Report Progetto LAM 2023
Bumpyroads

Koci Erik M. 0000997662
erik.koci@studio.unibo.it

June 2023

Indice

1 Introduzione all'applicazione

BumpyRoads, è un'applicazione innovativa che consente di **segnalare** e individuare i **difetti** presenti sulle **strade**. Le strade sono un elemento fondamentale delle nostre comunità, ma spesso possono presentare problemi come buche, mancanza di luce, rifiuti per strada o altre irregolarità che possono compromettere la **sicurezza** e il **comfort** dei **conducenti** e dei **pedoni**.

Con BumpyRoads, è possibile segnalare i difetti stradali in modo **rapido** ed efficiente. L'applicazione permette di rilevare e **registrare** le **imperfezioni** sulle strade utilizzando un dispositivo mobile. Basta avviare l'app e attivare la geolocalizzazione. Durante il tragitto, l'app attraverso una semplice interfaccia permetterà di registrare le informazioni sulle buche o altre irregolarità che incontri lungo il percorso.

Le **segnalazioni** raccolte dagli **utenti** vengono quindi elaborate e **salvate** in un **database** per permettere poi una **visualizzazione** a 360 gradi delle problematiche presenti. Essa inoltre potrebbe in futuro notificare le autorità locali e gli enti responsabili delle strade per avere una visione chiara e dettagliata dei difetti presenti sulla rete stradale. Questo consente loro di intervenire tempestivamente, programmando la manutenzione e **migliorando** la **qualità** delle **strade** per tutti i cittadini.

L'applicazione offre anche una serie di funzionalità aggiuntive, come la possibilità di **visualizzare** le **segnalazioni** fatte da altri utenti e di tenere traccia dello **stato** delle **riparazioni** e la possibilità di aggiungere un *mi piace* ad una segnalazione. Questo permette agli **utenti** di essere **informati** sugli interventi in corso e sulle aree che sono state riparate, **migliorando** ulteriormente l'esperienza di **guida** e la **sicurezza** stradale.

L'obiettivo di BumpyRoads è quello di creare strade più sicure e confortevoli per tutti grazie alla **partecipazione** della **comunità**.
Inoltre, BumpyRoads è sviluppata utilizzando la tecnologia **React Native**, un framework *open-source* per lo sviluppo di applicazioni mobili.

2 Elenco feature presenti

L'applicazione prevede di segnalare e memorizzare difetti per le strade. L'applicazione ha le seguenti feature:

- Tramite GPS **individua** la **posizione** dell'utente permettendo di inviare segnalazioni indicando:
  - Foto
  - Posizione
  - Gravità
- fornisce agli utenti una **mappa interattiva** dei **difetti** segnalati nella loro zona:
  - strade malformate,
  - carenza di luce per le strade
  - lampioni danneggiati
  - spazzatura
  - altro (eventuali pericoli)
- Possibilità di avere **percorsi personalizzati** applicando filtri di:
  - selezione veicolo
  - presenza di pericoli citati nel punto precedente
- possibilità di visionare la **cronologia** dei difetti **segnalati** e inoltre:
  - vederle
  - eliminarle
- durante uno spostamento, il dispositivo **vibra** per indicare un **avvicinamento** a un possibile **difetto** stradale
- visionare indicazioni stradali durante il tragitto
- Sezione **profilo** personale con la possibilità di:
  - Visionare le segnalazioni effettuate
  - Visualizzare le segnalazioni risolte
  - Tenere traccia del numero di like totali
  - Possibilità di modificare:
    - username
    - immagine di profilo
    - email
    - password

- Vedere un **resoconto** della **segnalazione** visualizzando:
  - l'immagine della segnalazione
  - l'utente che ha effettuato la segnalazione
  - data e ora
  - il luogo della via segnalata
  - il difetto
  - la gravità
  - possibilità di mettere un like
  - bottone per marcarla come risolta
  - eliminare la segnalazione

3 Scelte progettuali

Sono state fatte diverse scelte progettuali nello sviluppo di questa applicazione le principali sono:

1. L'utilizzo di ***React Native***:
   - React native offre diversi vantaggi tra cui il **Cross-platform**: che permette di avere l'app disponibile su entrambe le principali piattaforme mobili, **iOS** e **Android**. Questo significa che *un'ampia gamma di utenti* può accedere all'app e contribuire a segnalare i difetti stradali, indipendentemente dal dispositivo utilizzato.
   - **Sviluppo rapido**: React Native permette di ridurre significativamente i tempi di sviluppo, grazie alla sua architettura basata su **componenti** e alla possibilità di condividere il codice tra le diverse piattaforme. Ciò si traduce in un rilascio più tempestivo di aggiornamenti e nuove funzionalità per gli utenti di BumpyRoads.
   - **Esperienza utente fluida**: React Native consente di creare un'interfaccia utente intuitiva e reattiva, garantendo un'**esperienza** di utilizzo **piacevole** e senza intoppi.
2. Utilizzo di ***NativeBase***
   - La libreria NativeBase offre una vasta gamma di **componenti UI** predefiniti che **semplificano** lo sviluppo dell'interfaccia utente dell'app. NativeBase sfrutta la sua raccolta di componenti personalizzabili come pulsanti, carte, moduli e altro ancora. Utilizzando questi componenti, è stata creata una **UI coerente** e ben progettata per le diverse schermate dell'app. NativeBase fornisce anche **opzioni** di stile **flessibili**, consentendo di **adattare** l'aspetto dei **componenti** secondo le esigenze di design.

3. Integrazione di ***Mapbox***

    - Mapbox è una libreria potente per l'integrazione di **mappe interattive** e funzionalità di geolocalizzazione. Essa consente agli utenti di visualizzare mappe, aggiungere marcatori, **calcolare percorsi** e interagire con la mappa. Utilizzando i componenti forniti da Mapbox è stato inoltre possibile **personalizzare** lo stile della **mappa**, impostare le coordinate iniziali e aggiungere **marcatori** o sovrapposizioni aggiuntive, consentendo agli utenti di interagire con la mappa in modo **intuitivo**.

4. Backend con ***Firebase***:

    - Firebase è stata utilizzata come piattaforma di backend per lo sviluppo dell'app, tra cui **autenticazione** degli utenti, **archiviare** e **sincronizzare** i dati in tempo reale, **storage** delle immagini. Firebase ha semplificato l'implementazione di funzionalità come la gestione dei dati consentendo uno sviluppo più rapido ed efficiente.

5. Uso di ***Typescript***:

    - Permette di individuare errori di tipo durante la fase di sviluppo, migliorando la **qualità** e l'**affidabilità** del codice poiché *Type Safety*. Esso inoltre aiuta nel **Refactoring** automatizzando il tutto. La **dichiarazione** esplicita dei **tipi** aiuta a comprendere meglio il codice e a prevenire errori di integrazione. Ed infine è **scalabile** poiché particolarmente adatto per progetti di grandi dimensioni o complessi, in quanto aiuta a gestire meglio la complessità del codice e a evitare errori comuni.

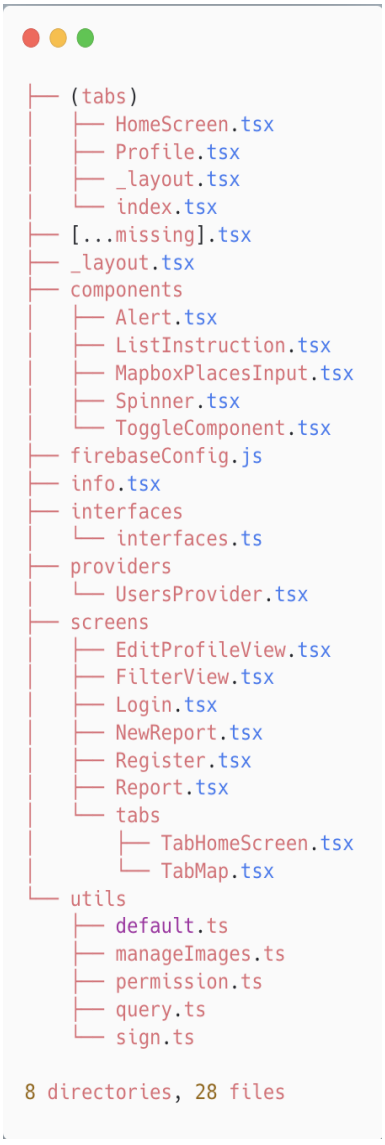
6. Strutturata in ***Tab***:

    - Per organizzare le diverse sezioni e funzionalità dell'app, è stata scelta una struttura a schede (Tab). Questo tipo di navigazione permette di **suddividere** le diverse **schermate** dell'app in schede separate, consentendo agli utenti di **passare facilmente** da una **scheda** all'altra. Tutto questo è stato realizzato usando la libreria *React Navigation*. Utilizzando il componente di navigazione delle schede, si sono definite le diverse schermate associate a ciascuna scheda. Nel mio caso è presente una scheda per la mappa, una scheda per creare nuovi report e una scheda per il profilo utente. L'utente può passare da una scheda all'altra semplicemente toccando la scheda corrispondente nella barra di navigazione inferiore.

3.1 Divisione e struttura del codice

L'organizzazione dei file e delle cartelle nel progetto segue un **pattern** *comune* per lo sviluppo di progetti *react*. Questo pattern aiuta a mantenere il codice **organizzato**, **facilitando** la navigazione e la manutenzione del **progetto**.

- La cartella **screens** contiene i componenti che rappresentano le diverse *schermate* dell'applicazione, come la schermata di accesso ("Login"), la schermata di registrazione ("Register") e la schermata di visualizzazione dei report ("Report"). All'interno della sottocartella "tabs" si trovano i componenti specifici per le schede dell'applicazione, come la schermata principale ("TabHomeScreen") e la mappa ("TabMap").
- La cartella *components* contiene i componenti riutilizzabili dell'applicazione, come gli **Alert**, l'**input** di posizione di Mapbox ("MapboxPlacesInput"), lo **spinner** di caricamento ("Spinner") e altri componenti. Questi componenti possono essere utilizzati in diverse schermate dell'applicazione per mantenere la coerenza nell'aspetto e nella funzionalità e favorire il riutilizzo di codice.
- La cartella *providers* contiene il file "UsersProvider.tsx", che rappresenta un provider di contesto per la gestione degli utenti. Questo provider è utilizzato per condividere i dati dell'utente *loggato*.
- La cartella *interfaces* contiene il file "interfaces.ts", che definisce le **interfacce** TypeScript utilizzate nel progetto. Queste interfacce aiutano a specificare la struttura dei dati e migliorano la **tipizzazione** del codice.
- La cartella *utils* contiene vari file di utilità, come "default.ts" per le **impostazioni** predefinite, "permission.ts" per la gestione dei **permessi** e "sign.ts" per le funzioni di **autenticazione**.



Questo pattern di organizzazione dei file e delle cartelle rende più facile la navigazione nel progetto e favorisce la **modularità** e la **riutilizzabilità** del codice, fornendo una struttura coerente e ben **organizzata**.

4 Analisi delle principali feature implementate

4.1 Divisione in tab

La struttura in tab utilizza la libreria react-navigation. Questa struttura consente di creare un'interfaccia utente con più schede (tab) nella parte inferiore dello schermo, consentendo all'utente di passare tra le diverse sezioni dell'applicazione.

```
<NavigationContainer independent={true} linking={config}
  fallback={<ActivityIndicator color="blue" size="large" />}>
  <Tab.Navigator screenOptions={({ route }) => ({
    tabBarIcon: ({ color }) => screenOptions(route, color),
  })}>
    {user ?
      <>
        <Tab.Screen name='Maps' component={TabMap} options={{ headerShown: false }}></Tab.Screen>
        <Tab.Screen name='NewReport' component={NewReport} options={{ headerShown: true }}>
        </Tab.Screen>
        <Tab.Screen name='Profile' component={Profile}></Tab.Screen>
      </>
      :
      <>
        <Tab.Screen name='Home' component={TabHomeScreen}></Tab.Screen>
      </>
    }
  </Tab.Navigator>
</NavigationContainer>
```

4.2 Rilevamento della posizione

L'applicazione utilizza un modulo di *expo* per ottenere la posizione dell'utente. La posizione viene quindi memorizzata nello stato *location*. Inoltre viene richiesta una grande precisione della localizzazione inserendo una *options* nel metodo *getCurrentPositionAsync*

```
export const getUserLocation = async (setLocation: React.Dispatch<React.SetStateAction<positionI>>,
setError: React.Dispatch<React.SetStateAction<string>>) => {
  let { status } = await Location.requestForegroundPermissionsAsync();
  if (status !== 'granted') {
    setError('Permission to access location was denied')
    return
  }
  const location = await Location.getCurrentPositionAsync({ accuracy: Location.Accuracy.Highest });
  const values: positionI = location.coords
  setLocation(values);
}
```

4.3 Mappa interattiva

Utilizzando la libreria *@rnmapbox/maps open-source* di **mapbox**, l'applicazione visualizza una mappa interattiva che mostra i difetti stradali segnalati nella zona dell'utente. Di seguito un breve *snippet* nel quale possiamo notare la necessità di un *TOKEN* per poter utilizzare la libreria.

```
import React from 'react';
import { StyleSheet, View } from 'react-
native';
import Mapbox from '@rnmapbox/maps';

Mapbox.setAccessToken('<YOUR_ACCESSTOKEN>');

const App = () => {
  return (
    <View style={styles.page}>
      <View style={styles.container}>
        <Mapbox.MapView style={styles.map} />
      </View>
    </View>
  );
}
```

4.4 Recupero di segnalazioni

I dettagli della segnalazione sono ricevuti da un database **Firebase** mediante la funzione *getSignalDefect*. Le segnalazioni sono state divise per difetto, e viene effettuata una ottimizzazione lato **render** utilizzando l'hook **useCallback**.

```
const getSignalDefect = useCallback(
  async () => {
    const data: SetStateAction<reportI[]> = []
    const querySnapshot = await getDocs(collection(db, "report"));
    querySnapshot.forEach((doc) => {
      data.push(doc.data() as reportI)
    });
    data.forEach(async (element) => {
      const image = ref(storage, `${element?.defect}/${element?.id}`)
      const imageURL = await getDownloadURL(image)
      element.imageURL = imageURL
    })
    setSavedDefect(data)
  }, [update])
```

[0]Nota: Nelle immagini del codice riguardo l'utilizzo della libreria di mapbox è stato utilizzato uno snippet presente nella documentazione ufficiale per non aggiungere troppo codice irrilevante.

4.5 Invio segnalazione

Le segnalazioni inviate dagli utenti vengono acquisiti dall'utente e inviati a un database Firebase mediante la funzione *sendSignalDefect*. La quale fa uso delle funzione *reverseGeocodeAsync* che permette di fare un **reverse** delle **coordinate** per ottenere un indirizzo.

```
const sendSignalDefect = async () => {
  if ((!image || !defect || !severity || !locationUser))
    return
  try {
    const address: AddressI = (await reverseGeocodeAsync(locationUser))[0] as AddressI
    let docRef = await addDoc(collection(db, "report"), {
      defect: defect,
      severity: severity,
      userUid: user.uid,
      createdAt: new Date().toString(),
      position: locationUser,
      like: [],
      address: address,
      isResolved: false
    });
    const imageURL = await uploadImage(image, `${defect}/${docRef.id}`) // upload
image setImage('')
    await setDoc(docRef, { id: docRef.id, imageURL: imageURL }, { merge: true })
    setSeverity('')
    setDefect('')
    props.navigation.navigate("Profile")
  } catch (e) {
    console.error("Error adding document: ", e);
  }
}
```

4.6 Vibrazione durante lo spostamento

VibrateOnNearReport viene utilizzata per controllare se l'utente si **avvicina** a un difetto stradale. Se l'utente si trova vicino a un difetto, viene attivata la vibrazione del dispositivo. Questa funzionalità ha necessitato di un calcolo accurato per verificare se il presunto difetto è abbastanza vicino.

```
const vibrateOnNearReport = () => {
  savedDefect.forEach(report => {
    if ((Math.abs(report.position.longitude - location.longitude) < 0.00008)
      && (Math.abs(report.position.latitude - location.latitude) < 0.000008))
{     Vibration.vibrate()
    }
  });
}
```

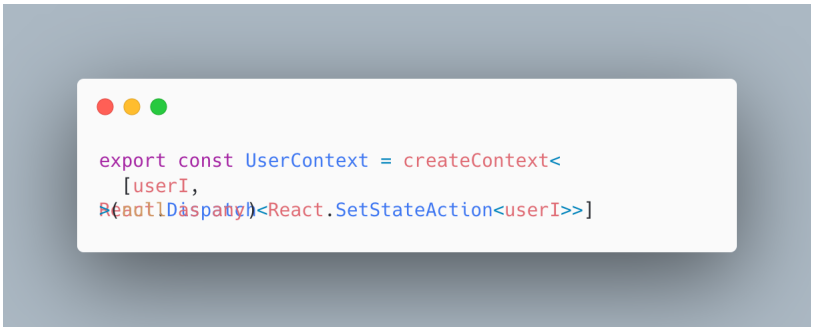
4.7 Filtri personalizzati per i percorsi

L'applicazione consente agli utenti di selezionare un veicolo e di applicare filtri per evitare determinati difetti stradali. I filtri sono rappresentati nello stato *avoidPoints*, che viene aggiornato tramite l'interfaccia utente. Quando l'utente avvia il percorso, la funzione *fetchRoute* utilizza i filtri selezionati per calcolare la **rotta ottimale**.

```
const fetchRoute = async () => {
  const avoidPoints: positionI[] = [{ longitude: 0, latitude: 0 }]
  Object.entries(avoid).map(([key, value]) => {
    if (value) {
      savedDefect.map(defect => {
        if (defect.defect == key) {
          avoidPoints.push({ longitude: defect.position.longitude, latitude: defect.position.latitude
          })
        }
      })
    }
  })
  const points = avoidPoints?.map(location => `point(${location.longitude} ${location.latitude})`);
  const reqOptions = {
    waypoints: [
      { coordinates: origin },
      { coordinates: destination },
    ],
    profile: vehicle,
    geometries: 'geojson',
    exclude: points.toString()
  };
  const res = await directionsClient.getDirections(reqOptions).send();
  const newRoute = makeLineString(res.body.routes[0].geometry.coordinates);
  setRoute(newRoute);
}
```

4.8 UserContext

UserContext viene definito come un contesto che memorizza una tupla contenente i valori dello *state* per aggiornare i dati dell'utente. Esso è molto utile per condividere i dati dell'utente *loggato*, consentendo loro di accedere e modificare i dati dell'utente senza dover passare esplicitamente le *props* attraverso la gerarchia dei componenti.



```
export const UserContext = createContext<
  [userI,
React.Dispatch<React.SetStateAction<userI>>]
```

4.9 Report like

La funzione *userLike* implementa la logica per gestire l'azione di mettere o togliere like a un report. Viene utilizzata una struttura dati (*Set*) per rimuovere eventuali duplicati dall'array dei like.

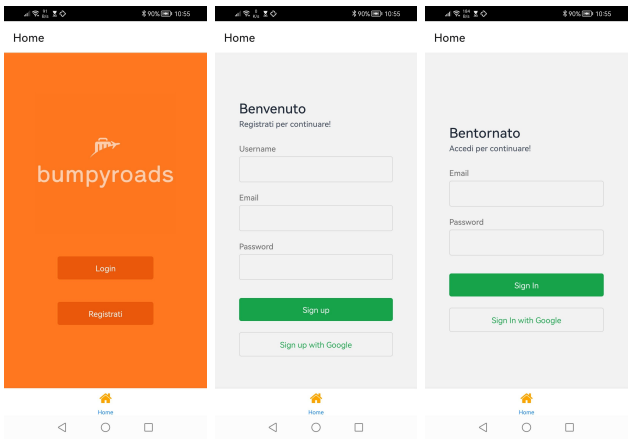
```
  const userLike = async (isLikedPost: boolean) => {
    try {
      let newLikes: string[] = reportData?.like!
      if (isLikedPost && !isLike) {
        setIsLike(true)
        setTotalLike(totalLike + 1)
        newLikes?.push(user.uid!)
      } else {
        if (newLikes.includes(user.uid!) && isLike) {
          setIsLike(false)
          newLikes = newLikes.filter(report => report !==
user.uid!)setTotalLike(totalLike - 1)
        }
      }
      const uniqueLikes = [...new Set(newLikes)];
      await updateDoc(doc(db, "report", reportData?.id!), {
        like: uniqueLikes
      });
    } catch (error) {
      console.log(error)
    }
  }
```

5 Screenshot interfaccia grafica

L'utilizzo di *nativeBase* (libreria UI) ha aiutato notevolmente durante lo sviluppo di bumpyroads per avere un tema coerente e privo di bug. Di seguito sono mostrate le pagine implementate nell'applicazione:

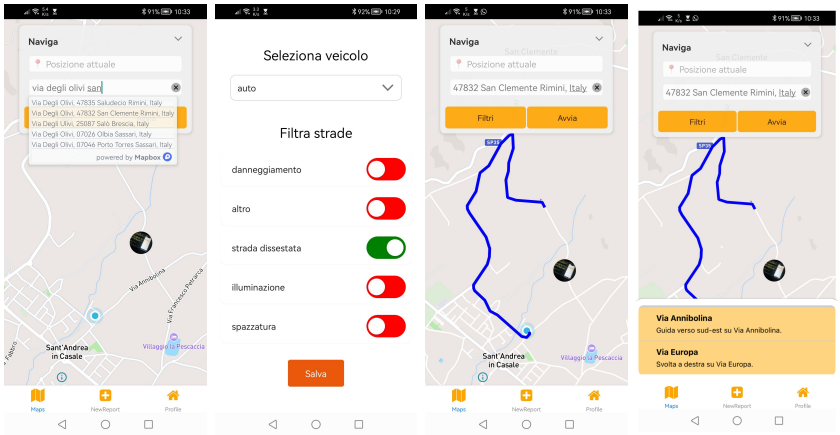
5.1 Interfaccia di avvio

La schermata di avvio si presenta facendo un login o una registrazione, la quale ha la possibilità di collegarsi ad un account google per rendere il tutto ancora più rapido.



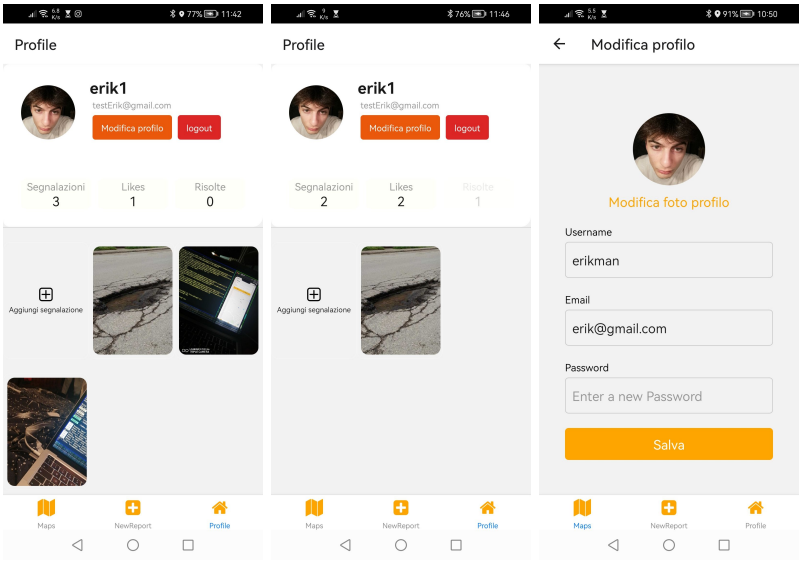
5.2 Index dell'applicazione

Nella schermata seguente schermate è possibile notare che grazie al filtro applicato, l'app segnala un percorso differente da quello di lunghezza ottimale.



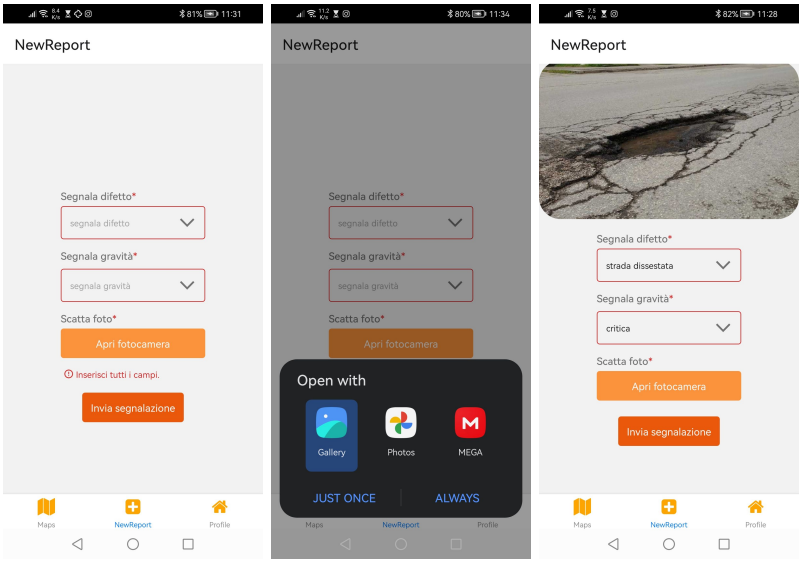
5.3 Sezione Profilo

La sezione del profilo è formata dalla sezione *Segnalazioni* dove sono presenti tutti i difetti dell'utente **non** risolti, mentre nella sezione *risolte* sono presenti quelle non più problematiche.



5.4 Schermata Nuovo Report

L'interfaccia presentata è molto semplice, e permette il caricamento della segnalazione in pochi e semplici passi.



5.5 Interfaccia Report

La panoramica del report caricata è data da questa pagina, la quale permette di visionare tutte le informazioni utili, la possibilità di inserire un *like* alla segnalazione e contrassegnarla come *risolta*.