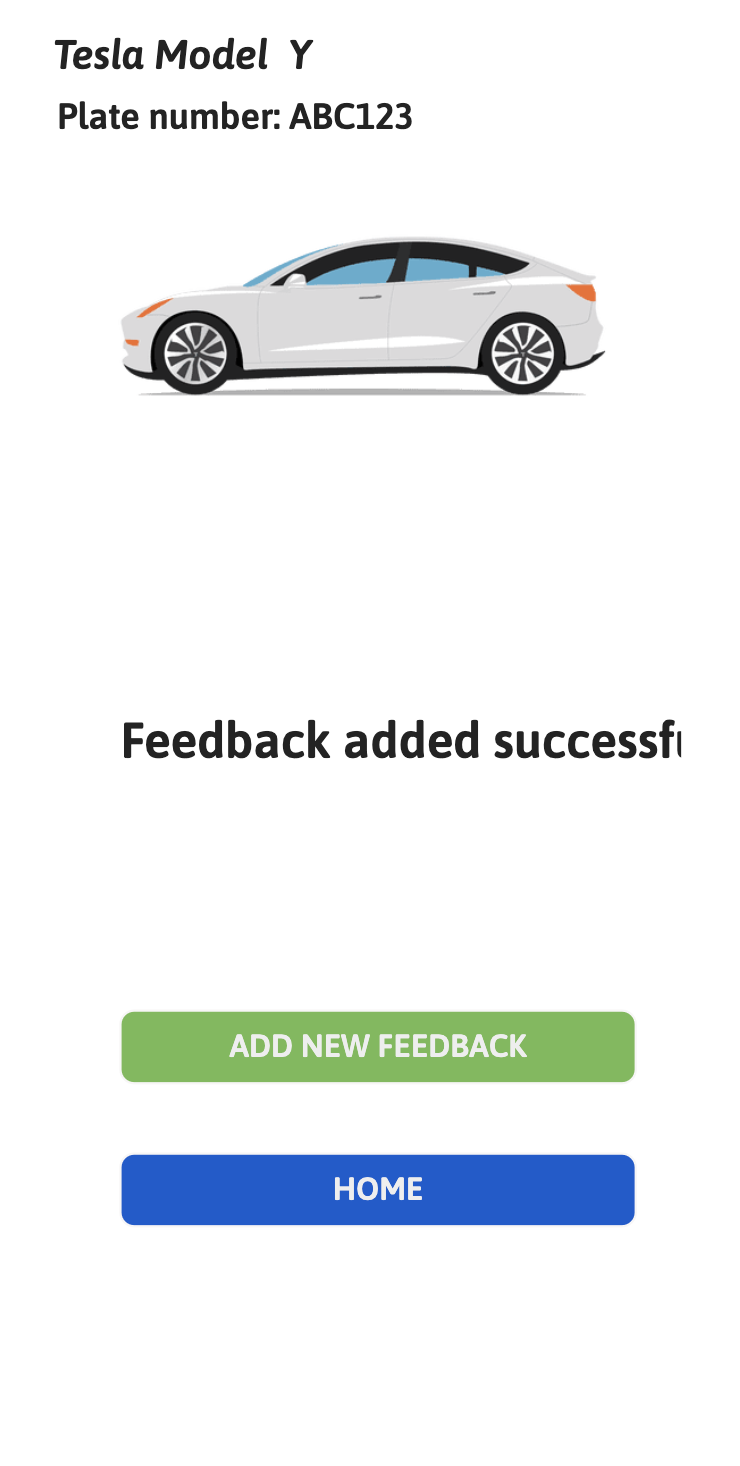


**Activity\_add\_comment:**

Questa activity viene eseguita nel momento in cui l’utente vuole lasciare un feedback.

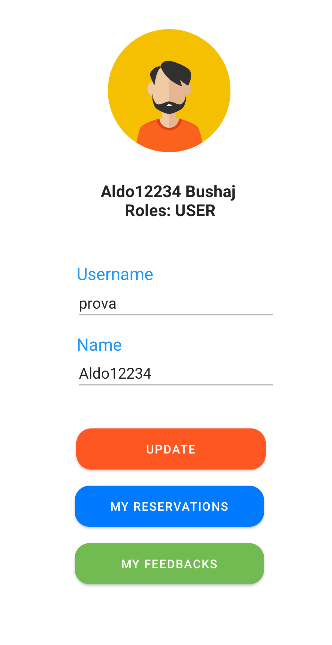
In questo caso abbiamo un campo di testo centrale dove l’utente andrà a scrivere il suo commento e il tasto *ADD FEEDBACK* che conferma l’operazione.

Questo è un modo prezioso per gli utenti di condividere le proprie esperienze e aiutare l'azienda a migliorare il proprio servizio.



**Activity\_comment\_added:**

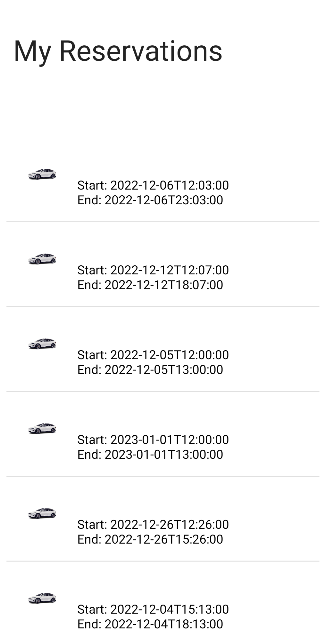
Nel momento in cui un utente ha aggiunto un nuovo feedback veniamo reindirizzati a questa pagina che ci notifica il corretto inserimento del commento, nella parte in alto vengono mostrate anche le informazioni principali del veicolo a cui si riferisce il feedback. Qui si ha la possibilità di aggiungere un nuovo commento tramite il tasto *ADD NEW FEEDBACK* oppure di ritornare alla pagina di prenotazione e scegliere un altro veicolo.



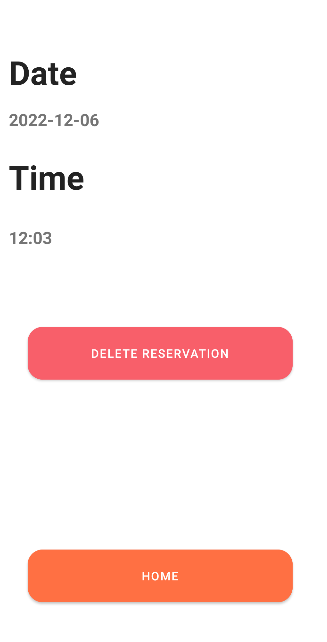
**Activity\_user\_details:**

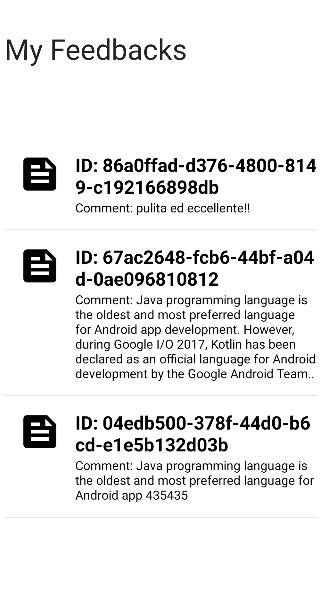
Questa è la parte dedicata al profilo del cliente che ha effettuato il login, in questa fase ci vengono mostrate informazioni quali l’username e i ruoli associati a quest’ultimo. Ricordiamo che in questa applicazioni ci saranno utenti registrati come clienti che usufruiscono del servizio e clienti registrati come proprietari dei veicoli messi a disposizione.

Tramite il tasto update è possibile modificare i propri dati personali, mentre tramite i tasti my reservation e my feedbacks è possibile passare alla sezione personale delle prenotazioni e dei feedbacks.



**Archive\_activity:**

Questa sezione è dedicata alle prenotazioni di un singolo utente, qui ci vengono mostrate le informazioni essenziali delle varie prenotazioni come l’ora di inizio e l’ora di fine corsa. Cliccando su una singola prenotazione passiamo invece ai suoi dettagli, dove ci viene mostrato anche un bottone DELETE RESERVATIONnel caso in cui volessimo cancellare la prenotazione selezionata, oppure ritornare al menù principale tramite il tasto *HOME*



**Activity\_feedbacks\_archive:**

Infine ricordiamo che nel database vengono salvati tutti i feedback inseriti dai vari utenti, questi sono visualizzabili nella seguente schermata dove troviamo i dettagli, con il commento scritto dall’utente e l’id univoco associato al suo profilo.

## Scelte Progettuali e Librerie Utilizzate

Tra le scelte progettuali più rilevanti troviamo quella di non utilizzare un database interno al dispositivo per le istanze degli utenti, macchine e gli elementi minori. Questo ha come vantaggio quello di rendere l’applicazione estremamente leggera, lo svantaggio è quello di avere una velocità minore in quanto bisogna scaricare gli elementi dal db ogni volta. L’applicazione si presta ad essere scalabile e riuscire a gestire anche un incremento del numero di utenti che la utilizzano senza perdere di performance mantenendo la qualità del servizio sempre costante, per questo motivo in implementazioni future, con il crescere del numero di utenti, sarebbe da valutare il salvataggio dei file relativi al proprio profilo in un database interno all’app.

### Librerie più significative:

Per lo sviluppo dell’applicazione sono state utilizzate diverse librerie tra cui:

**Jackson:** libreria utilizzata per analizzare, generare e manipolare i dati nel formato JSON. Questa ha permesso di rendere il lavoro più leggero e messo a disposizione un set di APIs per lavorare con JSON in Java.

**GSON**: libreria utilizzata nella conversione di oggetti Java in e da JSON. Utilizzata comunemente insieme a Retrofit per analizzare le risposte JSON da un server.

**Android Support Library:** la libreria di supporto Android è una raccolta di librerie che forniscono versioni retrocompatibili delle API del framework Android, nonché funzionalità che non sono disponibili nelle API del framework. Questa libreria è essenziale per lo sviluppo di app compatibili con le versioni precedenti di Android.

**Fast Android Networking Library** ([link](https://github.com/amitshekhariitbhu/Fast-Android-Networking)) **:** questa potente libreria viene utilizzata per eseguire qualsiasi tipo di richiesta di rete nelle applicazioni Android ed è realizzata su OkHttp Networking Layer. Fast Android Networking Library si occupa di gestire ogni fattore, non bisogna fare nulla, una volta effettuata la richiesta bisogna solo aspettare la risposta.

## Test

I test unitari (Unit Test) sono un tipo di test software che verifica la correttezza delle singole unità di codice. Nel contesto dello sviluppo Android, uno unit test è un test che verifica la correttezza di una sezione specifica del codice di un'app Android. Questi test sono in genere piccoli e si concentrano su una singola funzione o metodo, rendendoli facili da scrivere ed eseguire. Vengono spesso utilizzati per garantire che il codice sia corretto e affidabile prima che venga integrato con altre parti dell'app.

I test che sono stati effettuati sono sia gli Unit Test che gli Integration test i quali ci permettono di garantire il corretto funzionamento dell’applicazione verificando la correttezza dei risultati.

In particolare, ad esempio, sono stati effettuati dei test che assicurano la corretta registrazione di un nuovo utente, la creazione di una nuova macchina intestata ad un utente, l’aggiunta di un feedback (identificato con un ID) e l’assegnazione di tale feedback ad un veicolo preciso. Qui di seguito alleghiamo degli screen dei test effettuati:

