EUROPEAN UNIVERSITY OF LEFKE

FACULTY OF ENGINEERING

Graduation Project 2

Employee attendance and manage control using QR Codes

Manasbek Rakhmatulloev

184121

Attendance taking is one of the main priorities of the companies, because it affects productivity of the company, productivity of employees. There are many different ways to track employee attendance such as manually entering employee hours into a spreadsheet, card swiping, fingerprint and retina scans, tracking employees through a GPS system, a traditional attendance register and QR code scan. Among them QR code attendance becoming very popular, because of its simplicity, low cost, accessibility and fast attendance time.

There are two types of QR code attendance systems:

- First one is giving every employee a card with individual QR codes on it. Employees show this card to special device every time when they are attending. But it has many drawbacks. Among colleagues and acquaintances, these cards can be exchanged or copied to log in with their details and then mark their attendance. It is also depends on electricity and Wi-Fi.

- Second one is using employee's smartphone for scanning one QR code that was provided on the screen, door or wall. Employees scan QR code in special app on their smartphones. There

is no need for special devices. It works even when there is no electricity or Wi-Fi. Employees can't cheat in it. Employees can see their attendance results at any time on their smartphones.

Employee attendance and manage control using QR Codes uses the second approach (type), because it is more efficient and effective method than the first one.

Supervisor

Ezgi Deniz Ulker

Publish Date

08.06.2022

Table Of Contents

1. Introduction

Smartphones play an important part in our daily lives in this technological age. Smartphones can now handle the majority of problems fast and easily. With many social apps, business apps, problem-solving apps, educational apps, and marketing apps, it has made everyone's life simpler and easier. QR code attendance system for employee app is one of them. It uses employee's smartphone for scanning QR code that was provided on the screen, door or wall. There is no need for special devices. It works even when there is no electricity or Wi-Fi. Employees can't cheat in it. It uses GPS, unique device id and user registration by the authority of the administrator to make it secure. Employees can see their attendance results at any time on this app.

1.1 Problem definition

There are many other employee attendance systems.

They have many drawbacks:

- Slow Attendance Taking
- More Paper Waste
- Special Equipment Required
- Depends on Electricity and Wi-Fi
- Assembling and Disassembling is hard
- Special Training for employees
- Not Secure

1.2 Goals

The purpose of the project is to make attendance system simple, low cost, accessible, fast and secure.

- **Faster Attendance Taking:** It takes only a few seconds per employee for checking in.

- **Least Paper Waste:** There is no need for papers, because the whole attendance data will be stored in the online cloud. This data can be accessed at any time.

- **No Special Equipment Required**: No need for special equipment. It only needs smartphone and generated QR code for attending.

- **No Electricity:** It works even when there is no electricity or Wi-Fi.

- **Easy Assembling and Disassembling:** There is no need to worry about arranging laptops, staff members and stationery. It is easy to use a smartphone as a QR code scanner via mobile apps.

- **Simple Pre-preparation:** There is no need for special training for the employees, because smartphone QR code scanners are simple and easy to use.

- **Secure:** It is very hard cheat or hack system. It uses GPS, unique device id and user registration by the authority of the administrator for that.

2. Literature Survey

Using punch cards, log books, fingerprint systems, barcodes and QR codes among the different types of attendance systems that have been created, nevertheless causes several issues, such as supplying inaccurate information to users. The goal of a smartphone-based QR code attendance system is to computerize the conventional method of recording attendance and give an easy and efficient approach to track attendance at institutions today. Smartphones are the most prevalent device used in marketing and business.

- **Compare1:** It does not provide individual QR codes to its users like many other QR code attendance systems, instead it uses one QR code for all users for attending.

- **Compare2:** It uses smartphone for scanning the QR code that was provided, unlike many other systems which use individual cards and special equipment for attending.

- **Compare3:** Employees can see their attendance results through application on their smartphones, unlike many other systems in which employees don't know anything about their attendance.

- **Compare4:** It works even when there is no electricity, unlike many other attendance systems which depend on electricity and Wi-Fi.

- **Compare5:** It is more secure than other QR code attendance systems. It uses GPS, unique device id and user registration by the authority of the administrator.

3. Background Information

3.1 Required software

- **Flutter :**
  Flutter is the best SDK for mobile application development.

- **Android Studio :**
  Android Studio is the best IDE for mobile application development.

- **Django :**
  Django for backend.

- **PyCharm:**
  PyCharm is the best IDE for python projects.

3.2 Other software

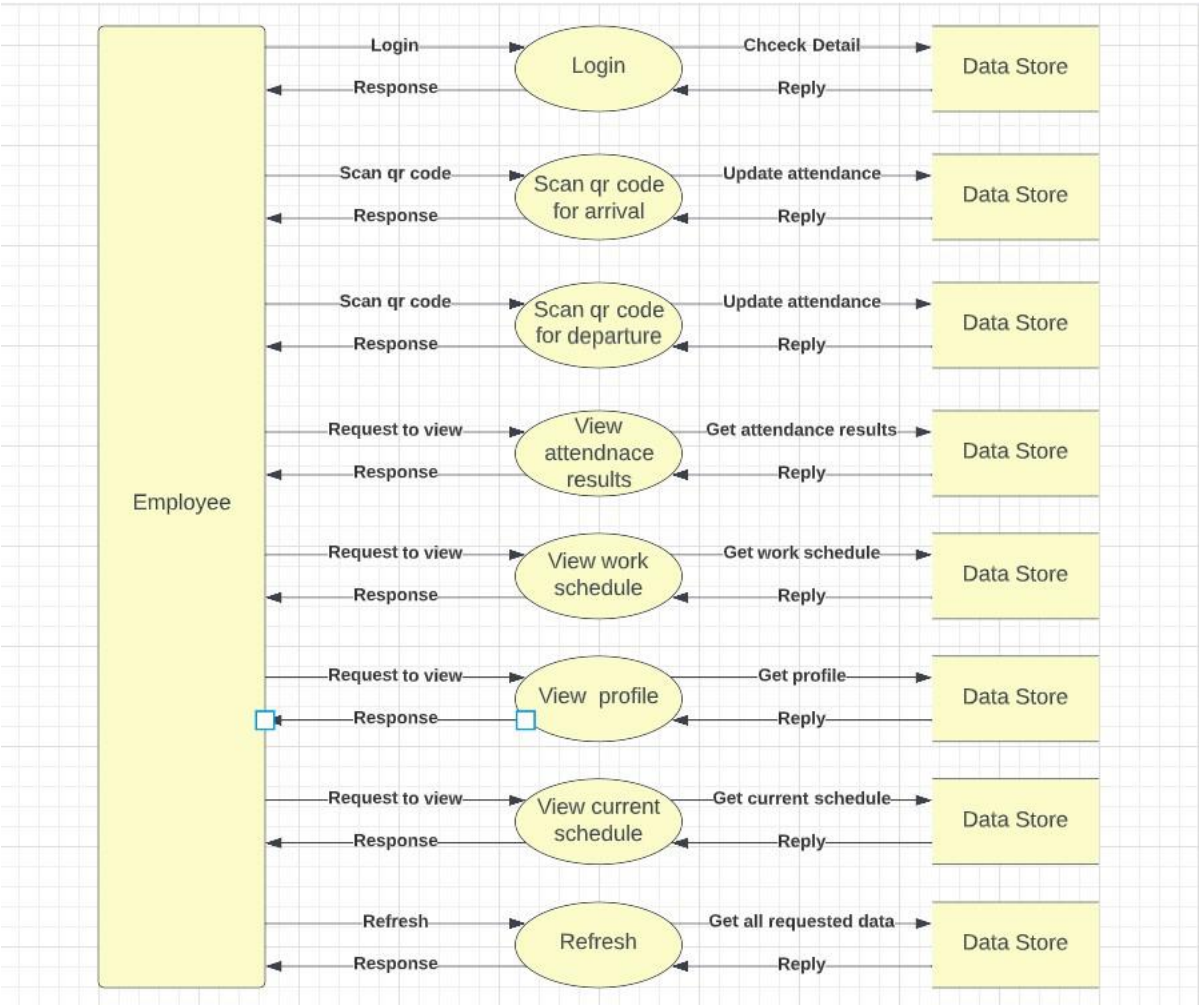
- **Adobe Photoshop and Flaticon :**
  For designing icons.

- **Git :**
  Used for repository.

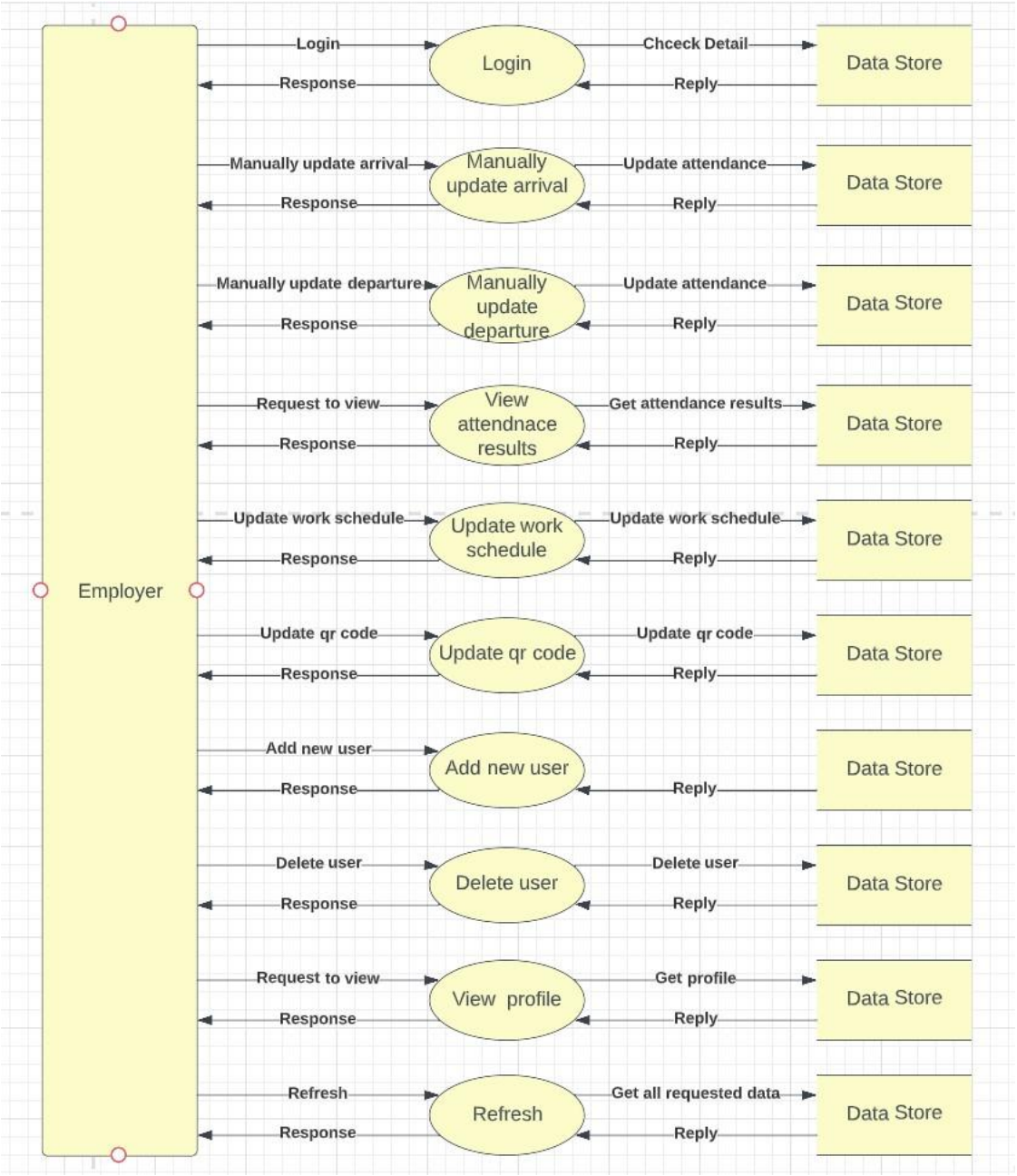
4. Design Documents

4.1 Data flow diagram

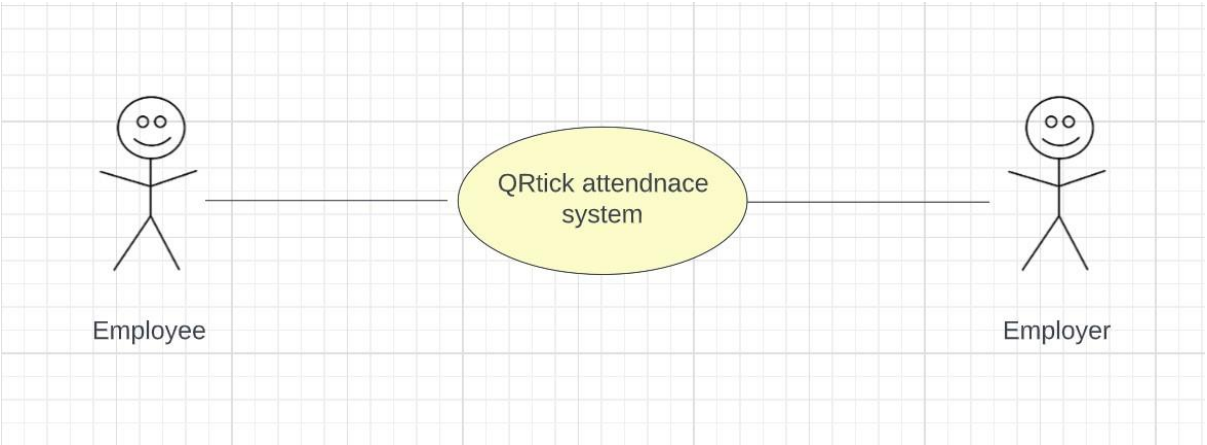
- **Employee data flow diagram:**



- **Employer data flow diagram:**



4.2 Your Context Diagram



5. Methodology

This project has both frontend and backend. I wrote frontend in Flutter SDK and backend in Django Rest Framework. I connected them using Flutter's http packages. I have not deployed backend to the online server, I used it as a local server. Phone can connect to the server using WI-FI or mobile hotspot.

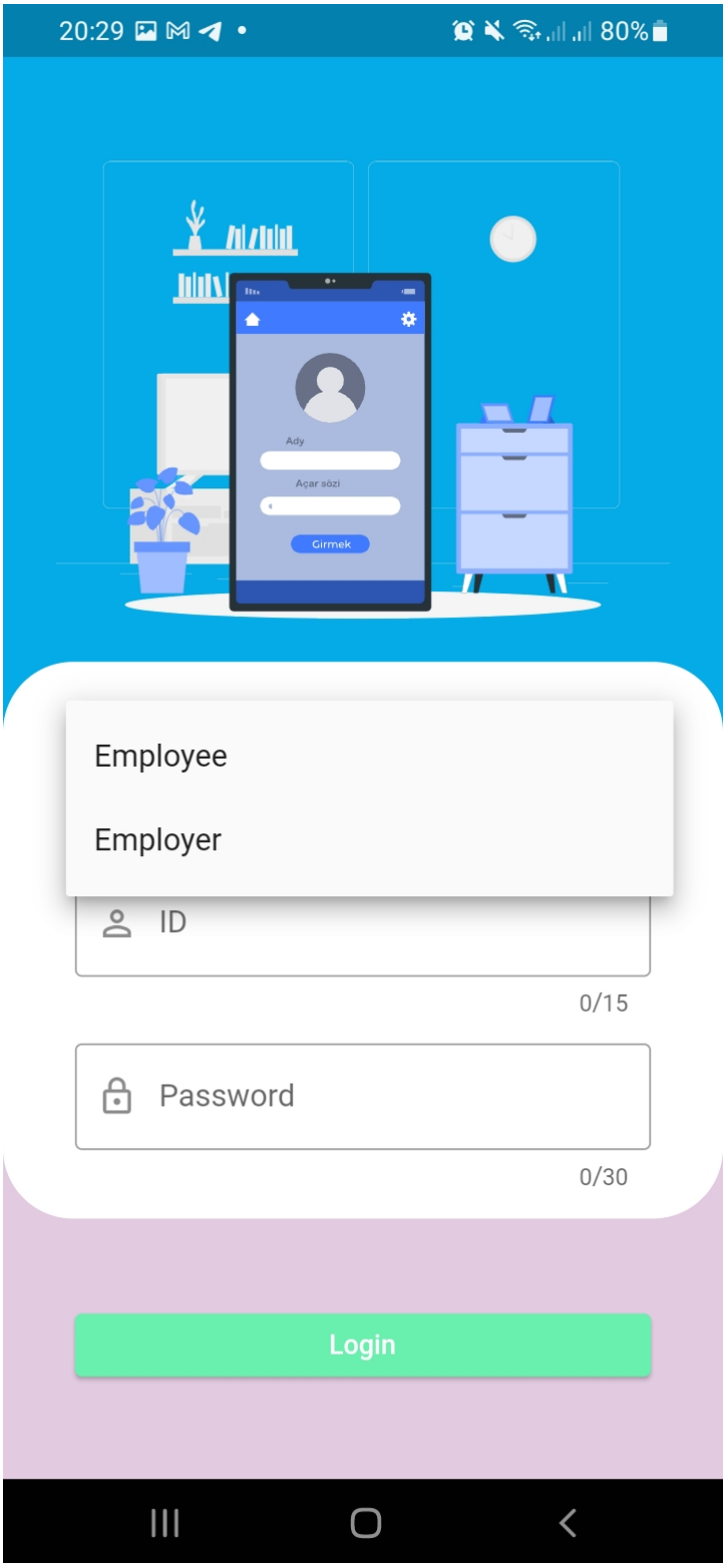
One of the most important part of the project is that it can work in multiple platforms, like iOS, Android, Web, Windows, macOS and Linux.

Flutter codes and Algorithms:

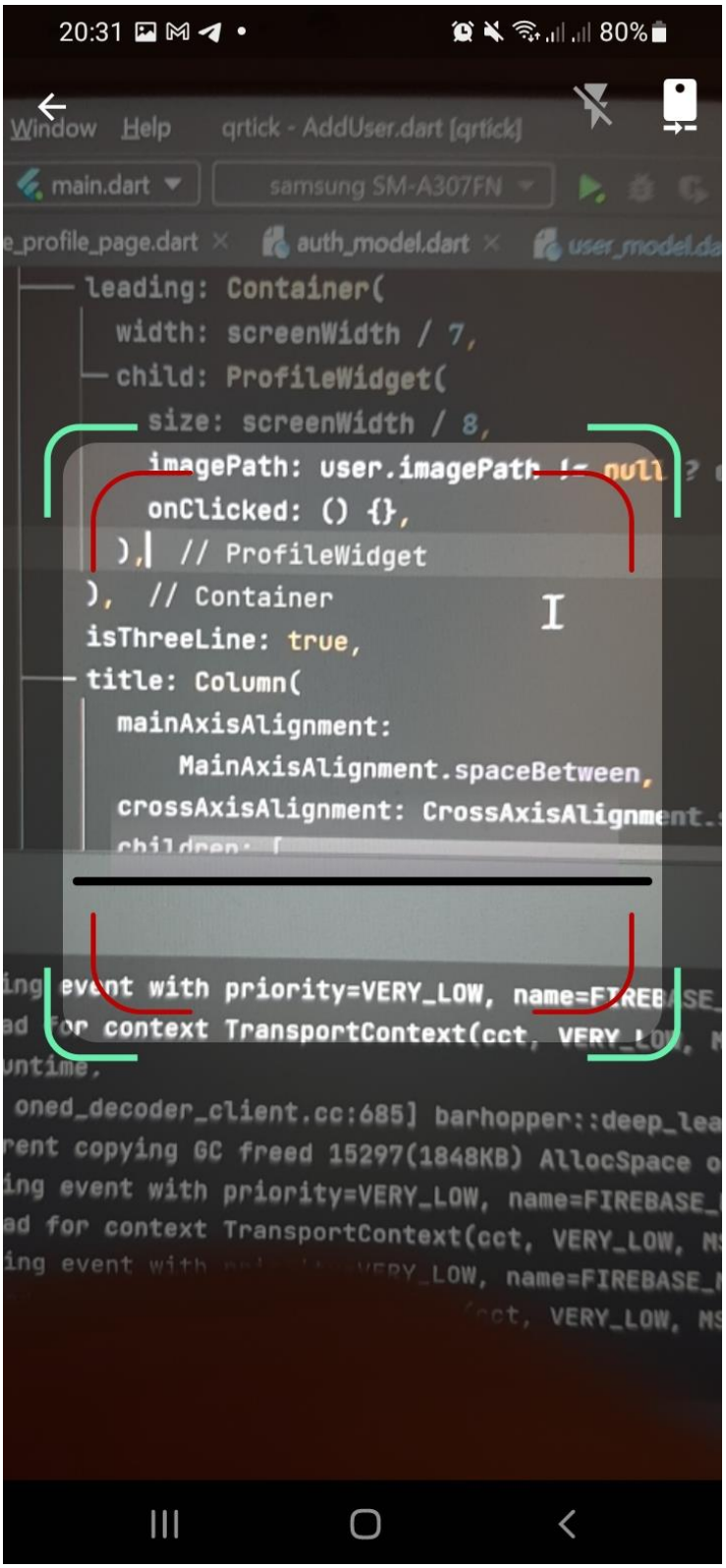
Pages:

There are in 8 screen pages in total:

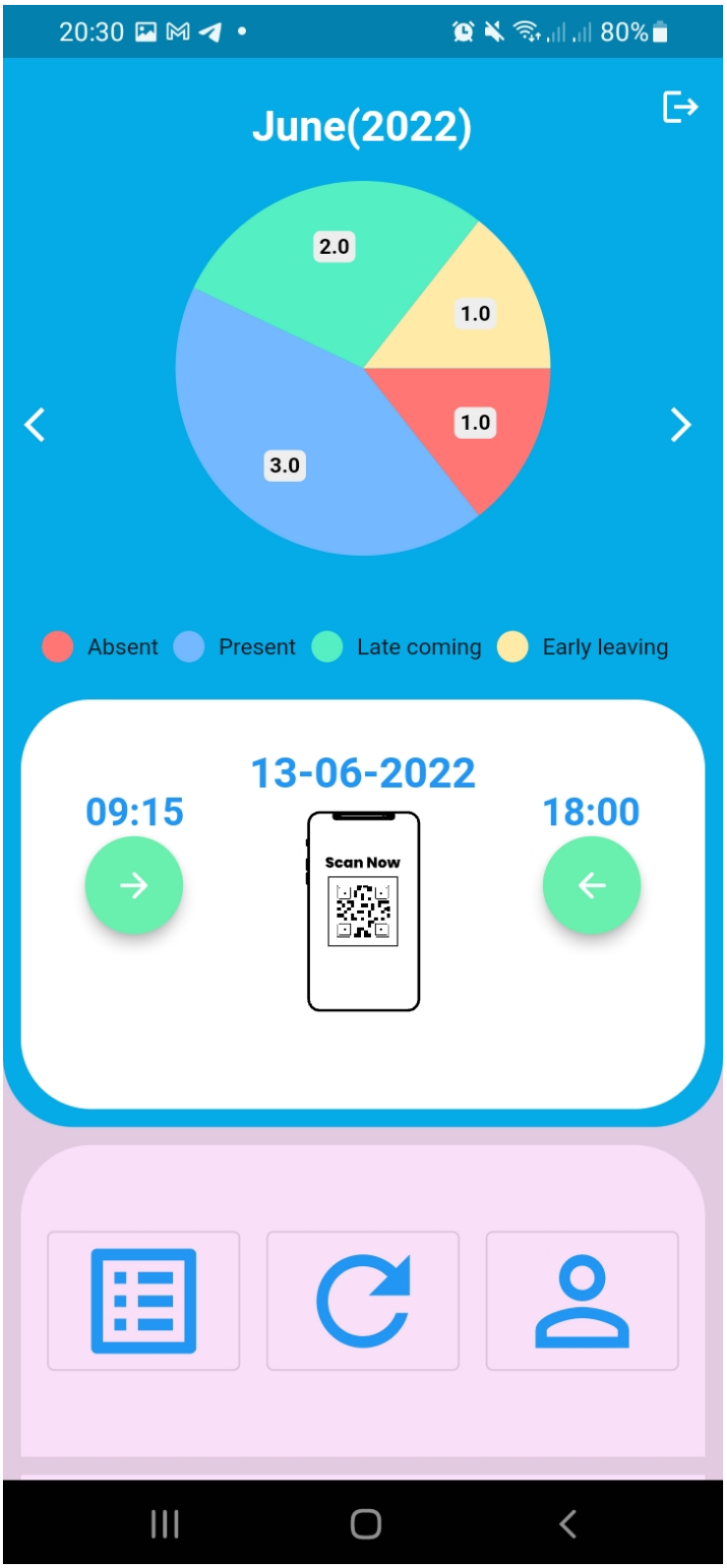
- **Login page**



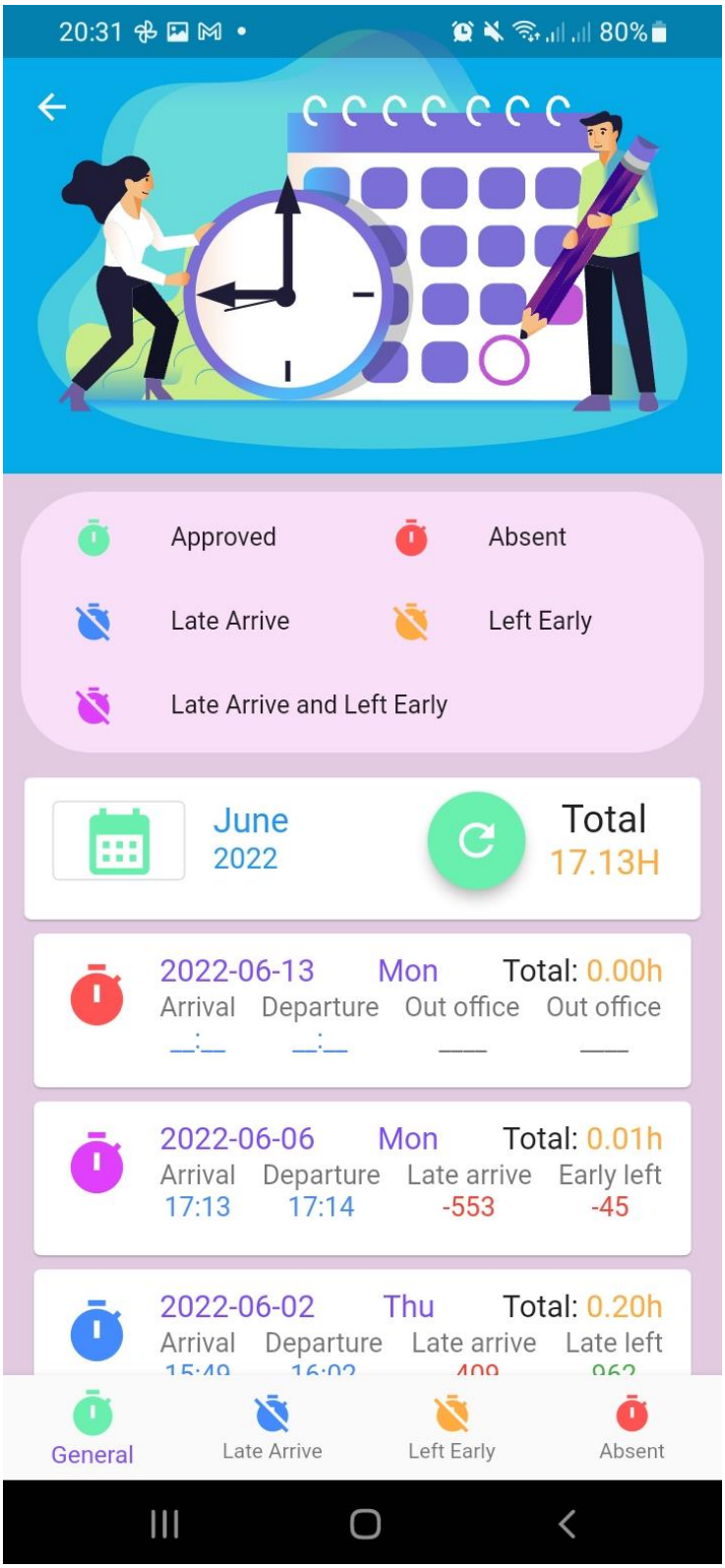
- **QR code scanner page**



- **Employee home page**



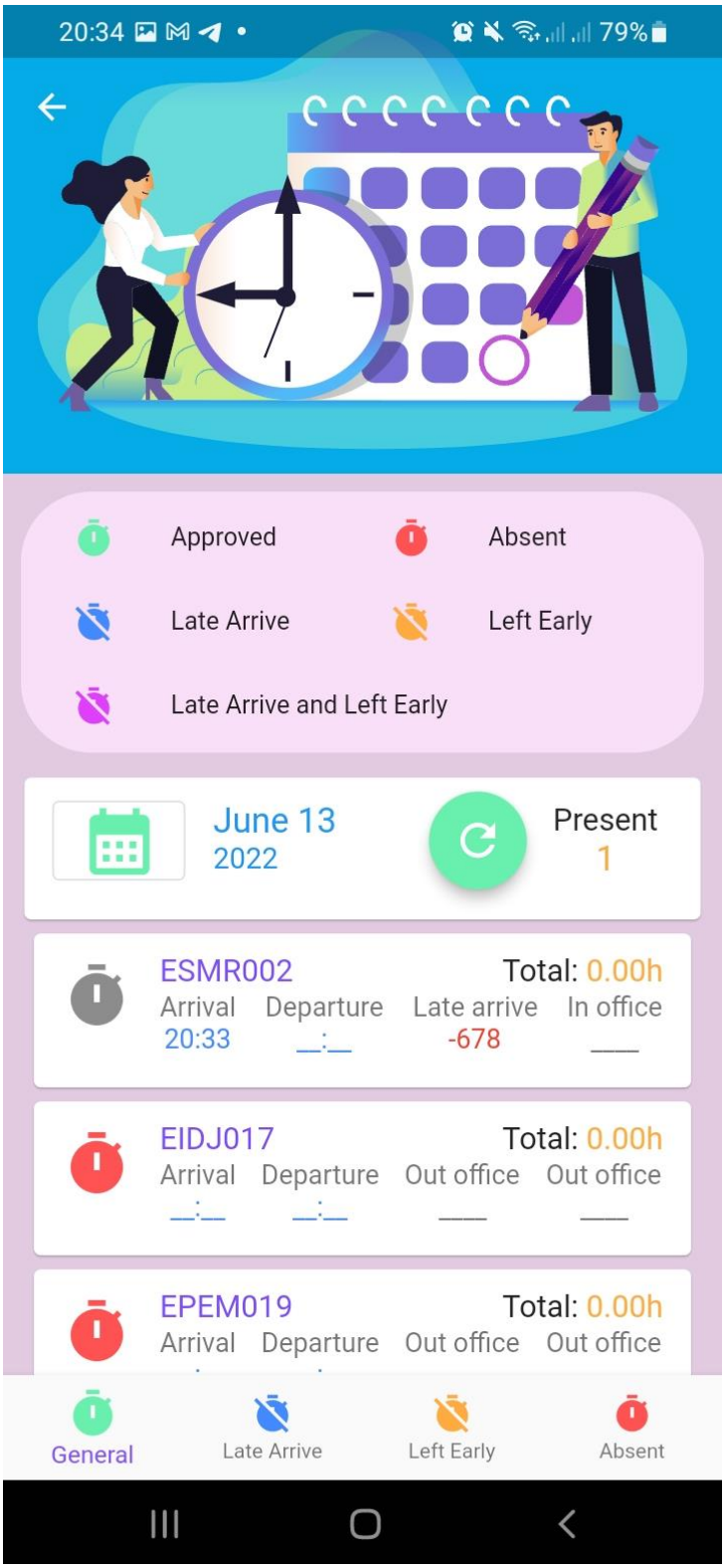
- **Employee attendance list page**



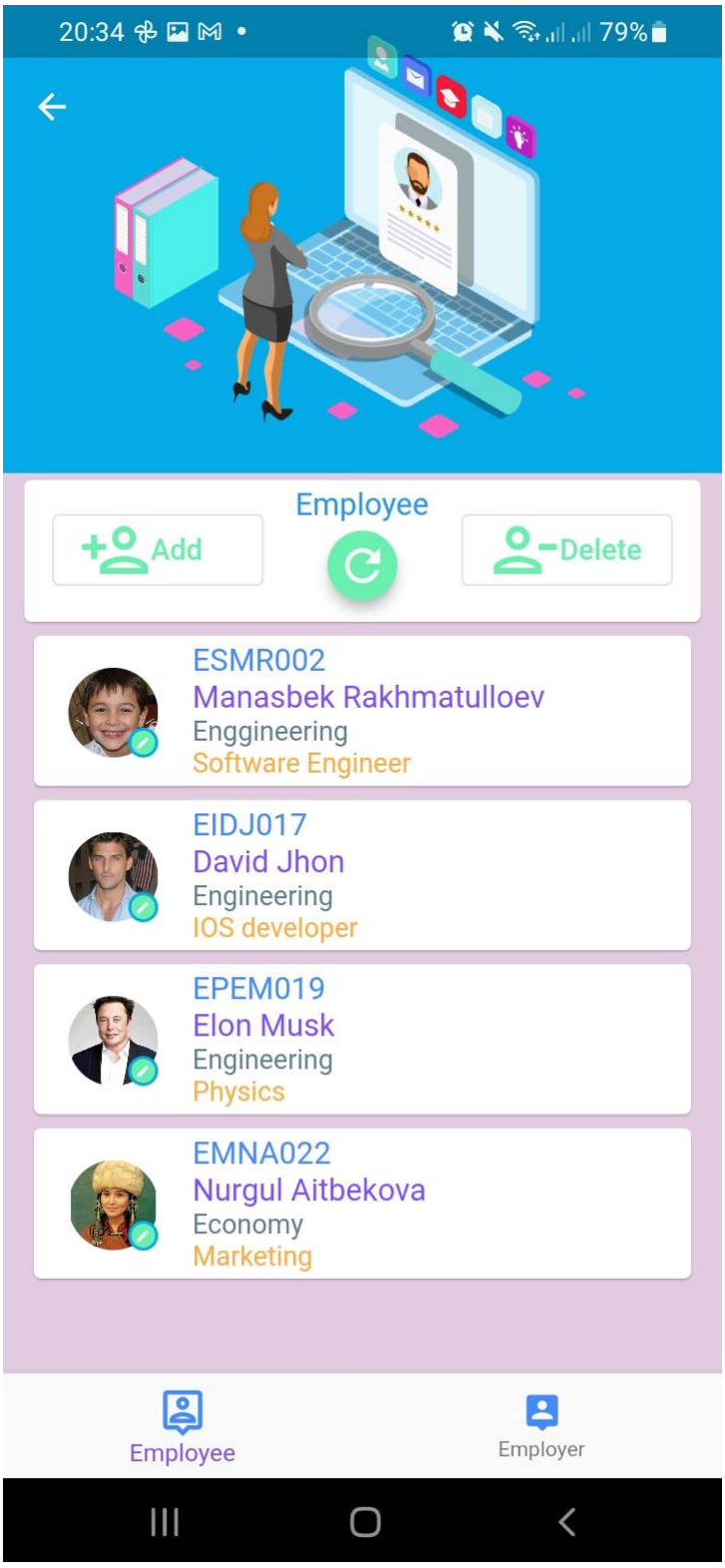
- **Employee Profile page**

- **Employer home page**

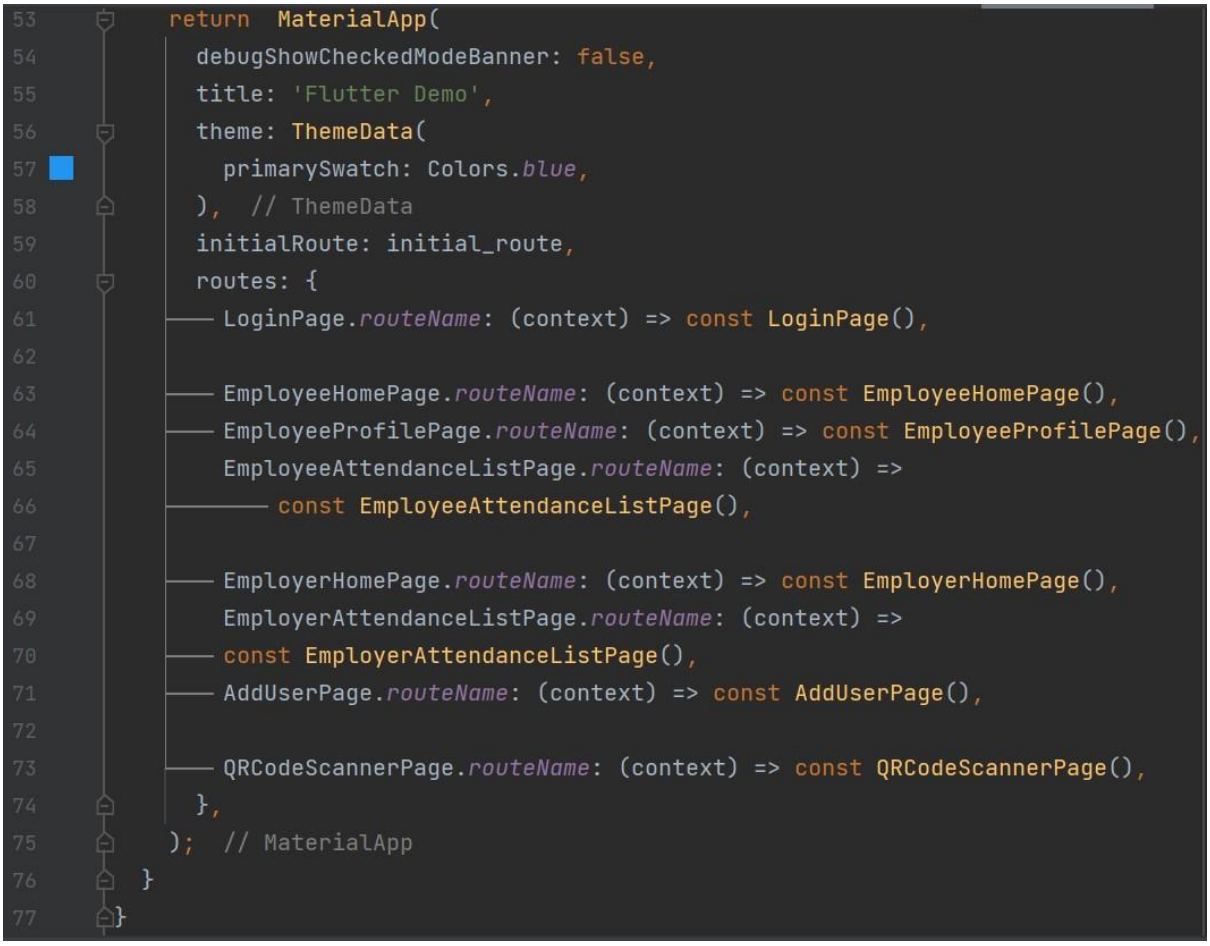
- **Employer attendance list page**



- **Employer add user page**



Registering pages to the router:



```
53      return  MaterialApp(
54        debugShowCheckedModeBanner: false,
55        title: 'Flutter Demo',
56        theme: ThemeData(
57          primarySwatch: Colors.blue,
58        ),  // ThemeData
59        initialRoute: initial_route,
60        routes: {
61          LoginPage.routeName: (context) => const LoginPage(),
62
63          EmployeeHomePage.routeName: (context) => const EmployeeHomePage(),
64          EmployeeProfilePage.routeName: (context) => const EmployeeProfilePage(),
65          EmployeeAttendanceListPage.routeName: (context) =>
66              const EmployeeAttendanceListPage(),
67
68          EmployerHomePage.routeName: (context) => const EmployerHomePage(),
69          EmployerAttendanceListPage.routeName: (context) =>
70          const EmployerAttendanceListPage(),
71          AddUserPage.routeName: (context) => const AddUserPage(),
72
73          QRCodeScannerPage.routeName: (context) => const QRCodeScannerPage(),
74        },
75      );  // MaterialApp
76    }
77  }
```

Dependencies:

```
dependencies:
  flutter:
    sdk: flutter
  pie_chart: ^5.3.1
  lottie: ^1.3.0
  syncfusion_flutter_datepicker: ^20.1.57
  mobile_scanner: ^1.1.1
  http: ^0.13.4
  flutter_riverpod: ^1.0.4
  shared_preferences: ^2.0.15
  intl: ^0.17.0
  qr_flutter: ^4.0.0
  path_provider: ^2.0.10
  gallery_saver: ^2.3.2
```

- **Flutter:** For flutter apps we need this SDK package

- **Pie_chart:** It is used for creating pie charts, that show daily and monthly attendance results of employees.

  Code:

```
Flexible(
  child: Stack(children: [
    PieChart(
      dataMap: attendanceResults,
      chartLegendSpacing: 40,
      legendOptions: const LegendOptions(
        legendTextStyle: TextStyle(fontSize: 11.8),
        showLegendsInRow: true,
        legendPosition: LegendPosition.bottom,
      ),  // LegendOptions
    ),  // PieChart
    Align(
      alignment: Alignment.centerLeft,
      child: IconButton(
        icon: const Icon(
```

- **Lottie:** It is used for showing lottie files, in other words animated files. Almost every page contains one lottie file.

  Code:

```
312                ),  // Text
313                Lottie.asset(
314                  'assets/images/qr_code_scanner.json',
315                ),
316              ],
```

- **syncfusion_flutter_datepicker:** It is used for date and time picking in the app.

  Code:

```
299          return SfDateRangePicker(
300            backgroundColor: const Color.fromRGBO(
301                252, 225, 251, 0.9),  // Color.fromRGBO
302            showActionButtons: true,
303            navigationMode:
304                DateRangePickerNavigationMode.snap,
305            view: DateRangePickerView.year,
306            showNavigationArrow: true,
307            confirmText: '          OK          ',
308            cancelText: '     CANCEL     ',
309            headerHeight: 200,
310            headerStyle: DateRangePickerHeaderStyle(
311                textAlign: TextAlign.center,
312                backgroundColor:
313                    app_colors.chartBlueBackground),  //
314            enableMultiView: false,
315            allowViewNavigation: false,
316            enablePastDates: true,
317            onSubmit: (value) async {
318              month = value.toString().substring(5, 7);
319              year = value.toString().substring(0, 4);
320              _onItemTapped(0);
321              await updateTable();
```

- **Mobile_scanner:** It used for scanning qr codes through camera.

  Code:

```
MobileScanner(
  controller: cameraController,
  allowDuplicates: false,
  onDetect: (barcode, args) async {
    if (barcode.rawValue == null) {
      debugPrint('Failed to scan Barcode');
    } else {
      final String code = barcode.rawValue!;
      debugPrint('Barcode found! $code');
      String token =
          ref.read(sharedPreferencesProvider).getString('token')!;
      Response response = await checkQRCode(code, token);
      if (response.statusCode == 200) {
        if (jsonDecode(response.body)['message'] == 'CORRECT') {
          String user_id = ref
              .read(sharedPreferencesProvider)
              .getString('user_id')!;
          bool is_arrival = (button_type.button_type == 'arrival');
          Response response1 =
              await arrivedAt(user_id, token, is_arrival);
          if (response1.statusCode == 200) {
            cameraController.stop();
            if (is_arrival &&
                ref
```

- **Http:** It is used for making http requests to the database.

  Code:

```dart
String ip_address = '192.168.0.106';

Future<client.Response> getToken(String username, String password) async {
  final client.Response response = await client.post(
      Uri.parse('http://$ip_address:8000/api/login/'),
      body: <String, String>{
        'username': username,
        'password': password,
      });

  return response;
}

Future<client.Response> getUserType(String user_id, String token) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/is_staff/'),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **Flutter_riverpod:** It is used as an provider for storing user preferences.
  Code:

```dart
final sharedPreferencesProvider = Provider<SharedPreferences>((ref) {
  throw UnimplementedError();
});  // Provider


Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final sharedPreferences = await SharedPreferences.getInstance();
  runApp(ProviderScope(
    overrides: [
      // override the previous value with the new object
      sharedPreferencesProvider.overrideWithValue(sharedPreferences),
    ],
    child: const MyApp(),
  ));  // ProviderScope
}
```

- **Shared_preferences:** It is used for saving user preferences like

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final sharedPreferences = await SharedPreferences.getInstance();
  runApp(ProviderScope(
    overrides: [
      // override the previous value with the new object
      sharedPreferencesProvider.overrideWithValue(sharedPreferences),
    ],
    child: const MyApp(),
  ));  // ProviderScope
}
```

- **Intl:** It used for formatting the date
  Code:

```
child: ListTile(
  title: Text(
    DateFormat.MMMM()
        .format(DateTime.parse('2022-$month-01')),
    style: TextStyle(fontSize: 18, color: Colors.blue),
  ),  // Text
  subtitle: Text(
    year,
    style: TextStyle(fontSize: 15, color: Colors.blue),
  ),  // Text
  trailing: Container(
    width: screenWidth / 2.8,
```

- **Qr_flutter:** It is used for generating QR code.
  Code:

```
child: QrImage(
    data:
        qr_code_update,
    gapless: false,
    embeddedImage:
        AssetImage(
            'assets/icons/app_icon_
    embeddedImageStyle:
        QrEmbeddedImageStyle(
      size:
          Size(50, 50),
    ),  // QrEmbeddedImageStyle
    backgroundColor:
        Colors.white,
    version:
        QrVersions.auto,
    size: screenWidth /
        1.8),  // QrImage
```

- **Path_provider:** It is used for getting specific path from mobile device.
  Code:

```
631
632                 final tempDir =
633                     await getTemporaryDirectory();
634
635                 var datetime =
636                     DateTime
637                         .now();
638
```

- **Gallery_saver:** It used for saving images to the gallery.
  Code:

```
646
647                 final path =
648                     '${tempDir.path}/$datetime.png';
649
650                 await GallerySaver
651                     .saveImage(
652                         path);
653             } catch (e) {
654               print(e
655                   .toString());
656             }
657
```

HTTP requests:

- **getToken:** This http request used for logging in and getting authorization token.

  Code:

```dart

import 'package:flutter/material.dart';
import 'package:http/http.dart' as client;
import 'dart:async';
import 'dart:io';

String ip_address = '192.168.0.106';

Future<client.Response> getToken(String username, String password) async {
  final client.Response response = await client.post(
      Uri.parse('http://$ip_address:8000/api/login/'),
      body: <String, String>{
        'username': username,
        'password': password,
      });

  return response;
}
```

- **getUserType:** This http request used for getting user type information, like if it is employee or employer.

  Code:

```dart
Future<client.Response> getUserType(String user_id, String token) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/is_staff/'),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **checkQRCode:** This http request used for comparing the scanned QR code with the QR code in the database.

  Code:

```
Future<client.Response> checkQRCode(String qr_code, String token) async {
  client.Response response = await client
      .post(Uri.parse('http://$ip_address:8000/api/check_qr_code/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  }, body: {
    'scanned_qr_code': qr_code
  });
  return response;
}
```

- **arrivedAt:** This http request used for saving arrival time and departure time in the database.

  Code:

```
Future<client.Response> arrivedAt(
    String user_id, String token, bool first_time) async {
  DateTime today = DateTime.now();
  String str_today = today.day.toString().padLeft(2, '0') +
      '_' +
      today.month.toString().padLeft(2, '0') +
      '_' +
      today.year.toString() +
      '_';
  client.Response response = await client.patch(
      Uri.parse('http://$ip_address:8000/api/arrival_departure/' +
          str_today +
          user_id +
          '/'),
      headers: {
        HttpHeaders.authorizationHeader: 'Token ' + token,
      },
      body: first_time
          ? {
              'arrival_time': '00:00',
              'departure_time': '00:00',
            }
          : {
              'arrival_time': '12:12',
              'departure_time': '12:12',
            });
  return response;
```

- **currentSchedule:** This http request used for getting current work schedule.

  Code:

```dart
Future<client.Response> currentSchedule(String token) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/current_schedule/'),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **workSchedule:** This http request used for getting weekly work schedule.

  Code:

```dart
Future<client.Response> workSchedule(String token) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/work_schedule/'),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **getUserMonthlyAttendance:** This http request used for getting monthly attendance results of the specific user.

  Code:

```
Future<client.Response> getUserMonthlyAttendance(
    String user_id, String token, String month, String year) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/arrival_departure/?search=' +
        '_' +
        month +
        '_' +
        year +
        '_' +
        user_id),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **updateQRCode:** This http request used for updating current QR code in the database.

  Code:

```
Future<client.Response> updateQRCode(String new_qr_code, String token) async {
  client.Response response = await client.patch(
      Uri.parse('http://$ip_address:8000/api/update_qr_code/qr_code_id/'),
      headers: {
        HttpHeaders.authorizationHeader: 'Token ' + token,
      },
      body: {
        'qr_code': new_qr_code,
      });
  return response;
}
```

- **updateWorkSchedule:** This http request used for updating current working schedule in the database.

  Code:

```
Future<client.Response> updateWorkSchedule(
    int index, String token, TimeOfDay time, bool is_from_time) async {
  client.Response response = await client.patch(
      Uri.parse('http://$ip_address:8000/api/work_schedule/$index/'),
      headers: {
        HttpHeaders.authorizationHeader: 'Token ' + token,
      },
      body: is_from_time
          ? {'from_time': '${time.hour}:${time.minute}'}
          : {'to_time': '${time.hour}:${time.minute}'});
  return response;
}
```

- **updateIsDayOff:** This http request used for updating weekly working and not working days.

  Code:

```
Future<client.Response> updateIsDayOff(
    int index, String token, bool is_day_off) async {
  client.Response response = await client.patch(
      Uri.parse('http://$ip_address:8000/api/work_schedule/$index/'),
      headers: {
        HttpHeaders.authorizationHeader: 'Token ' + token,
      },
      body: {
        'from_time': '',
        'to_time': '',
        'is_day_off': '${is_day_off}'
      });
  return response;
}
```

- **getUsersDailyAttendance:** This http request used for getting all the employees' daily attendance results.

  Code:

```
Future<client.Response> getUsersDailyAttendance(
    String token, String day, String month, String year) async {
  print(token);
  final client.Response response = await client.get(
    Uri.parse('http://$ip_address:8000/api/arrival_departure/?search=' +
        day +
        '_' +
        month +
        '_' +
        year),
    headers: {
      HttpHeaders.authorizationHeader: 'Token ' + token,
    },
  );

  return response;
}
```

- **getUserAuth:** This http request used for getting all the authenticated users' information from the database.

  Code:

```
Future<client.Response> getUserAuth(String token) async {
  client.Response response = await client
      .get(Uri.parse('http://$ip_address:8000/api/auth/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  });
  return response;
}
```

- **AddUser:** This http request used for adding new employee or in other words user to the database.

  Code:

```
Future<client.Response> AddUser(
    {required String token,
    required String name,
    required String surname,
    required String department,
    required String designation}) async {
  client.Response response = await client
      .post(Uri.parse('http://$ip_address:8000/api/profile/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  }, body: {
    'name': name,
    'surname': surname,
    'department': department,
    'designation': designation,
    'employed_date': '',
    'skills': '',
    'address': '',
    'phone_number': '',
    'email': '',
    'date_of_birth': '',
    'about_me': '',
    'profile_image': '',
  });
  return response;
}
```

- **DeleteUser:** This http request used for deleting user from the database.

  Code:

```
Future<client.Response> DeleteUser(
    {required String token,
      required int id,
    }) async {
  client.Response response = await client
      .delete(Uri.parse('http://$ip_address:8000/api/auth/$id/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  });
  return response;
}
```

- **AddStaff:** This http request used for adding new employer or in other words user to the database.

  Code:

```
Future<client.Response> AddStaff(
    {required String token,
      required String user_id,
      required String password,
      }) async {

  client.Response response = await client
      .post(Uri.parse('http://$ip_address:8000/api/auth/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  }, body: {
    "user_id": user_id,
    "password": password,
    "is_staff": "true"
  });
  return response;
}
```

- **GetUser:** This http request used for getting specific employer information form the database.

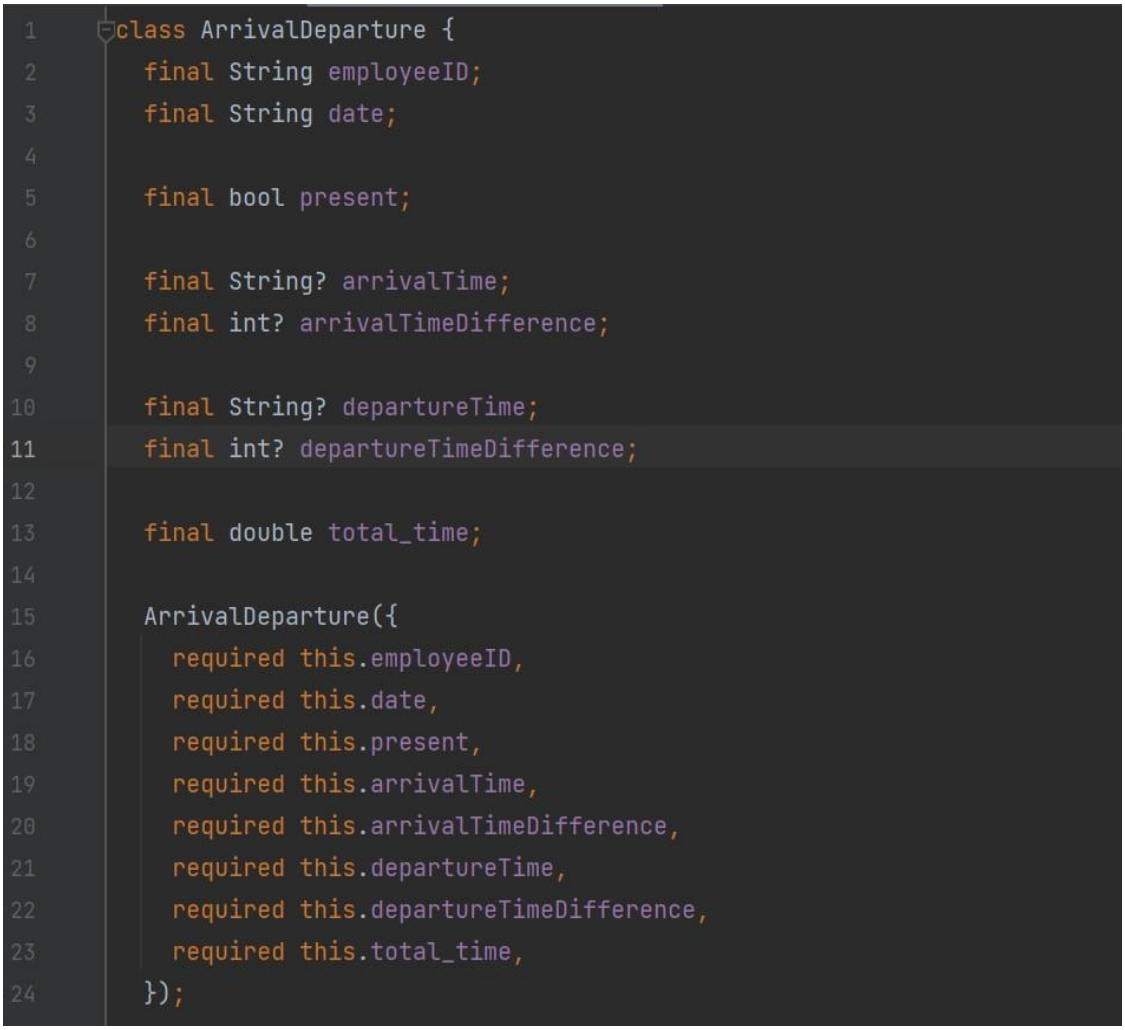
  Code:

```
Future<client.Response> GetUser(
    {required String token,
      required String user_id
    }) async {

  client.Response response = await client
      .get(Uri.parse('http://$ip_address:8000/api/profile/${user_id}/'), headers: {
    HttpHeaders.authorizationHeader: 'Token ' + token,
  });
  return response;
}
```

Models:

- **ArrivalDeparture:** This model is used for converting arrival departure model information to flutter objects or from flutter objects to arrival departure model.

  Code:



```dart
class ArrivalDeparture {
  final String employeeID;
  final String date;

  final bool present;

  final String? arrivalTime;
  final int? arrivalTimeDifference;

  final String? departureTime;
  final int? departureTimeDifference;

  final double total_time;

  ArrivalDeparture({
    required this.employeeID,
    required this.date,
    required this.present,
    required this.arrivalTime,
    required this.arrivalTimeDifference,
    required this.departureTime,
    required this.departureTimeDifference,
    required this.total_time,
  });
```

```dart
ArrivalDeparture.fromJson(Map<String, dynamic> json)
    : employeeID = json['user_id'],
      date = json['date'],
      present = json['present'],
      arrivalTime = json['arrival_time'],
      arrivalTimeDifference = json['arrival_time_difference'],
      departureTime = json['departure_time'],
      departureTimeDifference = json['departure_time_difference'],
      total_time = json['total_time'];

Map<String, dynamic> toJson() => {
      'user_id': employeeID,
      'date': date,
      'present': present,
      'arrival_time': arrivalTime,
      'arrival_time_difference': arrivalTimeDifference,
      'departure_time': departureTime,
      'departure_time_difference': departureTimeDifference,
      'total_time': total_time,
    };
}
```

- **Auth:** This model is used for converting user authentication model information to flutter objects or from flutter objects to user authentication model.
  Code:

```dart
class Auth {
  int id;
  String user_id;
  bool is_staff;


  Auth(
      {required this.id,
        required this.user_id,
        required this.is_staff
      });


  Auth.fromJson(Map<String, dynamic> json)
      : id = json['id'],
        user_id = json['user_id'],
        is_staff = json['is_staff'];


  Map<String, dynamic> toJson() => {
    'id': id,
    'user_id': user_id,
    'is_staff': is_staff
  };
}
```

- **User:** This model is used for converting user profile model information to flutter objects or from flutter objects to user profile model.
  Code:

```dart
class User {
  String user_id;
  String name;
  String surname;
  String department;
  String designation;
  String? employedDate;
  String? skills;
  String? address;
  String? phone;
  String? email;
  String? dob;
  String? aboutMe;
  String? imagePath;

  User(
      {required this.imagePath,
      required this.user_id,
      required this.name,
      required this.surname,
      required this.department,
      required this.designation,
      required this.employedDate,
      required this.skills,
      required this.address,
      required this.phone,
      required this.email,
      required this.dob,
      required this.aboutMe});
```

```
32
33      User.fromJson(Map<String, dynamic> json)
34          :
35            user_id = json['user_id'],
36            name = json['name'],
37            surname = json['surname'],
38            department = json['department'],
39            designation = json['designation'],
40            employedDate = json['employed_date'],
41            skills = json['skills'],
42            address = json['address'],
43            phone = json['phone'],
44            email = json['email'],
45            dob = json['date_of_birth'],
46            aboutMe = json['about_me'],
47            imagePath = json['profile_image'];
```

```
48
49      Map<String, dynamic> toJson() => {
50
51            'user_id': user_id,
52            'name': name,
53            'surname': surname,
54            'department': department,
55            'designation': designation,
56            'employed_date': employedDate,
57            'skills': skills,
58            'address': address,
59            'phone_number': phone,
60            'email': email,
61            'date_of_birth': dob,
62            'about_me': aboutMe,
63            'profile_image': imagePath,
64          };
65    }
```

- **WorkSchedule:** This model is used for converting work schedule model information to flutter objects or from flutter objects to work schedule model.
Code:

```dart
class WorkSchedule {
  final int id;
  final String day_of_week;
  final String? from_time;
  final String? to_time;
  final bool is_day_off;

  const WorkSchedule({
    required this.id,
    required this.day_of_week,
    required this.from_time,
    required this.to_time,
    required this.is_day_off,
  });

  factory WorkSchedule.fromJson(Map<String, dynamic> json) {
    return WorkSchedule(
      id: json['id'],
      day_of_week: json['day_of_week'],
      from_time: json['from_time'],
      to_time: json['to_time'],
      is_day_off: json['is_day_off']
    );
  }
}
```

Django REST API codes and Algorithms:

Routes:

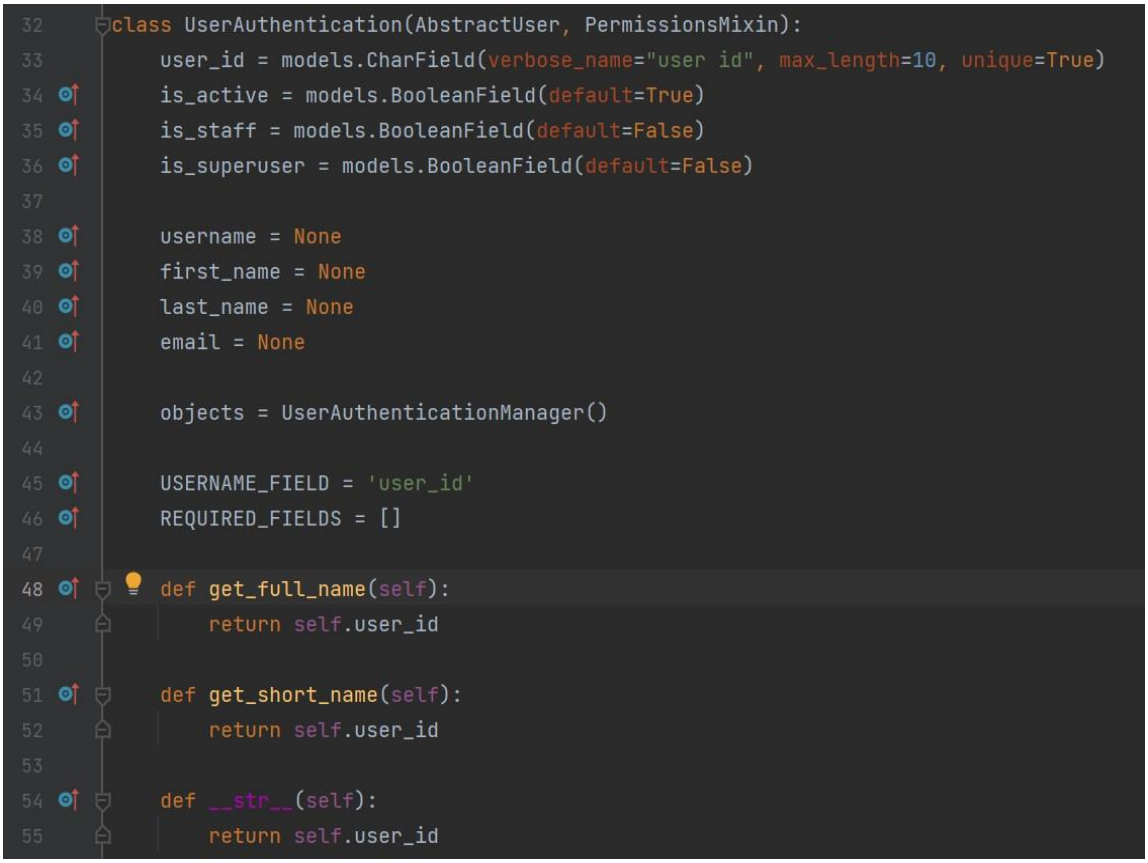
```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter

from qrtick_app import views

router = DefaultRouter()
router.register('auth', views.UserAuthenticationView)
router.register('profile', views.UserProfileView)
router.register('work_schedule', views.WorkScheduleView)
router.register('update_qr_code', views.QRCODEView)
router.register('arrival_departure', views.ArrivalDepartureView)


urlpatterns = [
    path('', include(router.urls)),
    path('login/', views.UserLoginView.as_view()),
    path('check_qr_code/', views.CheckQRCODEView.as_view()),
    path('is_staff/', views.IsStaffView.as_view()),
    path('current_schedule/', views.CurrentScheduleView.as_view()),
]
```

Dependencies:

```
Django>=3.2.13,<3.3.0
djangorestframework>=3.13.1,<3.14.0
psycopg2-binary>=2.9.3,<2.10.0
django-cors-headers>=3.12.0,<3.13.0
Pillow>=9.1.1,<9.2.0

django-phonenumber-field>=6.1.0,<6.2.0
phonenumbers>=8.12.48,<8.13.0

celery>=5.2.6,<5.3.0
redis>=4.3.1,<4.4.0
django-celery-results>=2.3.1,<2.4.0
django-celery-beat>=2.2.1,<2.3.0

uWSGI>=2.0.20,<2.1.0

```

Models:

- **UserAuthentication:** It is used for storing user authentication information.
  Code:



```
class UserAuthentication(AbstractUser, PermissionsMixin):
    user_id = models.CharField(verbose_name="user id", max_length=10, unique=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    username = None
    first_name = None
    last_name = None
    email = None

    objects = UserAuthenticationManager()

    USERNAME_FIELD = 'user_id'
    REQUIRED_FIELDS = []

    def get_full_name(self):
        return self.user_id

    def get_short_name(self):
        return self.user_id

    def __str__(self):
        return self.user_id
```

- **UserProfile:** It is used for storing user information.
  Code:

```python
def user_profile_image_file_path(instance, filename):
    ext = filename.split('.')[-1]
    filename = f'{instance.pk}.{ext}'
    return os.path.join('uploads/profile_images/', filename)


class UserProfile(models.Model):
    user_id = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, primary_key=True,
    name = models.CharField(max_length=50)
    surname = models.CharField(max_length=50)
    department = models.CharField(max_length=100)
    designation = models.CharField(max_length=100)
    employed_date = models.DateField(null=True)
    skills = models.TextField(max_length=50, null=True)
    address = models.CharField(max_length=100, null=True)
    phone_number = PhoneNumberField(null=True)
    email = models.EmailField(null=True)
    date_of_birth = models.DateField(null=True)
    about_me = models.TextField(max_length=100, null=True)
    profile_image = models.ImageField(null=True, upload_to=user_profile_image_file_path)

    def __str__(self):
        return f'{self.user_id}_{self.name}_{self.surname}'
```

- **ArrivalDeparture:** It used for storing user's arrival and departure information.

  Code:

```python
class ArrivalDeparture(models.Model):
    id = models.CharField(max_length=20, primary_key=True)
    user_id = models.ForeignKey(UserProfile, on_delete=models.CASCADE, unique=False,
    date = models.DateField(auto_now_add=True)
    present = models.BooleanField(default=False)

    arrival_time = models.TimeField(null=True)
    arrival_time_difference = models.IntegerField(null=True)

    departure_time = models.TimeField(null=True)
    departure_time_difference = models.IntegerField(null=True)

    total_time = models.FloatField(default=0)

    def __str__(self):
        return self.id
```

- **WorkSchedule:** It used for storing weekly work schedule.
  Code:

```
class WorkSchedule(models.Model):
    id = models.IntegerField(default=0, primary_key=True)
    day_of_week = models.CharField(max_length=10, null=False)
    from_time = models.TimeField(null=True)
    to_time = models.TimeField(null=True)
    is_day_off = models.BooleanField(default=False)

    def __str__(self):
        return self.day_of_week
```

- **QRCODE:** It is used for storing QR code.

```
class QRCODE(models.Model):
    id = models.CharField(default='qr_code_id', primary_key=True, max_length=20)
    qr_code = models.CharField(max_length=1000, default='')
```

Views:

- **UserAuthenticationView:** It is used for receiving http requests for user authentication model.
  Code:

```
class UserAuthenticationView(viewsets.ModelViewSet):
    serializer_class = serializers.UserAuthenticationSerializer
    queryset = models.UserAuthentication.objects.all()

    authentication_classes = (TokenAuthentication,)
    permission_classes = (permissions.UpdateOwnAuthorization, rf_permissions.IsAdminUser)

    filter_backends = (filters.SearchFilter,)
    search_fields = ('user_id',)
```

- **UserLoginView:** It is used for receiving http requests for logging in and getting authentication token.
Code:

```
class UserLoginView(ObtainAuthToken):
    permission_classes = (rf_permissions.AllowAny,)
    """Handle creating user authentication tokens"""
    renderer_classes = api_settings.DEFAULT_RENDERER_CLASSES

```

- **UserProfileView:** It is used for receiving http requests for user profile model.
Code:

```
class UserProfileView(viewsets.ModelViewSet):
    serializer_class = serializers.UserProfileSerializer
    queryset = models.UserProfile.objects.all()

    authentication_classes = (TokenAuthentication,)
    permission_classes = (permissions.UpdateOwnProfile,)

```

- **WorkScheduleView:** It is used for receiving http requests for work schedule model.
Code:

```
class WorkScheduleView(viewsets.ModelViewSet):
    serializer_class = serializers.WorkScheduleSerializer
    queryset = models.WorkSchedule.objects.all().order_by("id")

    authentication_classes = (TokenAuthentication,)
    permission_classes = (permissions.UpdateWorkSchedule,)

```

- **QRCODEView:** It is used for receiving http requests for QR code model.
Code:

```
class QRCODEView(viewsets.ModelViewSet):
    serializer_class = serializers.QRCODESerializer
    queryset = models.QRCODE.objects.all()

    authentication_classes = (TokenAuthentication,)
    permission_classes = (permissions.UpdateWorkSchedule,)

```

- **CheckQRCODEView:** It is used for receiving http requests and comparing QR codes.
Code:

```
class CheckQRCODEView(APIView):
    serializer_class = serializers.CheckQRCODESerializer

    authentication_classes = (TokenAuthentication,)
    permission_classes = (rf_permissions.IsAuthenticated,)

    def post(self, request):
        serializer = self.serializer_class(data=request.data)
        if serializer.is_valid():
            if serializer.validated_data.get('scanned_qr_code') == models.QRCODE.objects.
                return Response({'message': 'CORRECT'}, status=status.HTTP_200_OK)
            else:
                return Response({'message': 'INCORRECT'}, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

```

- **ArrivalDepartureView:** It is used for receiving http requests for arrival departure model.
Code:

```
class ArrivalDepartureView(viewsets.ModelViewSet):
    serializer_class = serializers.ArrivalDepartureSerializer
    queryset = models.ArrivalDeparture.objects.all().order_by('-date', 'arrival_time')

    authentication_classes = (TokenAuthentication,)
    permission_classes = (rf_permissions.IsAuthenticated,)

    filter_backends = (filters.SearchFilter,)
    search_fields = ('id', 'user_id__user_id__user_id', 'date__month', 'date__year', )

```

- **IsStaffView:** It is used for receiving http requests and checking if the user is staff user.
Code:

```
class IsStaffView(APIView):
    authentication_classes = (TokenAuthentication,)
    permission_classes = (rf_permissions.IsAuthenticated,)

    def get(self, request):
        if models.UserAuthentication.objects.get(id=request.user.id).is_staff:
            return Response({'message': True}, status=status.HTTP_200_OK)
        else:
            return Response({'message': False}, status=status.HTTP_200_OK)

```

- **CurrentScheduleView:** It is used for receiving http requests and returning current schedule back to the user.
Code:]

```
class CurrentScheduleView(APIView):
    authentication_classes = (TokenAuthentication,)
    permission_classes = (rf_permissions.IsAuthenticated,)

    def get(self, request):
        schedule = models.WorkSchedule.objects.get(id=datetime.now().weekday())
        from_time = schedule.from_time
        to_time = schedule.to_time
        if from_time is not None and to_time is not None:
            return Response({'from_time': from_time.strftime("%H:%M"), 'to_time': to_time.strftime("%H:%M"),
                             'date': date.today().strftime("%d-%m-%Y"), 'month_year': date.today().strftime("%B(%Y)"),

        return Response({'from_time': '--:--', 'to_time': '--:--',
                         'date': date.today().strftime("%d-%m-%Y"), 'month_year': date.today().strftime("%B(%Y)"),
```

Serializers:

- **UserAuthenticationSerializer:** It is used for serializing and deserializing user authentication model.
Code:

```
class UserAuthenticationSerializer(serializers.ModelSerializer):
    profile = UserProfileSerializer(many=False, read_only=True)

    class Meta:
        model = models.UserAuthentication
        fields = ('id', 'user_id', 'password', 'is_staff', 'profile')
        extra_kwargs = {
            'password': {
                'write_only': True,
                'style': {'input_type': 'password'}
            }
        }

    def create(self, validated_data):
        user = models.UserAuthentication.objects.create_user(
            user_id=validated_data['user_id'],
            password=validated_data['password'],
            is_staff=validated_data['is_staff']
        )

        return user
```

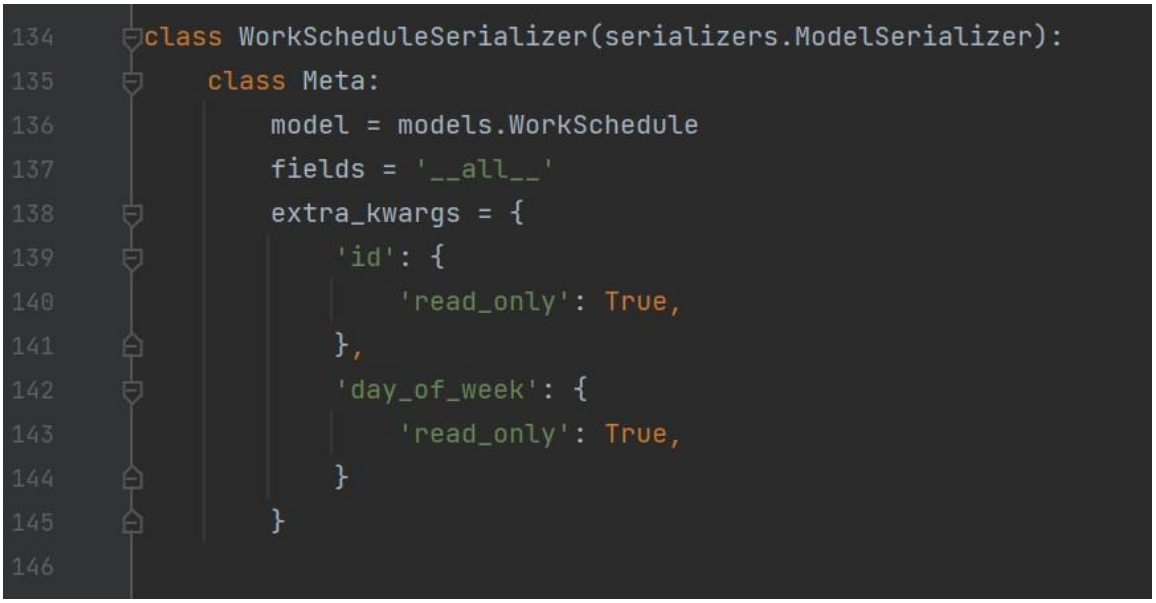
- **UserProfileSerializer:** It is used for serializing and deserializing user profile model.

Code:

```
class UserProfileSerializer(serializers.ModelSerializer):
    attendances = ArrivalDepartureSerializer(many=True, read_only=True,...)

    class Meta:
        model = models.UserProfile
        fields = '__all__'

        extra_kwargs = {
            'user_id': {
                'read_only': True,
            }
        }

    def create(self, validated_data):
        next_id = 1
        if models.UserAuthentication.objects.last() is not None:
            next_id = models.UserAuthentication.objects.last().id + 1

        user_id = validated_data['department'][0] + validated_data['designation'][0] + v
        user = models.UserAuthentication.objects.create_user(
            user_id=user_id,
            password=user_id
        )

        user_profile = models.UserProfile.objects.create(
```

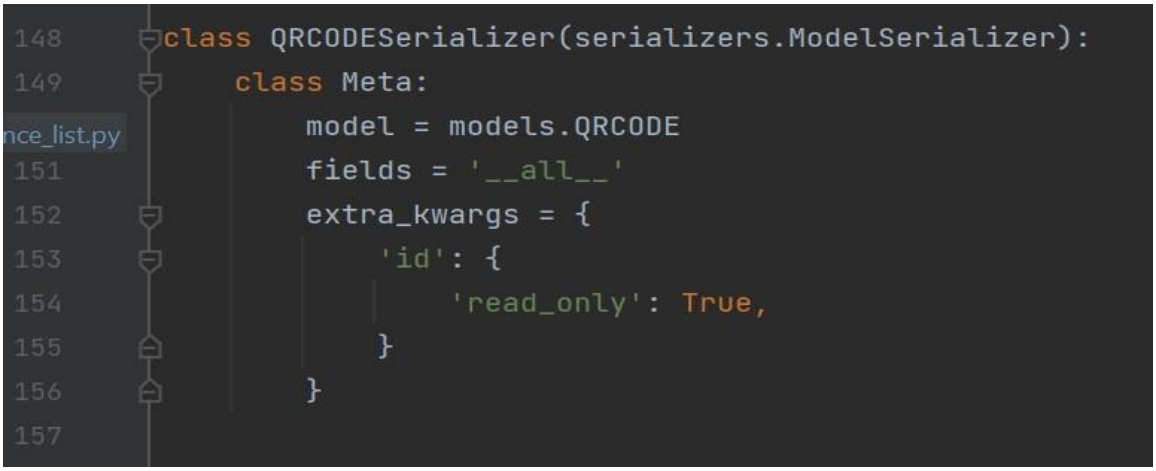
- **WorkScheduleSerializer:** It is used for serializing and deserializing work schedule model.

Code:



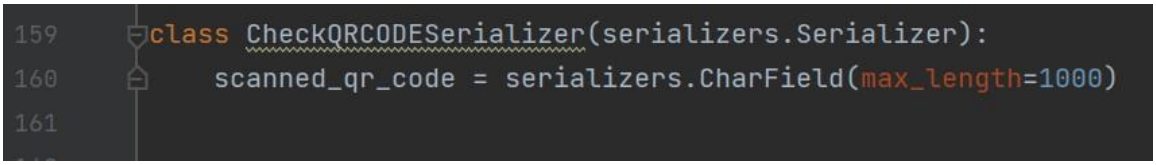
```
class WorkScheduleSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.WorkSchedule
        fields = '__all__'
        extra_kwargs = {
            'id': {
                'read_only': True,
            },
            'day_of_week': {
                'read_only': True,
            }
        }

```

- **QRCODESerializer:** It is used for serializing and deserializing QR code model.
  Code:



```
148  class QRCODESerializer(serializers.ModelSerializer):
149      class Meta:
nce_list.py          model = models.QRCODE
151                  fields = '__all__'
152                  extra_kwargs = {
153                      'id': {
154                          'read_only': True,
155                      }
156                  }
157
```

- **CheckQRCODESerializer:** It is used for serializing and deserializing request data from check QR code view.
  Code:

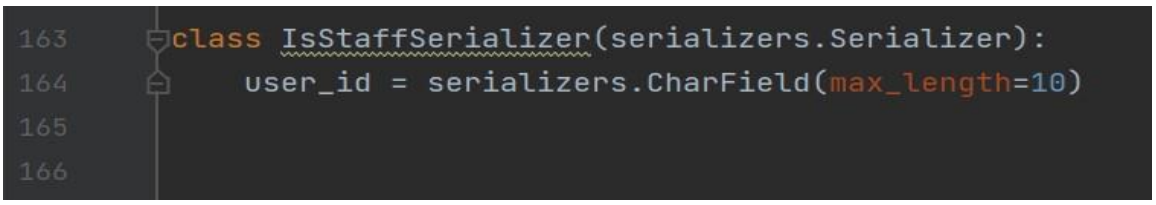


```
159  class CheckQRCODESerializer(serializers.Serializer):
160      scanned_qr_code = serializers.CharField(max_length=1000)
161
```

- **ArrivalDepartureSerializer:** It is used for serializing and deserializing arrival departure model.
Code:

```
class ArrivalDepartureSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.ArrivalDeparture
        fields = '__all__'

        ordering = ('-date',)
        extra_kwargs = {
            'id': {
              'read_only': True
            },
            'user_id': {
                'read_only': True,
            },
            'date': {
                'read_only': True,
            },
            'present': {
                'read_only': True,
            },
            'arrival_time_difference': {
                'read_only': True,
            },
            'departure_time_difference': {
                'read_only': True,
            },
            'total_time': {
```

- **IsStaffSerializer:** It is used for serializing and deserializing request data from is staff view.
Code:



```
class IsStaffSerializer(serializers.Serializer):
    user_id = serializers.CharField(max_length=10)
```

Cronjobs:

- **Every_2_hour_00:00: 0 */2 * * *:** This crontab executes some task every two hours every day.
  Code:

```python
from celery.schedules import crontab

broker_url = 'redis://redis:6379/0'
backend_url = 'redis://redis:6379/1'
accept_content = ['application/json']
task_serializer = 'json'
result_serializer = 'json'
beat_scheduler = 'django_celery_beat.schedulers:DatabaseScheduler'
timezone = 'Europe/Istanbul'
enable_utc = False
imports = ['qrtick_app.tasks']

beat_schedule = {
    'Every_2_hour_00:00': {
        'task': 'qrtick_app.tasks.create_daily_attendance_sheet',
        'schedule': crontab(minute='0', hour='*/2', day_of_week='*')
    }
}
```

- **Create_daily_attendance_list:** This task is used for creating daily attendance list for every working day.
  Code:

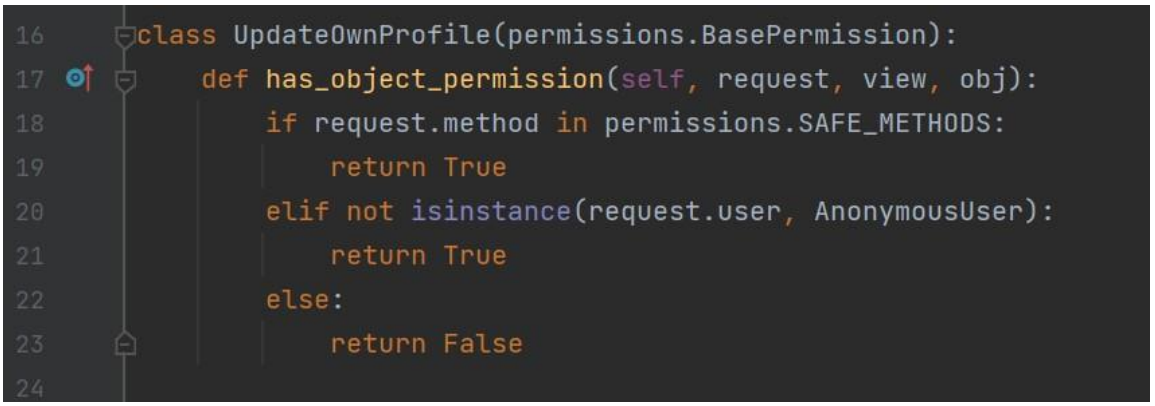
```python
from django.core.management.base import BaseCommand
from django.core import serializers
from qrtick_app import models
from datetime import date, datetime


class Command(BaseCommand):
    def handle(self, *args, **options):
        if not models.WorkSchedule.objects.get(id=date.today().weekday()).is_day_off:
            data = models.UserProfile.objects.all()
            for user in data:
                models.ArrivalDeparture.objects.get_or_create(
                    id=(datetime.now().strftime("%d_%m_%Y_") + user.user_id_id),
                    user_id=user, user_id_id=user.user_id_id,
                )
```

Permissions:

- **UpdateOwnAuthroization:** Employers can change or update employees' authorization information. Employees can change only their own authorization information.
  Code:

```
class UpdateOwnAuthorization(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.user.is_staff:
            return True
        else:
            if request.method in permissions.SAFE_METHODS:
                return True

            return obj.id == request.user.id
```

- **UpdateOwnProfile:** Only Employees can change or update their own information in the database.
  Code:



```
class UpdateOwnProfile(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        elif not isinstance(request.user, AnonymousUser):
            return True
        else:
            return False
```

- **UpdateWorkSchedule:** Only employers can change or update the work schedule. Employees can only get work schedule information.
  Code:

```
class UpdateWorkSchedule(permissions.IsAuthenticated):
    def has_object_permission(self, request, view, obj):
        if request.method in ('POST', 'DELETE'):
            return False
        elif request.method in ('GET',):
            return True
        elif request.method in ('PUT', 'PATCH') and request.user.is_staff:
            return True
        else:
            return False

```

- **UpdateArrivalTime:** Only employees can update their arrival and departure time.
  Code:

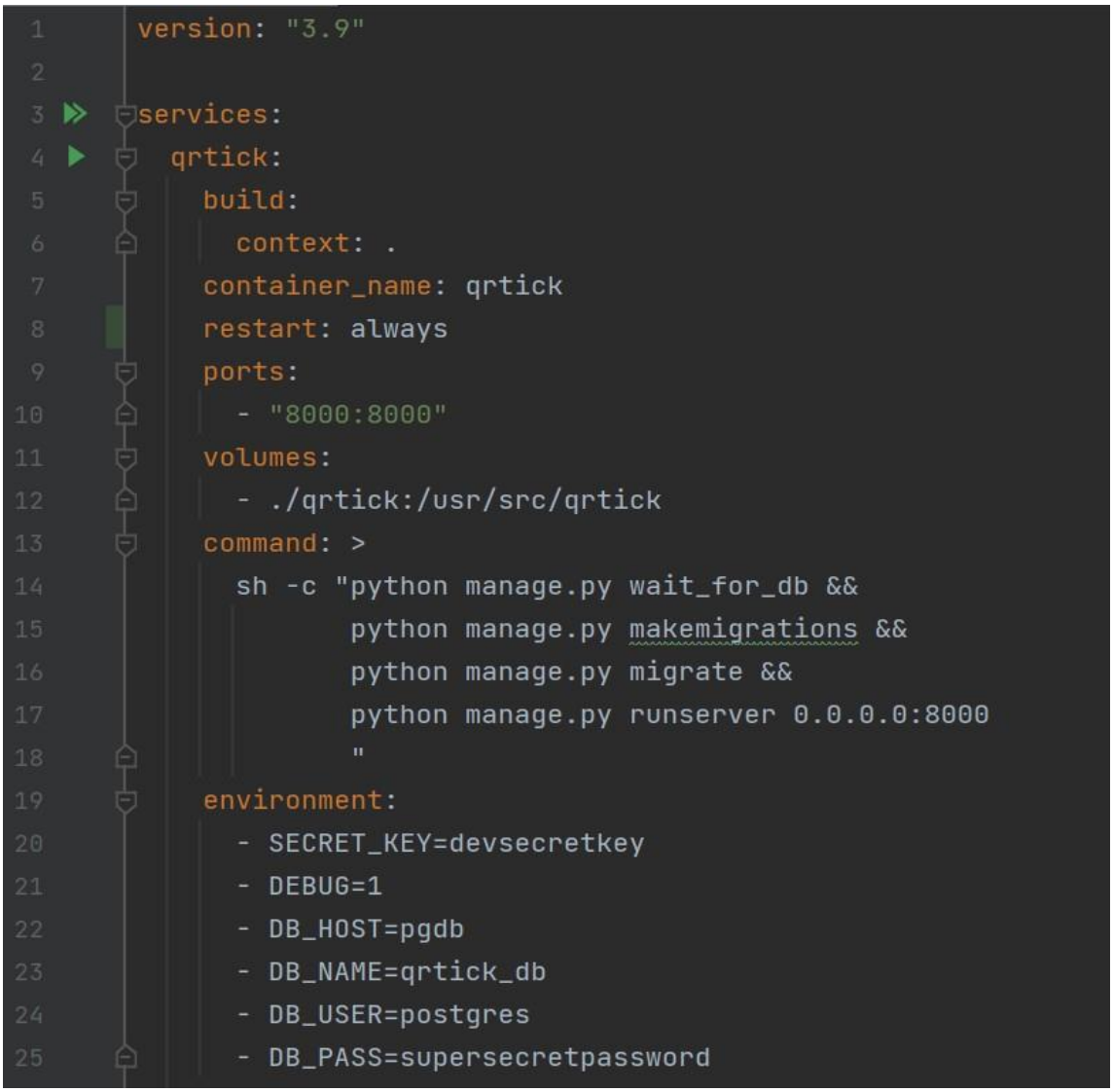
```
class UpdateArrivalTime(permissions.IsAuthenticated):
    def has_object_permission(self, request, view, obj):
        return obj.user_id_id == request.user.user_id
```

Docker:

- **Dockerfile:** This file creates python container in virtual machine.
  Code:

```
FROM python:3.10.4-bullseye
MAINTAINER MRR

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1



RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y netcat-openbsd gcc && \
    apt-get install -y libpq-dev && \
    apt-get clean




COPY ./qrtick /usr/src/qrtick
COPY ./requirements.txt /usr/src/qrtick
COPY ./scripts /scripts

WORKDIR /usr/src/qrtick
RUN pip install --upgrade pip
RUN pip install -r requirements.txt


RUN mkdir -p /vol/web/media
RUN mkdir -p /vol/web/static


RUN adduser --disabled-password --no-create-home --gecos '' app
RUN chown -R app:app /vol
RUN chmod -R 755 /vol
RUN chmod -R +x /scripts

ENV PATH="/scripts:$PATH"
USER app
EXPOSE 8000

#CMD ["run.sh"]
```

- **Docker-compose:** This file creates and executes all the containers in virtual machine.
  Code:



```
version: "3.9"

services:
  qrtick:
    build:
      context: .
    container_name: qrtick
    restart: always
    ports:
      - "8000:8000"
    volumes:
      - ./qrtick:/usr/src/qrtick
    command: >
      sh -c "python manage.py wait_for_db &&
             python manage.py makemigrations &&
             python manage.py migrate &&
             python manage.py runserver 0.0.0.0:8000
             "
    environment:
      - SECRET_KEY=devsecretkey
      - DEBUG=1
      - DB_HOST=pgdb
      - DB_NAME=qrtick_db
      - DB_USER=postgres
      - DB_PASS=supersecretpassword
```

```
    depends_on:
      - pgdb

  celery:
    restart: always
    build:
      context: .
    command: >
      sh -c "celery -A qrtick worker --loglevel=info &&
             celery -A qrtick  beat -l info --scheduler
      django_celery_beat.schedulers:DatabaseScheduler
             "
    volumes:
      - ./qrtick:/usr/src/qrtick
      - ./data/web:/vol/web
    environment:
      - SECRET_KEY=devsecretkey
      - DB_HOST=pgdb
      - DB_NAME=qrtick_db
      - DB_USER=postgres
      - DB_PASS=supersecretpassword
    depends_on:
      - pgdb
      - redis
      - qrtick

  pgdb:
    image: postgres:14.3-bullseye
    restart: always
    container_name: pgdb
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=qrtick_db
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=supersecretpassword
    volumes:
      - pgdata:/var/lib/postgresql/data/

  redis:
    restart: always
    image: redis:7.0.0-bullseye

volumes:
  pgdata:
```

6. Conclusion

The purpose of this project is to make attendance system simple, low cost, accessible, fast and secure.

6.1 Benefits

a. Benefits to users:

1. **Faster Attendance Taking:** It takes only a few seconds per employee for checking in.

2. **Least Paper Waste:** There is no need for papers, because the whole attendance data will be stored in the online cloud. This data can be accessed at any time.

3. **No Special Equipment Required**: No need for special equipment. It only needs smartphone and generated QR code for attending.

4. **No Electricity:** It works even when there is no electricity or Wi-Fi.

5. **Easy Assembling and Disassembling:** There is no need to worry about arranging laptops, staff members and stationery. It is easy to use a smartphone as a QR code scanner via mobile apps.

6. **Simple Pre-preparation:** There is no need for special training for the employees, because smartphone QR code scanners are simple and easy to use.

b. Benefits to me:

1. Increase my knowledge in mobile and backend development

2. Increase my self-confidence as a Mobile developer

3. In-depth understanding of QR codes

4. Increase my project management and problem solving skills

5. Increase my creative and critical thinking skills

Why did I choose this project?

I chose this project because wanted to learn more about QR codes. I think nowadays every programmer should know about it. Recent years, I began to notice QR codes everywhere, from Wi-Fi passwords up to COVID-19 test results, so started to interest in this topic. I searched some QR code projects from the internet and found that there are only a few projects about the QR code attendance system. These projects had some drawbacks, so I wanted to do a new better one. I think there are also many fields in which QR codes can be used, but I started it with QR code attendance system.

6.2 Ethics

- Employees must use smartphones which have Camera and GPS.

- Employees must bring their smartphones that were registered by the admin, to the work.

- Employees must use QR code attendance application for scanning QR codes that were provided by the employer.

- Employees must allow access to Camera and GPS for QR code attendance application on their smartphones.

- Employees must scan QR codes that were provided by the employer with their smartphones, while arriving and leaving.

- If there are problems with attending, employees must visit employer (admin) office for manual attendance.

- If employees change their phone to another one, they must inform about it to the employer (admin) for using QR code attendance application on their new phones.

- Attendance can be quantified and verified, and employers keep track of employee attendance and leave records.

6.3 Future Works

For now, I have no idea, but if there is something I can add or improve I will do it.

7. References

[1]: QR Attendance System => https://www.attendezz.com/#

[2]: Using QR codes for employee and task tracking => https://www.allgeo.com/blog/employee-and-task-tracking-using-qr-codes

[3]: QR Code Attendance for Faster Attendance Taking => https://hrmlabs.com/qr-code-attendance-for-faster-attendance-taking/

[4]: Как использовать QR-коды для отслеживания посещаемости и участия? => https://pageloot.com/ru/qr-коды-для/отслеживание-посещаемости/

[5]: Managing Employee Attendance => https://www.shrm.org/resourcesandtools/tools-and-samples/toolkits/pages/managingemployeeattendance.aspx