

DOKUMENTACJA

aplikacja mobilna

UrzER - wypożyczalnia samochodów
w stylu Ubera

Informatyka III, lab. 3

Patryk Krawiec

Filip Konior

Łukasz Kowalski

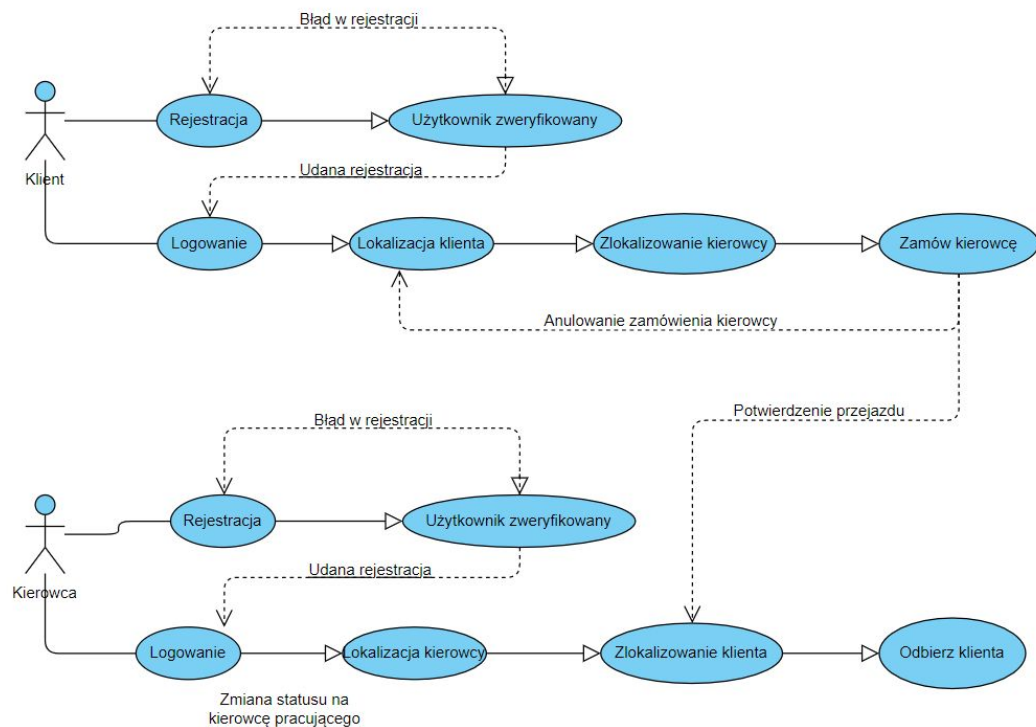
1. Cele projektu

Celem naszego projektu było stworzenie aplikacji mobilnej na system android, która umożliwia użytkownikowi (klientowi) zamówienie UrzER'a (kierowcy).

2. Wykorzystane technologie

Aplikację stworzyliśmy w środowisku programistycznym Android Studio. Logiczna część aplikacji napisana została w języku Java, natomiast część graficzna została zaimplementowana w plikach XML. Jako bazy danych użyliśmy FireBase oraz takich dodatków jak Google Maps SDK oraz GeoFire, który pozwolił nam m. in. przechowywać lokalizację użytkowników aplikacji.

3. Diagram przypadków użycia



4. Opis projektu

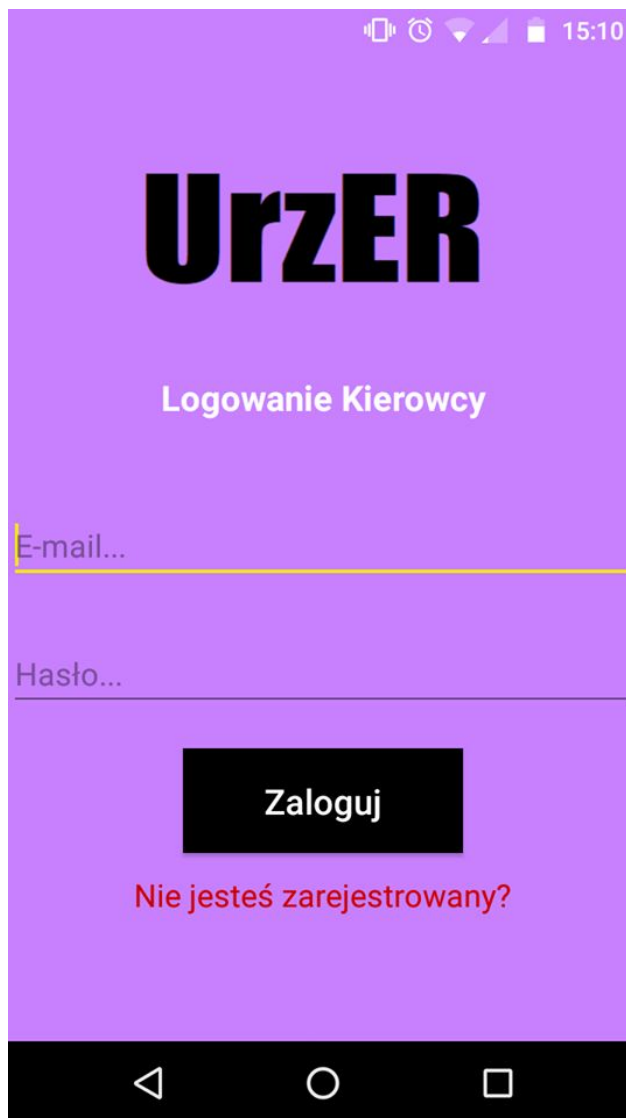
Aplikacja składa się z 6 activities, po uruchomieniu aplikacji, wita nas ona oknem wyboru swojego typu użytkownika.

Ekran główny (MainActivity)



W tym kroku użytkownik aplikacji wybiera czy jest/chciałby zostać Kierowcą UrZER'a, czy skorzystać z usługi jako Klient.

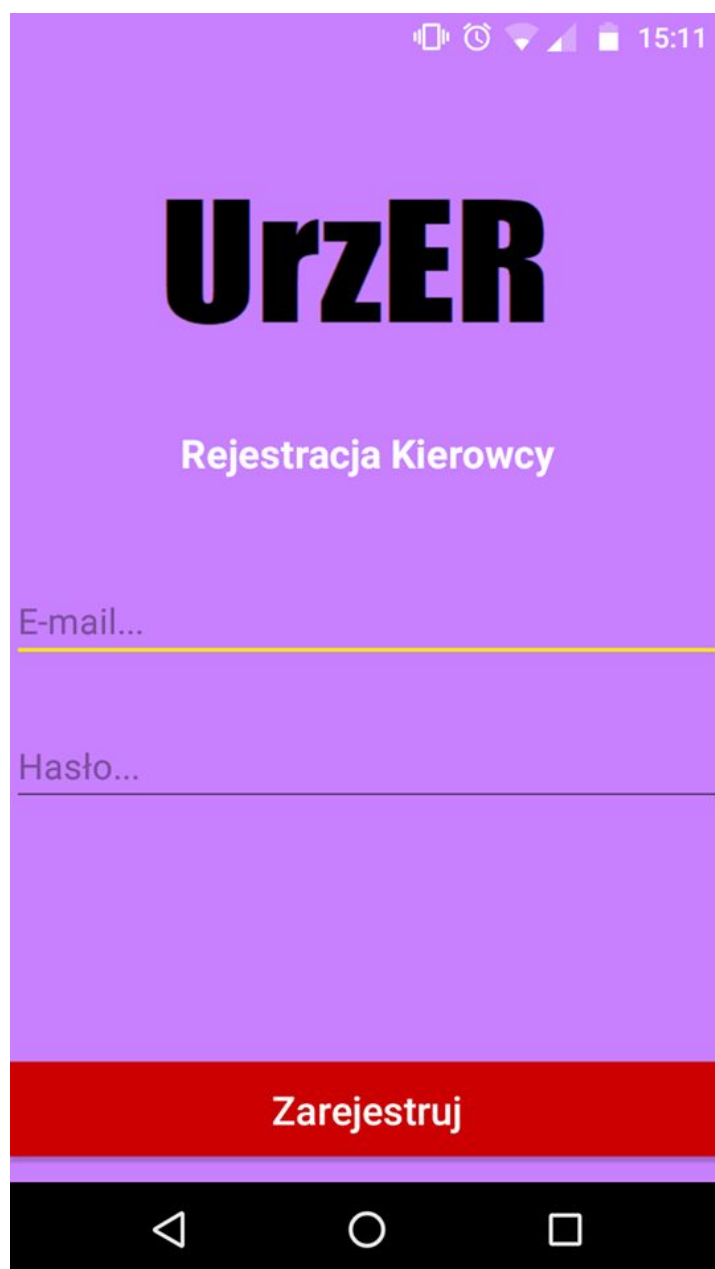
Jestem Kierowcą (DriverLoginRegisterActivity)



The screenshot shows a mobile application interface with a purple background. At the top, the status bar displays various icons and the time 15:10. The app's logo, 'UrzER', is prominently displayed in large, bold, black letters. Below the logo, the text 'Logowanie Kierowcy' (Driver Login) is centered in white. There are two input fields: the first is labeled 'E-mail...' and the second is labeled 'Hasło...' (Password...). Below these fields is a black button with the white text 'Zaloguj' (Login). At the bottom of the form area, there is a red link that says 'Nie jesteś zarejestrowany?' (Are you not registered?). The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

Deklarując się, że jesteśmy kierowcą musimy się zalogować do naszego uprzednio stworzonego konta. Jeżeli jesteśmy nowymi użytkownikami UrzER'a - klikamy "Nie jesteś zarejestrowany?".

Nie jesteś zarejestrowany? (DriverLoginRegisterActivity)



UrZER

Rejestracja Kierowcy

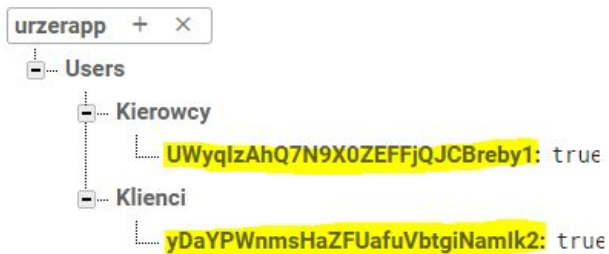
E-mail...

Hasło...

Zarejestruj

Wynikowo pokaże nam się okno z panelem rejestracyjnym. Użytkownik rejestrując się walidowanym e-mailem i hasłem jest wpisywany do naszej bazy danych - Firebase.

Wyszukaj według adresu e-mail, numeru telefonu lub identyfikatora UID użytkownika				Dodaj użytkownika	↺	⋮
Identyfikator	Dostawcy	Utworzono	Zalogowano	Identyfikator UID użytkownika ↑		
test@gmail.com	✉	5 sty 2019	6 sty 2019	aqOvtYkAjfhK72WklwkM7e7WEF...		
test2@gmail.com	✉	5 sty 2019	6 sty 2019	yDaYPWnmsHaZFUafuVbtgiNamIk2		
Liczba wierszy na stronę: 50 1-2 z 2 < >						



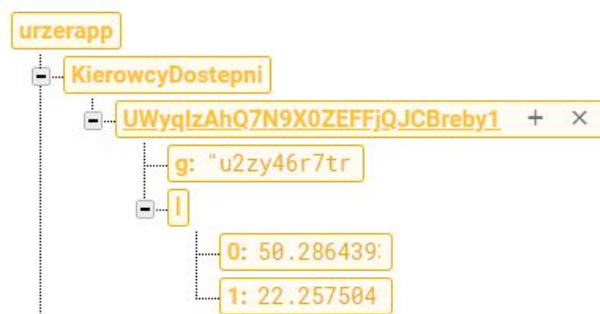
Identyfikatory (ID) - podświetlone żółtym kolorem

Zarejestrowany? Zalogowany? (DriverMapActivity)



Po pomyślnym procesie rejestracji/logowania Kierowca zostaje przeniesiony do okna z mapą. Zostaje zlokalizowana jego pozycja i jest zaznaczona na mapie.

Kierowca staje się “Kierowcą Dostępnym”. Do dyspozycji w interfejsie ma przycisk “WYLOGUJ”, który zlikwiduje jego dostępność. Lokalizacja geograficzna kierowcy odświeżana jest co 1 sek.



Jestem Klientem (CustomerLoginRegisterActivity)

UrzER

Logowanie Klienta

E-mail...

Hasło...

Zaloguj

Nie jesteś zarejestrowany?

Deklarując się, że jesteśmy Klientem musimy się zalogować do naszego uprzednio stworzonego konta. Jeżeli jesteśmy nowymi użytkownikami UrzER'a - klikamy "Nie jesteś zarejestrowany?".

Nie jesteś zarejestrowany? (CustomerLoginRegisterActivity)



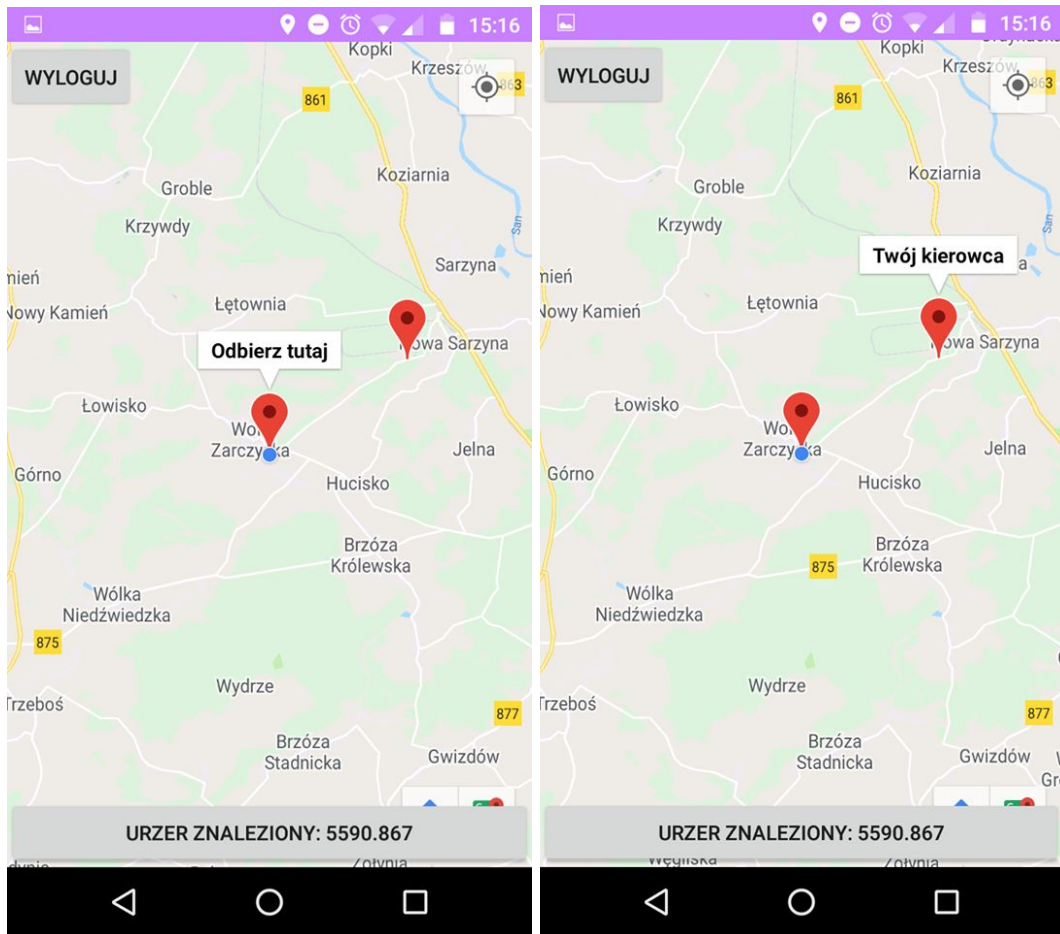
The screenshot shows a mobile application interface for registration. At the top, the status bar displays various icons and the time 15:11. The app's logo, 'UrZER', is prominently displayed in a large, bold, black font. Below the logo, the text 'Rejestracja Klienta' (Customer Registration) is centered. There are two input fields: the first is labeled 'E-mail...' and the second is labeled 'Hasło...' (Password...). Both fields have a light blue border. At the bottom of the form, there is a red button with the text 'Zarejestruj' (Register) in white. The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps icons.

Podobnie jak przy rejestracji kierowcy, wynikowo pokaże nam się okno z panelem rejestracyjnym. Użytkownik rejestrując się walidowanym e-mailem i hasłem jest wpisywany do naszej bazy danych - Firebase.

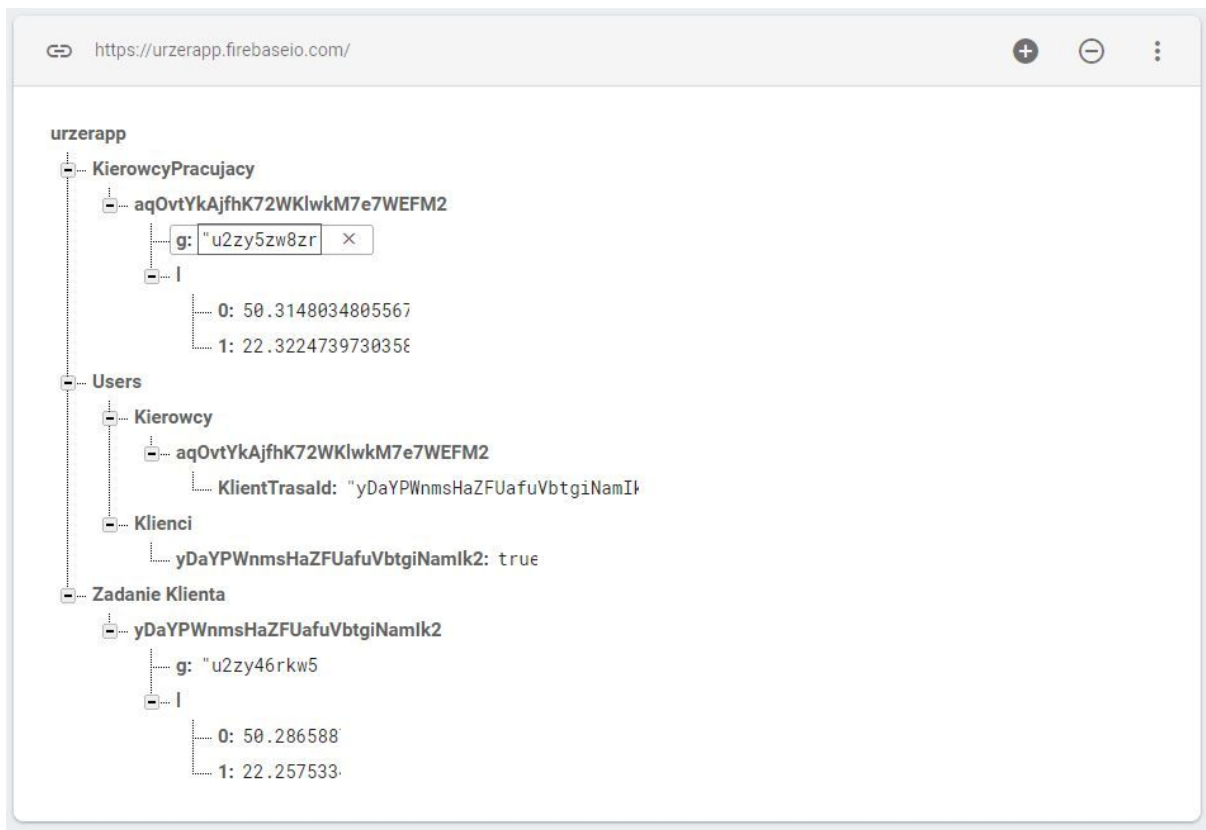
Zarejestrowany? Zalogowany? (CustomerMapActivity)



Po pomyślnym procesie rejestracji/logowania Klient zostaje przeniesiony do okna z mapą. Zostaje zlokalizowana jego pozycja i jest zaznaczona na mapie. Klient ma możliwość zamówienia UrzER'a, po kliknięciu w przycisk - aplikacja zacznie szukać lokalizacji najbliższego dostępnego Kierowcy w okolicy. Dolny przycisk zmieni swoje aktywności. Do dyspozycji w interfejsie Klient ma również przycisk "WYLOGUJ", który przeniesie Klienta do MainActivity.



Jeżeli UrzER (Kierowca) zostanie zlokalizowany jako “Dostępny”, zmieni on swoją aktywność w bazie na “Kierowca Pracujący”. Klient w swoim oknie aplikacji zobaczy dwa znaczniki określające jego położenie oraz położenie jego kierowcy. Dodatkowo na przycisku zostanie wyświetlona informacja w jakiej odległości od Klienta jest UrzER.



W bazie danych pojawiły się “Żądania klienta” czyli “request”, który wysyła klient do bazy jeżeli potrzebuje UrzER’a, określone są w nim: ID Klienta, lokalizacja.

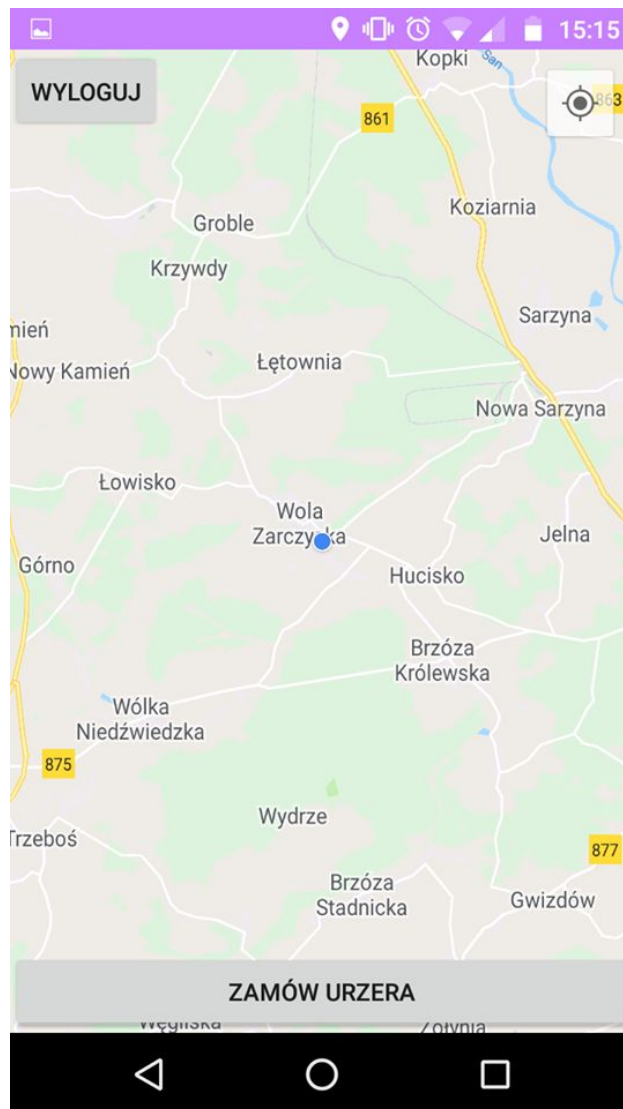
Kierowca zmienił swoją aktywność z “Kierowca Dostępny” na “Kierowca Pracujący”, dodatkowo w encji Kierowcy pojawiła się “KlientTrasald”.



Jeżeli Klient zamówił UrzER'a i aplikacja przypisała go do Kierowcy, to Kierowca wynikowo zobaczy mapę ze znacznikiem określającym położenie Klienta.



Kiedy Kierowca dotrze do Klienta, Klientowi wyskoczy komunikat, że jego kierowca jest na miejscu.



W przypadku gdy Klient zrezygnuje z usługi Kierowcy albo podróż już się zakończy to zostaje mu kliknąć w przycisk na dole aplikacji gdzie wyświetlane były komunikaty. Wynikowo Klient może zamówić UrzER'a ponownie, a kierowca zmienia swoją aktywność z "Kierowca Pracujący" na "Kierowca Dostępny" - zostaje przerwana usługa między nimi.

5. Wyjaśnienie fragmentów kodu

```
private void SignInDriver(String email, String password)
{
    if(TextUtils.isEmpty(email))//Warunek jeżeli nie wpisał maila
    {
        Toast.makeText( context: DriverLoginRegisterActivity.this, text "Wpisz e-mail...", Toast.LENGTH_SHORT).show();
    }
    if(TextUtils.isEmpty(password))//Warunek jeżeli nie wpisał maila
    {
        Toast.makeText( context: DriverLoginRegisterActivity.this, text "Wpisz hasło...", Toast.LENGTH_SHORT).show();
    }
    else //Jeżeli nie puste - logowanie
    {
        loadingBar.setTitle("Logowanie Kierowcy");
        loadingBar.setMessage("Proszę czekać, logowanie trwa...");
        loadingBar.show();

        mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(task -> {
                if(task.isSuccessful()) //Jeżeli zostało authenticated
                {
                    Intent driverIntent = new Intent( packageContext: DriverLoginRegisterActivity.this, DriverMapActivity.class);
                    startActivity(driverIntent);

                    Toast.makeText( context: DriverLoginRegisterActivity.this, text "Logowanie pomyślne", Toast.LENGTH_SHORT).show();
                    loadingBar.dismiss();
                }
                else
                {
                    Toast.makeText( context: DriverLoginRegisterActivity.this, text "Logowanie niepowodzenie", Toast.LENGTH_SHORT).show();
                    loadingBar.dismiss();
                }
            });
    }
}
```

Metoda “SignInDriver”, to metoda logowania Kierowcy.

Następuje w niej walidacja logowania Kierowcy, znajduje się w niej implementacja loading bar’u, który wyświetla się użytkownikowi podczas logowania.

Skorzystaliśmy

z Firebase Authentication tworząc listener, który w pętli sprawdza sukcesywność logowania. Ta metoda jest podobna do metod, które wykorzystaliśmy w procesach rejestracji użytkowników i logowania Klienta.

```

mLogout = (Button) findViewById(R.id.logout);
mLogout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        isLoggingOut = true;

        disconnectDriver();

        FirebaseAuth.getInstance().signOut();
        Intent intent = new Intent( packageContext, DriverMapActivity.this, MainActivity.class);
        startActivity(intent);
        finish();
        return;
    }
});

```

```

private void disconnectDriver(){
    LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient, locationListener: this);
    String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
    DatabaseReference ref = FirebaseDatabase.getInstance().getReference( path: "KierowcyDostepni");

    GeoFire geoFire = new GeoFire(ref);
    geoFire.removeLocation(userId);
}

@Override
protected void onStop() { //metoda onStop - potrzebna, bo kiedy driver sie rozlaczy to usuwamy jego lokalizacje, nie jest dostepny
    super.onStop();

    if (!isLoggingOut){
        disconnectDriver();
    }
}
}

```

Ustawione zostało w OnClick'u na przycisku wyloguj wywołanie metody "disconnectDriver", która to rozłączy naszego kierowcę, czyli usunie wszystko związane z jego aktualną lokalizacją oraz z bazy danych zostanie wyrzucony z encji "Kierowcy Dostępni".

```

    getAssignedCustomer();
}
private void getAssignedCustomer()
{
    String driverId = FirebaseAuth.getInstance().getCurrentUser().getUid();
    DatabaseReference assignedCustomerRef = FirebaseDatabase.getInstance().getReference().child("Users").child("Kierowcy").child(driverId).child("KlientTrasaId");
    assignedCustomerRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists())
            {
                customerId = dataSnapshot.getValue().toString();
                getAssignedCustomerPickupLocation();
            }else{ //ten else bedzie sie wykonywal za kazdym razem gdy klienttrasaid zostanie usuniete spod rodzica
                customerId = "";
                if(pickupMarker != null){
                    pickupMarker.remove();
                }
                if(assignedCustomerPickupLocationRefListener != null) {
                    assignedCustomerPickupLocationRef.removeEventListener(assignedCustomerPickupLocationRefListener);
                }
            }
        }
    });
}

```

Najpierw w zmiennej typu string zapisywany jest pobrany z bazy ID Kierowcy. Dalej wykonuje się referencja bazy danych, czyli tworzy się encja “dziecko” - “KlientTrasaId”. Do tej encji będziemy wpisywać customerId, które jest zapisane w Listenerze we wcześniej wspomnianej Referencji w pętli if. Za każdym razem jak chcemy “przeczytać” dane z bazy danych otrzymujemy je jako dataSnapshot. Jeżeli owe dataSnapshot istnieją (bez tego warunku aplikacja się crashowała) to wykonują się instrukcje pętli - do customerId przypisujemy stringa, wywołuje się metoda gACPL. Jeżeli nie spełni się warunek pierwszego if’a to customerId staje się pusty, znacznik na mapie się usuwa. Jeżeli nie ma lokalizacji odbioru klienta to nie wykonują się żadne Listenerzy.

```

Marker pickupMarker;
private DatabaseReference assignedCustomerPickupLocationRef;
private ValueEventListener assignedCustomerPickupLocationRefListener;
private void getAssignedCustomerPickupLocation()
{
    assignedCustomerPickupLocationRef = FirebaseDatabase.getInstance().getReference().child("Zadanie Klienta").child(customerId).child("l");
    assignedCustomerPickupLocationRefListener = assignedCustomerPickupLocationRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists() && !customerId.equals(""))
            {
                List<Object> map = (List<Object>) dataSnapshot.getValue();
                double locationLat = 0;
                double locationLng = 0;
                if(map.get(0) != null) //jak bd 0 to sie zcrashuje wiec niezbedny warunek
                {
                    locationLat = Double.parseDouble(map.get(0).toString());
                }
                if(map.get(1) != null)
                {
                    locationLng = Double.parseDouble(map.get(1).toString());
                }
                LatLng driverLatLng = new LatLng(locationLat, locationLng);
                pickupMarker = mMap.addMarker(new MarkerOptions().position(driverLatLng).title("Lokalizacja odbioru"));
            }
        }
    });
}

```

Na początku tworzona jest globalna instancja referencji bazy, bo będziemy pobierać z niej lokalizację (jest ona pod oznaczeniem "l" - latitude, longitude) oraz globalna instancja ValueEventListener - nasza metoda implementuje ten interfejs i użyje go do odbierania zdarzeń dot. zmian danych w lokalizacji. W metodzie onDataChange w pętli if wykonuje się: tworzy listę obiektów mapy, pobiera je jako dataSnapshot'y. Tworzy dwie zmienne locationLat, locationLng, które będą odpowiadały za współrzędne (szerokość, długość geograficzna). Dalej wykonuje się if jeżeli lat nie jest nullem i lng nie jest nullem to do zmiennych locLat, locLng przypisujemy wartości w stringu pobrane z listy obiektów map. Dalej tworzony jest obiekt LatLng i na jego

podstawie na mapie tworzy się znacznik “Lokalizacja odbioru”.

```
public void onLocationChanged(Location location) {
    if(getApplicationContext() != null) {
        mLastLocation = location;

        LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());

        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng)); //przesuwanie kamery za userem
        mMap.animateCamera(CameraUpdateFactory.zoomTo( 11));

        //dodawanie lokalizacji do bazy
        String userId = FirebaseAuth.getInstance().getCurrentUser().getUid(); //pobierze id zalogowanego
        DatabaseReference refAvailable = FirebaseDatabase.getInstance().getReference( path: "KierowcyDostepni");
        DatabaseReference refWorking = FirebaseDatabase.getInstance().getReference( path: "KierowcyPracujacy");
        GeoFire geoFireAvailable = new GeoFire(refAvailable);
        GeoFire geoFireWorking = new GeoFire(refWorking);
        switch (customerId){
            case "": //nie ma zadnych klientow, kierowca nie moze wykonac pracy
                geoFireWorking.removeLocation(userId);
                geoFireAvailable.setLocation(userId, new GeoLocation(location.getLatitude(), location.getLongitude()));
                break;

            default:
                geoFireAvailable.removeLocation(userId);
                geoFireWorking.setLocation(userId, new GeoLocation(location.getLatitude(), location.getLongitude()));
                break;
        }
    }
}
```

Do zmiennej `mLastLocation` przypisywana jest `location` pobrana wraz z wywołaniem metody. Tworzona jest zmienna `latLng` określająca położenie. Dalej określone zostały: poruszanie kamery zgodnie z wartościami w zmiennej `latLng` oraz zoom kamery ustawiony na 11 w skali do 21 jednostek (chyba). Do zmiennej `userId` przypisane zostało ID zalogowanego Kierowcy. Dalej instancje referencji dot. Kierowców Pracujących i Dostępnych. GeoFire - API, które przechowuje w swoim sercu lokalizację za pomocą kluczy w stringu, opiera się ono o FireBase'a, możemy z niego “wyciągać” i do niego “wkładać” klucze w czasie rzeczywistym. Tworzymy dwa obiekty GeoFire na podstawie dwóch referencji. Korzystając ze switcha, jeżeli

nie ma żadnego klienta usuwana jest z lokalizacja Kierowcy Pracującego i usuwana z bazy danych oraz ustawiana jest lokalizacja Kierowcy Pracującego i dodawana do bazy. W przypadku instrukcji switcha “default” zachodzą odwrotne czynności.