Introduction

In Egypt, there are many people who travel daily, and their work may be in a governorate which they don't live in. Accordingly, it is necessary to have a railway linking these governorates to facilitate movement within the country. But sometimes people face a problem locating the trains and this leads to wasting a lot of time waiting and worrying about where it is now. Therefore, we thought in our graduation project to solve the problem of not knowing the location of the train and have mercy on people from the tension they are experiencing about their location, which stations are coming, and which trains pass now and will reach the desired destination and other services. TrainDar is our graduation project and the solution, it depends on the user to know the location of the trains. TrainDar is a phone application that the user can download and a site that the user can deal with. This site works on all devices and adapts to them. In this document, we present some of the details of TrainDar like:

a) **The problem.**
b) **The TrainDar services.**
c) **Technologies.**
d) **UI-UX design.**
e) **User stories.**
f) **ERD analysis.**
g) **Data preparing.**
h) **Main functions implementation.**
i) **Hosting.**
j) **Expected behavior scenarios.**
k) **Simulation and Test.**

We wish you a great browsing …

The problem

There are many people who travel many times and often they cannot know where the train is or even predict where it is. This leads to waiting at train stations and not knowing how long the train needs to reach the station. One of the problems that people can face is being late for an already late train! In addition to worrying about not knowing where I am when I am traveling!

On a larger scale, officials can run TrainDar in the cargo transportation sector to know when it will arrive and what its expected location is now. Officials can also make statistics that indicate whether the trains arrive on time or not.

Finally, the main problem which the project revolves is the delay of trains from their official times, which leads to many problems in managing them and wasting people's time.

To make sure that people suffer from these problems, we made a survey and published this questionnaire on the social networking site Facebook, to reach the largest number of people participating and to cover more than one layer. A very large percentage of the answers included the train being late, not knowing where it was, wasting time, worrying about knowing the next station and most of the people liked to know which trains are entering a particular station … It is worth noting that Google Maps does not contain directions for the Egyptian railways.

Finally, we asked people if they thought that this problem needs a solution and effort to solve it, and we found more than 245 answer was "Yes, it is a problem" and only 11 answer was "No, it is not a problem"

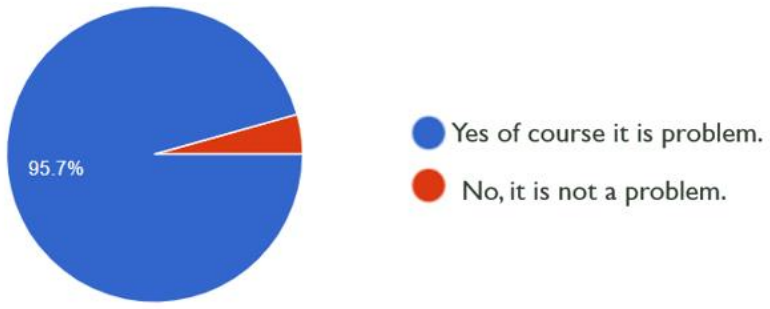


Figure 1: the answer for it is problem or not

The TrainDar services

TrainDar gives the clear answer to the user for the questions of what is the expected train location right now? which trains are entering a specific station? what is the next station? and what are the nearest stations?

As we mentioned in introduction section that the users is the core of our project because for determine the expected location of the train, each passenger share the location of his train. Then, TrainDar filters locations and ignores fake ones (out of range of railway or outlier). Then, TrainDar expects every train location and then creates a map for each train when the user searches. Before talking about our features and its mechanism, we will talk about TrainDar points system. So, what is TrainDar points system?

TrainDar points system

To encourage people to use and follow our app, we have created a points system that goes up and down according to user behavior and what he does in TrainDar. The main question is when does user get points?

The user gets points in these situations:

a) First Login.
b) Sharing TrainDar.
c) Sharing a correct train location.
d) Points sent from another user.
e) Buying points.

Another main question is when does the user lose points?

The user loses points in these situations:

a) Using TrainDar features.
b) Sharing a fake location.
c) Sending points.

TrainDar features

Due to the lack of time, we could not finish everything that was on our minds, we did the main features only:

a) Search for a train.
b) Check upcoming trains.
c) Check out the nearest stations

a- Search for a train

The user chooses the train ID from the dropdown list, then click search. TrainDar responses by a map that shows the expected location for a train.

Note: the list contains the active trains only, and active train means that the train has a shared user.

b- Check upcoming trains

This feature helps the user to know which trains will arrive the start station and will stop at the destination station. So, user will choose two stations from the dropdown list, then TrainDar response with a table that contains the trains and the estimated time for each one to arrive to the station.

Note: if user wants to know where the train is, he just has to choose the train and press a button then the TrainDar will create the map for selected train.

c- Check out the nearest stations

This feature helps the user to know which stations the chosen train pass will throw and its estimated time to arrive for each one.

The technologies

After researching and taking the opinion of professionals, we decided to do our ui-ux design using adobe XD, our Back-end using Java and spring boot framework, our Front-end using angular typescript, and our mobile app using Flutter platform.

UI-UX design

We took into our account that the user interface should be simple for ease of use and quick access to results. In addition to the colors to comfort the eye and not be confused.

Website ui-ux design

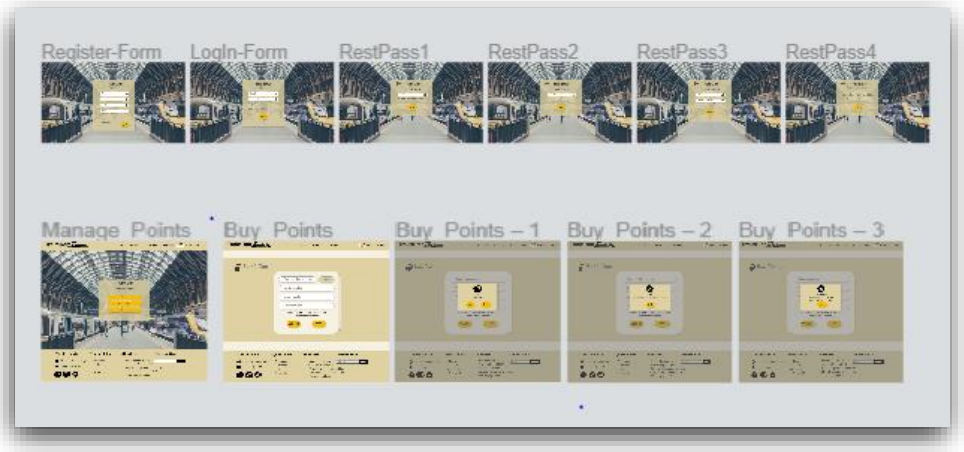


Figure 2: ui-ux for web points and login screens

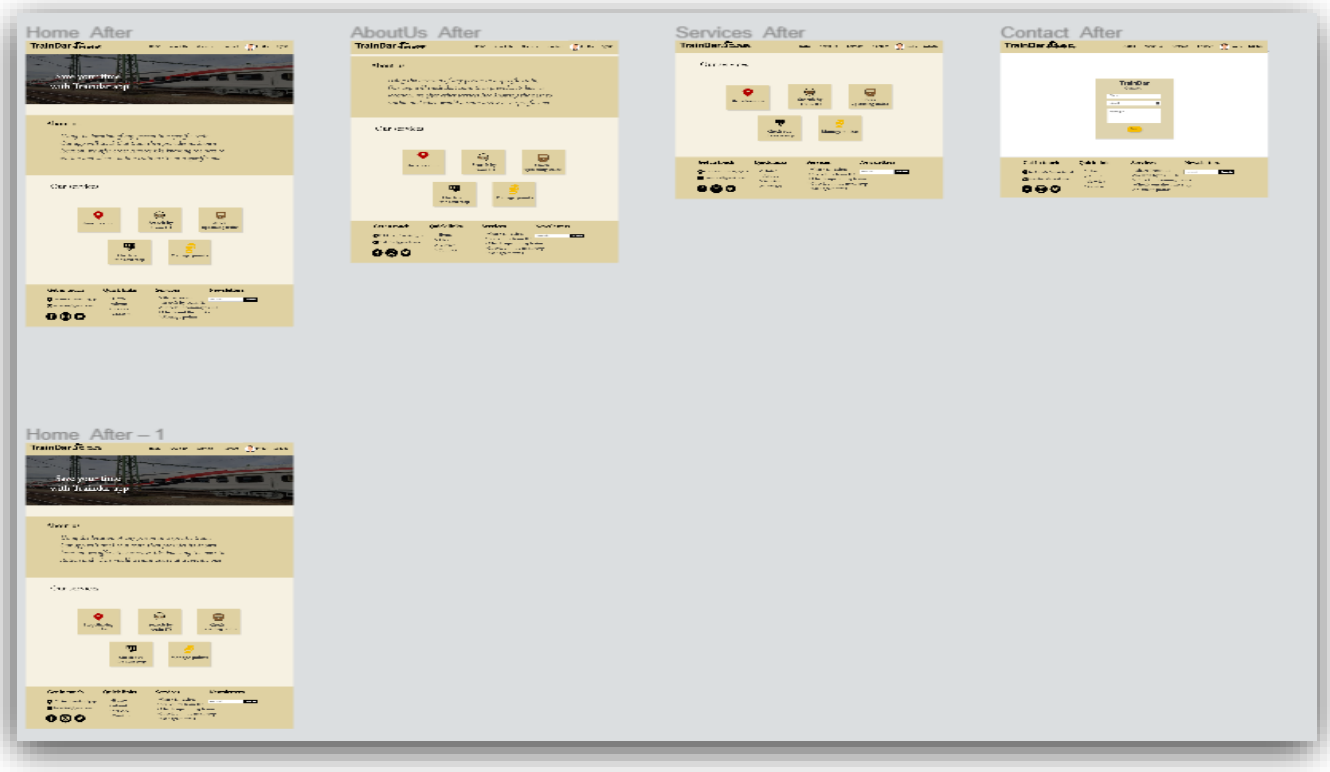


Figure 3: after login

What about mobile app?...

Mobile app ui-ux design

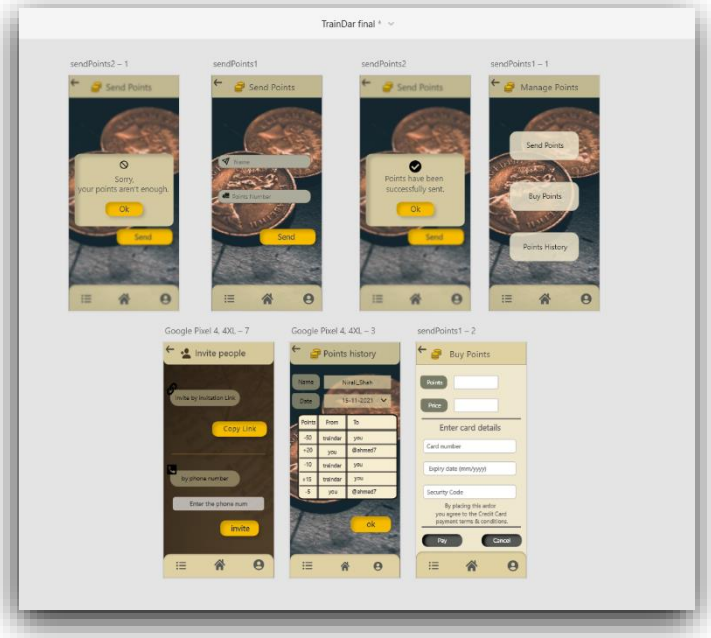


Figure 6: points management mobile app ui-ux design

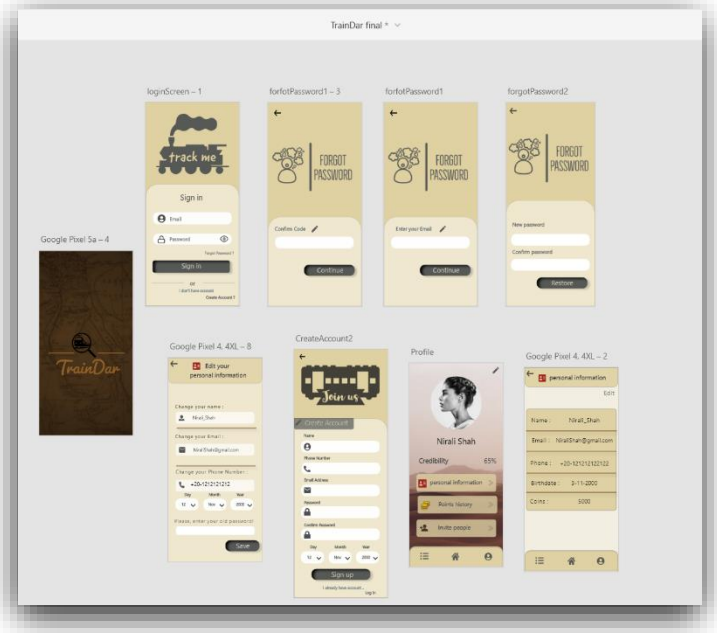


Figure 7: profile mobile ui-ux design

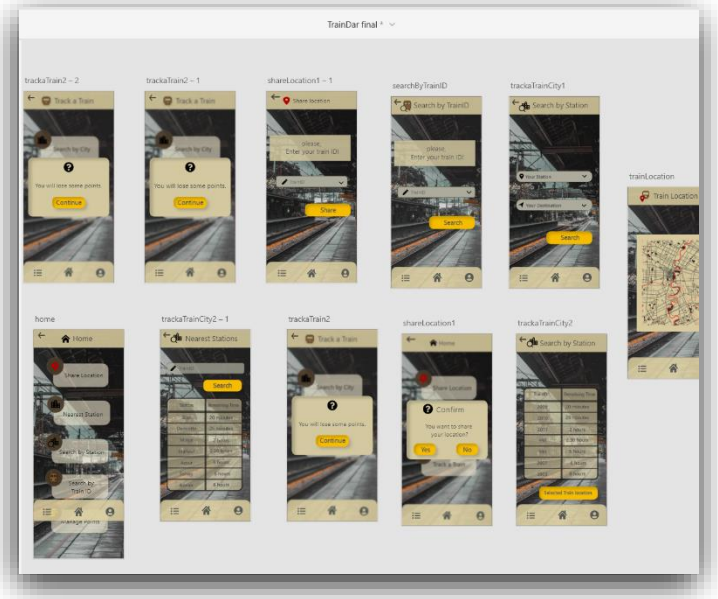


Figure 5:features mobile app ui-ux design

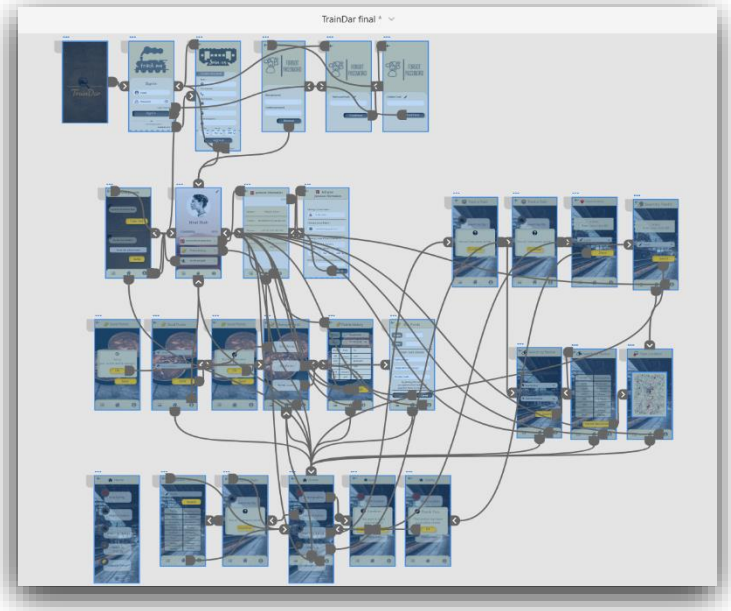


Figure 4: transitions in ui-ux mobile using adobe xd

User Stories

Some of scenarios that will be happen…

1-As a new user,

I want to be able to create a new account with my details and gain some points.

So that: I could reuse my details to log in afterwards.

Scenario: the new user signs up for a new account with a new username and password.

Given**:**" I am a new user, and I am on the sign-up page ".

When**:** I fill in the details and a valid username and password and click the sign-up button.

Then**:** I have created a new account and can use the created username and password afterwards and gain some points.

2-As a logged-out user,

I want to be able to sign into the website.

So that: I can log in my account.

Scenario**:** the registered user signs in with a valid username and password.

Given: I am a logged-out user from the website.

When: I fill in the username and password fields with my valid credentials and I click the sign-in button.

Then: I signed into my account.

3-As a logged in user,

I want to be able to share my location when I am on a specific train.

So that: I can help others and gain some points over time.

Scenario: the logged in user wants to share his location while riding a train.

Given: I am a logged in user, and I am on the home page.

When: I click on share my location and choose the train number from a specific list.

Then: I would be able to give out my current location and gain points over time.

5-As a logged in user,

I want to be able to know which trains are coming to specific station.

So that: I could know trains details that are coming.

Scenario: the logged in user wants to see which trains are near.

Given: I am a logged in user, I am on the home screen.

When: I click on check upcoming trains button on the home page.

Then: I would be able to see the nearest trains for specific station.

5-As a logged in user,

I want to be able to know which what the next station is.

So that: I could know the next station.

Scenario: the logged in user wants to see which next station.

Given: I am a logged in user, I am on the home screen.

When: I click on check out the next stop button on the home page.

Then: I would be able to see the nearest station for specific train.

4-As a logged in user,

I want to be able to search for the location of a specific train using train I.

So that: I can know the train's location.

Scenario: the logged in user wants to search for a specific train.

Given: I am a logged in user, I am on the home screen.

When: I click on search for a train button.

Then: I would be able to search for a train using its ID.

5-As a logged in user,

I want to be able to see my points and send some of them to another user.

So that: I could manage my points.

Scenario: the logged in user wants to manage his points.

Given: I am a logged in user, I am on the home screen.

When: I click on my points button on the home page.

Then: I would be able to see my points and send some of them to a specific user.

6-As a logged in user,

I want to be able to invite others to install and use the app.

So that: I could gain some points.

Scenario: the logged in user wants to invite others to install the app.

Given: I am a logged in user, and I am on the home page.

When: I click on invite people on the home page.

Then: I would be able to gain points after the other people have installed the app or registered on our website.

7-As a logged in user,

I want to be able to be logged out of the web site or app.

So that: I can turn off the app and close the site.

Scenario: the logged in user wants to log out of the app and website.

Given**:** I am a logged in user, I am on the home screen.

When**:** I click on the log out button on the home page screen.

Then**:** I would be able to log out of the website or close the app.

8- There are more features we will add after finishing the previous features (Insha'Allah).

Database relationship analysis

At this section we will explain the entity relationship diagram relations. The app contains from main entities AppUser, Train, Station, PathPoints, PointsHistory, and Cards for recharge points.

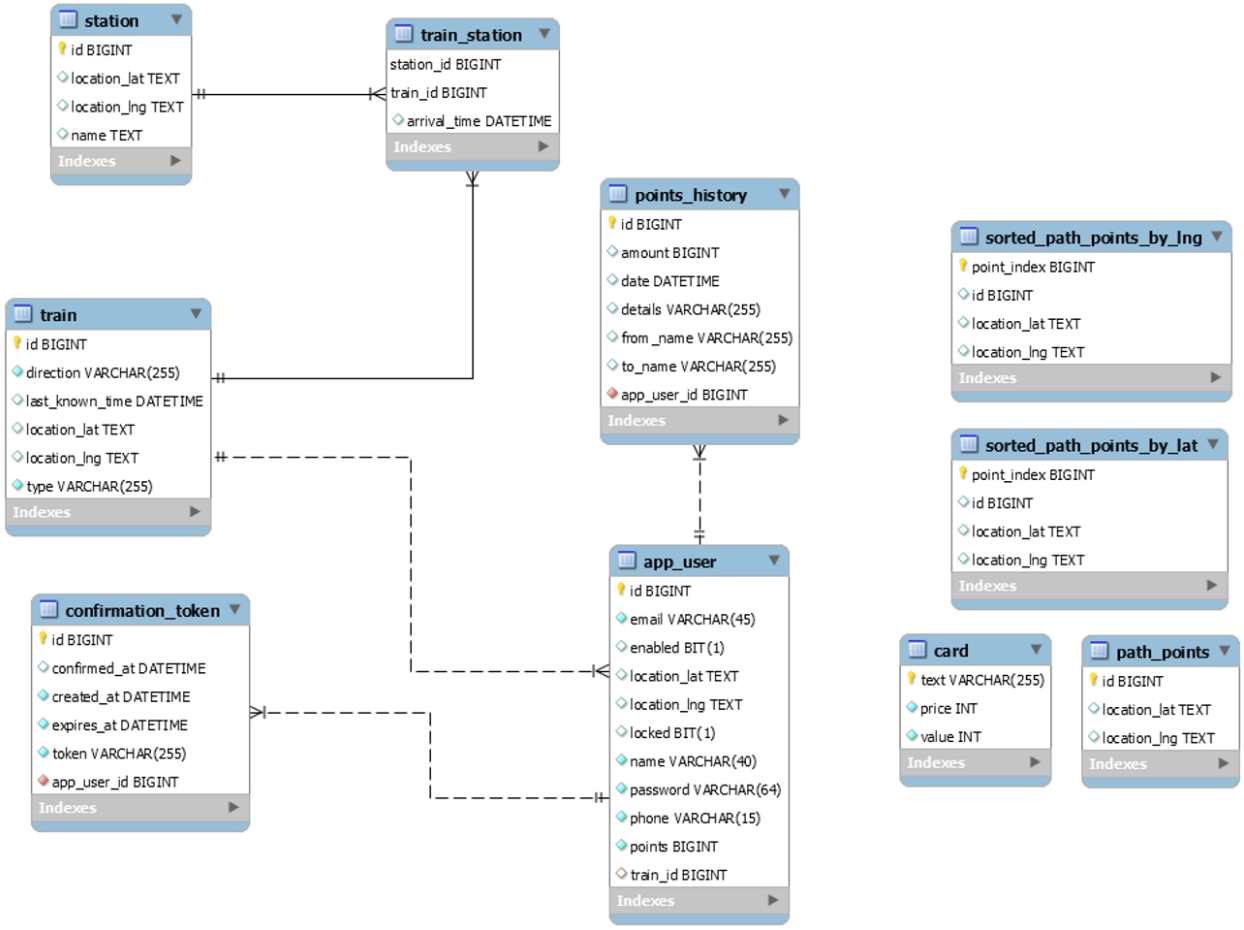


Figure 8: ERD for TrainDar

Station – Train relation

We use this many-to-many relation to know:

- for each train what are the stations that it will path through and the time of arrival.
- for each station what are the trains that will path through it and the time of their arrival.

We use a bridge named train-station to help us to do this task. This relation helps us to complete the search by station and nearest station functions.

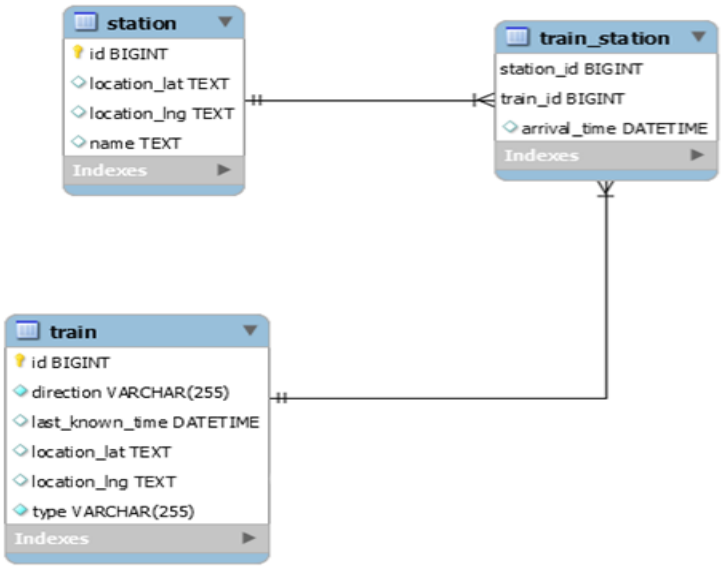


Figure 9: Station - train relation

```
@OneToMany(mappedBy = "train")
private List<TrainStation> stations;
```

Figure 10: Station-train relation implementation

Train – AppUser relation

We use this one-to-many relation to know which users are sharing a specific train, and the train that user shares. This relation helps us to complete the share location function.

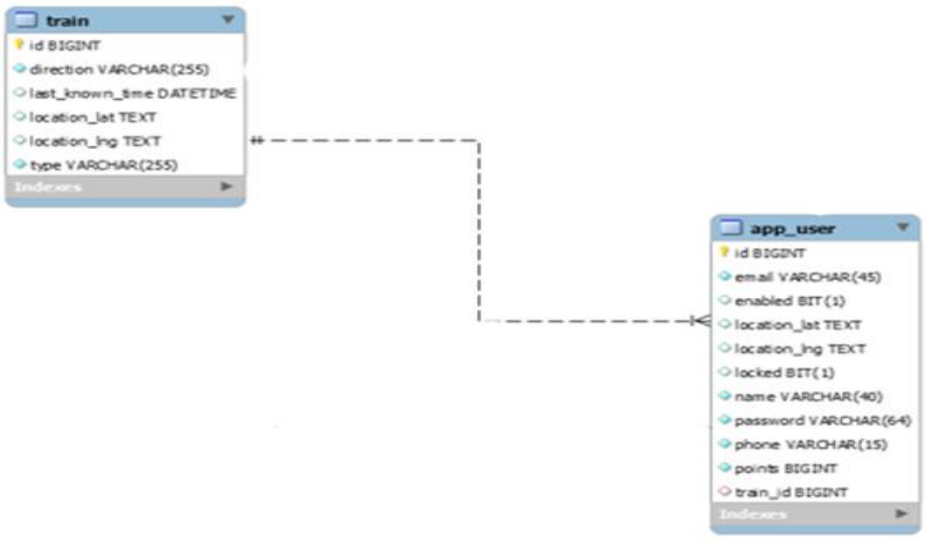


Figure 13: Train-AppUser relation

```
@ManyToOne
@JoinColumn(name="train_id")
@JsonIgnore
private Train train;
```

Figure 12: Train - AppUser relation implementation -class AppUser

```
@OneToMany(fetch = FetchType.EAGER, mappedBy = "train")
private List<AppUser> sharedUsers;
```

Figure 11:Train - AppUser relation implementation -class Train

Confirmation_Token – App_User relation

We use this one-to-many relation to know that which user is related to specific token. This relation helps us to complete the registration and reset password functions.

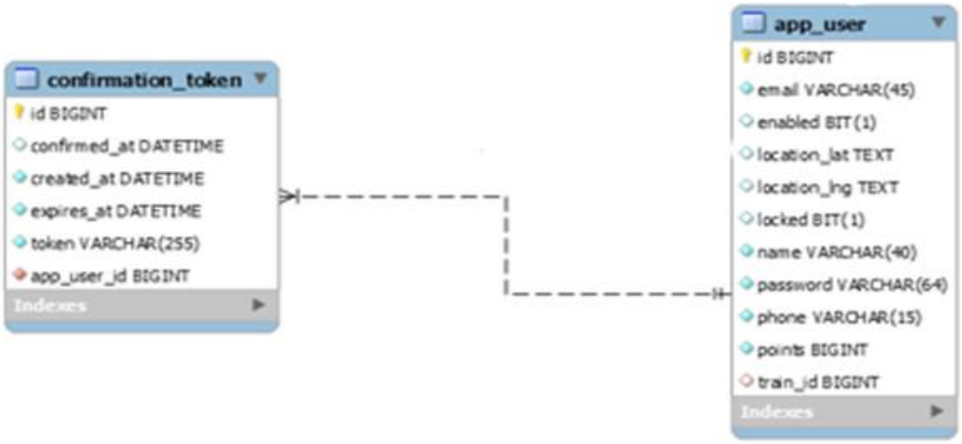


Figure 14: confirmation token and app user relation

```
@ManyToOne
@JoinColumn(nullable = false,name = "app_user_id")
private AppUser appUser;
```

Figure 15: Confirmation token and app user relation implementation

PointsHistory – AppUser relation

We use this one-to-many relation to know the points history for each user. This relation helps us to complete the manage points function.

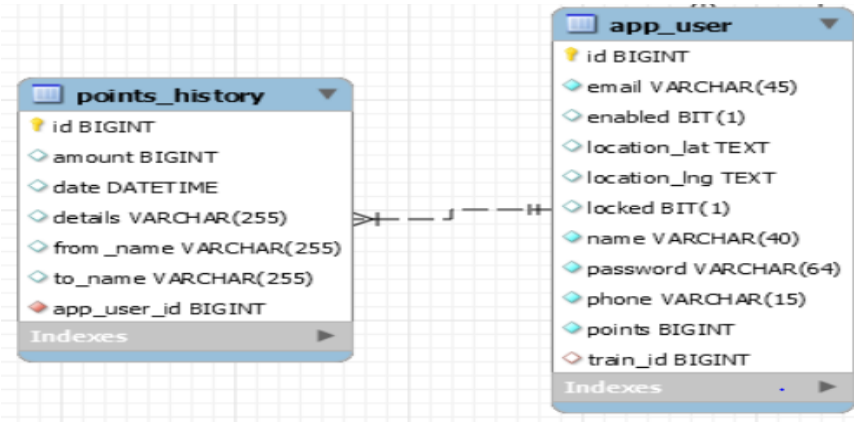


Figure 16:PointsHistory and AppUser relation

```
@OneToMany(fetch = FetchType.EAGER, mappedBy = "appUser")
private List<PointsHistory> pointsHistory;
```

Figure 17: PointsHistory and AppUser relation implementation

Data preparing

For our program we need a huge data that represents the stations, trains, stations that each train stop on, and the path of railway. In the following we will talk about each part of these data:

1-The path of railway

The path of railway contains the points with latitude and longitude for each step that train pass throw. After searching for a tool for help us to record this path, we found PostGIS.

PostGIS is spatial extensions for PostgreSQL, Geographic Information System, used to store, edit, and analyze geographical data, presents data in layers, provides a powerful way of viewing information spatially. Spatial database is database that stores mappable (spatial) data, stores spatial or GIS data (vector, raster), and Includes coordinates. There are a lot of data types at this database like points, lines, polygons, …

Because of this technology is very huge, hard to be learned, and we haven't time to do that, we search more for another tool to help us to record the path and do our queries. We found an application called "My Tracks" that can record where I go (figure 2) and record a lot of data, including the Time, Latitude, Longitude, Altitude, Accuracy(m), Speed(m/s), and bearing. After recording the path, I can export the path to many types like (CSV, GPX, KML, KMZ)- figure3 -, and we prefer to export it as a CSV file to make it easy to maintain and edit the rows and columns to create the insert statements to our database.

Figure 20: Record path

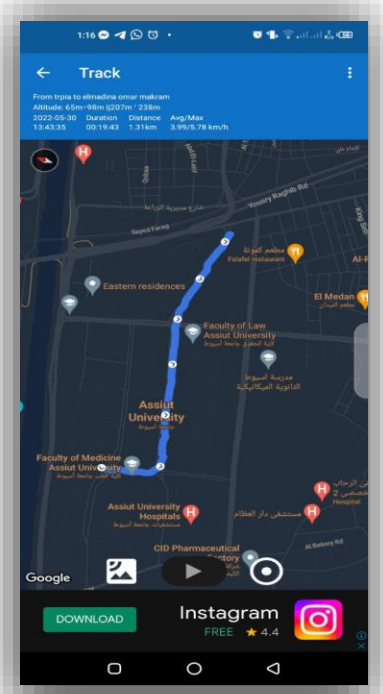


Figure 19: Recorded path

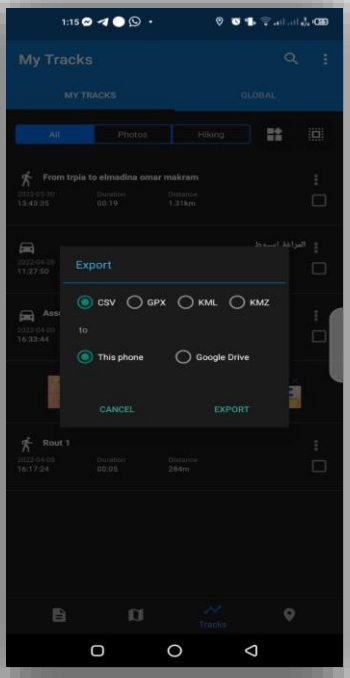


Figure 18: Export path

Exported path

As I indicated before, the path file contains the Time, Latitude, Longitude, Altitude, Accuracy(m), Speed(m/s), and bearing. We need just latitude and longitude from all this data to determine the locations of the railway.

Location	Time	Latitude	Longitude	Altitude	Accuracy(m)	Speed(m/s)	Bearing
1	2022-04-20T14:33:49.623Z	27.181265	31.187582	69.1	70.77	1.2907847	206.82863
2	2022-04-20T14:33:59.716Z	27.18112	31.187607	69.200005	56.05	2.0371046	177.00304
3	2022-04-20T14:34:06.518Z	27.180958	31.18812	68.6	49.994	4.4856005	121.4236
4	2022-04-20T14:34:11.703Z	27.180777	31.18857	69.1	49.749	5.940854	121.21955
5	2022-04-20T14:34:18.559Z	27.180502	31.188606	69.1	59.179	4.647406	144.59993
6	2022-04-20T14:34:23.339Z	27.18036	31.188639	69.4	122.4	4.1324472	151.23155
7	2022-04-20T14:34:28.160Z	27.18016	31.188847	69.3	132.45	4.7674704	145.51855
8	2022-04-20T14:34:33.224Z	27.179983	31.189001	69.3	368.816	4.809087	144.6344
9	2022-04-20T14:34:40.000Z	27.180357	31.188684	69.3	96	0.28268164	175.76389
10	2022-04-20T14:34:43.000Z	27.180527	31.188513	69.3	6.733	0.31571347	178.57878
11	2022-04-20T14:34:46.000Z	27.180418	31.188581	69.3	96	0.46456128	179.62671

Figure 21: exported file

After that we create the insertion statements using excel functions after erase the columns that we don't need. The function we use is CONCATENATE (), it takes a collection of cells and concatenates them together.

We have three entities for path points, there are path_points, sorted_path_points_by_lat, sorted_path_points_by_lng. So, for the path_points entity we insert it as it recorded, for the sorted_path_points_by_lat entity we sort all points by latitude then insert the points, and for the sorted_path_points_by_lng entity we sort all points by longitude then insert points.

A1 =CONCATENATE(B1,C1,D1,E1,F1,G1,H1)

	A	B	C	D	E	F	G	H
1	insert into path_points (id,location_lat,location_lng) values(1,26.551245,31.699417);	insert into	1	,	26.55125	,	31.69942	);
2	insert into path_points (id,location_lat,location_lng) values(2,26.551273,31.699373);	insert into	2	,	26.55127	,	31.69937	);
3	insert into path_points (id,location_lat,location_lng) values(3,26.551304,31.699335);	insert into	3	,	26.5513	,	31.69934	);
4	insert into path_points (id,location_lat,location_lng) values(4,26.551334,31.699291);	insert into	4	,	26.55133	,	31.69929	);

Figure 22: Path points entity

B1 insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(

	A	B	C	D	E	F	G	H	I	J
1	insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(1,1116,27.181265,31.187582);	insert into sort	1	,	1116	,	27.18127	,	31.187582	);
2	insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(2,1115,27.18112,31.187607);	insert into sort	2	,	1115	,	27.18112	,	31.187607	);
3	insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(3,1114,27.180958,31.18812);	insert into sort	3	,	1114	,	27.18096	,	31.18812	);
4	insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(4,1107,27.180527,31.188513);	insert into sort	4	,	1107	,	27.18053	,	31.188513	);
5	insert into sorted_path_points_by_lng (point_index,id,location_lat,location_lng) values(5,1113,27.180777,31.18857);	insert into sort	5	,	1113	,	27.18078	,	31.18857	);

Figure 23: sorted path points by longitude entity

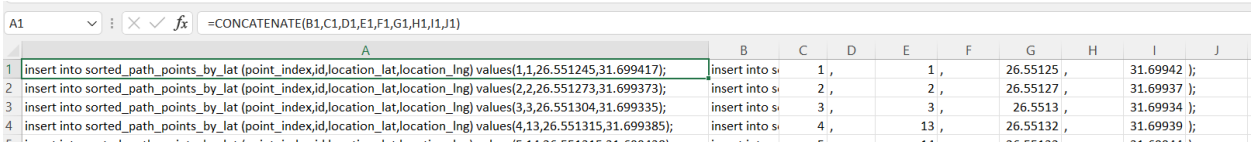


Figure 24: sorted path points by latitude entity

2-Trains and Stations

There are many trains in Egypt, and there is not available api save all trains in Egypt, so after a lot of searches we found a sites and apps that have all trains and all stations which trains pass throw. For each train we store its id, direction, type, and which stations where train stop on. For each station we store its name, location (latitude, longitude), and its id.

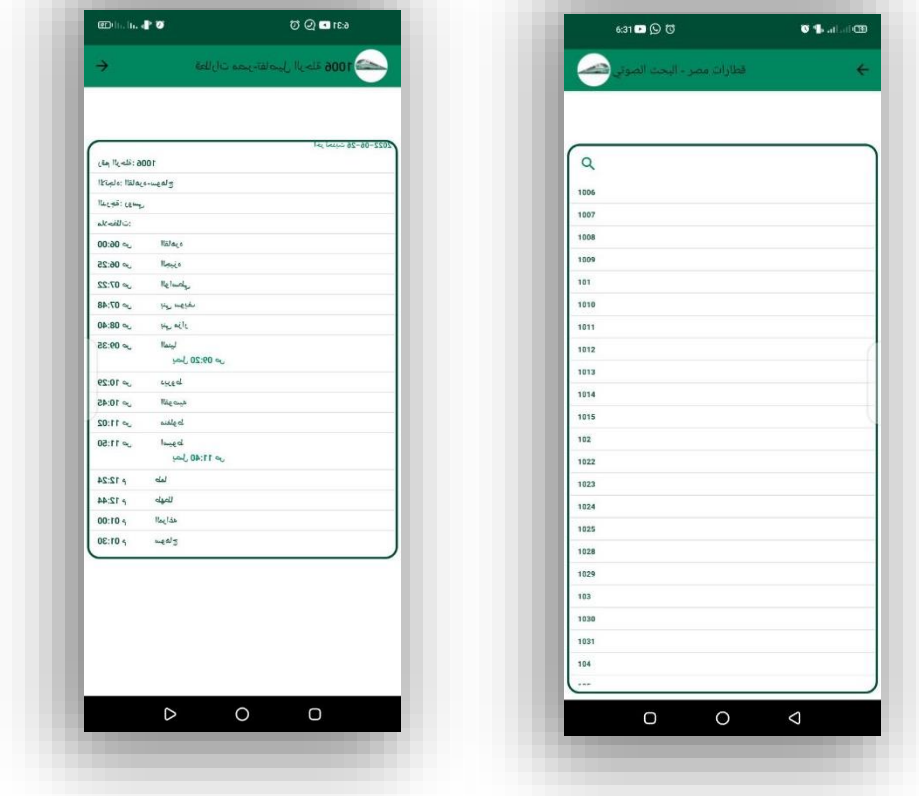


Figure 25: Trains and Stations resource

Insert trains and stations in database:

We insert stations from the South to the North (from Aswan to Alexandria).

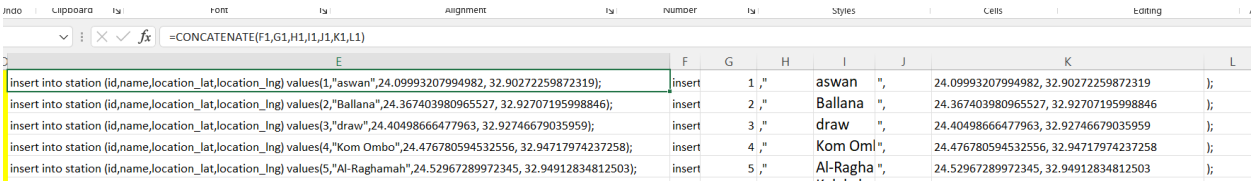


Figure 26:stations data

Then insert trains data:

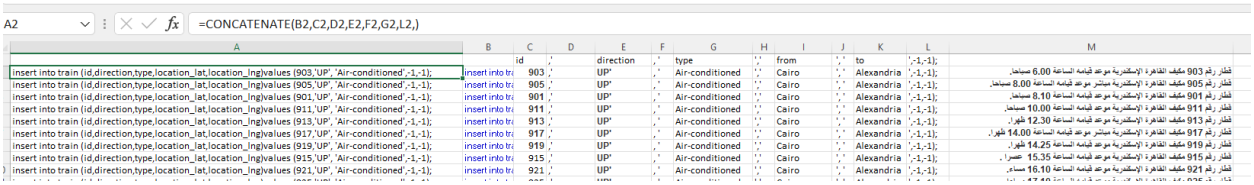


Figure 27: Trains data

Then insert the stations which train pass throw:

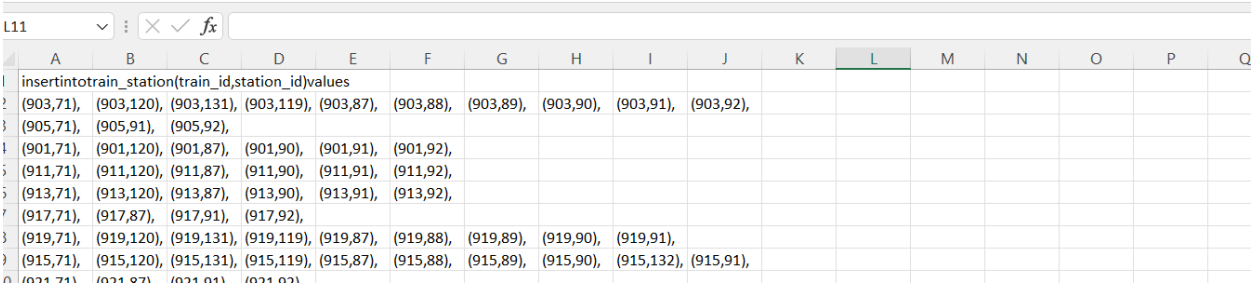


Figure 28: train stations for relation

Note: *we get the stations locations using google maps to get latitude and longitude.*

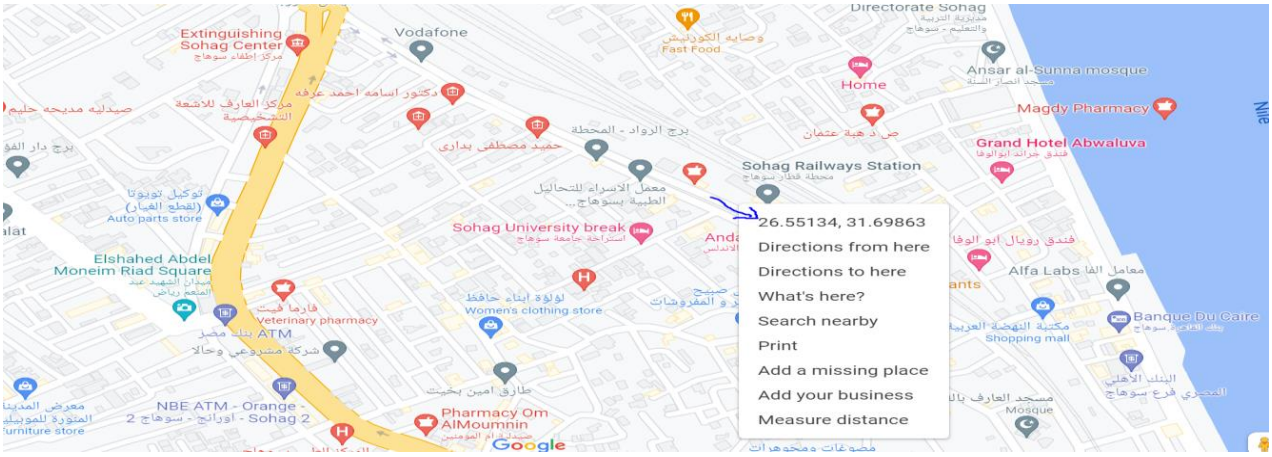


Figure 29: Getting latitude and longitude from google maps

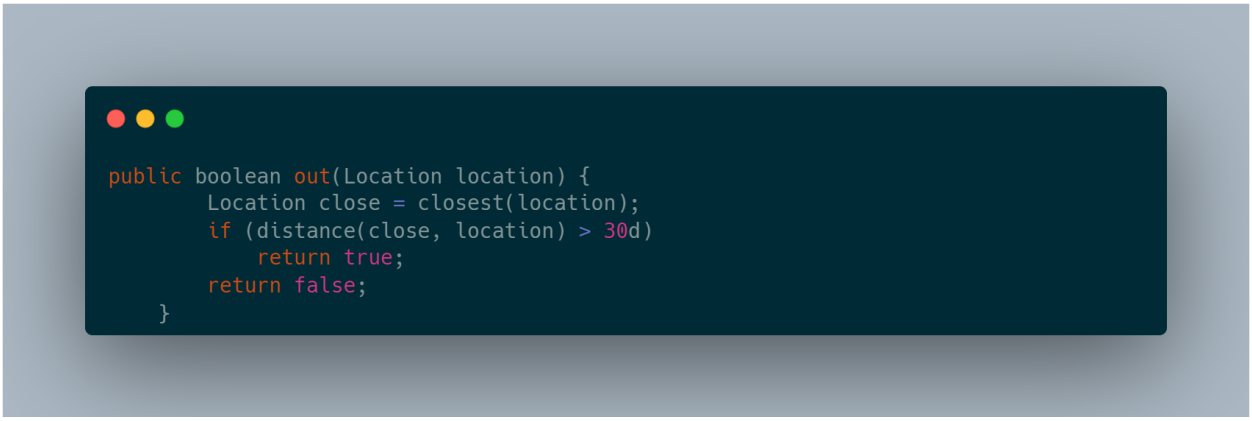
Highlights about implementation in Back-End

For first filter -as we mentioned before- we should classify the in-range users and out-range users of our railway. So, for implement that we create the function inRange (), that take a list of users as a parameter and return another list of in-range users-figure … .

```
public List<AppUser> inRange(List<AppUser> users) {
        List<AppUser> ins = new ArrayList<>();
        Location shared;
        for (AppUser user : users) {
            shared = new Location(user.getLocationLat(), user.getLocationLng());
            if (!out(shared)) {
                ins.add(user);
            }
        }
        return ins;
    }
```

Figure 30: InRange function

In this process we check for each user location and if it is in-range add him to the ins list, we use function out () to determine that-figure….



```
public boolean out(Location location) {
        Location close = closest(location);
        if (distance(close, location) > 30d)
            return true;
        return false;
    }
```

Figure 31: out function

Out () function takes a location of user as a parameter, get the closest point in our path to this location, then get the distance between the closest location and the user location using distance () function. If this distance is more than 30 meters, we consider that location is out-range and return true, if the distance is less than 30 meters this function out will return false.

Distance () takes two locations and return the distance between them, and we assume that the earth is spherical.

```
public double distance(Location point1, Location point2) {
        double locationDistance = 0;
        BigDecimal r2d = BigDecimal.valueOf((180.0D) / (3.141592653589793D));
        final BigDecimal d2r = BigDecimal.valueOf(3.141592653589793D / 180.0D);
        final BigDecimal d2km = r2d.multiply(BigDecimal.valueOf(111189.57696D));
        BigDecimal x = d2r.multiply(point1.getLocationLat());
        BigDecimal y = d2r.multiply(point2.getLocationLat());
        double a = Math.sin(x.doubleValue()),
                b = Math.sin(y.doubleValue()),
                c = Math.cos(x.doubleValue()),
                d = Math.cos(y.doubleValue()),
                e = d2r.doubleValue(),
                f = point1.getLocationLng().doubleValue(),
                g = point2.getLocationLng().doubleValue();
        locationDistance = Math.acos(a * b + c * d * Math.cos(e * (f - g))) * d2km.doubleValue();
        return locationDistance;
    }
```

Figure 32: Distance function

Closest () function takes a location and then returns the closest point in our path that we recorded before.

```
public Location closest(Location shared) {
        Location closestLat = closestLat(shared);
        Location closestLng = closestLng(shared);
        if (distance(shared, closestLat) < distance(shared, closestLng))
            return closestLat;
        return closestLng;
    }
```

Figure 33: closest function

In closest function we call closestLat () to get the closest location using sorted path points by latitude, and closestLng () to get the closest location using sorted path points by longitude using binary search algorithm to improve performance. After getting the both, we compare them which one is closest then return the closest location.

As we show in figure …, we get all path points from database that we sort it by latitude then search on them using binary search algorithm to get the closest latitude.

```
public Location closestLat(Location shared) {
        ArrayList<PathPoints> pathPointsByLat=pathPointsRepository.getAllPointsByLat();
        int first = 0;
        int last = pathPointsByLat.size() - 1;
        int middle = (first + last) / 2;
        double closeValue = Location.CLOSE;
        Location middleLocation = new Location();
        while (first <= last) {
            middleLocation = new Location(pathPointsByLat.get(middle).getLocationLat(),
pathPointsByLat.get(middle).getLocationLng());
            if (Math.abs(distance(shared, middleLocation)) <= closeValue) {
                return middleLocation;
            } else if (shared.getLocationLat().doubleValue() > middleLocation.getLocationLat().doubleValue()) {
                first = middle + 1;
            } else if (shared.getLocationLat().doubleValue() < middleLocation.getLocationLat().doubleValue()) {
                last = middle - 1;
            }
            middle = (first + last) / 2;
        }
        return middleLocation;
    }
```

Figure 34: closest latitude function

As we show in figure …, we get all path points from database that we sort it by longitude then search on them using binary search algorithm to get the closest longitude.

```
public Location closestLng(Location shared) {
        ArrayList<PathPoints> pathPointsByLng=pathPointsRepository.getAllPointsByLng();
        int first = 0;
        int last = pathPointsByLng.size() - 1;
        int middle = (first + last) / 2;
        double closeValue = Location.CLOSE;
        Location middleLocation = new Location();
        while (first <= last) {
            middleLocation = new Location(pathPointsByLng.get(middle).getLocationLat(),
                    pathPointsByLng.get(middle).getLocationLng());
            if (Math.abs(distance(shared, middleLocation)) <= closeValue) {
                return middleLocation;
            } else if (shared.getLocationLng().doubleValue() > middleLocation.getLocationLng().doubleValue()) {
                first = middle + 1;
            } else if (shared.getLocationLng().doubleValue() < middleLocation.getLocationLng().doubleValue()) {
                last = middle - 1;
            }
            middle = (first + last) / 2;
        }
        return middleLocation;
    }
```

Figure 35: closest longitude function

After this process we can get the in-range users and out-range users, then we can move on the second filter. Will all locations that have survived from this filter be all true?

Not all of them. Due to a lot of factors, not all locations that are on the train tracks are valid for our filtration. Some locations might be old or worst, some locations may belong to people walking on the train tracks! For this specific kind of filtration, we used a wide known data mining concept called box plot.

Box Plot: It is a type of chart that depicts a group of numerical data through their quartiles. It is a simple way to visualize the shape of our data. It makes comparing characteristics of data between categories very easy. A box plot gives a five-number summary of a set of data which is-

Minimum: It is the minimum value in the dataset excluding the outliers.

First Quartile (Q1): 25% of the data lies below the First (lower) Quartile.

Median (Q2): It is the mid-point of the dataset. Half of the values lie

below it and half above.

Third Quartile (Q3): 75% of the data lies below the Third (Upper) Quartile.

Maximum: It is the maximum value in the dataset excluding the outliers.

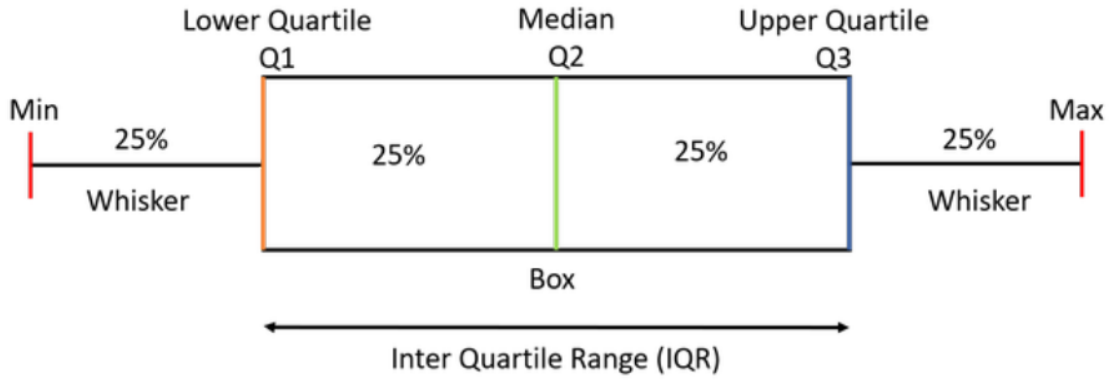


Figure 36: IQR

The area inside the box (50% of the data) is known as the Inter Quartile Range. The IQR is calculated as **IQR = Q3-Q1**. Outliers are the data points below and above the lower and upper limits. The lower and upper limit is calculated as:

Lower Limit = Q1 - 1.5*IQR

Upper Limit = Q3 + 1.5*IQR

The values below and above these limits are considered outliers and the minimum and maximum values are calculated from the points which lie under the lower and upper limits. We first start by dividing the locations into latitudes and longitudes and sorting them increasingly. Then we calculated the quartiles for each of them. After that, we calculated the estimated value of IQR and the lower and upper limits.

Then we calculated the outliers as the values below and above these limits are considered outliers and should not be counted in our train location estimation process.

What is after that?

We simply now have all the 'Good' locations. We could easily estimate the train location by calculating their average.

But Due to the curves in the railway, the average will not always be on the railway. Therefore, we need to take that average and match it to the nearest point in our path that represents train tracks. All these filtrations happen periodically every 30 seconds. We iterate over a list of all trains and get active users for each train. apply the first filtration then the second filtration and then calculate the nearest point on the track. We will then finally get the final estimated train location. These filters run for all trains each 30 seconds and update the trains locations by calling updateLocation function that in TrainService class - figure….

Figure 37:Schedule update trains' locations

Note: *If the user goes out of range, the app will stop his sharing.*

Points management system

As we mentioned before there is increasing and decreasing in user points during using app, it did by calling the function points () that in PointsHistoryService class and pass only the user id and the subject of action. In addition to sending points -figure … -and buy points – figure … -.

```
public void sendPoints(Long from, String emailto, Long amount){...}
```

```
public void points(Long id, String subject) {
        String details = "";
        AppUser user = appUserRepository.findById(id).get();
        PointsHistory pointsHistory = new PointsHistory();
        //Done : build massage
    if (subject == "Bad Share"){...}
    else if (subject == "Good Share"){...}
    else if (subject == "Up Coming Trains"){...}
    else if (subject == "Nearest Station") {...}
    else if (subject == "Search"){...}
    else if (subject == "Invitation") {...}
    else if (subject == "Gift") {...}
    else if (subject == "Registeration"){...}
     pointsHistory.setDetails(details);
        pointsHistory.setDate(LocalDateTime.now());
        user.getPointsHistory().add(pointsHistory);
        pointsHistory.setAppUser(user);
        appUserRepository.saveAndFlush(user);
        pointsHistoryRepository.saveAndFlush(pointsHistory);
}
```

Figure 38:Points functions

So now we have the trains' locations, lets talk about implementation of our light services as we mentioned before:

Search for a train

The backend team create an API that returns the location of the train by passing a train id and user id parameters, then front end takes this location and display the location on our map.

```
@GetMapping(path = {"/view"})
  public Location getTrainLocation(@RequestParam("user-id")Long userId,
                                   @RequestParam("train-id") Long trainId) {
      pointsHistoryService.points(userId,"Search");
      return trainService.findById(trainId);//return location
  }
```

Figure 39: get train location api

Upcoming trains service

As we mentioned before this feature take start station and the destination then return the trains that will pass throw the start station and the destination station. Back-end create an API that return the result. So, what is the steps of this process?

First, get all trains that pass through first-City, then get all valid trains (active trains which have a shared users), then if train direction not equal to stations direction the train won't be added to upcoming trains, then check if current train has already passed the first station or not yet. After checking these conditions add the train to upcoming trains to this start station.

```
public List<UpcomingTrain> getUpcomingTrains(String firstCityName, String secondCityName) {
        var firstCity = stationRepository.findByName(firstCityName);
        var secondCity = stationRepository.findByName(secondCityName);
        List<UpcomingTrain>upcomingTrains = new ArrayList<>();
        TreeSet<Long>trainsTreeSet = new TreeSet<>();
        for(var trainStation : firstCity.getTrains()){
            trainsTreeSet.add(trainStation.getTrain().getId());
        }
        for(var trainStation : secondCity.getTrains()){
            var currTrain = trainStation.getTrain();
            if(!trainsTreeSet.contains(currTrain.getId())){
                continue;
            }
            if(!locationService.isTrainActive(currTrain)){
                continue;
            }
            String stationDirection = locationService.stationDirection(firstCityName, secondCityName);
            if(!stationDirection.equals(currTrain.getDirection())){
                continue;
            }
            if(locationService.hasTrainPassedCity(
                      currTrain
                    , new Location(firstCity.getLocationLat(), firstCity.getLocationLng())
                    , new Location(secondCity.getLocationLat(), secondCity.getLocationLng())
                    )){
                continue;
            }
            upcomingTrains.add(new UpcomingTrain(currTrain.getId(),
                    locationService.timeLeft(
                            new Location(currTrain.getLocationLat(), currTrain.getLocationLng())
                            , new Location(firstCity.getLocationLat(), firstCity.getLocationLng())
                    )
                    ));
        }
        Collections.sort(upcomingTrains);
        return upcomingTrains;
    }
```

Figure 40: upcoming function

Nearest stations service

The user will give us the train ID, then the front will send it to server to get the nearest stations. This process begins by getting all stations that the train pass in and determine which stations that the train hasn't pass throw yet. Then return the nearest station name and the time left to arrive this station.

```
    public List<NearestStation> checkoutNearestStations(Long trainId) {
        Train train = trainRepository.getById(trainId);
        List<NearestStation> nearestStations = new ArrayList<>();
        List<TrainStation> trainStations = train.getStations();
        for (var station : trainStations) {
            Location trainLocation = new Location(train.getLocationLat(), train.getLocationLng()),
                    stationLocation = new Location(station.getStation().getLocationLat(),
station.getStation().getLocationLng());
            if (train.getDirection().equals(locationService.DirectionUpDown(trainLocation, stationLocation)))
{
                nearestStations.add(
                        new NearestStation(station.getStation().getName(),
                                locationService.timeLeft(trainLocation, stationLocation)));
            }
        }
        if (train.getDirection().equals("DOWN")){
            Collections.reverse(nearestStations);
        }
        return nearestStations;
    }
```

Figure 41:The nearest stations function

By the way to get the direction between two locations we can use function DirectionUpDown that can determine the direction between from and to locations. "UP" refers to the direction is to the North and "DOWN" refers to the South.

```
public String DirectionUpDown(Location from, Location to) {
        //path points inserted from aswan to cairo
        from = closest(from);
        to = closest(to);
        int fromIndex = pathPointsRepository.getIDByLatLng(from.getLocationLat(),
                from.getLocationLng()),
                toIndex = pathPointsRepository.getIDByLatLng(to.getLocationLat(),
                to.getLocationLng());
        if (fromIndex > toIndex) {
            return "DOWN";
        } else if (fromIndex < toIndex) {
            return "UP";
        } else return "SAME";
    }
```

Figure 42: Direction function

Highlights about implementation in Front-End (website)

Integrate Google Maps in Angular:

Google Maps is a map service provided by Google that supports a wide variety of configuration settings. Adding Google Maps to your application can provide users with more contextual information than a street address or set of coordinates.

Angular Google Maps are components that provide Angular applications with tools to utilize the Google Maps APIs. We used the @agm/core library and create a map with a marker, and A Google Maps JavaScript API Key.

Steps:

1- A Google Maps JavaScript API Key:

- This will require a Google account.
- Signing into the Google Cloud Platform Console.
- Creating a new project.
- Enabling the Google Maps JavaScript API for the project.
- Creating credentials for an API Key.

2- Loading the Maps JavaScript API:

The Maps JavaScript API is loaded using a script tag, which can be added inline in your HTML file or dynamically using a separate JavaScript file- figure 1-.

Figure 43: Loading the Maps JavaScript API

In the examples above one of that several attributes are set on the script tag:

'Src': The URL where the Maps JavaScript API is loaded from, including all of the symbols and definitions you need for using the Maps JavaScript API. The URL in this example has parameter: key, where you provide your API key,

3- From your project folder, run the following command to install @agm/core:

npm install @agm/core@1.1.0

4-Open app.module.ts in your code editor and modify it to support @agm/core:

As shown in figure 2.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AgmCoreModule } from '@agm/core';
@NgModule({
  declarations: [
    AppComponent
                ],
  imports: [
    BrowserModule,
    CommonModule,
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyBy1TT8oOUraxv2IRV5bx5Zm0Ps8FfXHSM',//YOUR-API-KEY-HERE
      libraries: ['places'],
    }),
                ],
})
```

Figure 44: import @agm/core to app.module.ts

5- Map DOM Elements:

```
<div id="map"></div>
```

Figure 45: html div element for containing the map

For the map to display on a web page, we must reserve a spot for it. We do this by creating a named div element and obtaining a reference to this element in the browser's document object model (DOM).

6-The Map Object:

```
const map = new google.maps.Map(
        document.getElementById("map") as HTMLElement,
        {
          zoom: 13,
          center: myLatlng,
        });
```

Figure 46: the map object

The JavaScript class that represents a map is the Map class. Objects of this class define a single map on a page. (You may create more than one instance of this class — each object will define a separate map on the page.) We create a new instance of this class using the JavaScript new operator.

When you create a new map instance, you specify a HTML element in the page as a container for the map. HTML nodes are children of the JavaScript document object, and we obtain a reference to this element via the document.getElementById () method.

This code defines a variable (named map) and assigns that variable to a new Map object. The function Map () is known as a constructor and its definition is shown below:

Constructor	Description
`Map(mapDiv:Node, opts?:MapOptions )`	Creates a new map inside of the given HTML container — which is typically a DIV element — using any (optional) parameters that are passed.

Figure 47: Map constructor definition

Full function:

```
MapGoogleCode(): void {
    const myLatlng = { lat: this.trainLocation.locationLat, lng: this.trainLocation.locationLng };
    const map = new google.maps.Map(
        document.getElementById("map") as HTMLElement,
        {
          zoom: 13,
          center: myLatlng,
        });
    const marker = new google.maps.Marker({
        position: myLatlng,
        map,
        title: "Click to zoom",
    });
    map.addListener("center_changed", () => {
        // 3 seconds after the center of the map has changed, pan back to the marker.
        window.setTimeout(() => { map.panTo(marker.getPosition()as google.maps.LatLng );}, 3000);
    });
    marker.addListener("click", () => {
        map.zoom++;
        map.setCenter(marker.getPosition()as google.maps.LatLng );
    });

  }
```

Figure 48: Display Map function

Map:

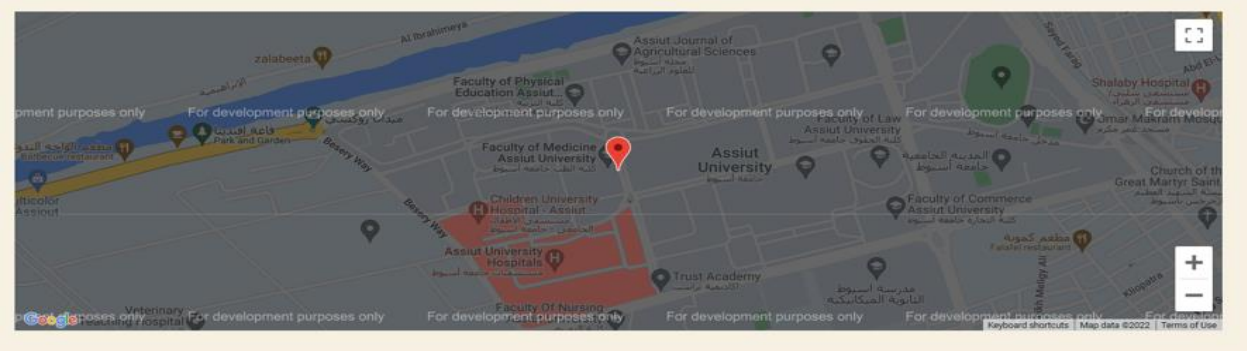


Figure 49: Map in web site

Forms validation in Angular:

Every time the value of a form control changes, Angular runs validation and generates either a list of validation errors that results in an INVALID status, or null, which results in a VALID status.

In case of INVALID status:

The <input> element is surrounded by red color and custom message displayed.

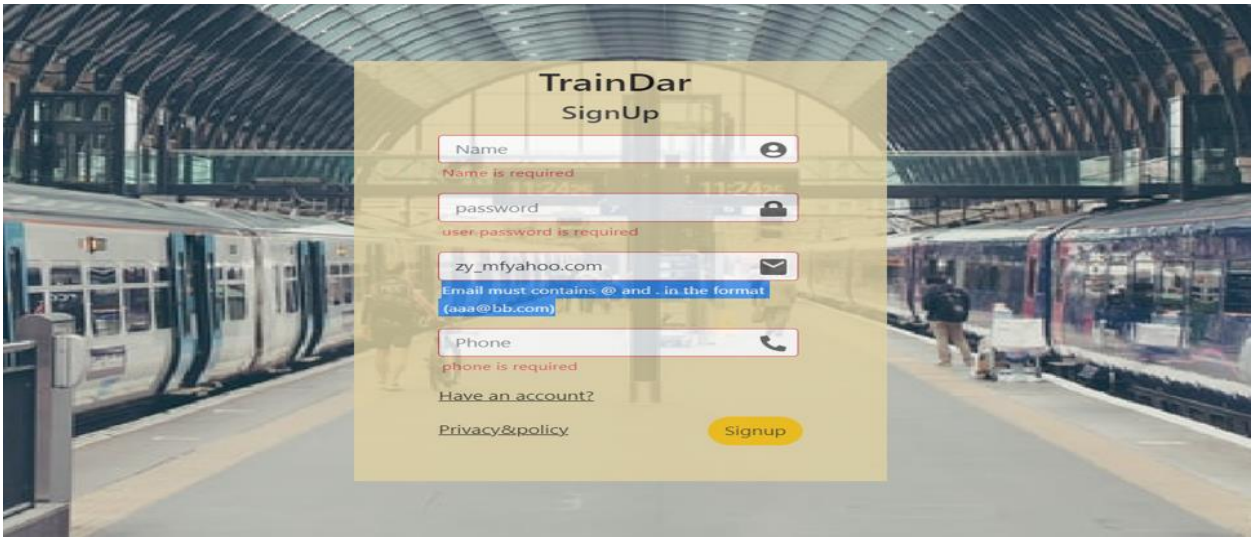


Figure 50: Signup form

```
<div class="form-outline mb-3">
     <input type="email" class="form-control email" placeholder="Email"
            [class.is invalid]="txtemail.invalid && txtemail.touched" pattern="[^@\s]+@[^@\s]+\.[^@\s]+"
            name="textemail" #txtemail="ngModel" required  [(ngModel)]="userModel.email" />
     <small class="text-danger" [class.d-none]="!txtemail.errors?.['required']|| txtemail.untouched">
       Email is required
      </small>
   <small class="text-danger" [class.d-none]="!txtemail.errors?.['pattern'] || txtemail.untouched">
       Email must contains @ and . in the format <br> (aaa@bb.com)
   </small>
</div>
```

Figure 51: Input text validation for email

Integrate SweetAlert2 in Angular:

Every now and then, you'll have to show an alert box to your users to let them know about an error or notification. The problem with the default alert boxes provided by browsers is that they're not very attractive. When you're creating a website with great color combinations and fancy animation to improve the browsing experience of your users, the unstyled alert boxes will seem out of place.

We used a library called SweetAlert2, which allows us to create all kinds of alert messages that can be customized to match the look and feel of our own website.

Steps:

1- From your project folder, run the following command to install SweetAlert2:

npm install sweetalert2

2- Include library in your project and call the sweetAlert2-function:

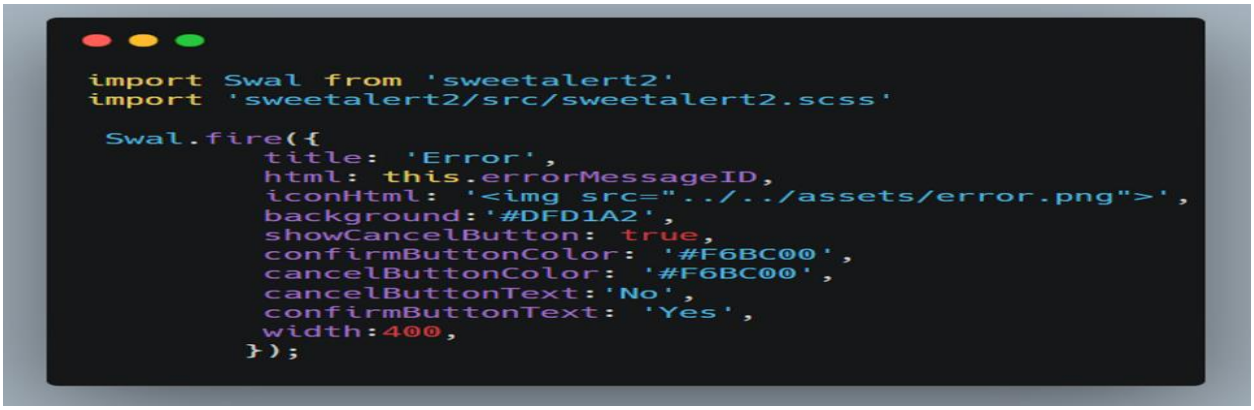


Figure 52:SweetAlert2 library and function

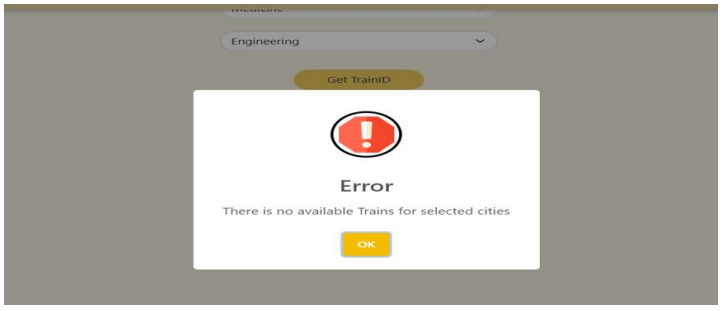


Figure 53: sweetAlert2 in web site

Responsive Web Design:

Web pages can be viewed using many different devices: desktops, tablets, and phones, it should look good, and be easy to use, regardless of the device and should not leave out information to fit smaller devices, but rather adapt its content to fit any device. We used SCSS, HTML and Bootstrap to resize, shrink, enlarge, or move the content to make it look good on any screen.

Media query in stylesheet:

```
//labtop
@media ( min-width: 768px)and  ( max-width:1300px)
 { body{width:400%;}
}

// my phone
@media ( min-width: 400px)and  ( max-width: 500px)
 { body{width: 115%;}
}

// small web
@media ( min-width: 500px)and  ( max-width: 768px)
 { body{width: 100%;}
}


// small pfone
@media ( max-width:400px)
 { body{width:180%;}
}}
```

Figure 54:Media query for body width

Here are some examples:

Nav bar example

```
<nav class="navbar navbar-expand-lg navbar-light fixed-top w3-container w3-card  ">
  <div class="container-fluid">
    <b><a class="navbar-brand " routerLink="/home"><b>TrainDar</b><img src="../../assets/154101.png" >
</a></b>
          
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse smalllap" id="navbarSupportedContent">
      <ul class="navbar-nav mb-2 mb-lg-0">
        <pre>
    </pre>
        <li class="nav-item me-4">
          <a class="nav-link active" aria-current="page" href="/home">Home</a>
        </li>
        <li class="nav-item me-4">
          <a class="nav-link active" aria-current="page" href="#about">About Us</a>
        </li>
        <li class="nav-item me-4">
          <a class="nav-link active" aria-current="page" href="#service">Services</a>
        </li>
        <li class="nav-item me-4">
          <a class="nav-link active" aria-current="page" href="/contact">Contact</a>
        </li>
        <li class="nav-item me-4">
          <a class="nav-link active" aria-current="page" routerLink="/signinform">Login/Register</a>
        </li>
           
      </ul>
    </div>
  </div>
</nav>
```

Figure 55: Html and bootstrap class for navbar

Lap view:

Figure 56: navbar view in Lap

Mobile view:

Figure 57: navbar view in mobile

If user click the icon that I refer to this list of options -figure… -.

Figure 58: navbar after clicking icon

Profile example

Here we work for design a profile page for user and make it available to be viewed using many different devices.

Lap view: As shown in figure ….

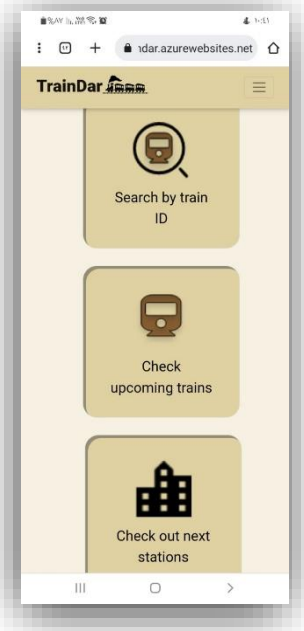
Figure 59: profile view in web site

Mobile view: As shown in figure ….

Figure 18: profile view in web site in mobile

Service view:

The figure … shows the view of web site at lap, and figure… shows the view of website at mobile.



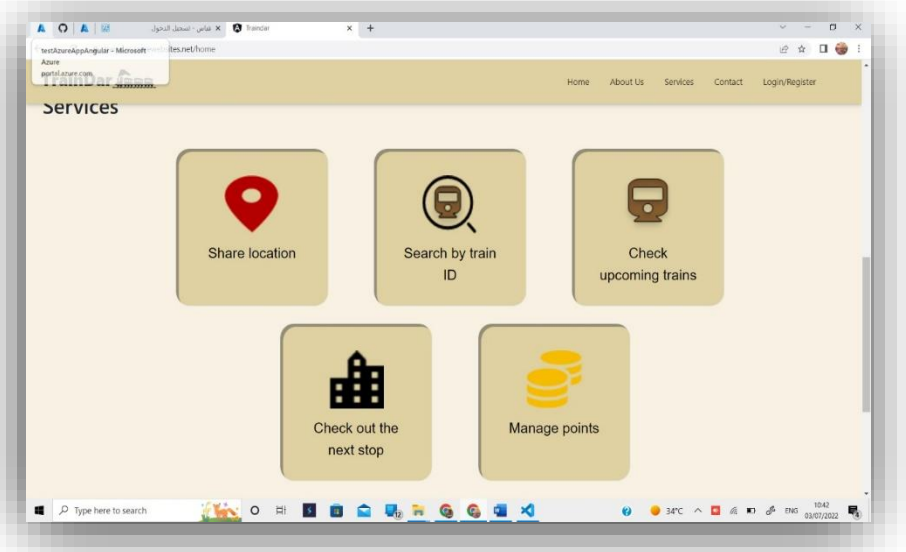


Figure 18: services view in web site

Highlights about implementation in Front-End (mobile app)

We use Flutter because it is a cross-platform framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase, so we use it in our Graduation project.

Description of TrainDar Mobile:

We divide the project into:

- Models.
- Functions about API.
- Modules.
- Shared Components.
- Using Packages.

First, we will talk about models:

Models describe our data in classes which have variables and converter functions fromJson() and to son(), we divide them into three primary entities (Trains, Users, Stations) and have two sub-entities (Nearby Trains, and Close Stations).

```dart
import 'dart:convert';

User userFromJson(String str) => User.fromJson(json.decode(str));

String userToJson(User data) => json.encode(data.toJson());

class User {
  User({
    required this.id,
     this.points=500,
    required this.name,
    required this.email,
    required this.password,
    required this.phone,
  });

  int id;
  int points;
  String name;
  String email;
  String password;
  String phone;

  factory User.fromJson(Map<String, dynamic> json) => User(
    id: json["id"],
    points: json["points"],
    name: json["name"],
    email: json["email"],
    password: json["password"],
    phone: json["phone"],
  );

  Map<String, dynamic> toJson() => {
    "id": id,
    "points": points,
    "name": name,
    "email": email,
    "password": password,
    "phone": phone,

  };
}
```

Figure 60:User Model

Second, Functions of APIS:

We create a class that has future functions to handle with APIS. In these Future Functions, we use HTTP methods to do these operations (get, post, delete, update, put).

using async, await, and future because the data took more time to come from the back, after that we parse URL and Headers in the HTTP method which returns the response body, and we check if this response doesn't have any exceptions, if not we navigate data from API to modules to display it, such as figure …

```
Future<List<NearbyTrains>> getNearbyTrains(
    {required String station1, required String station2}) async {
  List<NearbyTrains> nearbyTrains = [];
  var response = await http.get(
    Uri.parse(
        "https://train-dar.azurewebsites.net/api/v1/train/show-upcoming-trains?"
            "user-id=${UserAPI.currentUserId}"
            "&first-city=$station1"
            "&second-city=$station2"),
    headers: URI.headers,
  );
     if(response.statusCode==200) {
    var body = jsonDecode(response.body);
    for (var item in body) {
      nearbyTrains.add(NearbyTrains.fromJson(item));
    }
    return nearbyTrains.toList();
  }
     nearbyTrains.add(NearbyTrains(trainId: 0, timeLeft: '0'));
  return nearbyTrains.toList();
}
```

Figure 61:Get Nearby Trains

Third, Modules:

In this category, we display our design and data by creating classes extend from two main Widgets in Flutter (Stateful and Stateless), but at first, we create a layout by composing widgets to build more complex widgets and use initState(),setState() to handle the State Management.

```dart
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color.fromRGBO(223, 209, 164, 0.85),
    body: screens[currentIndex],
    bottomNavigationBar: BottomNavigationBar(
      type: BottomNavigationBarType.fixed,
      iconSize: 30,
      showUnselectedLabels: false,
      backgroundColor: const Color.fromRGBO(223, 209, 164, 1),
      unselectedItemColor: const Color.fromRGBO(87, 89, 86, 1),
      currentIndex: currentIndex,
      onTap: (ind) {
        setState(() {
          currentIndex = ind;
        });
      },
      fixedColor: const Color.fromRGBO(87, 89, 86, 0.5),
      items: const [
        BottomNavigationBarItem(
          icon: Icon(Icons.view_list_rounded),
          label: "Menu",
        ),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon(Icons.house),
          label: "Home",
        ),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon(Icons.account_circle),
          label: "Profile",
        ),  // BottomNavigationBarItem
      ],
```

Figure 62:Build function at home layout class

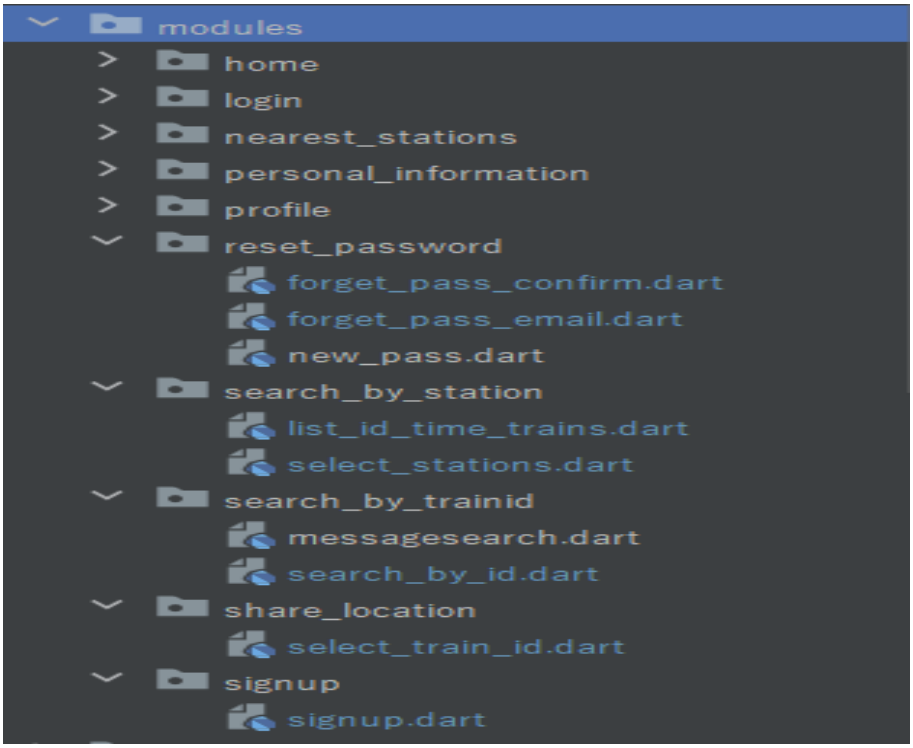


Figure 63:our modules in the project

Fourth, Shared Components:

There are some identical widgets, so we write one code in a separate class and then call it when needed it.

```dart
import 'package:flutter/material.dart';
Widget defaultButton ({
  required Function func,
  required String text,
   double width =double.infinity, }) => SizedBox(
    width: width,
    height: 40.0,
    child: MaterialButton(
      child: Text(
        text,
        style:  const TextStyle(
          fontSize: 20,
        ),  // TextStyle
      ),  // Text
      onPressed: func(),
      color: const Color.fromRGBO(87, 89, 86, 100),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(30),
        side: const BorderSide(color: Color(0×ff707070)),
      ),  // RoundedRectangleBorder
    ));  // MaterialButton, SizedBox
```

Figure 64:default button in the UI

Fifth, Using Packages:

Packages:

1. Shared Preference:

We use this package to store the condition of the user's location to check if this user shares his location or not.

```
late StreamSubscription<LocationData> locationSubscription;
Future<bool> getValidShare() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  setState(() {
    _isValidShare = prefs.getBool("isShare")!;
  });
  return Future(() => _isValidShare);
}
void setValidShare() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  setState(() {
    prefs.setBool("isShare", _isValidShare);
   //| print('$_isValidShare+set');
  });
}
```

Figure 65: storing the connection of user

2. Location:

This plugin for flutter handles getting a location on Android and iOS. It also provides callbacks when the location is changed and checks if the GPS is enabled or not.

```
void _operationsOfLocation() {
  Location location = Location();
  locationSubscription =
      location.onLocationChanged.listen((LocationData currentLocation) async {
        bool res = await ShareAPI().sharedLocation(
        locationLat: currentLocation.latitude as double,
        locationLng: currentLocation.longitude as double);
    if (!res) {
      setState(() {
        ScaffoldMessenger.of(context)
            .showSnackBar(
          SnackBar(
            content:const  Text(
              "you share a wrong location",
              style: TextStyle( ... ),  // TextStyle
            ),  // Text
            backgroundColor: const Color.fromRGBO(211, 200, 160, 0.85),
            duration: const Duration(seconds: 10),
            padding: const EdgeInsets.all(10),
            action: SnackBarAction( ... ),  // SnackBarAction
          ),  // SnackBar
        );
        locationSubscription.cancel();
        _isValidShare = false;
        setValidShare();
        _shareText = "Share Location";
        ShareAPI().deleteShare(trainId: HomeScreen.trainId);
      });}
  });
```

Figure 66:on changed Location function

3. HTTP:

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. We use it to deal with APIS.

```
Future<bool> loginUser(String email, String password) async {
  final response = await http.post(
      Uri.parse(URI.login),
      body: jsonEncode(<String, dynamic>{
        'email': email,
        'password': password,
      }),
      headers: URI.headers);
  if (response.statusCode == 200) {
    //print(response.body);
    var responseBody = jsonDecode(response.body);
    currentUserId = responseBody as int;
    return true;
  } else {
    //print(response.body);
    return false;
  }
}
```

Figure 67:HTTP package

4. Google_Maps_Flutter:

A Flutter plugin that provides a Google Maps widget from google cloud.
we create a project on the cloud and enable google map services to get the key and add it in the permission access to flutter code.

```dart
import 'package:location/location.dart';
class LocationServices{
  static bool getPermission=false;
Future<LocationData> getLocation () async{
  Location _location= Location();
  bool _serviceEnabled;
  PermissionStatus _permissionGranted;
  LocationData _data;
  _serviceEnabled= await _location.serviceEnabled();
  if(!_serviceEnabled) {
    _serviceEnabled=await _location.requestService();
  }
  if(!_serviceEnabled){
    throw Exception("open GBS");
  }

  _permissionGranted=await _location.hasPermission();
  if(_permissionGranted==PermissionStatus.denied ||
  _permissionGranted==PermissionStatus.deniedForever){
    _permissionGranted=await _location.requestPermission();
  }
  if(_permissionGranted≠PermissionStatus.granted){
    throw Exception('give me Permission');
  }
  _data=await _location.getLocation();
  getPermission=true;
  return _data;
}
}
```

Figure 68:test permissions

Hosting

We use Microsoft Azure for hosting the front end and database. **So, what is Azure?**

The Azure cloud platform is more than 200 products and cloud services designed to help you bring new solutions to life—to solve today's challenges and create the future. Build, run, and manage applications across multiple clouds, on-premises, and at the edge, with the tools and frameworks of your choice.

Why Azure?

In addition to azure is a free trial for students, Microsoft maintains a growing directory of Azure services, with more being added all the time. All the elements necessary to build a virtual network and deliver services or applications to a global audience are available, including Virtual machines, SQL databases, Azure Active Directory Domain services, Application services, and Storage.

For what did we use Azure?

First, we create an MYSQL server, and we deploy our database. Then we create our App-service and deploy the application using GitHub as a deployment center. We did that also for site's front-end. After that azure does the rest of the tasks.

Expected behavior scenarios

1. Sign up:

Create an account to use Traindar app, the user should have account.

a. **Wrong email format:**
   i. Back: throw exception “bad email format”
   ii. Front: will give a massage.
b. **Email already exists:**
   i. Back: throw exception “already exists”
   ii. Front: alert it is already exists
c. **Password differ confirm password:**
   i. Back: it should given from front there are the same or not.
   ii. Front: alert that.
d. **All attributes are filled and good:**
   i. Back: returns the user token and add user to the database.
   ii. Front: go to sign in page.

2. Sign in:

Login using existed account, email must be signed up.

a. **Wrong email format:**
   i. Back: throw exception
   ii. Front check the format.
b. **Wrong password:**
   i. Back: throw exception “wrong password”
   ii. Front: alert that “email or pass is not correct.”
c. **Email doesn’t exist:**
   i. Back: throw exception “email not found”
   ii. Front: alert that
d. **All attributes are good:**
   i. Back: returns the user’s id
   ii. Front: (go to home page)

3. Forget password:

a. **Email format is wrong:**
   i. Back: throw exception “bad format”
   ii. Front: alert that
b. **Email doesn’t exist:**
   i. Back: throw exception “not found”
   ii. Front: alert not found
c. **Wrong confirmation code: After this step user can reset password.**
   i. Back: throw exception “valid email and token”
   ii. Front: (alert “it Is good”)
d. **Password differs from confirm password:**
   i. Back: return exception “there are not the same”

ii. Front: alert that

e. **If email and token are right:**

i. Front: go to sign in page.

4. **Profile:**
   a. **Edit profile.**
   b. **Points history.**
   c. **Invitation.**
5. **Share Location:**
   a. **Trains list:**
      i. Back: server returns a list of trains.
      ii. Front: receive the list and display it using drop down list.
   b. **Send location:**
      i. Front: mobile sends his current location to server each 30 sec.
      ii. Back: receives this location and analyses this location.
   c. **Share fake location:**
      i. Back: returns true that means the location is not good.
      ii. Front: mobile alert that he is share not good location and he will lose amount of points (2) & stop sharing.
   d. **Share good location:**
      i. Back: after analysis, if it returns false then it is good sharing, and his points will increase by 2 every 30 sec.
      ii. Front: no action and let open sharing, at mobile, it should be working at background.
   e. **Stop sharing location:**
      i. **Back: remove him from train users.**
      ii. Front: stop sharing and change icon.
6. **Search for a train:**
   a. **Trains list:**
      i. Back: returns a list of active trains (which trains have users)
      ii. Front: display this list at drop down list.
   b. **Trains list:**
      i. Front: If the list is empty, front will display there is no active train as alert.
   c. **Haven't enough points:**
      i. Back: throw exception "you have not enough points"
      ii. Front: alert that massage,
   d. **Have enough points:**
      i. Back: will return the expected location (lat, lng) for this train and decrease his points by 3.
      ii. Front: display the location using google maps, alert a massage that user points will decrease by 3(didn't work yet).
7. **Check upcoming trains:**
   a. **Stations list: (always has values)**
      i. Back: returns a list of stations sorted from South to north.

ii. Front: display this list at a dropdown list for start station and destination station.

b. Haven't enough points:

i. Back: throw exception "you have not enough points"
ii. Front: alert that "the user should have more than 3 points".

c. Have enough points:

i. Back: returns a list of active trains that will stop on chosen station and its time left, and decrease user points by 3.
ii. Front: display the trains and its time left.

d. Empty trains list:

i. Front: alert that there is no active trains.
ii. Back: don't decrease his points.

e. Choose train:

i. Front: go to the page of "Search for a train" and pass the selected train id (note: at the page of search for a train will check his points).

8. Check nearest stations:

a. Trains list:

i. Back: returns a list of active trains (which has shared users)
ii. Front: display this list as a drop-down list.

b. Trains list:

i. Front: if the list is empty, alert there is no active trains.

c. Haven't enough points:

i. Back: throw exception "you haven't enough points"
ii. Front: alert that "you must have more than 3 points"

d. Have enough points:

i. Back: returns a list of stations and its time left for the train to reach.
ii. Front: display the list of stations and its time left at a Table.

e. Stations list is empty:

i. Front: alert "there is no stations that the train will stop on"
ii. Back: don't decrease user points.

9. Manage points:

a. Send points.
b. Buy points.
c. Points history.

Simulation and Testing

For testing, we test a hard code test by putting values in database and run our app, see the result and compare them with what we expected. Then after that we go to simulate our project and make a simple model that acts as a railway, trains, and stations. In the following pages we will show the test and what appears in mobile and web site (on mobile & lap).

Trains:

Train id	Direction	Stations
1	Up	2,3,4
2	Up	2,3,5
3	Down	6,3,2
4	Down	1,2
5	Up	1,5,6

Stations:

1. Medicine
2. Dentistry
3. Engineering
4. Law
5. Building
6. Omar Makram gate

1-Share train location Tests:

a. All users share correct locations; The train location is the location of users.

Figure 69:Test1, all users are correct locations.

When we search for train 1, it will appear in the location of people.

At Lap browser:

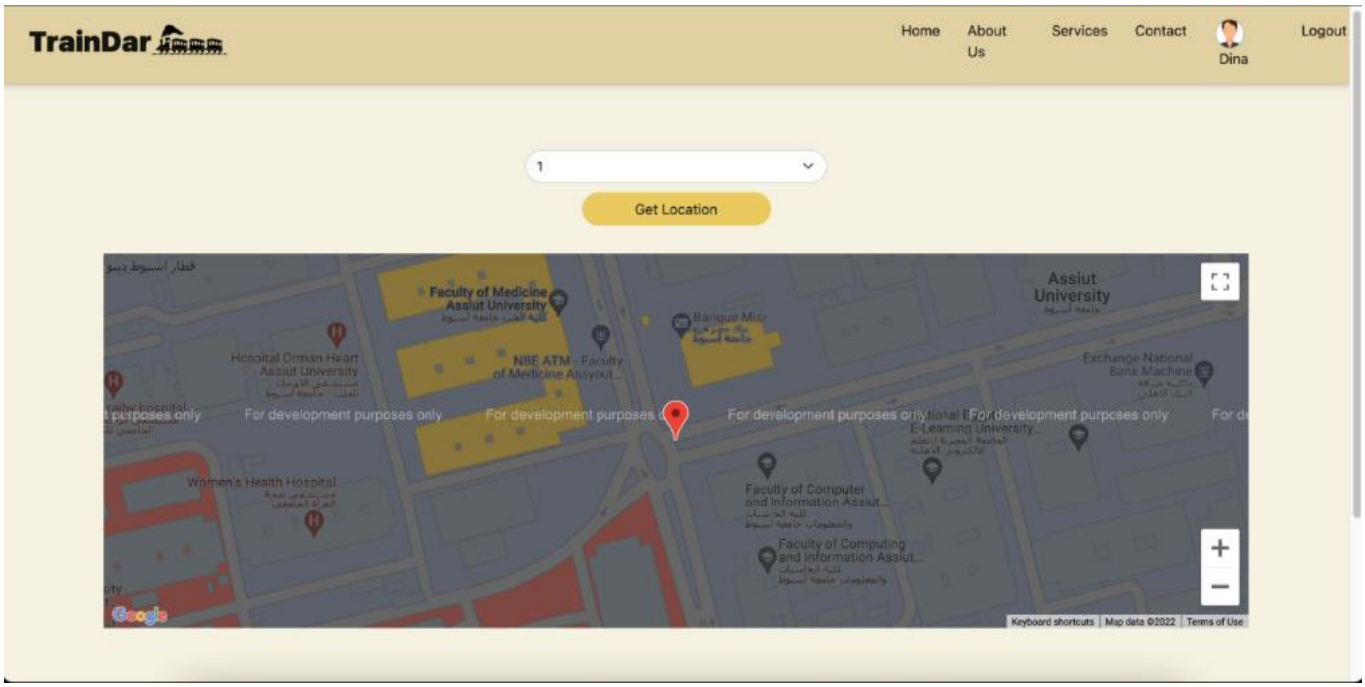


Figure 70 Test 1, all users have correct locations, Lap view

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

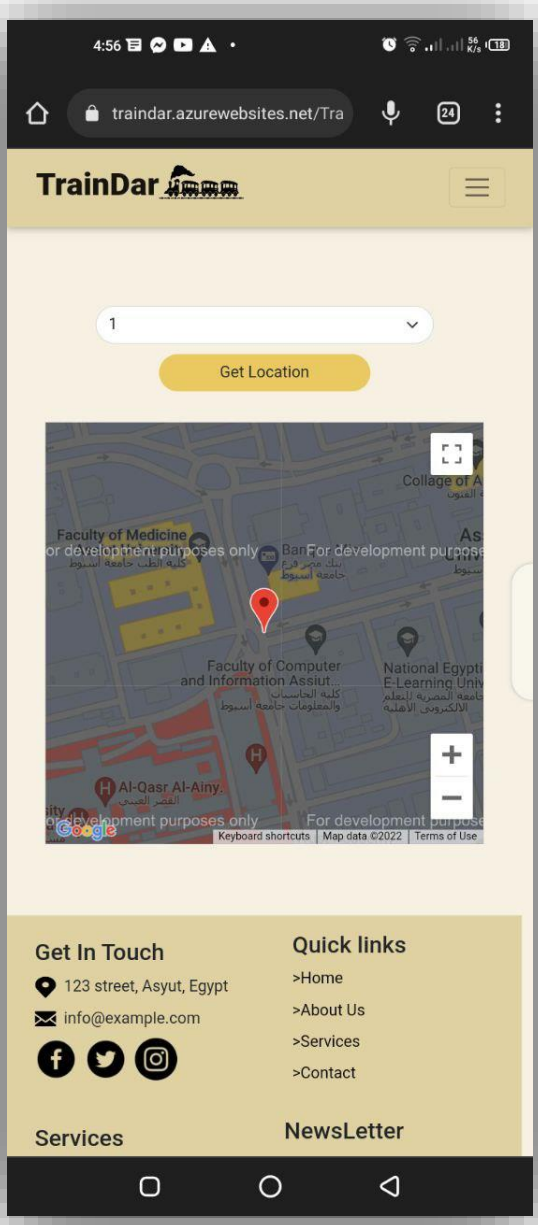


Figure 71, mobile app view for test1

b. One of users is outlier.

When we search for train 1, it will appear in a location between the outlier and real people, so it will shift a little percentage to UP.

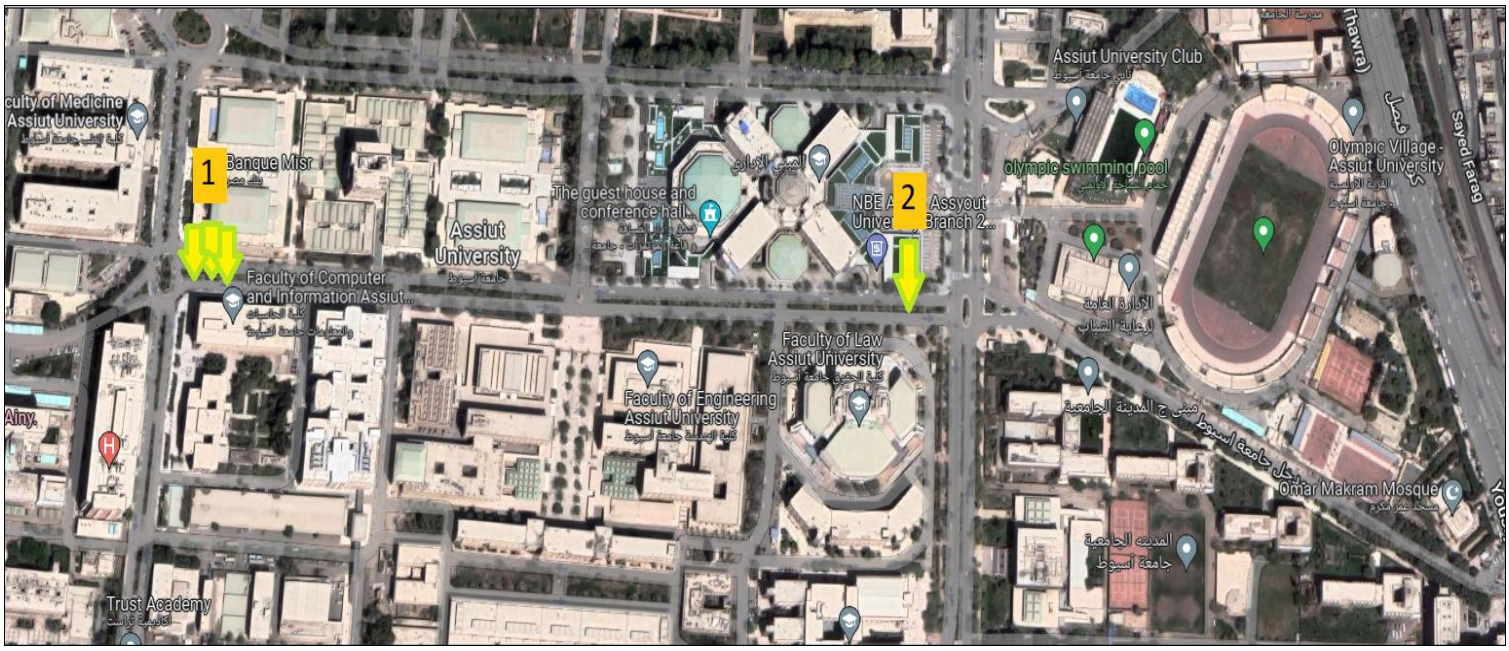


Figure 72 Test2, one of users is outlier

At lap browser:

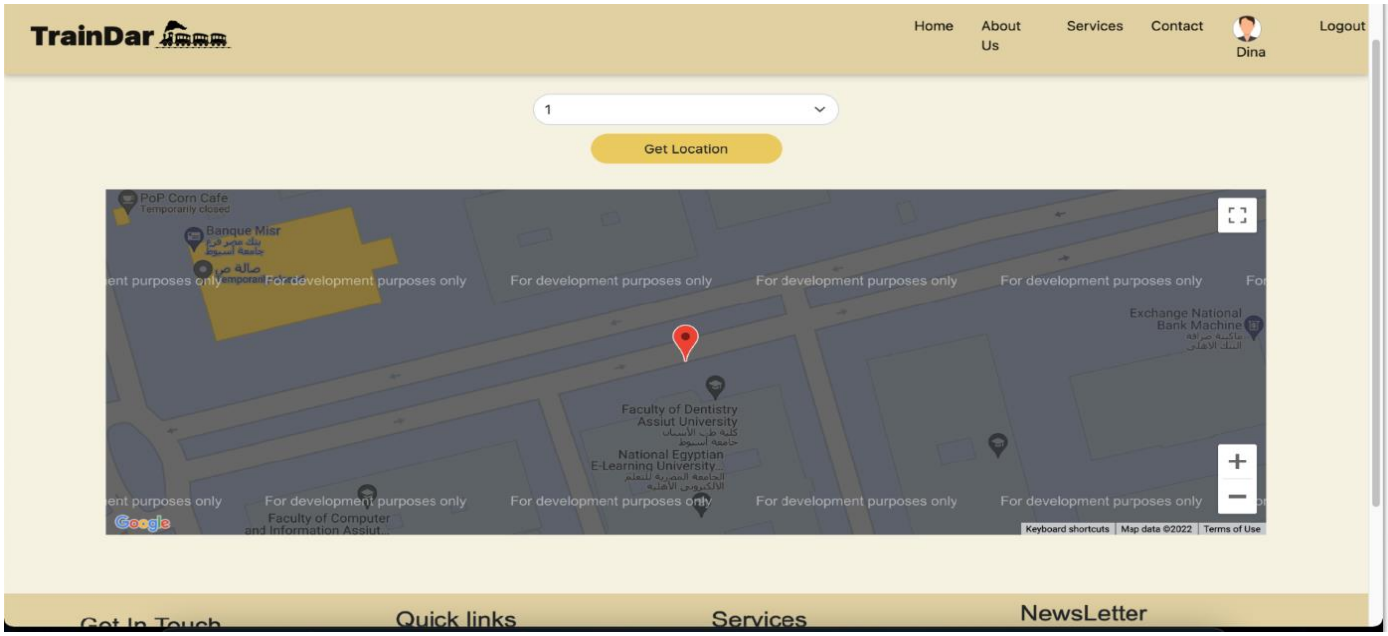
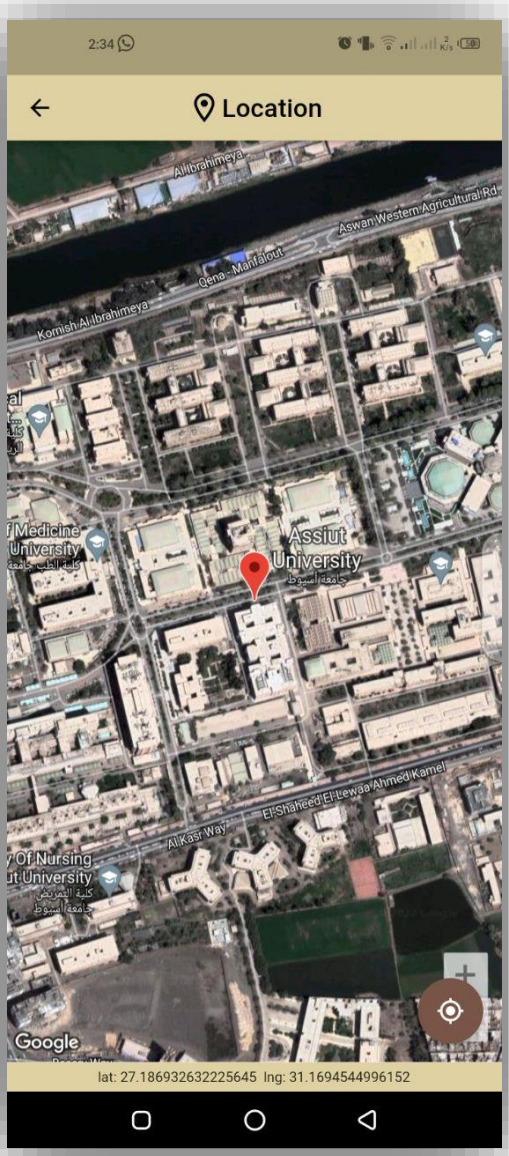


Figure 73: Lab browser view for test2

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.



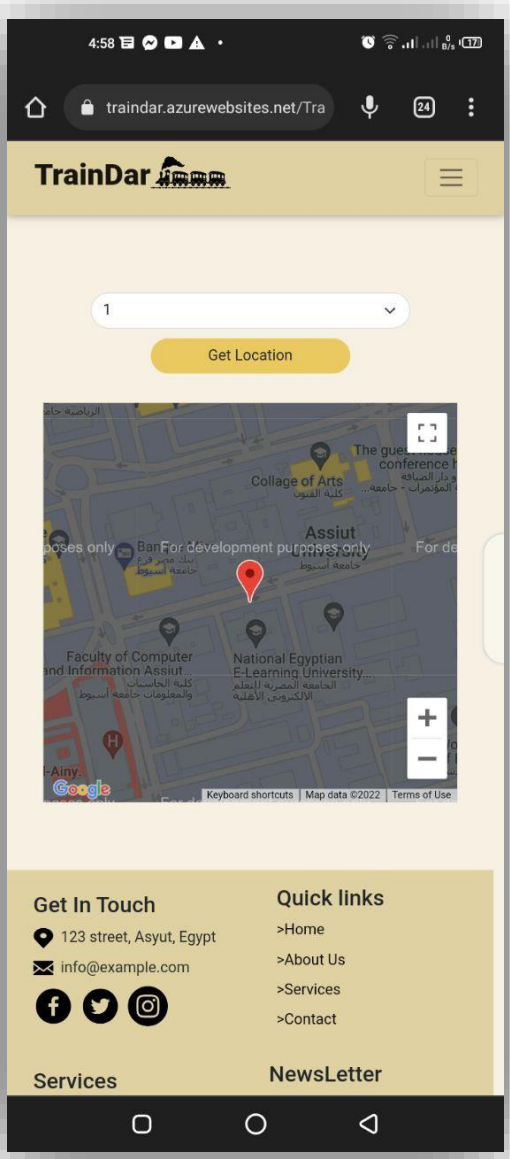


Figure 74, mobile app view for test 2

c. users are outlier, out of range, and correct.

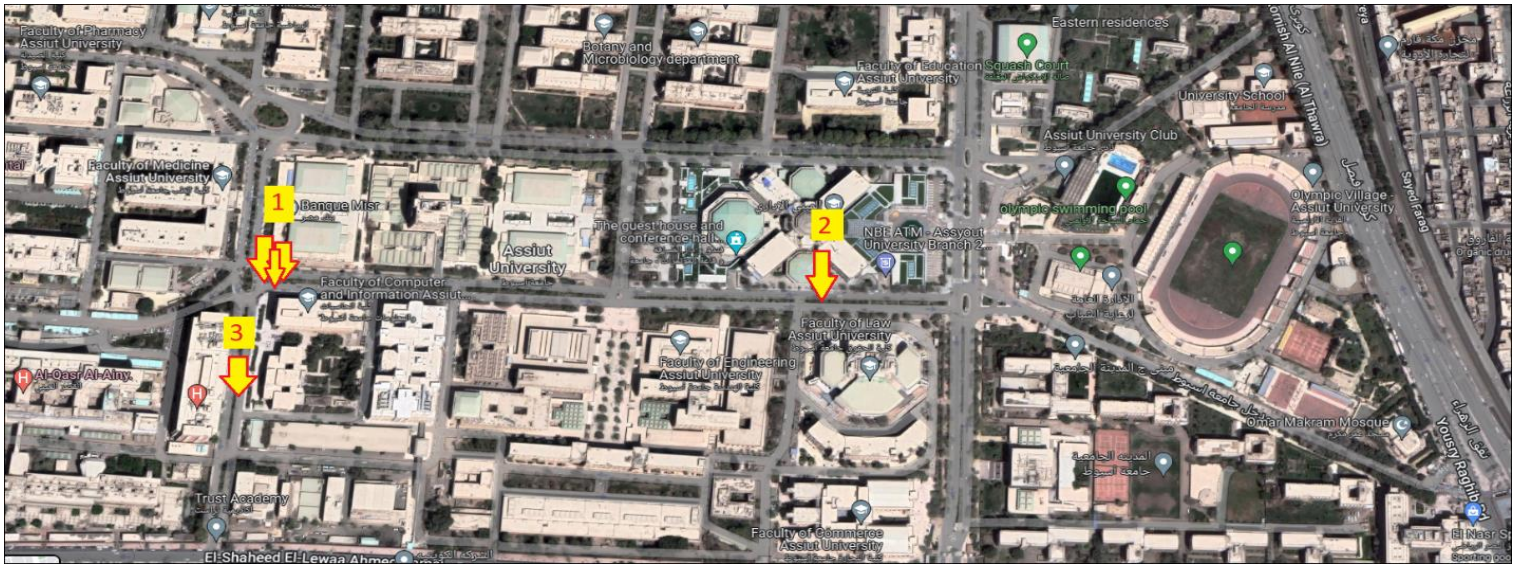


Figure 75: test 3, correct, out of range, and outlier users

For out-of-range users the app will turn of sharing location and ignore it.

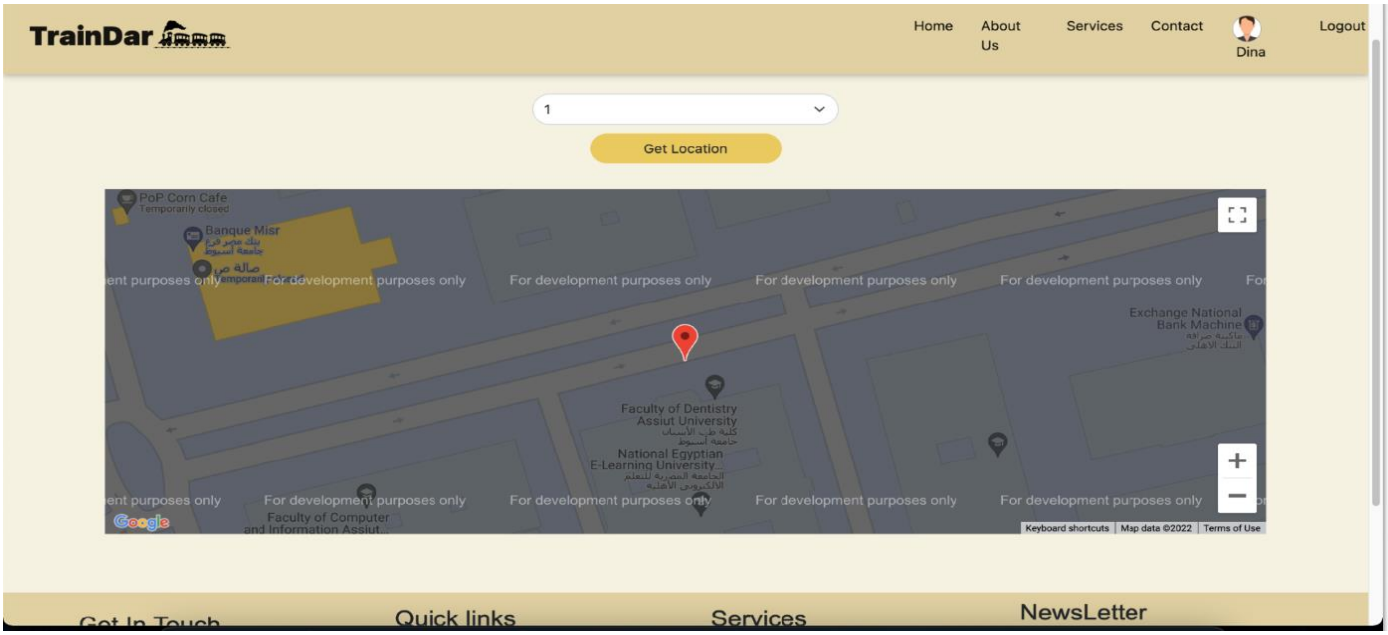


Figure 76, web view in lap browser for test 3

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

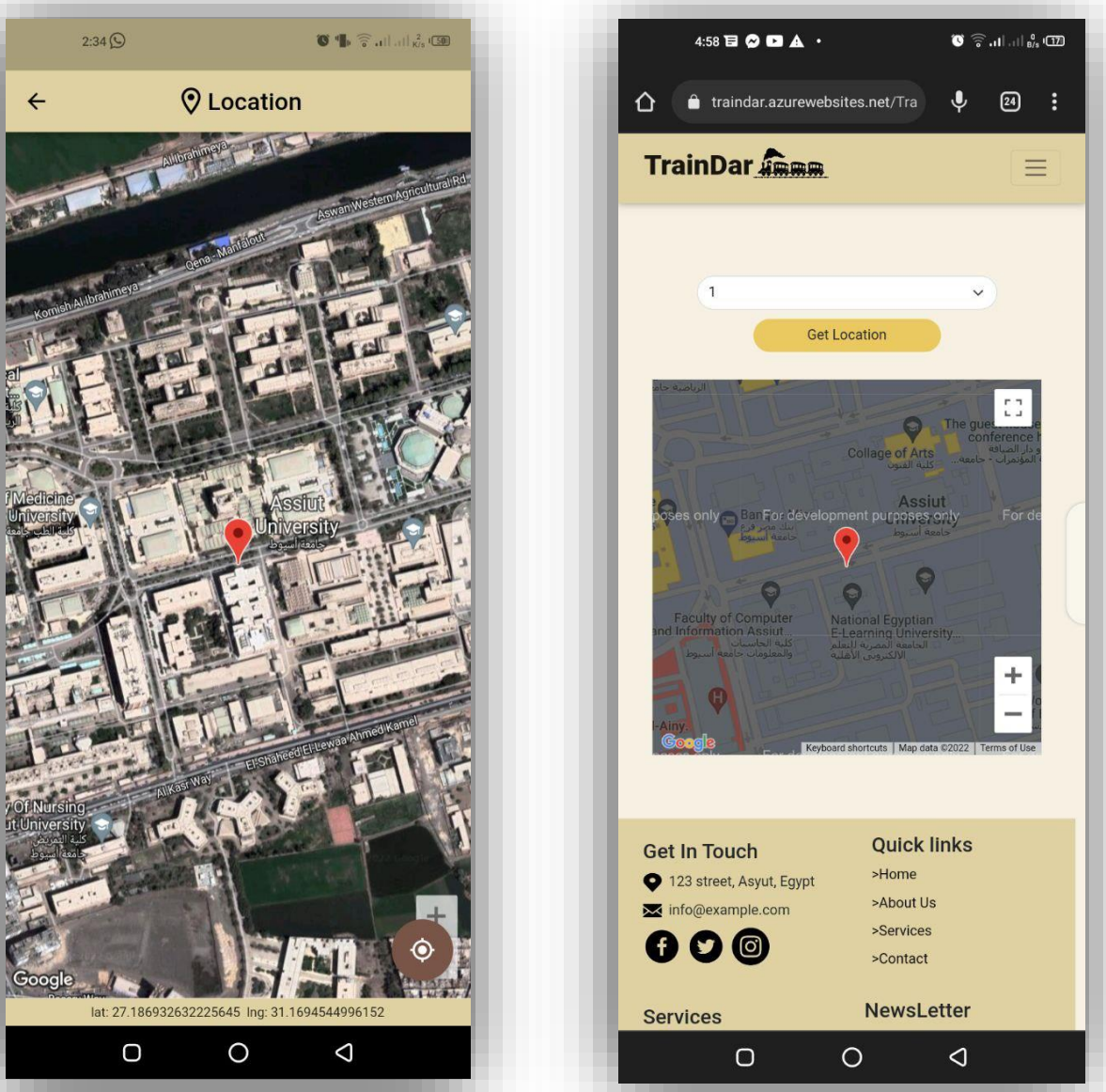


Figure 77: mobile view for test 3

d. All shared users are sharing train 1 and out of range (ignore all).

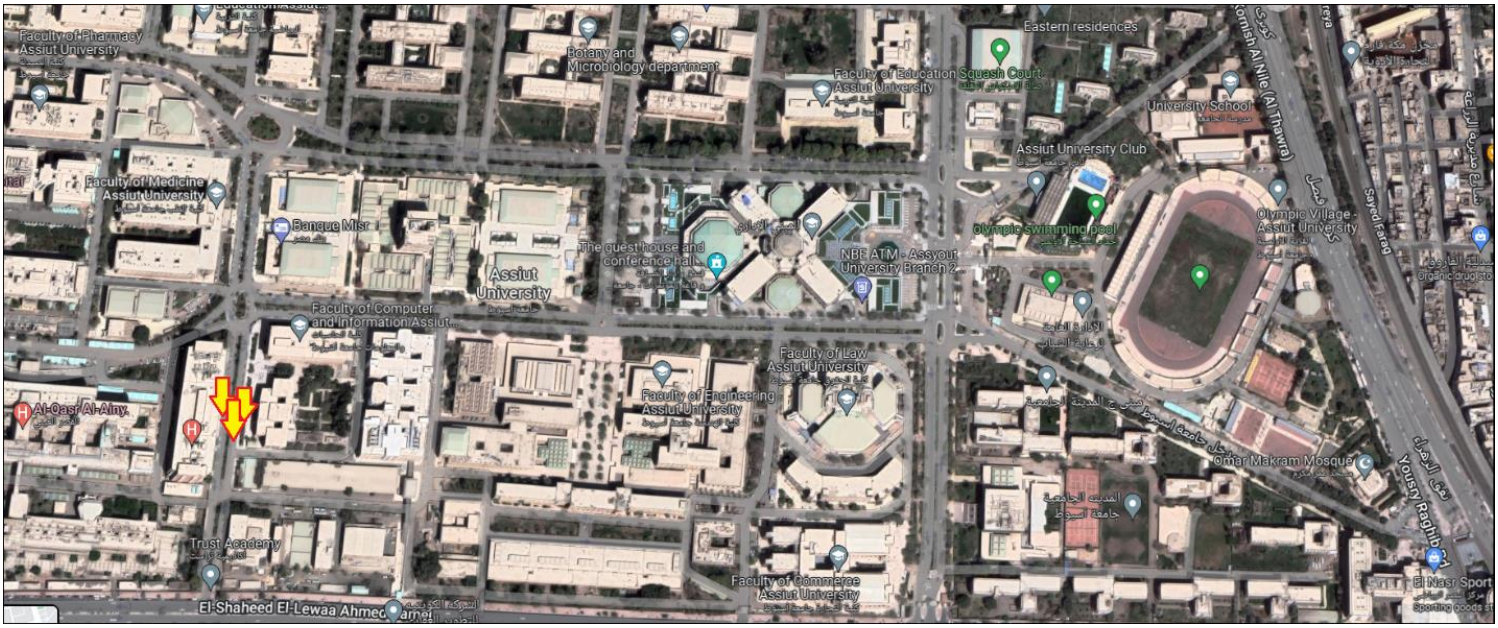


Figure 78, test 4, all users are out of range

At this test, train 1 is not active train so it will not appear in trains list.

Web view:

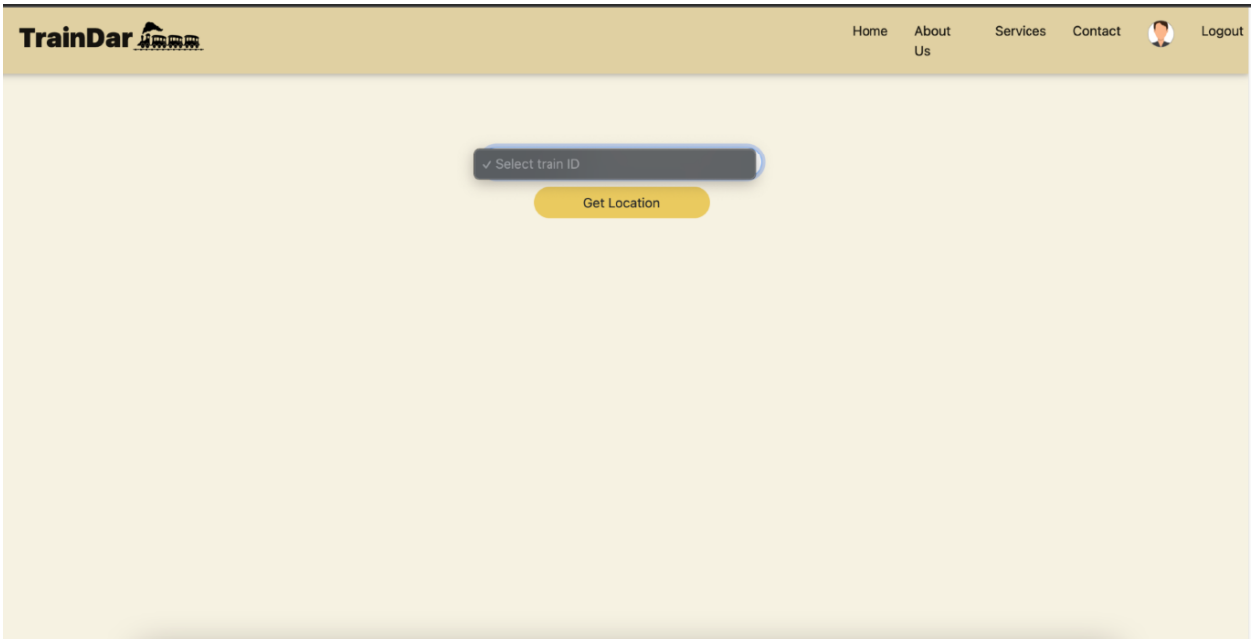


Figure 79: web view in lap browser for test 4

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

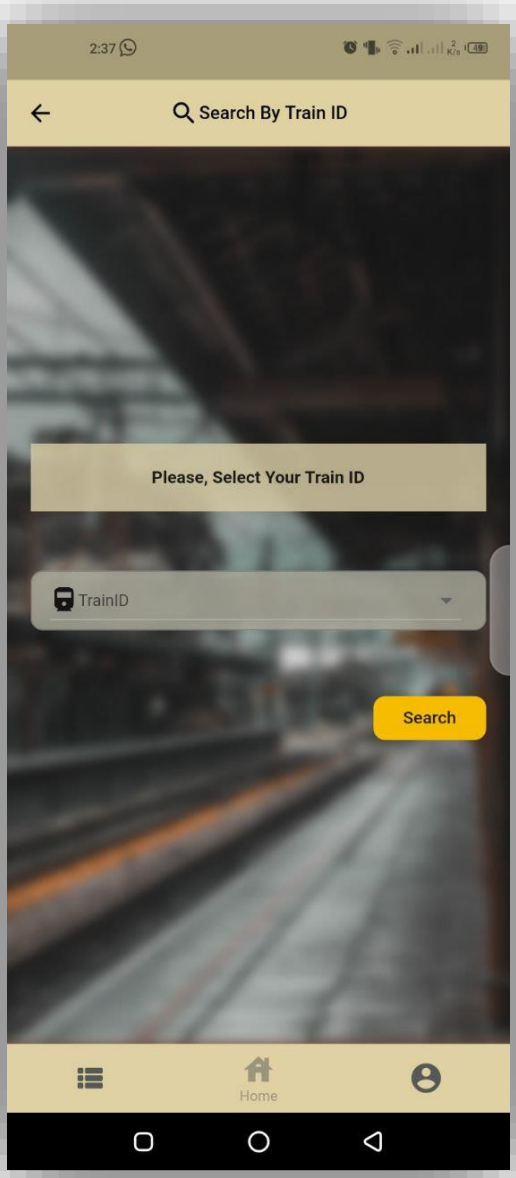


Figure 80: mobile app view for test4

e. If all users are on our railway and share a location for specific train but they are sharing un correct train id, our app can't detect that, and the expected train location and real location will be different. (1 is expected, 2 is real).

Figure 81: test 5, all users share not correct location but not out of range

2-Nearest stations:

For this feature user select train ID

a. Train 1 is before station 2 so it must show 2,3,4

Figure 82: test 6, train 1 is before dentistry station

The web site view:

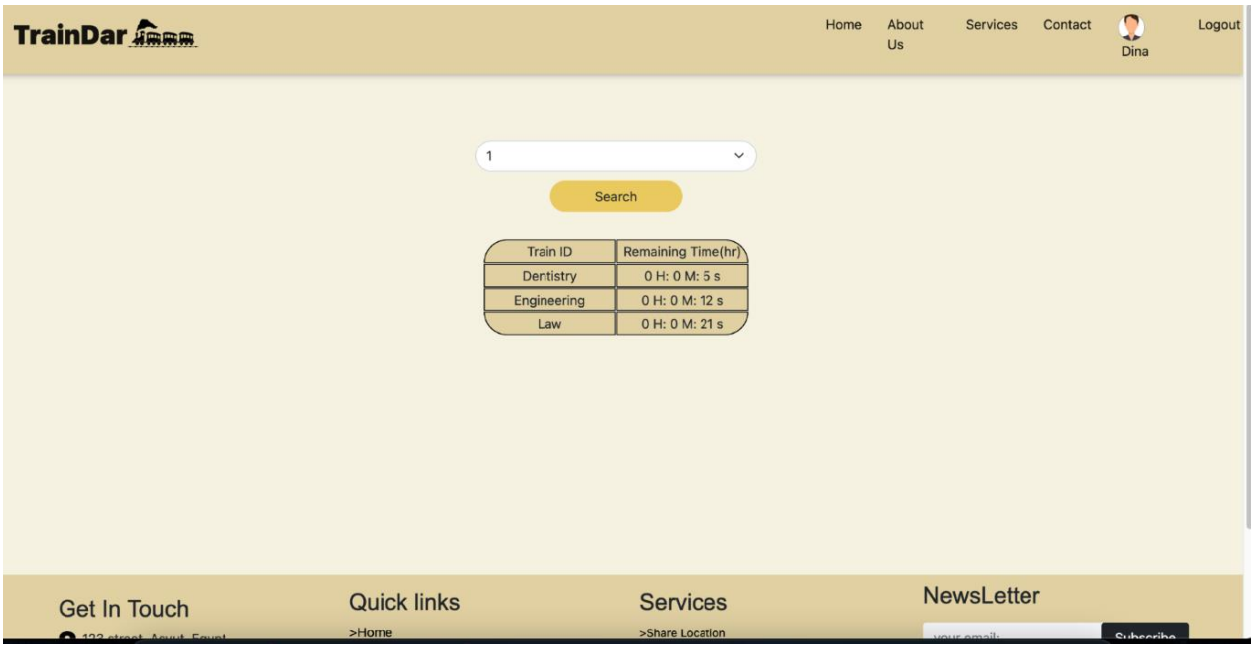


Figure 83: web view in lap browser for test 6

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

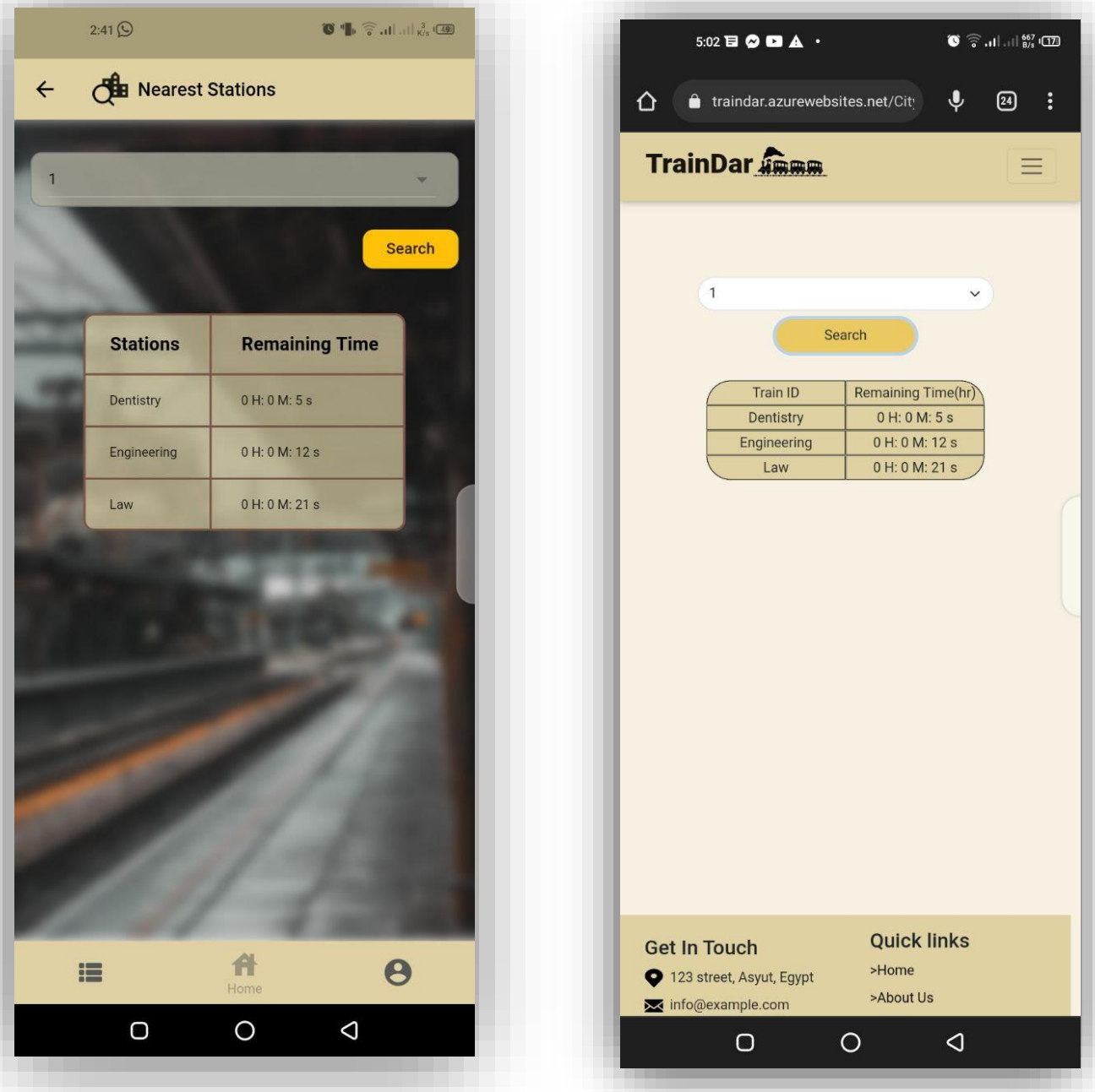


Figure 84: mobile app view for test 6

b. The train 1 is between 2,3 so it must show station 3,4

Figure 85: test 7, train 1 is between station 2 and 3

The web view:

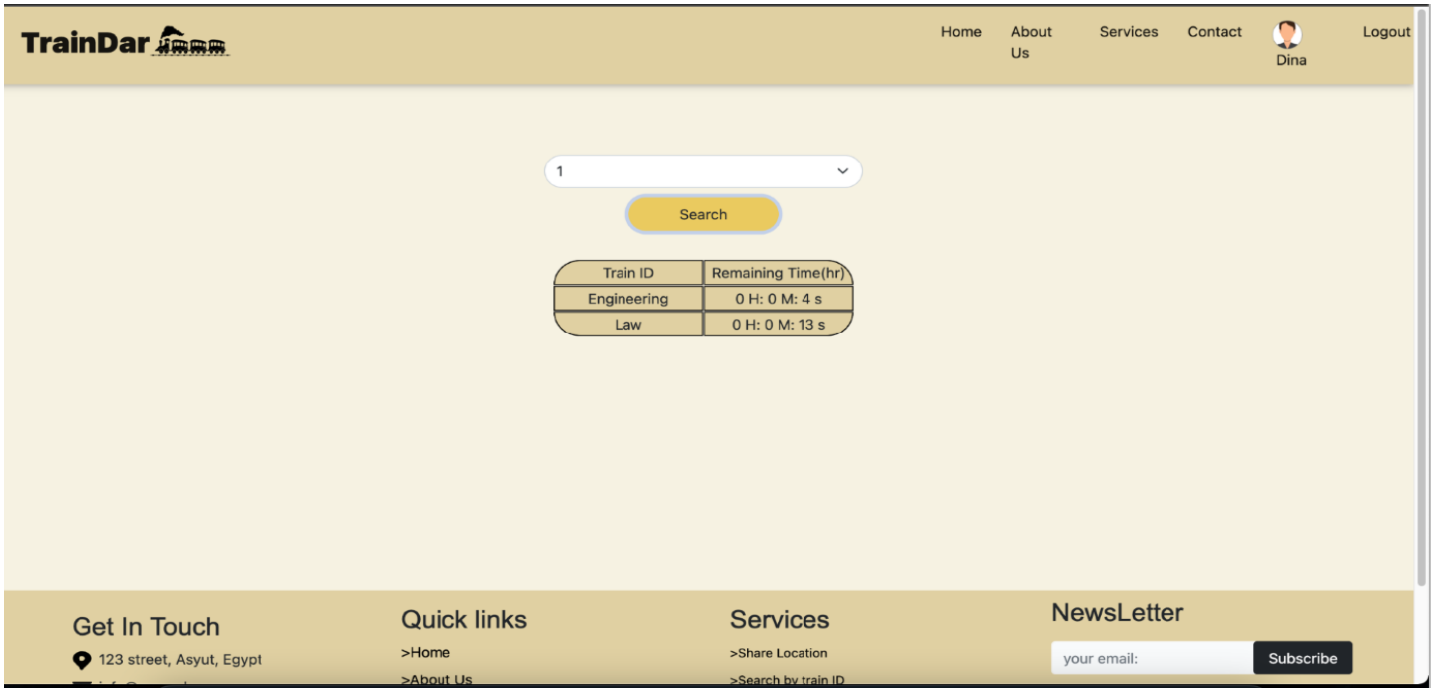


Figure 86:web view in lap browser for test7

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

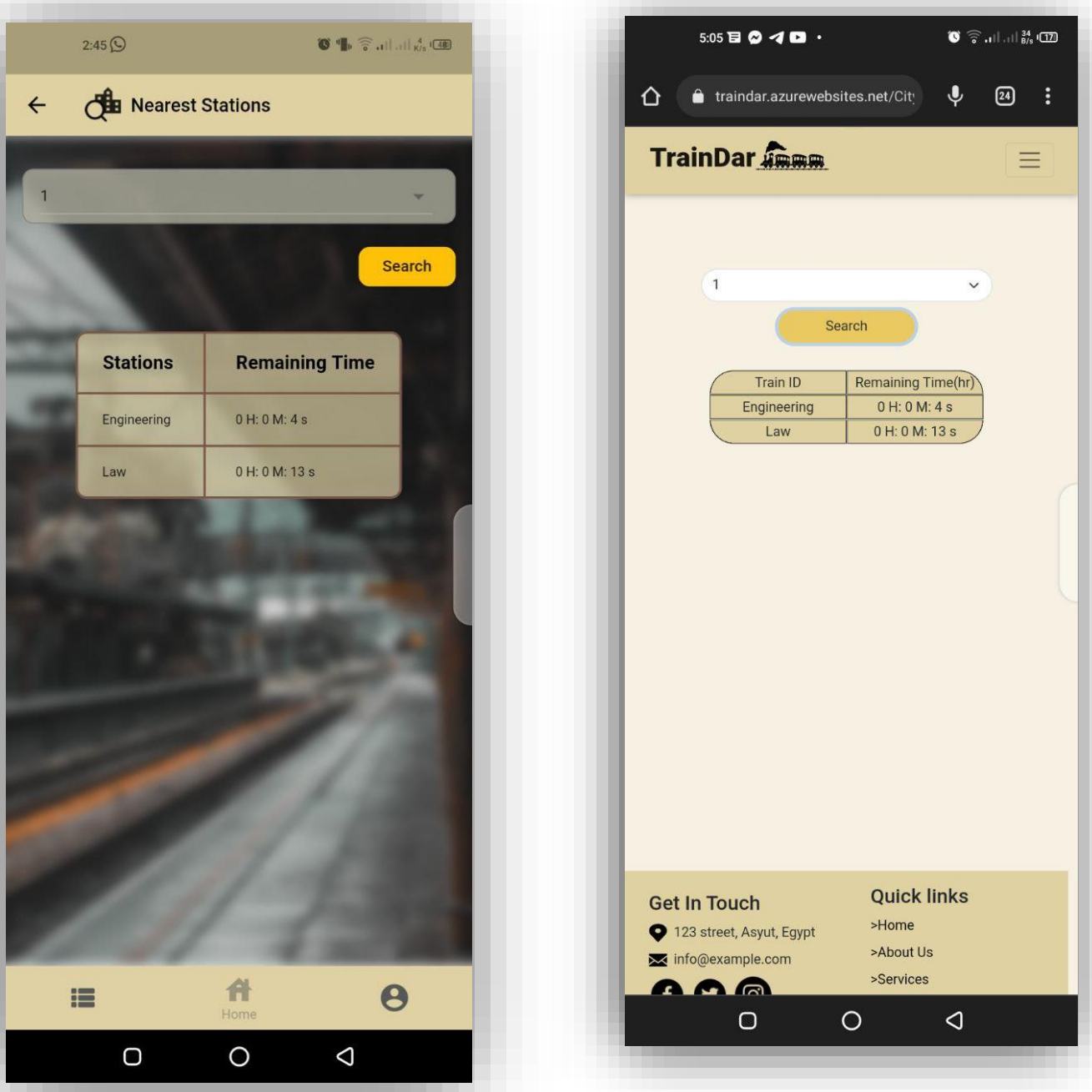


Figure 87: mobile app view for test 7

The train 1 is after station 4 so it will give a message tell there is no stations available.

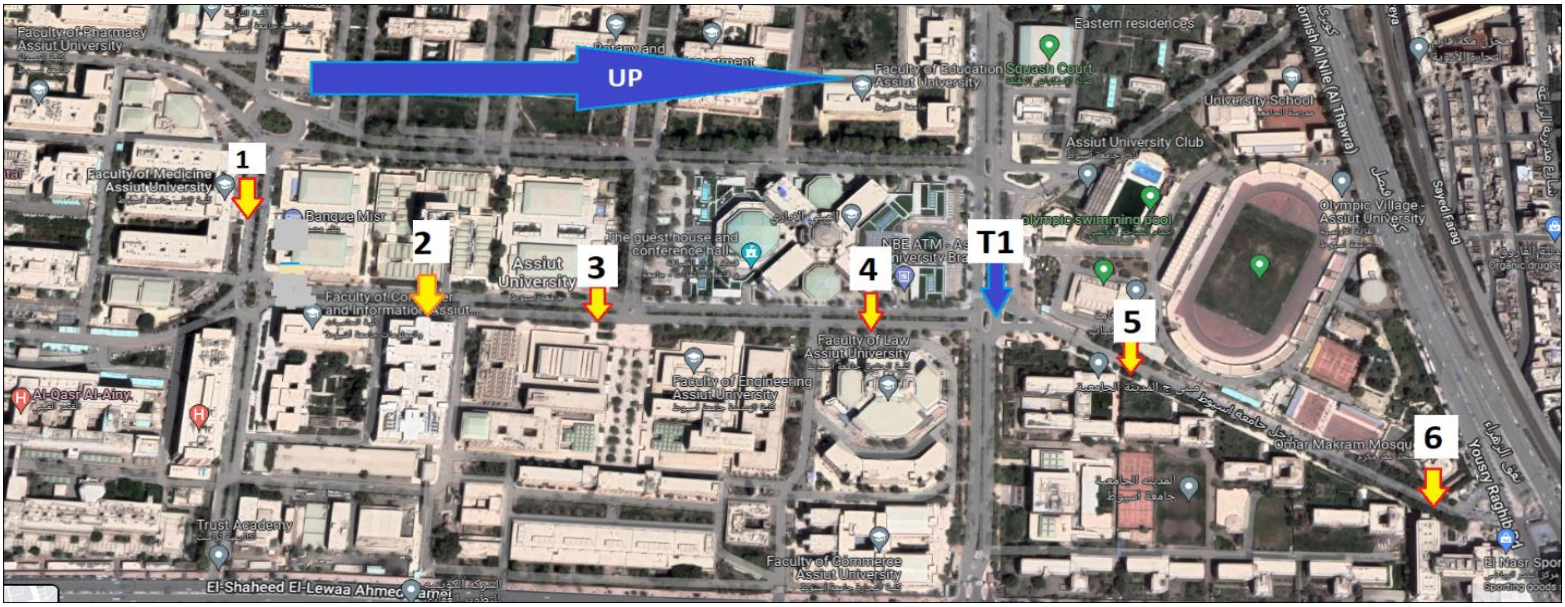


Figure 88: test 8, there is no available stations.

Web view:

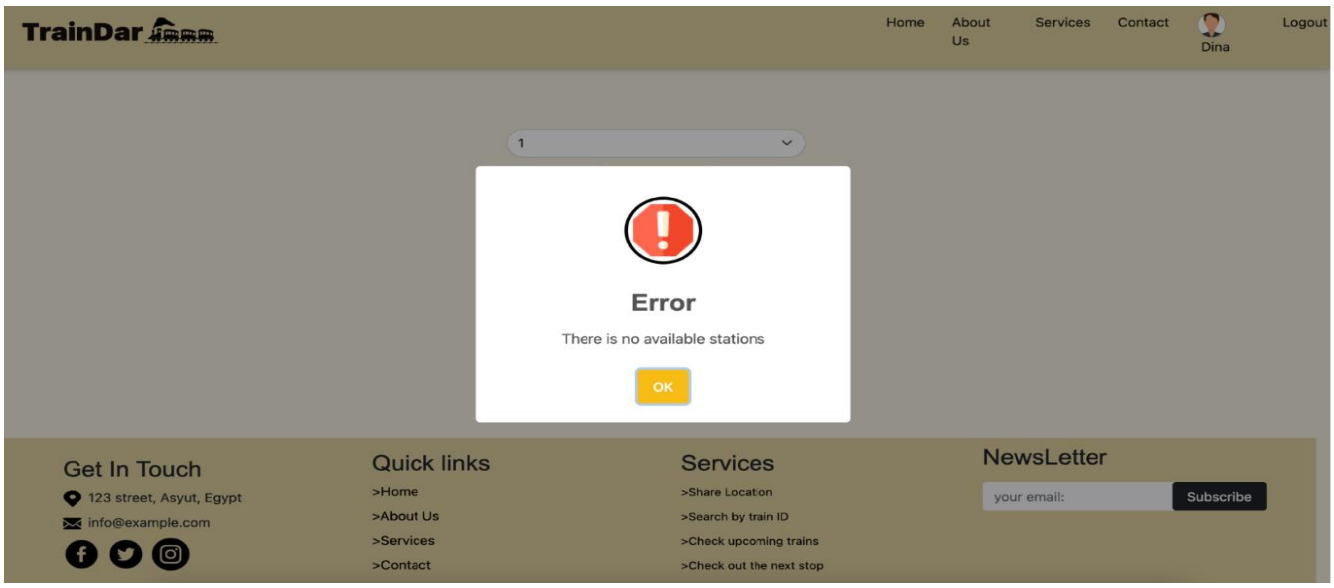


Figure 89: web view in lap browser for test 8

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

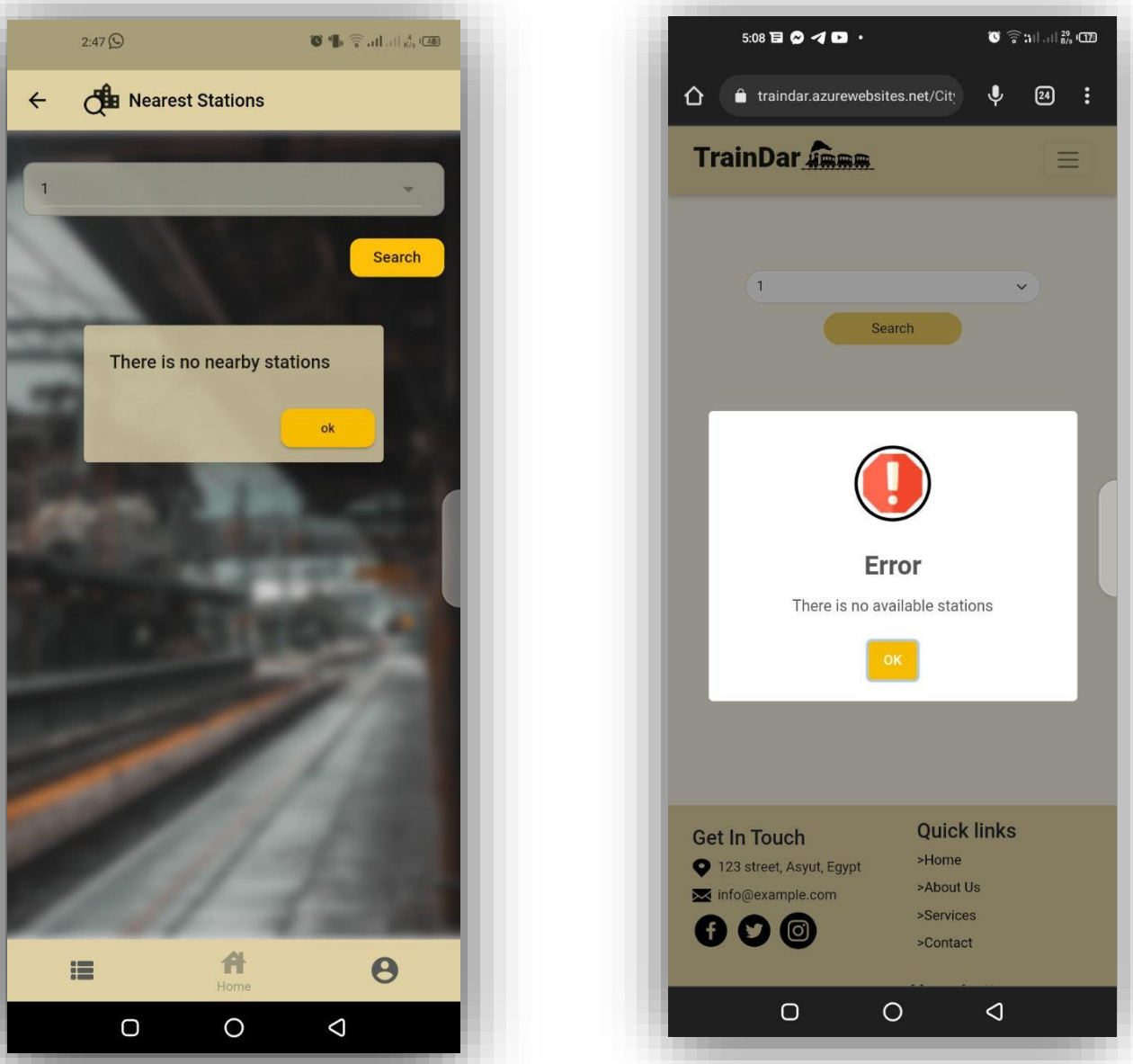


Figure 90: mobile app view for test 8

3-Upcoming trains:

we have train 1,2 up, 3 down, 2 doesn't pass station 4. It will show train 1, start station is station 3, and the destination is station 4.

Figure 91: test 9, upcoming trains

Web view:

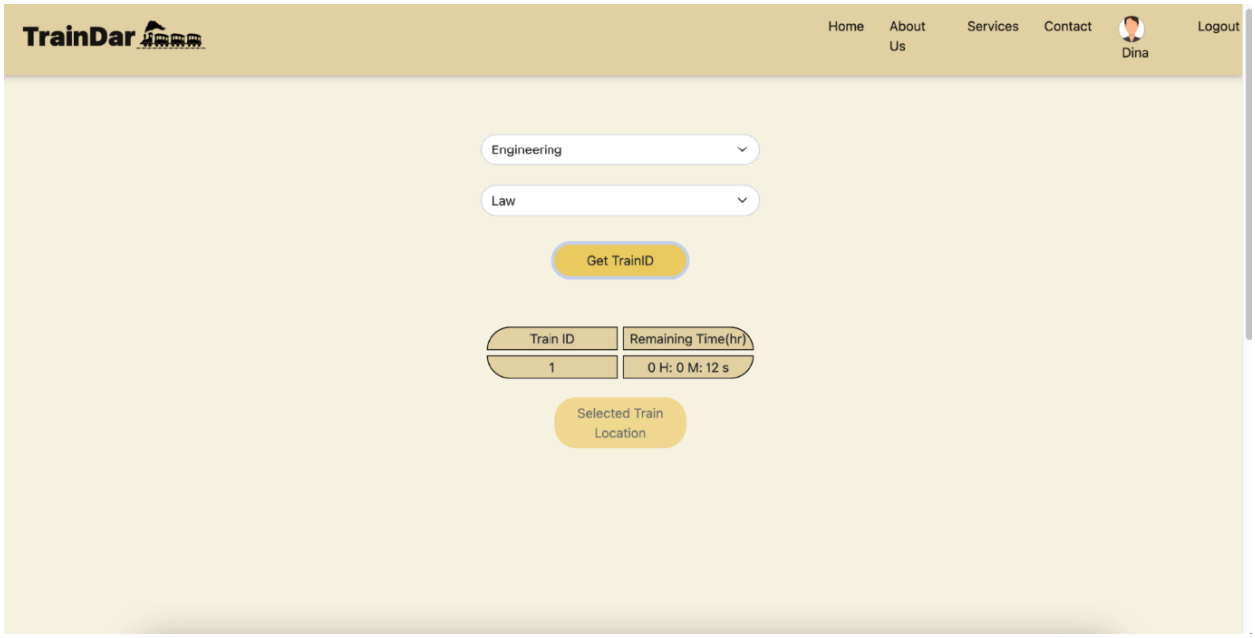
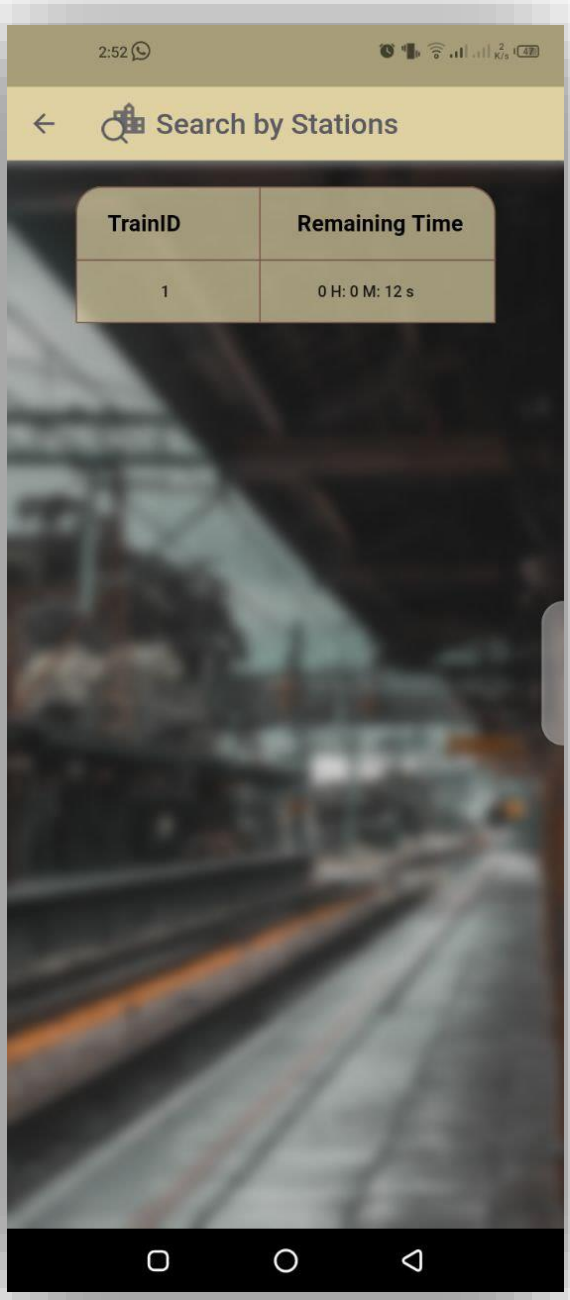


Figure 92, web view in lap browser for test 9

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.



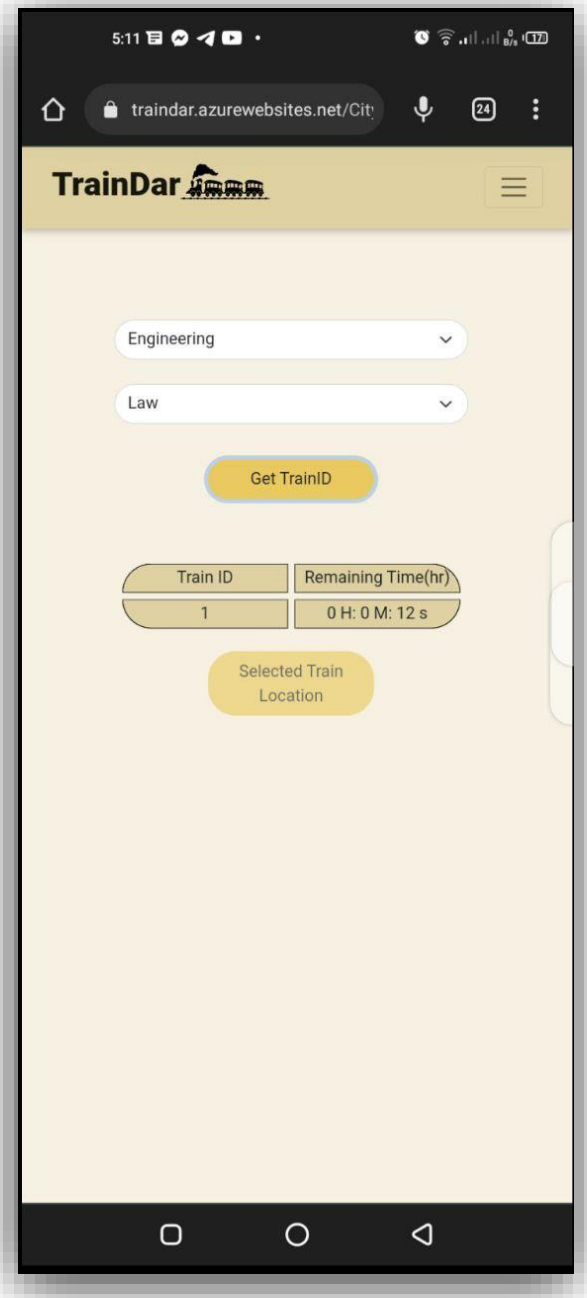


Figure 93: mobile app view for test 9

If the user selects a train from the result list, it will open the location of train at the map.

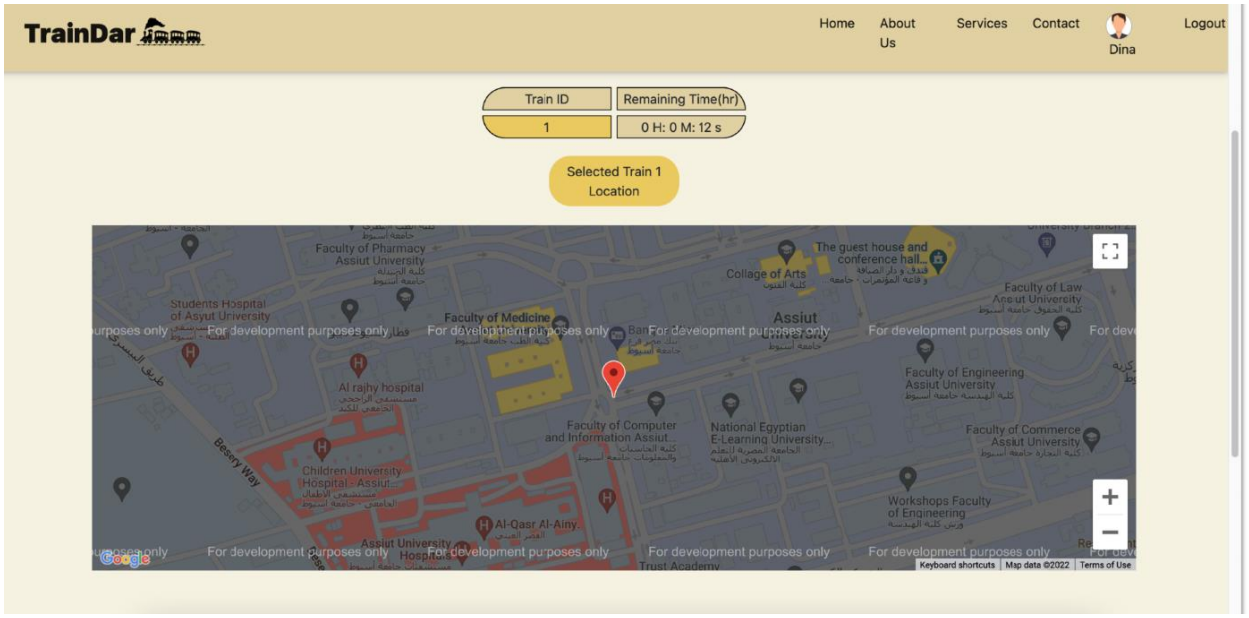


Figure 94: web view for the selected train

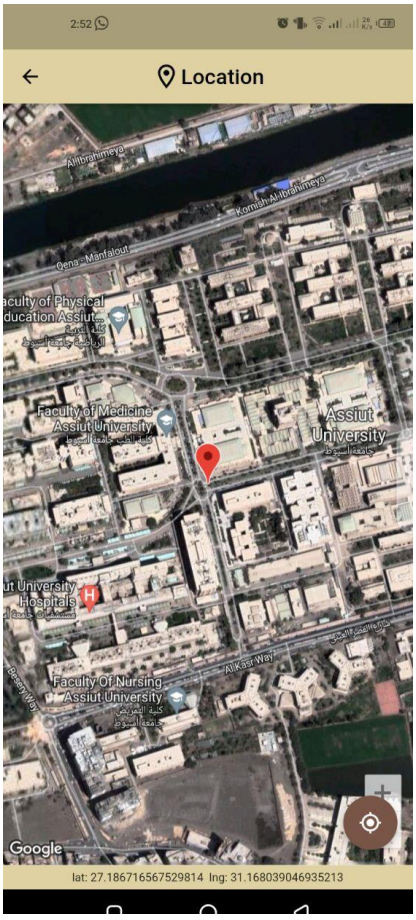


Figure 95: mobile app view for selected train id

b. We have train 1,2 up, 3 down. It will show train 1,2 sorted by the time left. Start station is station 2, and destination is station 3.

Figure 96: test 10, more than 1 train sorted

Web view:

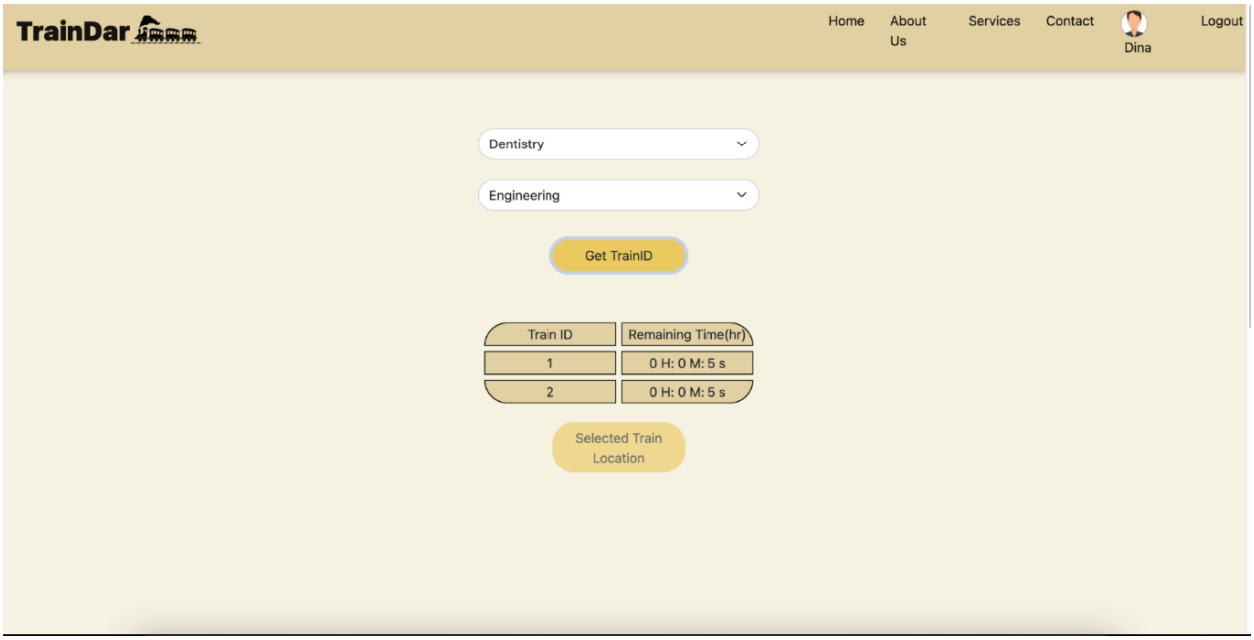
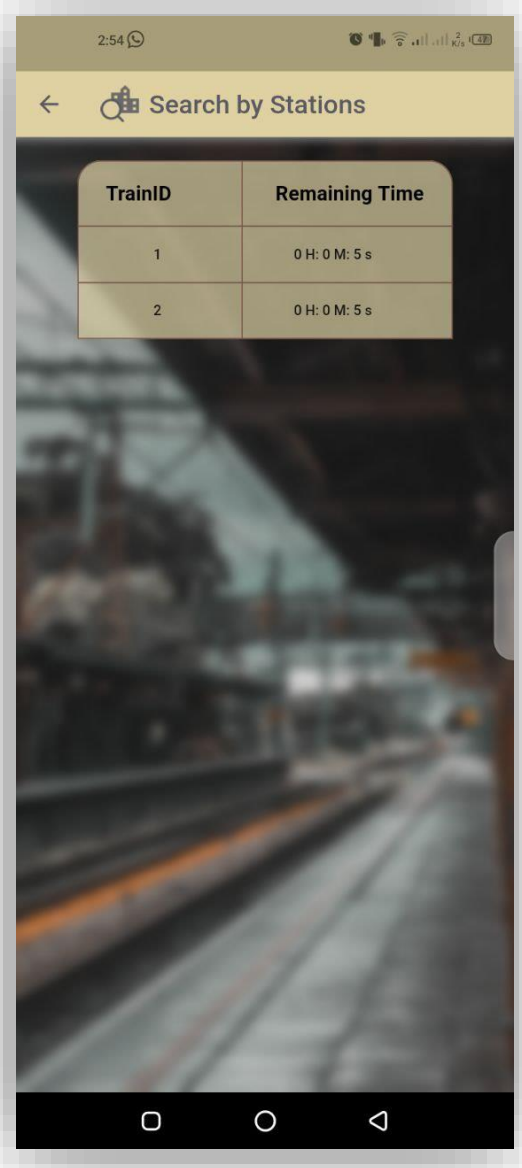


Figure 97: web view in lap browser for test 10

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

Note that you can select a train and view it.



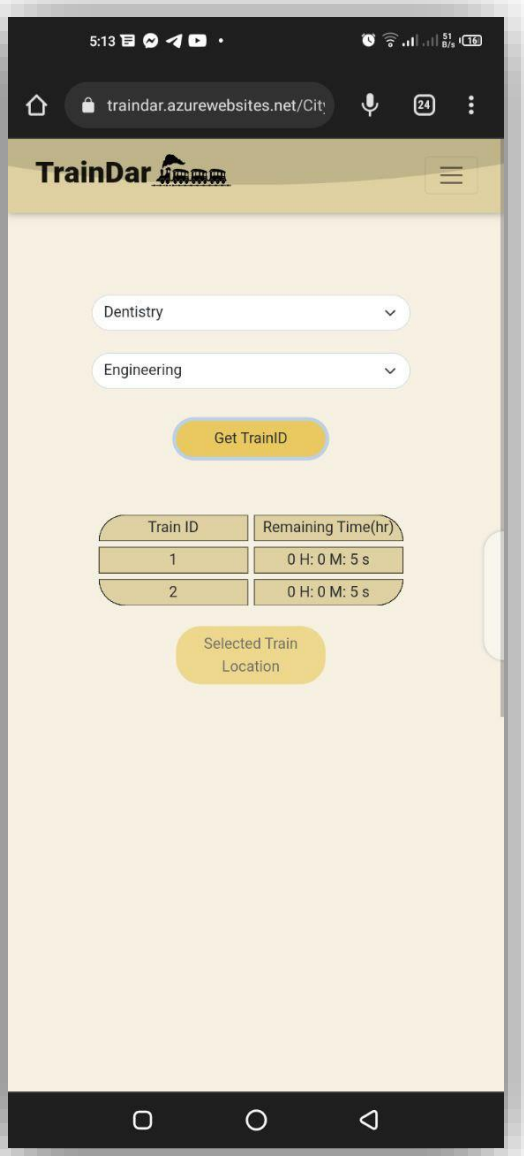


Figure 98: mobile app view for test 10

c. No trains available. Start 3 and the destination is 4.

Figure 99: test 11, there is no available trains

Web view:

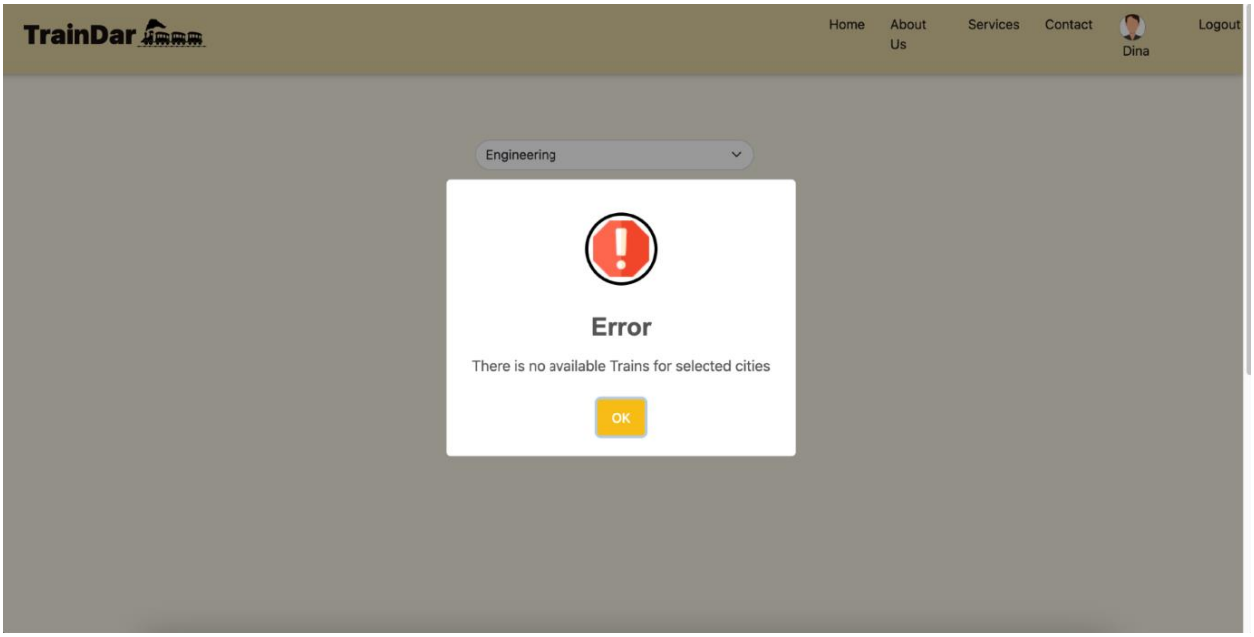
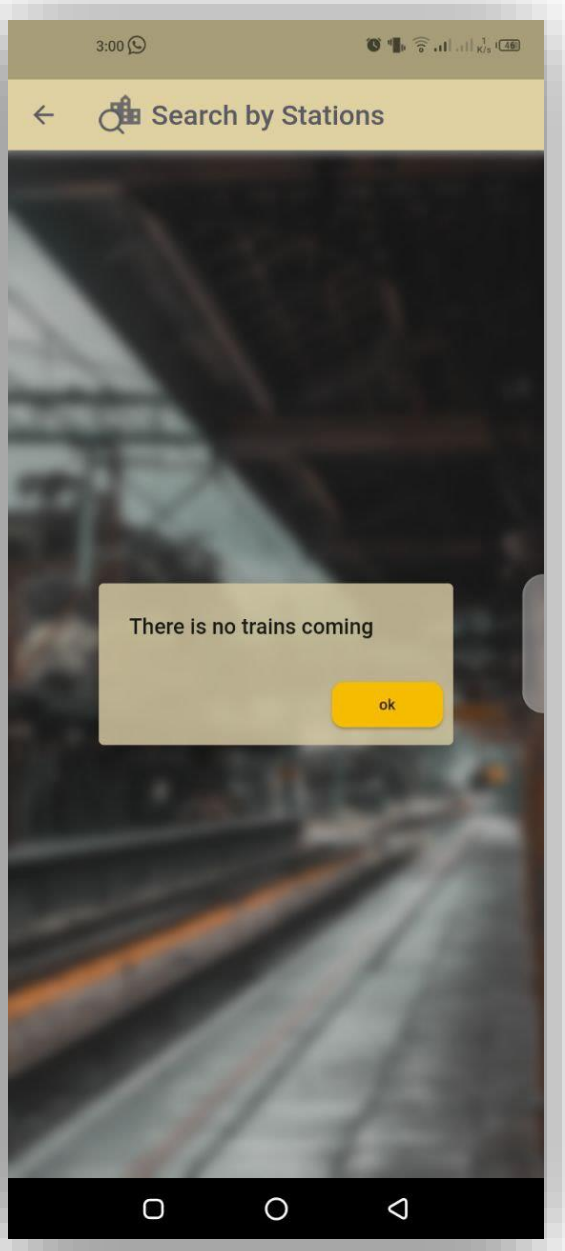


Figure 100: web view in lap browser for test 11

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.



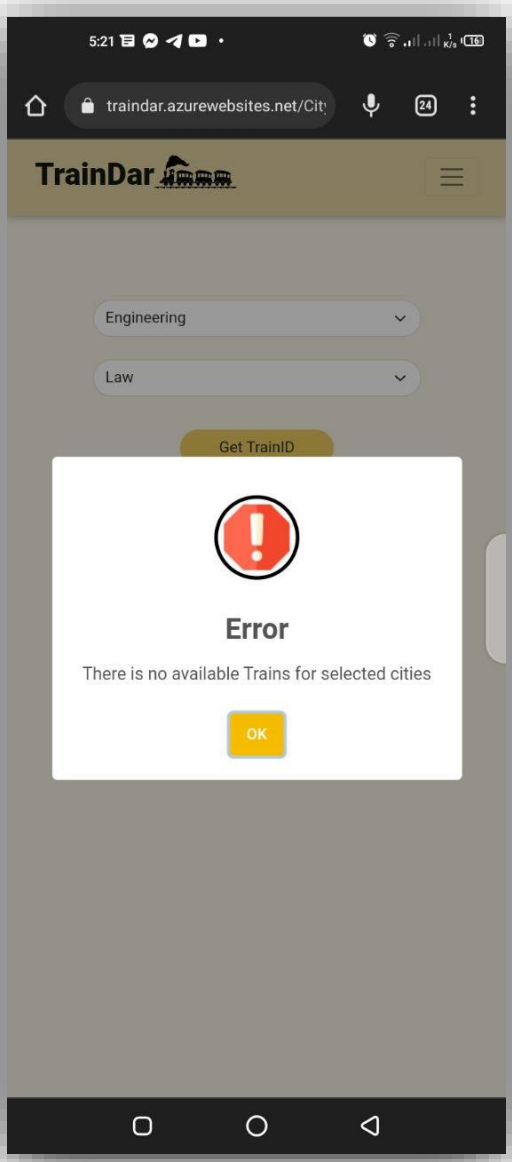


Figure 101: mobile app view for test 11

c. It will show no available trains with station 3,4.

Figure 102: test 12, no available trains (train 2 don't pass throw station 4)

Web view:

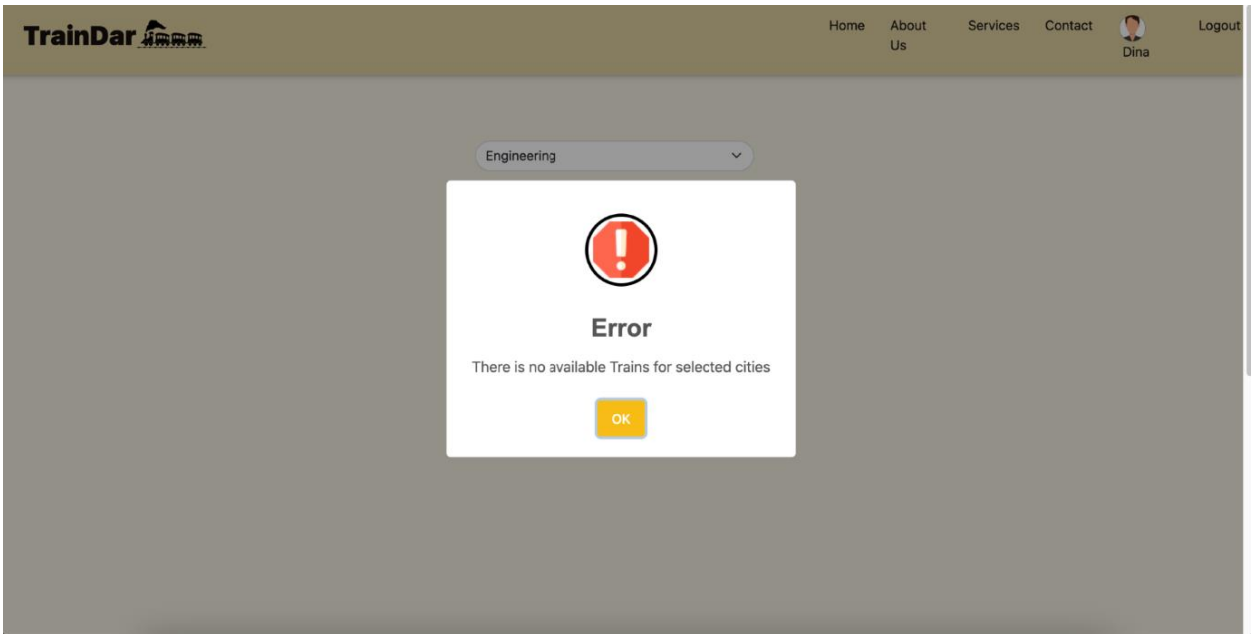
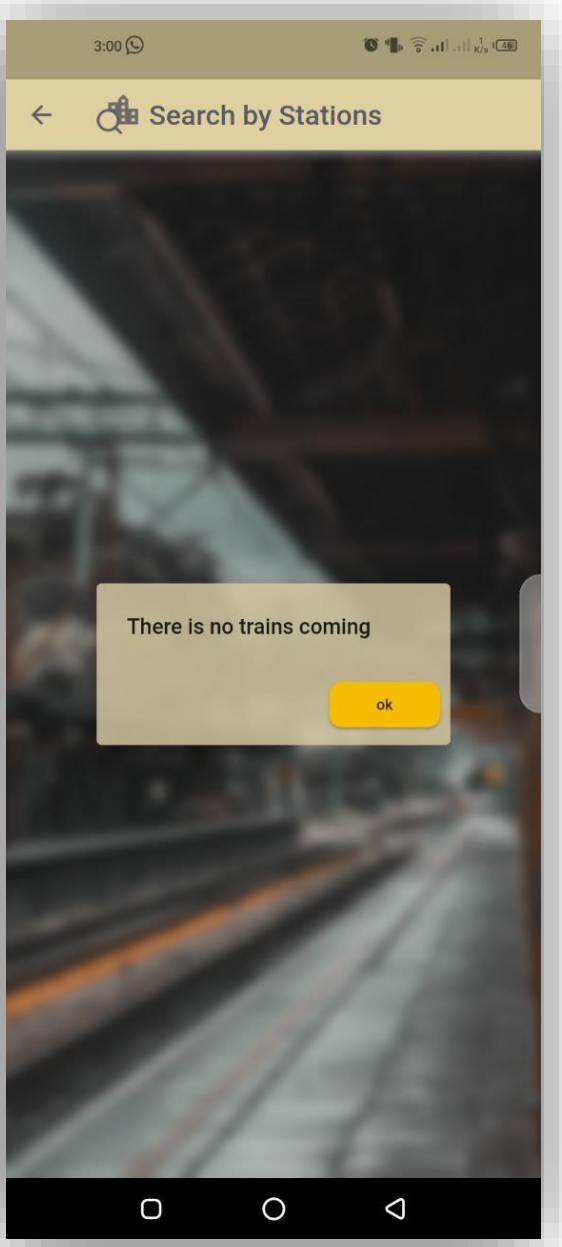


Figure 103: web view in lap browser for test 12

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.



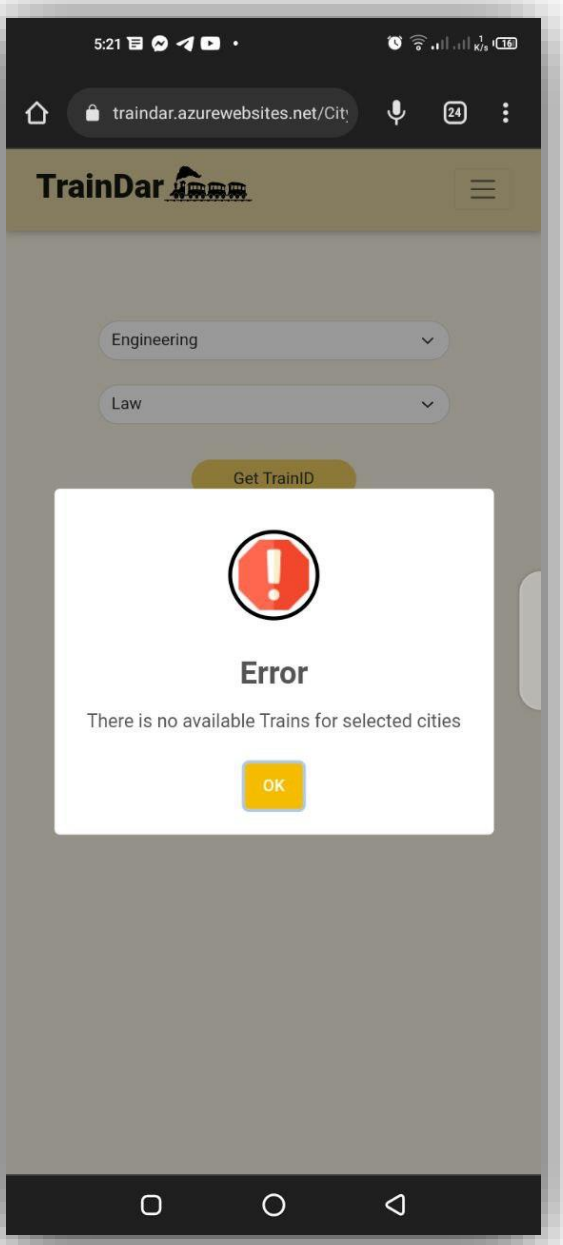


Figure 104: mobile app view for test 12

d. Note : train 5 pass throw station 3,4 but the direction is 4,3. It will show only train 1.

Figure 105: test 13, train 5 is down

Web view:

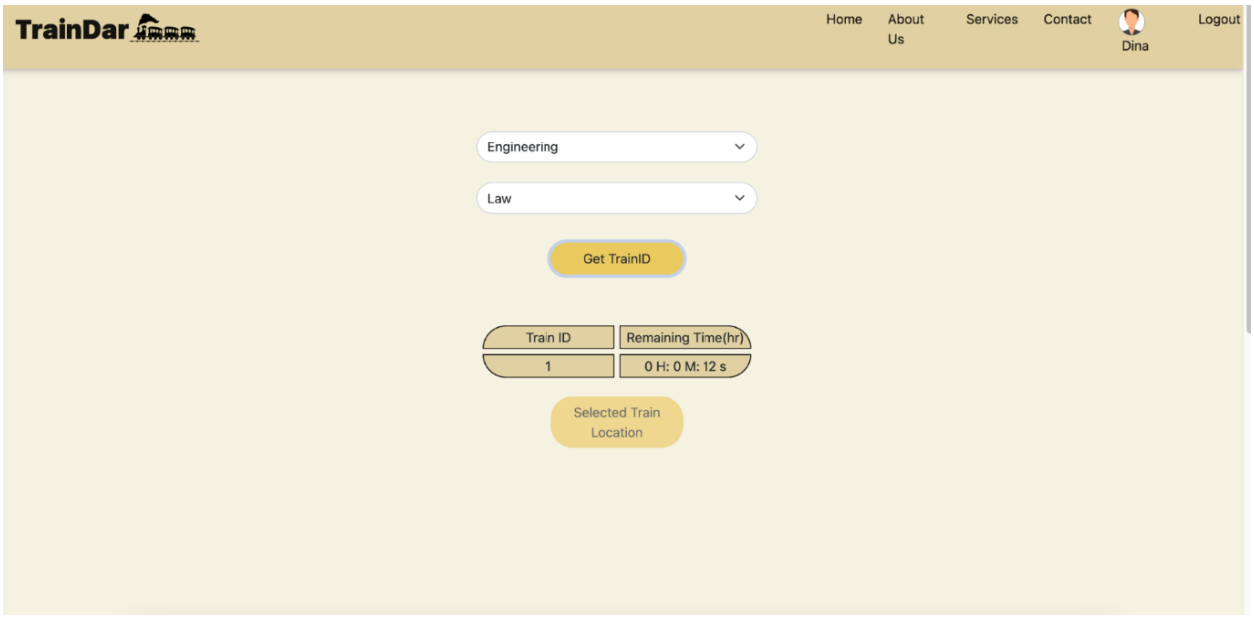


Figure 106: web view in lap browser for test 13

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.

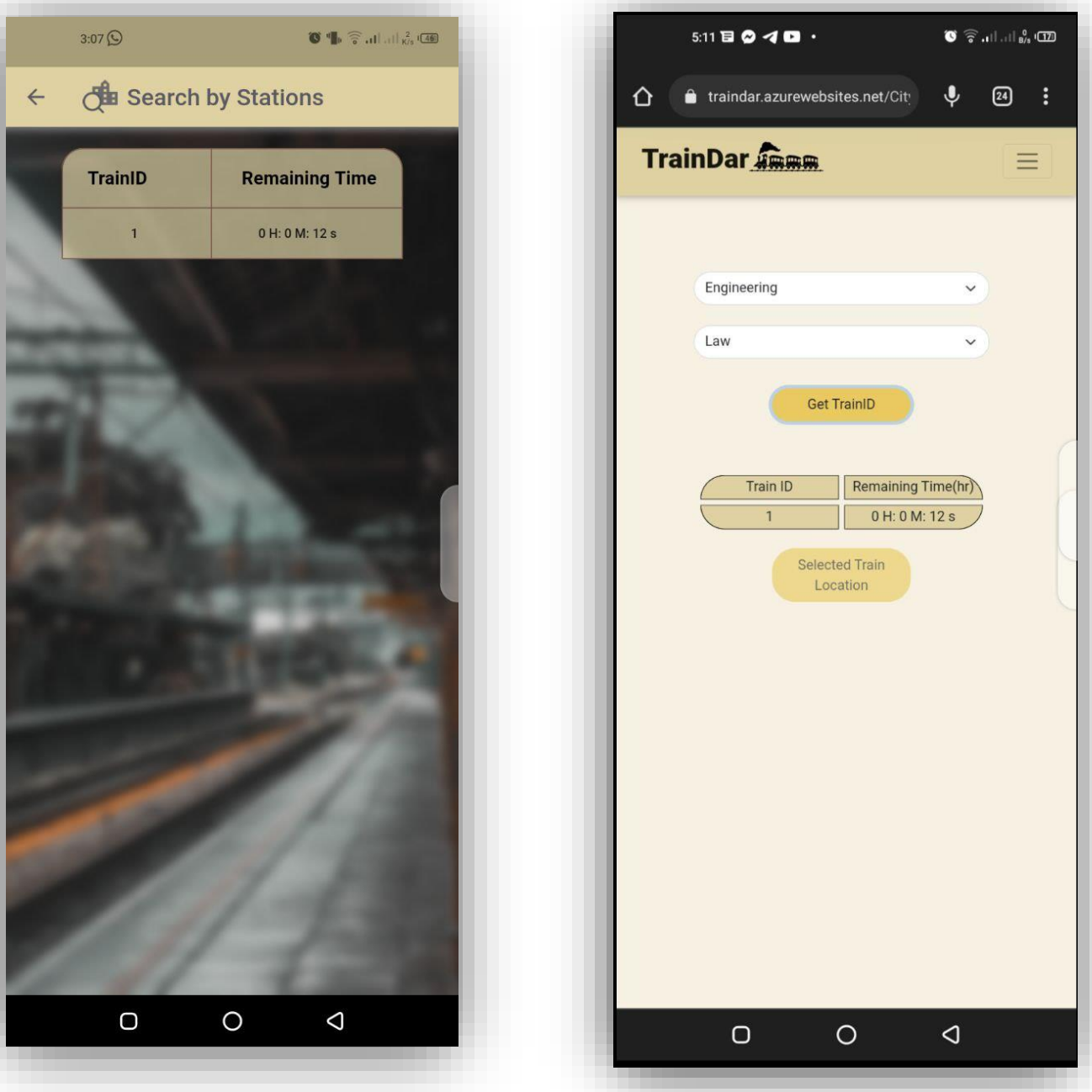


Figure 107: mobile app view for test 13

f. Train1,2 up, 3 down. It will show train 1,2 sorted by the time left. Start station is station 2, and destination is station 3.

Figure 108: test 14, for sorted results

Web view:

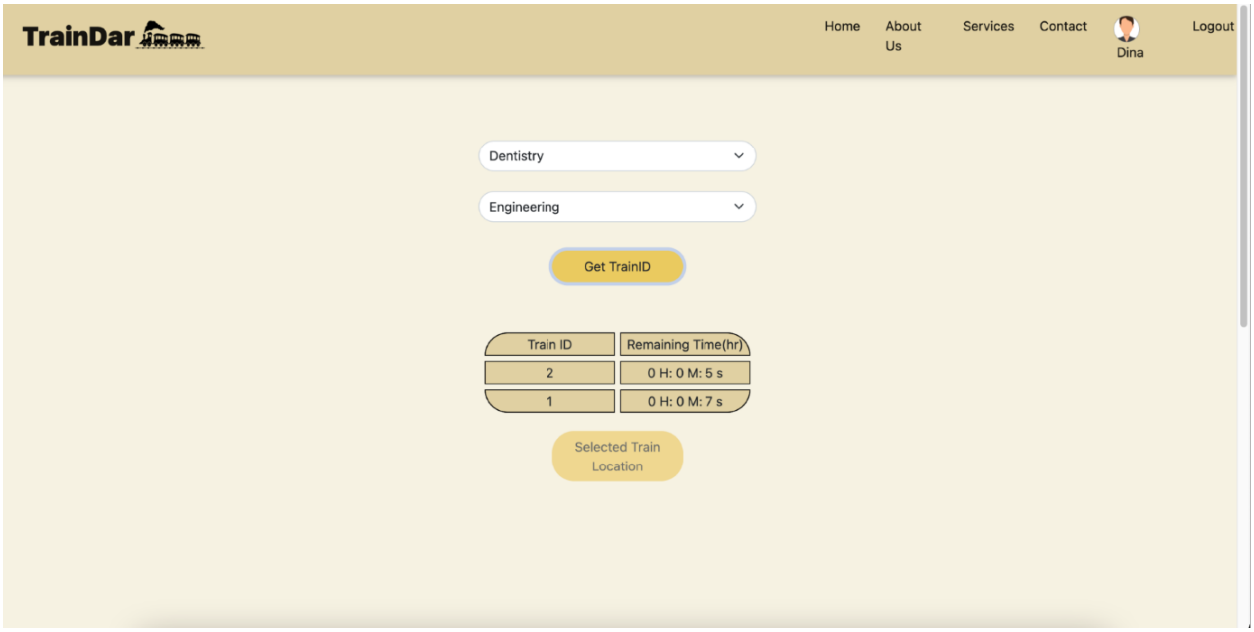
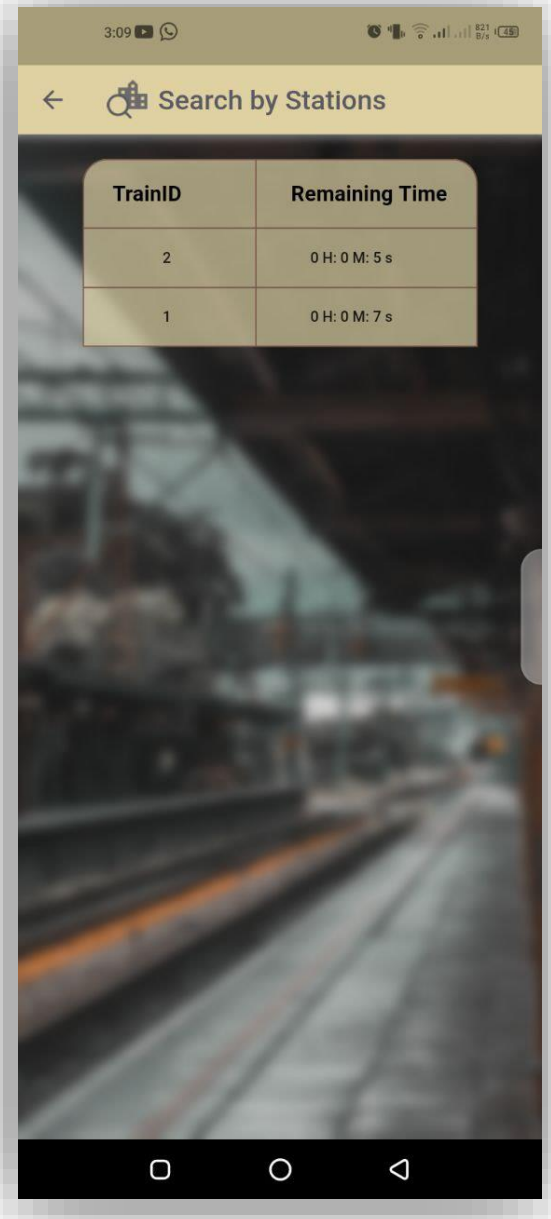


Figure 109: web view in lap browser for test 14

The figure … shows the results at mobile browser, the figure … shows the app results in mobile.



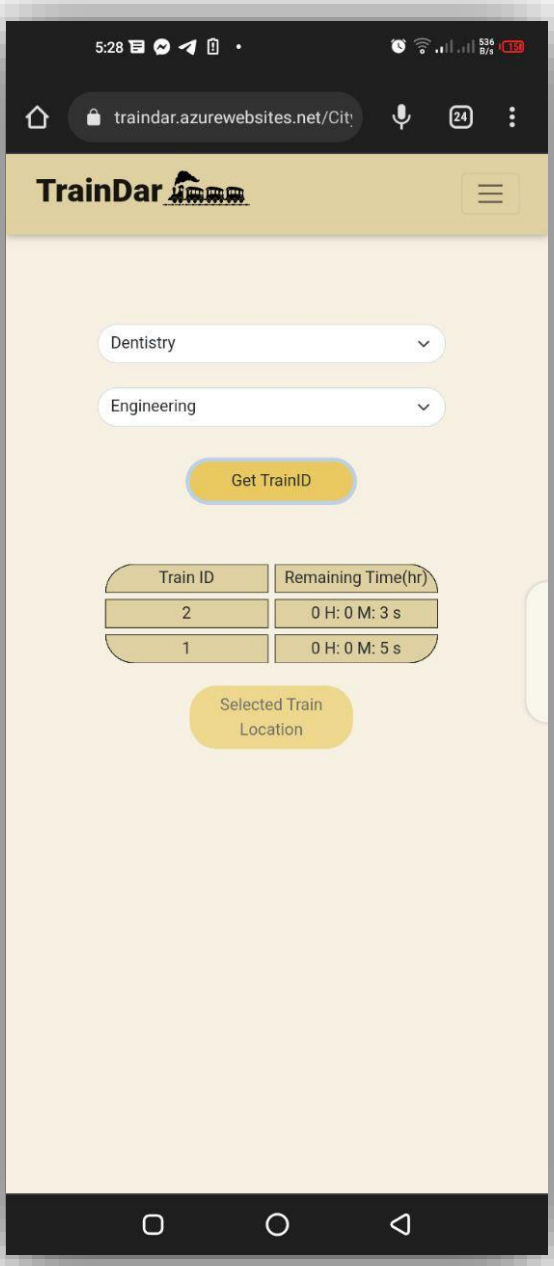


Figure 110: mobile app view for test 14