Fleet management system

Overview and introduction

ℹ️	**This documentation is available online here.**
ℹ️	Navigate to Report 1
ℹ️	Navigate to Report 2
ℹ️	Navigate to Report 3
ℹ️	Navigate to Report 4

Team

- Monika Rosa 239113
- Godfrey Mghase 239195
- Stanisław Puławski 239111
- Wiktor Muraszko 239109

Description

The main objective of this project is to create a IT System for managing car fleet for the company, which consists of a headquarter and few branch offices located in different cities (Lodz, Warsaw, Cracow). Our solution connects the information systems of company's headquarters and its branches and allows enterprise to manage their car fleet. We have prepared working Web Service and Flutter Android client.

Repositories

- Backend HQ repository here.
- Backend BO repository here.
- Frontend repository here.

Documentation, scripts (and special frontend for 1st repo) here

Technology stack

- Database
  - MySQL `MySQL Server 8.0.23-1debian10`
- Backend
  - Java Spring `Spring Boot (v2.4.3)`
- Frontend
  - Flutter `Flutter 2.2.2`
    - Android Flutter Android
    - Web Flutter Web

- Web container Flutter Web container

Production deployment

Production deployment links

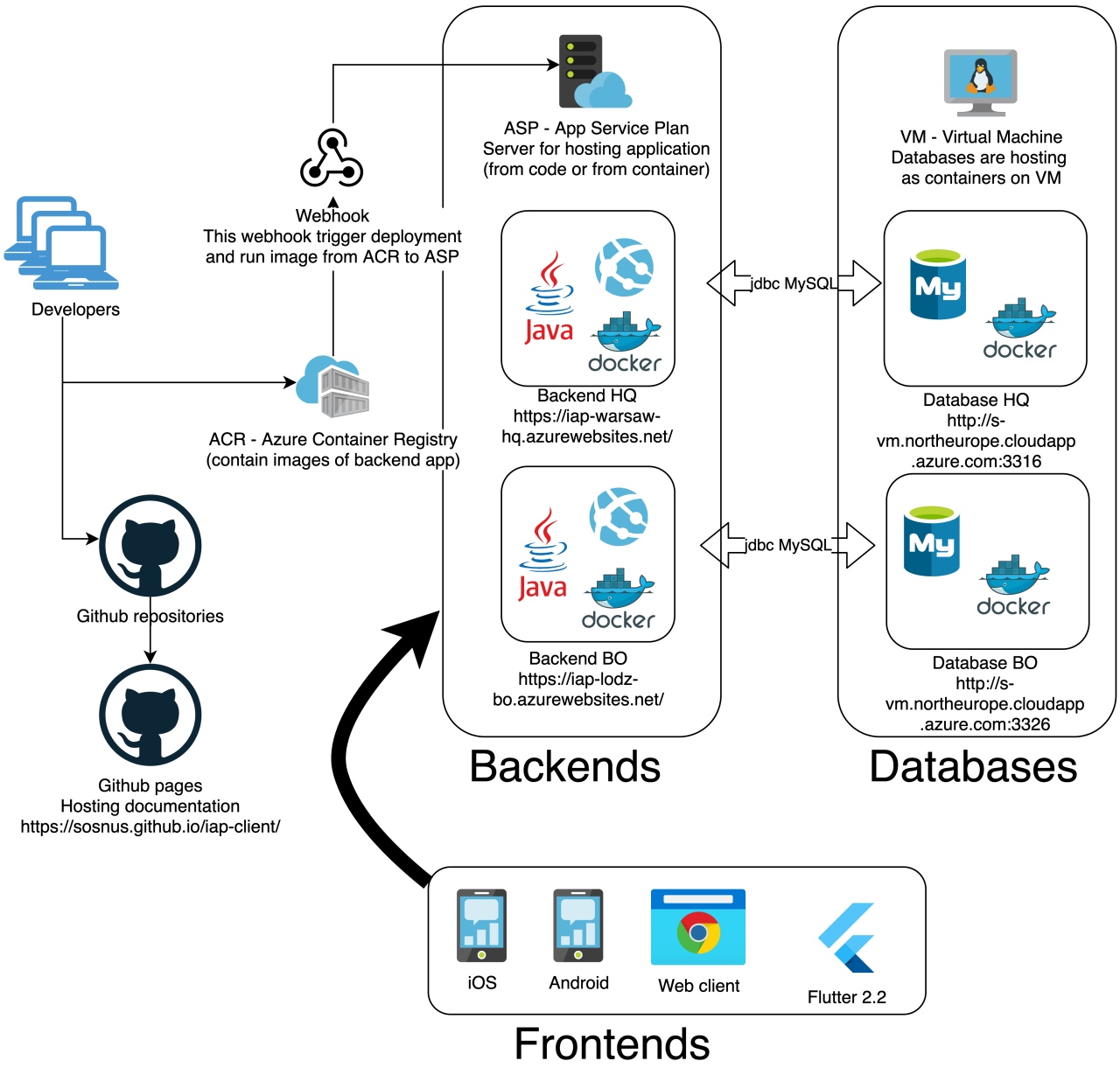
- HQ Warsaw office
- BO Lodz office

Backend containers are hosted on ASP (App Service Plan) with 28 other applications. ASP based on tier B2 and have: 3.5GB RAM and 2vCPU cores

Databases containers are hosted on Azure Virtual Machine with a lot of others containers and services. VM size: Standard B1ms 1vCPU, 2GB RAM. Containers are separated and are ready to migration.

Deployment diagram

Fleet Management System
Internet Application Programming



When developer build and push image, webhooks will deploy container at ASP.

```
docker login sosnuscontainers.azurecr.io
# login: ********
# password: *********

docker build -t sosnuscontainers.azurecr.io/iap-warsaw-hq .
docker push sosnuscontainers.azurecr.io/iap-warsaw-hq

docker build -t sosnuscontainers.azurecr.io/iap-lodz-bo .
docker push sosnuscontainers.azurecr.io/iap-lodz-bo

docker build -t sosnuscontainers.azurecr.io/iap-cracow-bo .
docker push sosnuscontainers.azurecr.io/iap-cracow-bo
```

Report 1 - Feasibility study of communication between systems

> ℹ️ **the purpose of the first report is presentation connection between database, backend and frontend applications**

First, test deploy consist of 3 parts:

- MySQL database
- Spring boot backend service
- Flutter Android client

test deployment

For communication test purpouse, database and backend was deployed on docker containers, on the same Virtual Machine. VM size: Standard B1ms 1vCPU, 2GB RAM

- Backend address here
- Database address here

Before container deployment, it is necessary to enable new firewall rules:

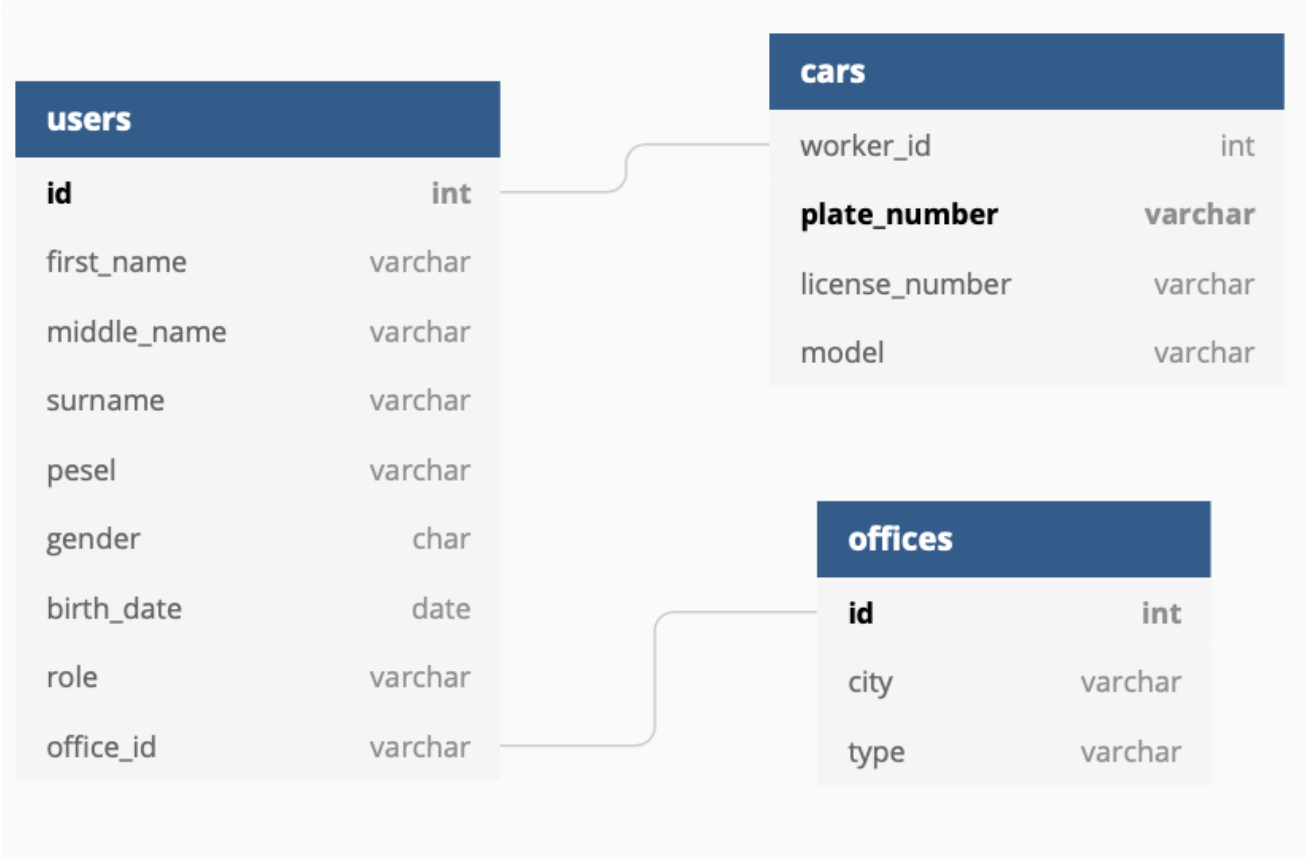
```
ufw allow 22
ufw allow 3306
ufw allow 8081
ufw reload
```

Database - deploy and test

For database deployment, we use simple bash script to run new docker container from Docker Hub

```
docker run --name some-mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=my-secret-p
```

We create first sql schema for this project using dbdiagram.io tool. Probably we will have some changes here in the future. Online documentation for our schema is here



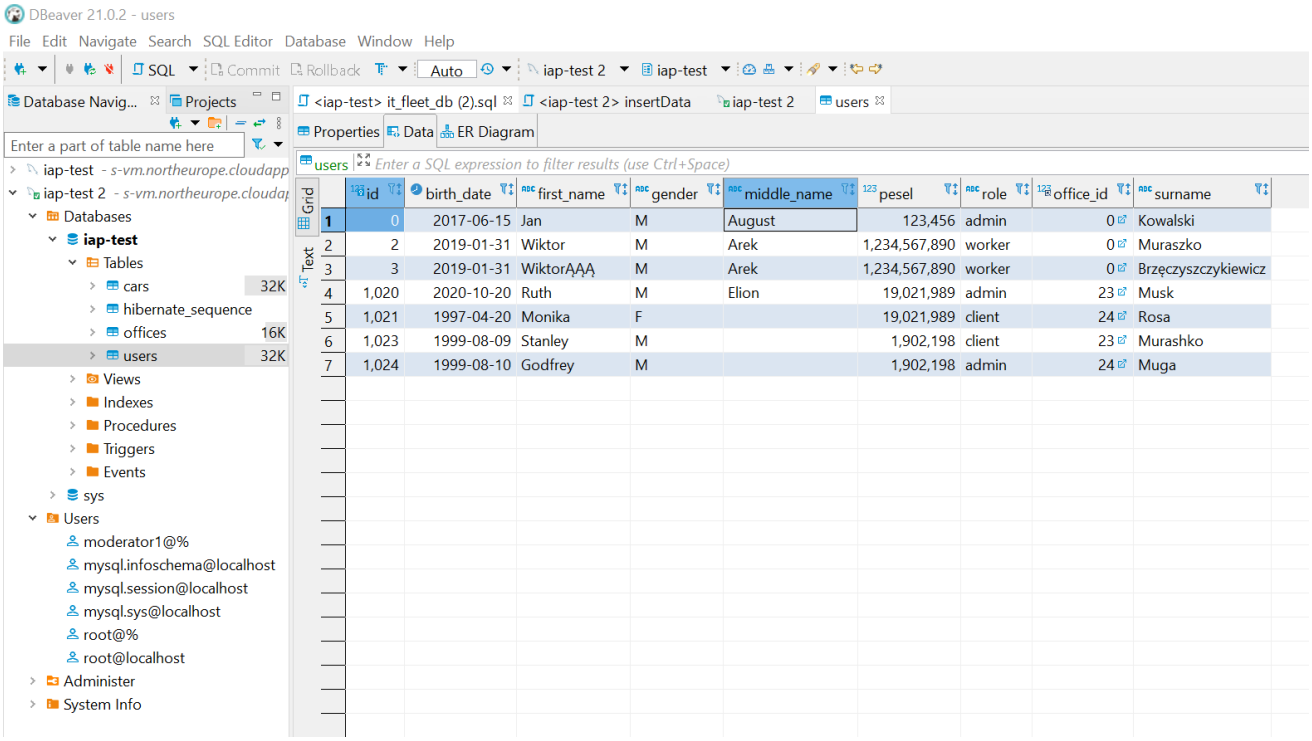
Next we create new database users to enable easy synchronous access for the rest of the team members. It is important for future deployments, we need independent user for every backend instance. Here is how we created the users.

```
CREATE USER 'moderator1'@'%' IDENTIFIED BY '1234';
GRANT ALL PRIVILEGES ON * . * TO 'moderator1'@'%';
FLUSH PRIVILEGES;
```

Next we add dump data for testing using the script below.

```
INSERT INTO `users` (`id`, `first_name`, `middle_name`, `surname`, `pesel`,

(1020, 'Ruth', 'Elion', 'Musk', 19021989, 'M', '2020-10-20', 'admin', '23'),
(1021, 'Monika', '', 'Rosa', 19021989, 'F', '1997-04-20', 'client', '24'),
(1023, 'Stanley', '', 'Murashko', 1902198, 'M', '1999-08-09', 'client', '23'
(1024, 'Godfrey', '', 'Muga', 1902198, 'M', '1999-08-10', 'admin', '24');

INSERT INTO `cars` (`worker_id`, `plate_number`, `license_number`, `model`)
(1020, '1520', '5060', 'Toyota'),
(1021, '1521', '5061', 'Nissan'),
(1023, '1522', '5062', 'Hundai'),
(1024, '1523', '5063', 'Toyota');

INSERT INTO `offices` (`id`, `city`, `type`) VALUES
(23, 'Lodz', 'HQ'),
(24, 'Warsaw', 'BO');
```

We test Our database deployment using DBeaver desktop app:

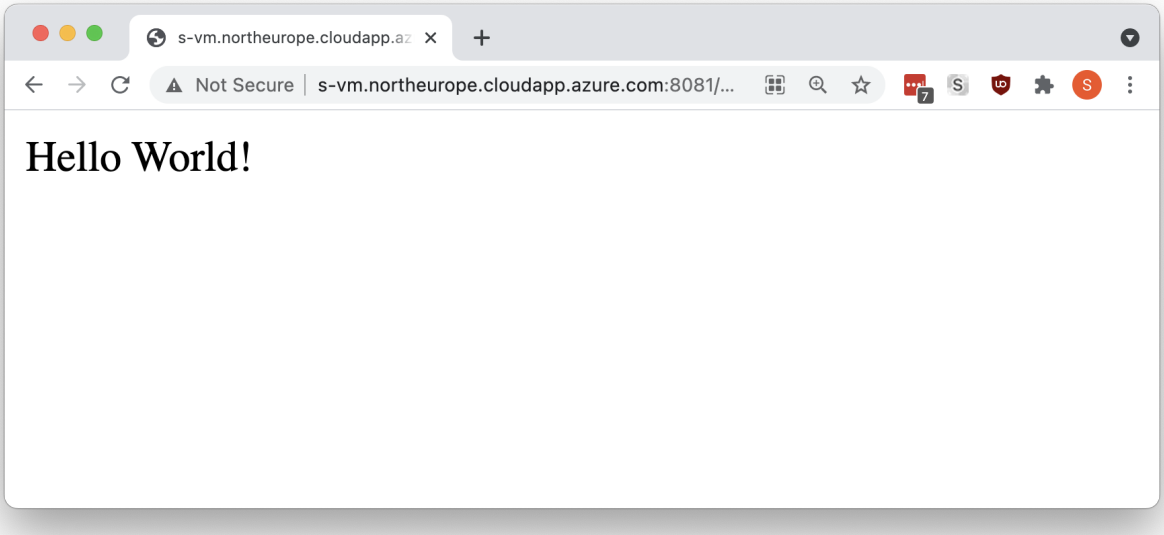


Backend - deploy and test

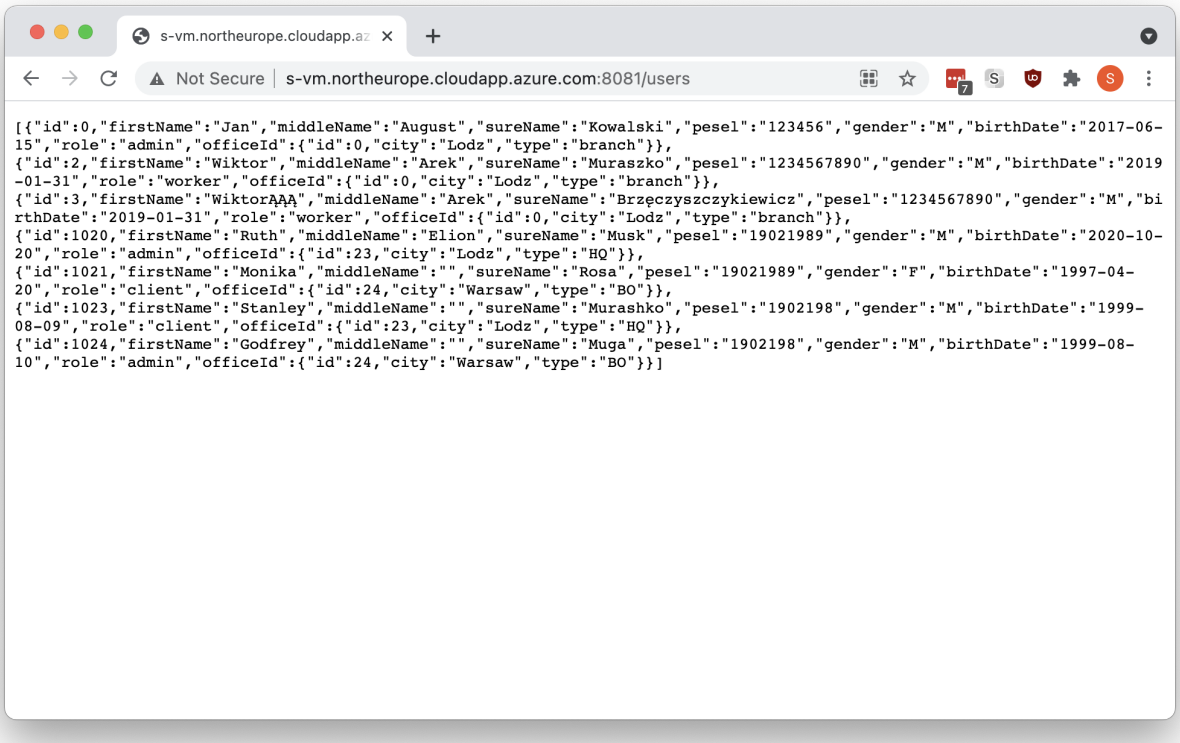
To deploy backend application, we need build container from source code and run it, using commands below:

```
git clone https://github.com/Wredter/IAP_project_1
cd ./IAP_project_1
docker stop iap-back-container -t 1
docker rm iap-back-container
docker build --no-cache -t iap-back .
docker run -d -p 8081:80 --name=iap-back-container iap-back
```

Now we can test backend project, by send http get request on `/hello` endpoint. In Our case, we can see it on address`



On endpoint `/users` we can see list of `elements` from `users` collections



Frontend - test

For first project iteration, we need implement a few features in frontend application:

- Connection to API
- Users model
- User list view

Connection to API

Class `FleetService` contains access to backend API using `package:http/http.dart` library

Users

User class is very simple, and help us to present data from Users collection from backend. We add it for test purpose, in next iteration this class will be modified, and contain expanded constructors and other methods

```
class User {
  int id;
  String pesel;
  String firstName;
  String sureName;

  User({this.id, this.pesel, this.firstName, this.sureName});
}
```

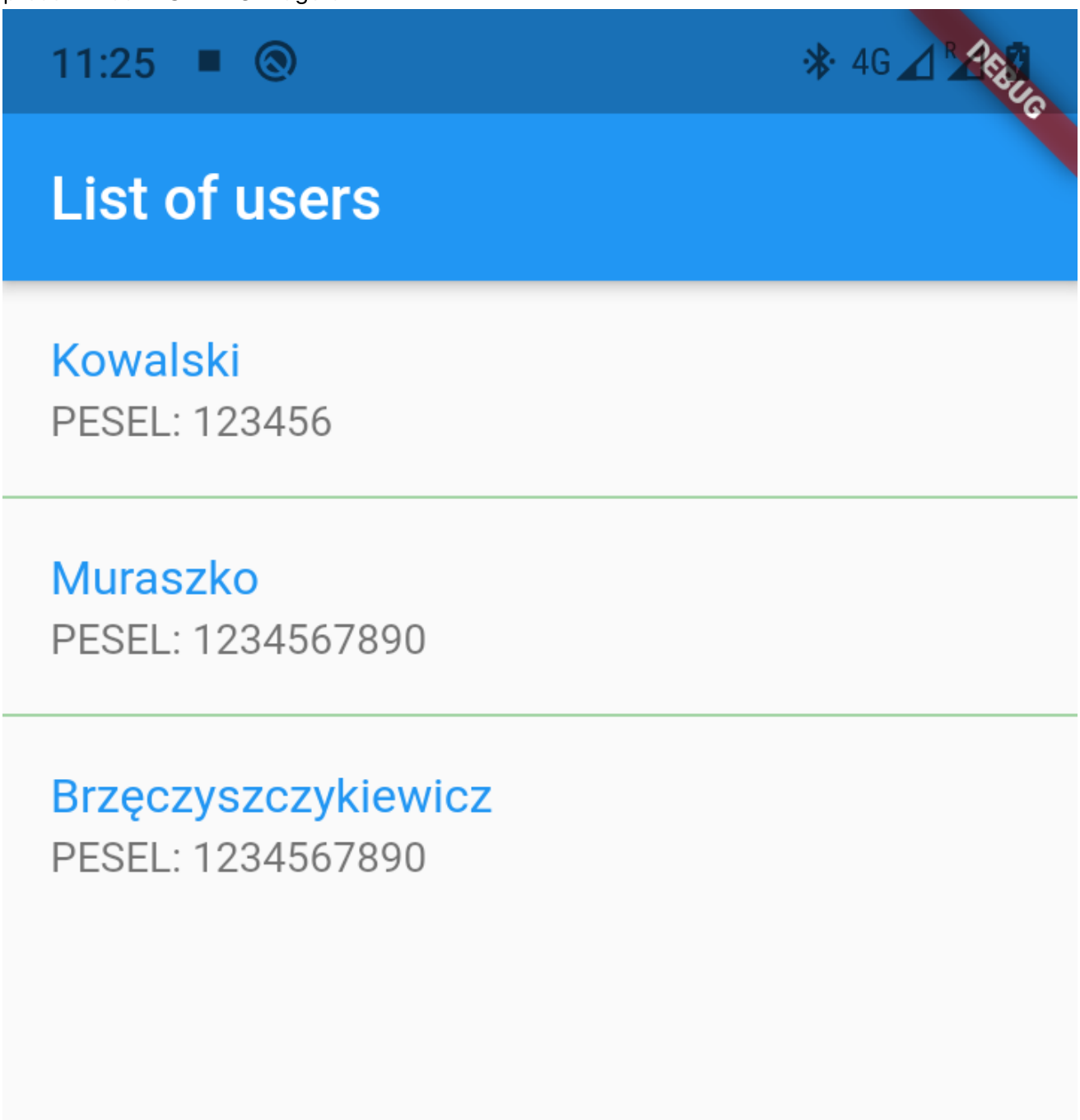
User list view

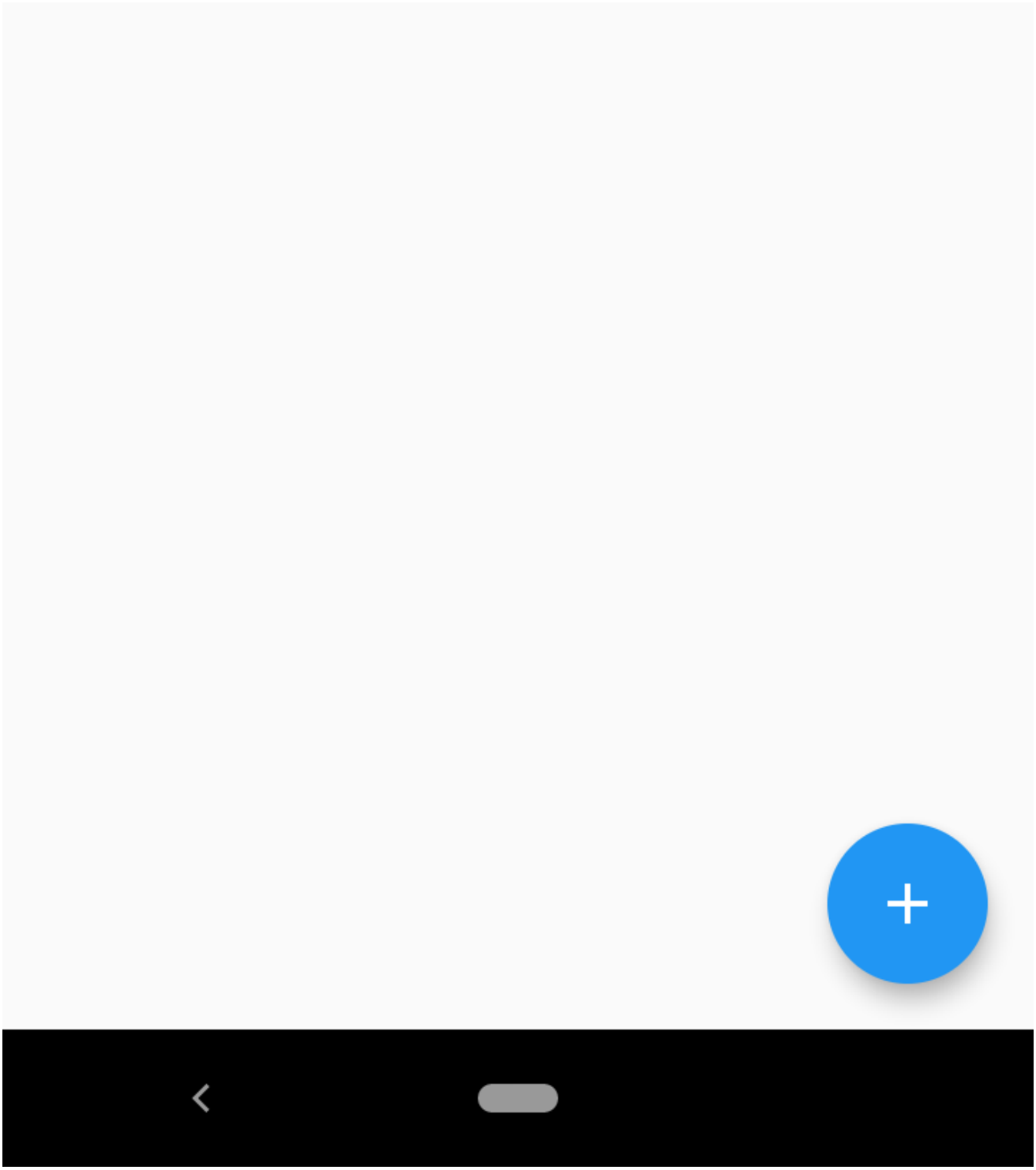
Main part of view in this project is builder, which can dynamic add new elements to ListView collection

```
Builder(
        builder: (_) {
          if (_isLoading) {
            return Center(child: CircularProgressIndicator());
          }


          if (_apiResponse.error) {
            return Center(child: Text(_apiResponse.errorMessage));
          }

          return ListView.separated(
            separatorBuilder: (_, __) =>
                Divider(height: 1, color: Colors.green),
            itemBuilder: (_, index) {
              return Dismissible(
                key: ValueKey(_apiResponse.data[index].id),
                direction: DismissDirection.startToEnd,
                onDismissed: (direction) {},
                confirmDismiss: (direction) async {
                  final result = await showDialog(
                      context: context, builder: (_) => UserDelete());
                  print(result);
                  return result;
                },
                background: Container(
                  color: Colors.red,
                  padding: EdgeInsets.only(left: 16),
                  child: Align(
                    child: Icon(Icons.delete, color: Colors.white),
```

```
                    alignment: Alignment.centerLeft,
                  ),
                ),
                child: ListTile(
                    title: Text(
                      _apiResponse.data[index].sureName,
                      style: TextStyle(color: Theme.of(context).primaryColor
                    ),
                    subtitle: Text('PESEL: ${_apiResponse.data[index].pesel}
                    onTap: () {}),
              );
            },
            itemCount: _apiResponse.data.length,
          );
        },
      ),
```

Application get list of users using service `fleet_service`, convert it into list of `User` objects, and present it as `ListTile` widgets:





RESOURCES for Report 1

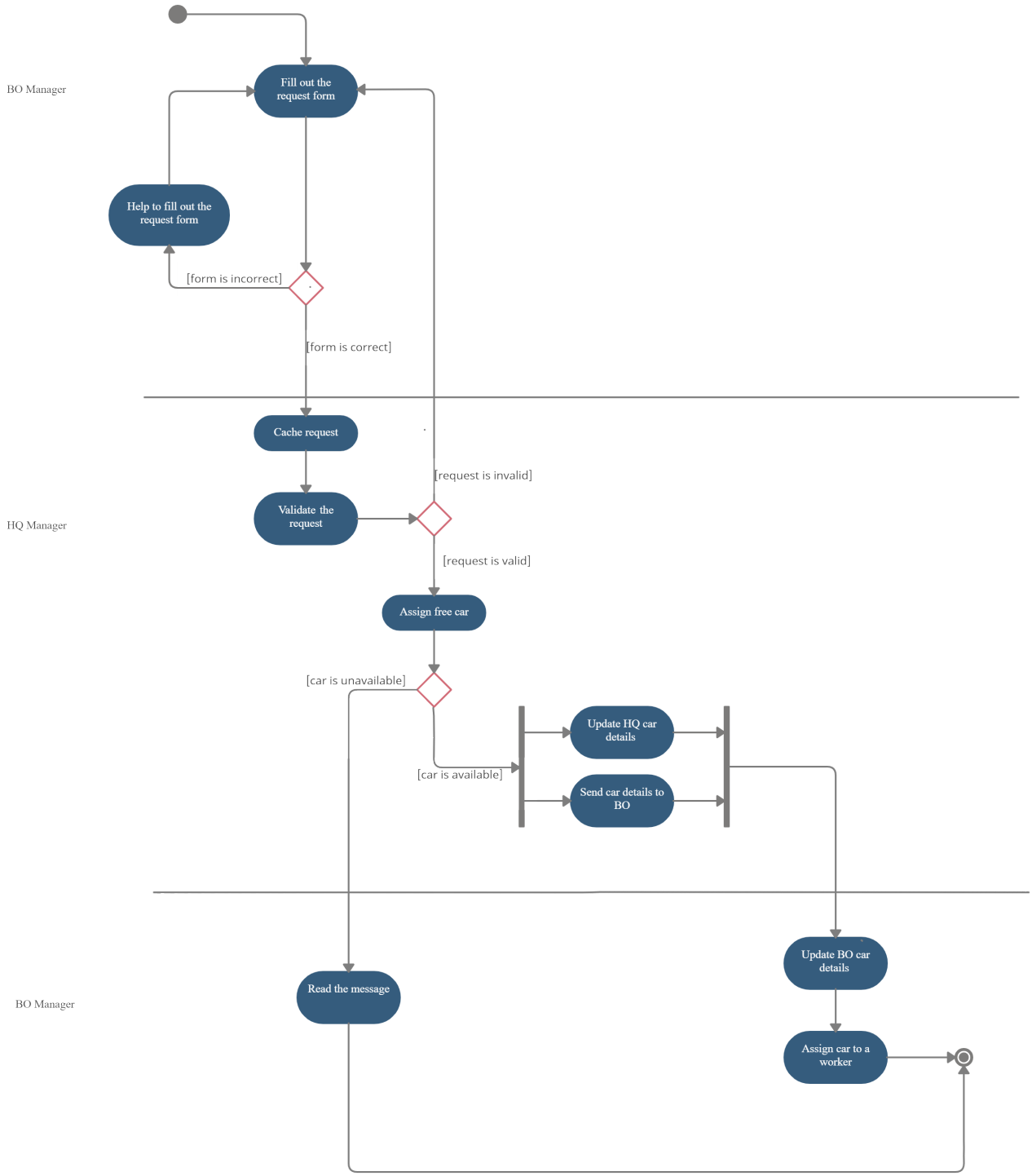
- REST API in flutter
- Flutter documentation
- Create database users

Report 2 - Establish the business context, sketch the system architecture, select technology

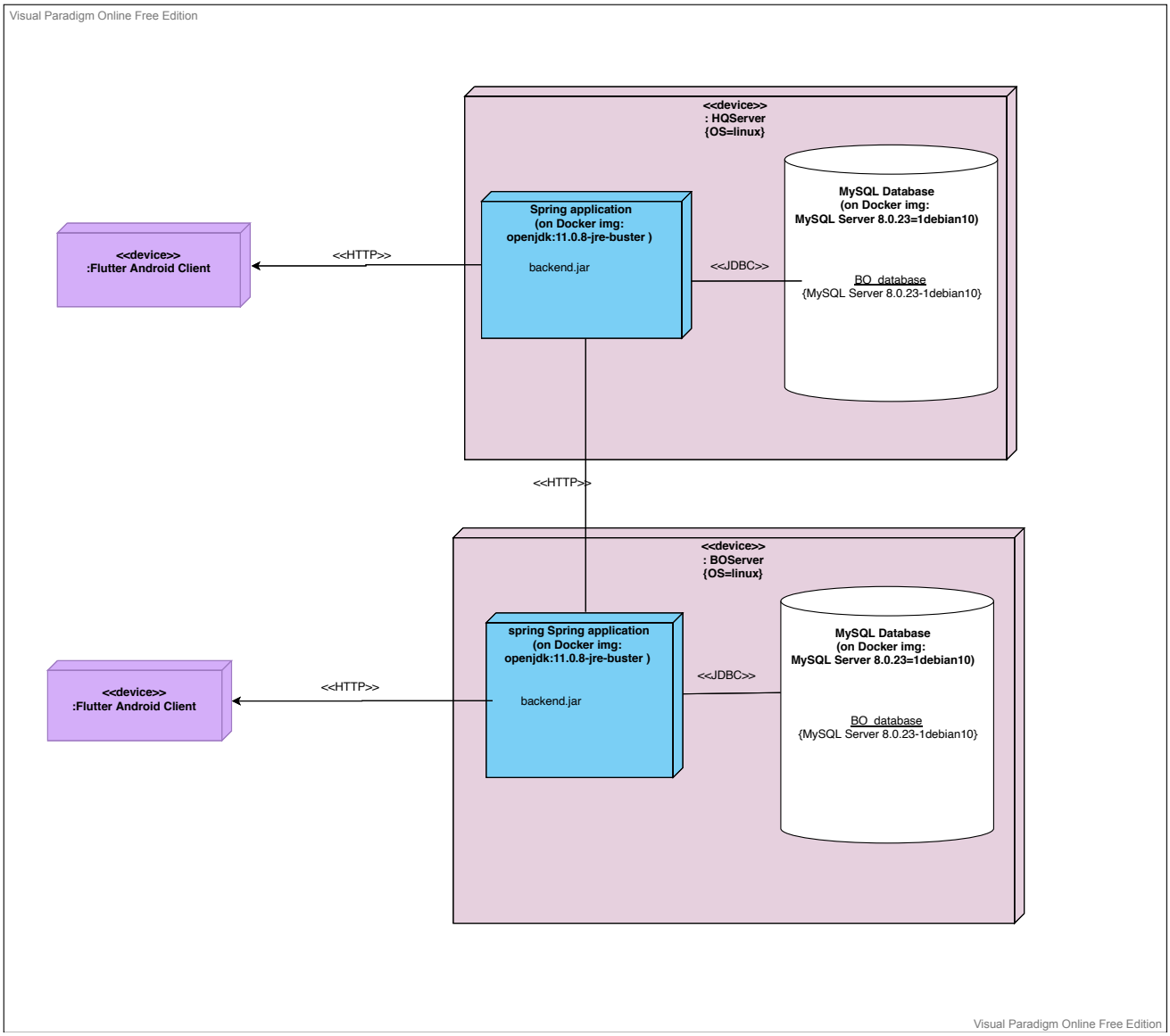
System actors

- Headquarters Manager - Headquarters Manager represents a system role with the authorization to assign car to the branch office, update HQ car details.
- Branch Office Manager - Branch Office Manager represents a system role with the authorization to fill out the request form for the car, assign car to a worker, update BO car details.

Activity diagram



Deployment diagram



Description of use cases

1. Fill out the request form (when worker needs new car)

- Type: general
- Users: Branch Office Manager
- Initial conditions: Branch Office Manager confirmed their identity via login and password.
- Typical step sequence:

1. Branch Office Manager chooses an option to fill out the request form.
2. BO Manager fills out the request form

- Alternative sequence of steps:
  2a. Request form was filled incorrectly, BO Manager repeats filling process. 2b. Request form was filled incorrectly, BO Manager displays help video and repeats the process.
- Final conditions: the request form has been successfully filled.

2. Assign free car (to branch)

- Type: general
- Users: Headquarters Manager

- Initial conditions: Headquarters Manager confirmed their identity via login and password.
- Typical step sequence:

1. Cache requests
2. The request validation
3. HQ Manager assigns free car
4. Update HQ car details
5. Send car details to BO

- Alternative sequence of steps: 2a. The request is invalid (error message is sent to BO)
  3a. There is no free car (notification is sent to BO)
- Final conditions: the free car has been assigned to BO

3. Assign car to a worker

- Type: general
- Users: Branch Office Manager
- Initial conditions: Branch Office Manager confirmed their identity via login and password.
- Typical step sequence:

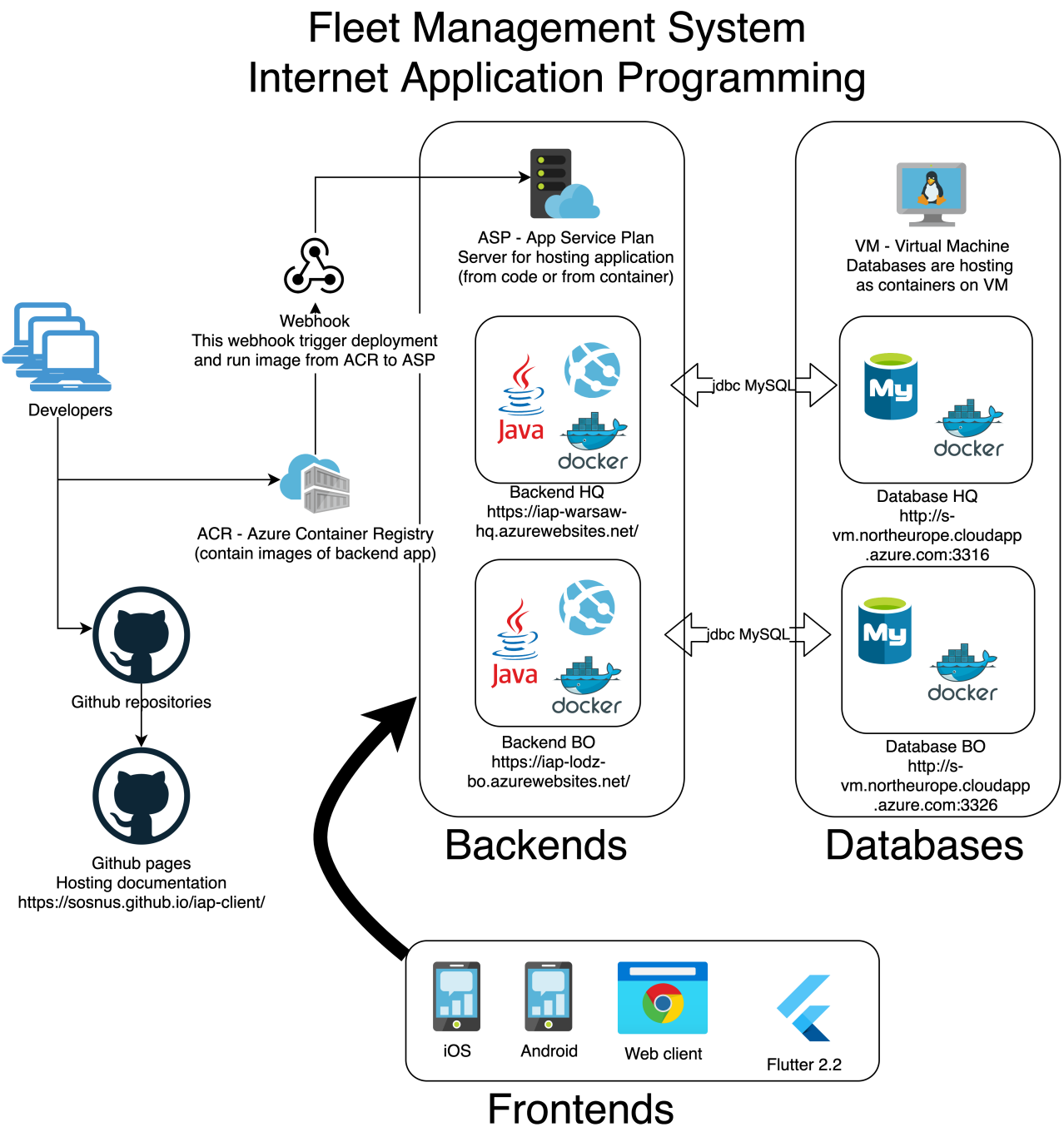
1. Branch Office Manager assigns car to the worker.
2. BO car details are updated.

- Final conditions: car is assigned to worker.

RESOURCES for report 2

- UML deployment diagram guide
- UML activity diagram
- http://www.agilemodeling.com/artifacts/deploymentDiagram.htm
- http://www.agilemodeling.com/style/activityDiagram.htm

Report 3 - Implementation of data exchange and synchronization (headquarters(HQ) / offices(BO))

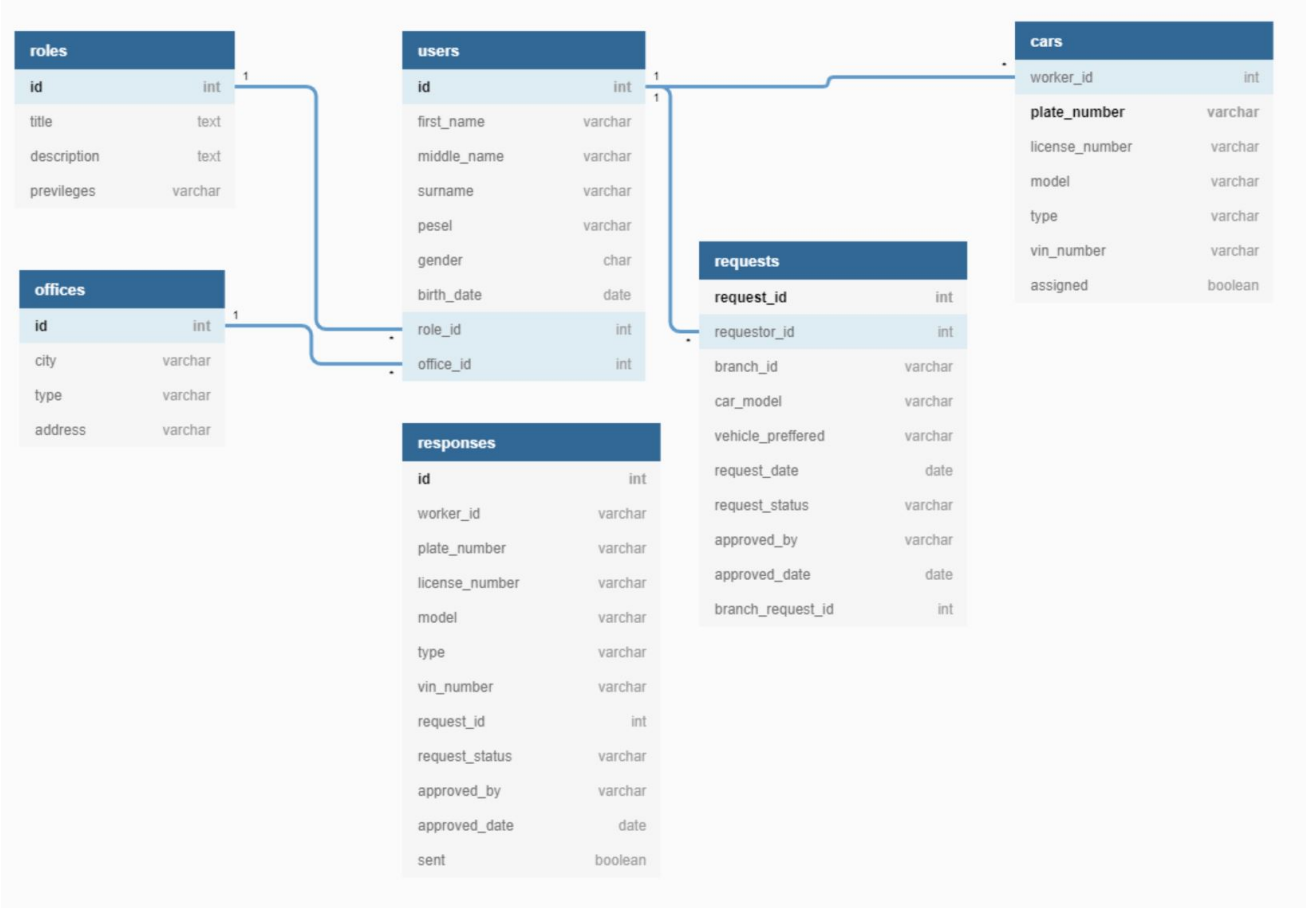


Application - Backend

Data Models

As of now the data models for headquarters and branch office look pretty similar except for the responses model which takes care of all the responses the hq have ever sent to the branch office and some columns entries at branch office model are meant for data synchronization purposes.

HQ Model

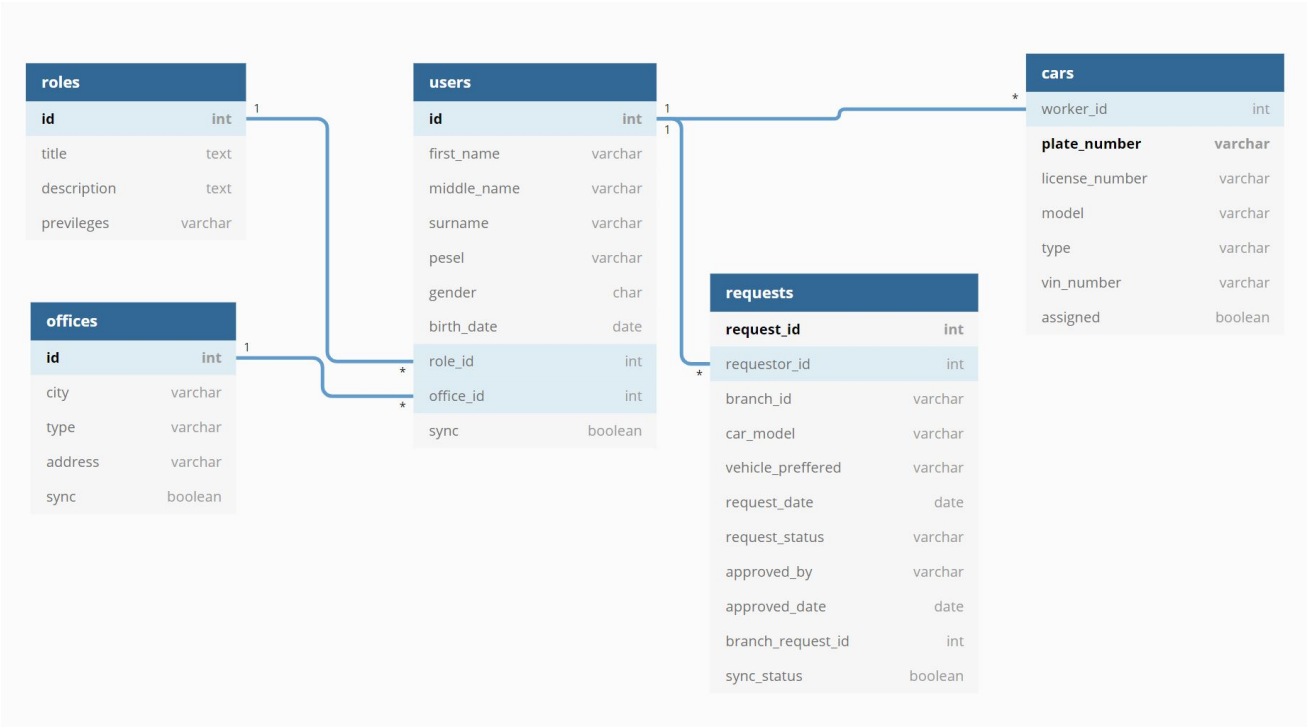


Below we paste a sample code for the HQ office entity as implemented in our application.

```
@Entity
@Table(name = "offices")
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Builder
public class Office {
    @Getter
    @Setter
    @Id
    @NotNull
    @Column(name = "id")
    public long id;

    @Getter
    @Setter
    @Column(name = "city")
    public String city;

    @Getter
    @Setter
    @Column(name = "type")
    public String type;

    @Getter
    @Setter
    @Column(name = "address")
```

```
    public String address;
}
```

BO Model



Below we paste a sample code for the bo users entity as implemented in our application.

```
@Entity
@Table(name = "users")
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class User {
    @Getter
    @Setter
    @Id
    @NotNull
    @Min(40000)
    @Max(60000)
    private long id;


    @Getter
    @Setter
    @Column(name = "first_name")
    private String firstName;

    @Getter
    @Setter
    @Column(name = "middle_name")
    private String middleName;
```

```
    @Getter
    @Setter
    @Column(name = "surname")
    private String surname;

    @Getter
    @Setter
    @Size(min = 10, max = 12)
    @Column(name = "pesel")
    private String pesel;

    @Getter
    @Setter
    @Column(name = "gender")
    private char gender;

    @Getter
    @Setter
    @Temporal(TemporalType.DATE)
    @Column(name = "birth_date")
    private Date birthDate;



    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "office_id", nullable = false, referencedColumnName =
    private Office officeId;

    @Getter
    @Setter
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "role_id", nullable = false, referencedColumnName = '
    private Role roleId;

    @Getter
    @Setter
    @Column(name = "sync")
    private Boolean sync;


    public User(String firstName, String middleName, String surname, String
        this.firstName = firstName;
        this.middleName = middleName;
        this.surname = surname;
        this.pesel = pesel;
        this.gender = gender;
        this.birthDate = birthDate;
        this.roleId = roleId;
        this.officeId = officeId;
    }
}
```

Implementation of data exchange and synchronization

We use RestTemplate class under org.springframework.web.client package to manage data exchange, and for synchronization, we use TaskSchedular interfaces in spring boot by enabling the scheduling at the main application and use @ annotation for each method/class we need to schedule.

Enabling Scheduling in spring boot main application, @EnableScheduling and @EnableSwagger2 for documentation.

```
package com.IAP.car_exchange;

@RestController
@SpringBootApplication
@EnableScheduling
@EnableSwagger2
public class CarExchangeApplication {

	public static void main(String[] args) {
		SpringApplication.run(CarExchangeApplication.class, args);
	}

	public Docket productApi() {
		return new Docket(DocumentationType.SWAGGER_2).select()
		    .apis(RequestHandlerSelectors.basePackage("com.IAP.car_e
		   }

	@GetMapping("/hello")
	public String hello(@RequestParam(value = "name", defaultValue = "Wo
		return String.format("Hello %s!", name);
	}

}
```

Data exchange and synchronization for branch office

Firstly, assuming the office, role and user have been commissioned already, A user will create a request, this request will be stored in a local branch database and the RestTemplate.postForObject() method will be invoked to transfer this request to headquarter office. Incase of any failure, scheduling/synchronization has been enabled in such a way that after every 10 seconds the failed requests will be retransmitted again. Entities like User,Office, and Request have beeen synchronized such that there is a copy of each of them at the HQ server/database.

a. Synchronization properties definition

```
package com.IAP.car_exchange;

public class SynchronizationConfiguration {

	// connection to HQ
	public static final String uriToHq = "http://s-vm.northeurope.cloud
	public static final String uriToHqUser = "http://s-vm.northeurope.cl
	public static final String uriToHqOffice = "http://s-vm.northeurope
```

```
        public static final String uriFromOffice = "http://s-vm.northeurope.

        // connection from HQ
        public static final String  uriFromHq = "details";

        // delay sync timer in millisecond
        public static final int delay = 10000;



}
```

b. Synchronization service controller

```
@RestController
public class SyncronizationController {


        @Autowired
        SyncronizationService syncronizationService;
        @Autowired
        SynchronizeUserService synchronizeUserService;
        @Autowired
        SynchronizeOfficeService synchronizeOfficeService;

        private static final Logger logger = LoggerFactory.getLogger(Syncror
        private static final DateTimeFormatter dateFormat = DateTimeFormatte


        @Scheduled(fixedDelay=SynchronizationConfiguration.delay)
        public void synchronizeRequests() {
                logger.info("Scheduled Synchronization Task for requests ::
                try {
                        ResponseEntity<String> sync = syncronizationService.
                } catch(Exception ex) {
                        logger.error("We have run into an error: we will rec
                }
        }


        @Scheduled(fixedDelay=SynchronizationConfiguration.delay)
        public @ResponseBody void synchronizeUsers() {
                logger.info("Scheduled Synchronization Task for users :: Exe

                  try { ResponseEntity<String> sync = synchronizeUserServic
                  catch(Exception ex) {
                  logger.error("We have run into an error: we will reconnec
                  throw ex;
                  }
        }

        @Scheduled(fixedDelay=SynchronizationConfiguration.delay)
        public @ResponseBody void synchronizeOffices() {
                logger.info("Scheduled Synchronization Task for Offices :: E
```

```
                    try { ResponseEntity<String> sync = synchronizeOfficeServ
                    catch(Exception ex) {
                    logger.error("We have run into an error: we will reconnec
                    throw ex;
                    }
            }

    }
```

c. Sychronization service implementation for requests as an example.

```
    @Repository
    @Data
    public class SyncronizationService {
            final RequestRepository requestRepository;

            RequestData requestData = new RequestData();

              public SyncronizationService(RequestRepository requestRepository)
              this.requestRepository = requestRepository; }


            public List<Request> getAllUnsycedRequests(){

                    return requestRepository.getAllUnyced();
            }

            public ResponseEntity<String> tryToSync(){


                    /*
                     * List<Request> requests = syncronizationService.getAllUns
                     * if(requests.isEmpty()) { System.out.println("am empty "+
                     */
                    List<Request> requests = getAllUnsycedRequests();
                    if (requests.isEmpty() == false) {
                            for(Request r:requests) {
                                    //System.out.println(r.getRequestorId());

                                    // Prepare the data to be sent to HQ for thi

                                    requestData.setRequestorId(r.getRequestorId
                                    requestData.setBranchId(r.getBranchId());
                                    requestData.setCarModel(r.getCarModel());
                                    requestData.setVehiclePreffered(r.getVehicle
                                    requestData.setRequestDate(r.getRequestDate
                                    requestData.setRequestId(r.getRequestId());

                                    // Now send it
                                    //String uri = "http://localhost:8080/reques
                                    RestTemplate restTemplate = new RestTemplate
                                    try {
```

```
                                        try {
                                        RequestData result = restTemplate.postForOb
                                        } catch(Exception e) {
                                                System.out.println("We are sorry so
                                                //System.out.println(requestData.ge
                                                //e.printStackTrace();
                                                throw e;
                                        }

                                        // If success, change the sync status
                                        r.setStatus(true);
                                        requestRepository.save(r);
                                }
                        }

                        return ResponseEntity.ok(null);
                }

        }
```

Data exchange and synchronization for head quarter office

Here, we transfer and synchnronize all the responses we have ever sent to the branch office. After we receive the car request from the branch office, we log it into the requests table and then the headquarter manager can check the pending requests, find the requested car by filtering, copy the requestid and just hit assign button and the car assignment process will proceed. Under the hood, if the car is found it's detail is packed and sent to the branch office and the necessary updates in the hq table entities is performed. If not found then the rejection notification is appended on the response. Again, the neccessary fields are updated to tell that this particular request has been processed and it is rejected.

Background services

a. Sychronization properties definition

```
package com.IAP.car_exchange;

public class SynchronizationConfiguration {

        // connection to BO
        public static final String uriToBo = "http://s-vm.northeurope.cloud

        // connection from BO
        public static final String  uriFromBo = "request";

        // delay sync timer in millisecond
        public static final int delay = 5000;

}
```

b. Responses synchronization controller

```
@RestController
public class ResponsesSyncController {
        @Autowired
        ResponsesSycService responsesSycService;

        private static final Logger logger = LoggerFactory.getLogger(Respons
        private static final DateTimeFormatter dateFormat = DateTimeFormatte


        @Scheduled(fixedDelay=SynchronizationConfiguration.delay)
        public void synchronizeResponses() {
                logger.info("Scheduled Synchronization Task :: Execution Tim
                try {
                        ResponseEntity<String> sync = responsesSycService.tr
                } catch(Exception ex) {
                        logger.error("We have run into an error: we will rec
                }
        }

}
```

c. Responses synchronization service

```
@Repository
@Data
public class ResponsesSycService {

        final ResponsesRepository responsesRepository;

        Response response = new Response();

        public ResponsesSycService(ResponsesRepository responsesRepository)
                this.responsesRepository = responsesRepository;
        }

        public List<Responses> getAllUnsycedResponses(){

                return responsesRepository.getAllUnsyc();
        }


        public ResponseEntity<String> tryToSync(){

                List<Responses> responses = getAllUnsycedResponses();
                if (responses.isEmpty() == false) {
                        for(Responses r:responses) {
                                // Prepare the data to be sent to HQ for thi
                                response.setWorkerId(r.getWorkerId());
                                response.setPlateNumber(r.getPlateNumber());
```

```
                                response.setLicenseNumber(r.getLicenseNumber
                                response.setModel(r.getModel());
                                response.setType(r.getType());
                                response.setVinNumber(r.getVinNumber());
                                response.setRequestId(r.getRequestId());
                                response.setRequestStatus(r.getRequestStatus
                                response.setApprovedBy(r.getApprovedBy());
                                response.setApprovedDate(r.getApprovedDate()

                                // Now send it

                                RestTemplate restTemplate = new RestTemplate
                        try {
                        Response result = restTemplate.postForObject(Synchro
                        } catch(Exception e) {
                                System.out.println("We are sorry something h
                                        throw e;
                                }

                                // If success, change the sync status
                                r.setSent(true);
                                responsesRepository.save(r);
                        }
                }

                return ResponseEntity.ok(null);
        }


}
```

Service layer descriptions

Here, we explain how we implemented the car request service. We should note that, CRUD operations have also been implemented for all the data models. Follow along the following steps:-

1. A bo user create a request -- Data access layer

```
@Data
public class RequestData {
   Long requestId;
   Long requestorId;
   Long branchId;
   String carModel;
   String vehiclePreffered;
   Date requestDate;
   String requestStatus;
   String approvedBy;
   Date approvedDate;
   String assignedCar;
   Boolean status;
```

```
}
```

Data access interface - the repository

```
package com.IAP.car_exchange.repository;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import com.IAP.car_exchange.Model.Request;

public interface RequestRepository extends JpaRepository<Request,Long>
   @Query(value = "SELECT * FROM requests r ORDER BY r.request_id DESC
   Request getLastRecord();

   @Query("SELECT r FROM Request r WHERE r.requestId=?1 AND r.assignedC
   Request getUnassignedCar(Long requestId);

   @Query("SELECT r FROM Request r WHERE r.status != true")
   List<Request> getAllUnyced();
}
```

2. The request is stored in the database --- service layer

A request controller

```
@RestController
public class RequestController {
   @Autowired
   Querries DataAccess;

   @PostMapping("request")
   @JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
    public @ResponseBody
    ResponseEntity<String> createRequest(@RequestBody RequestData dataH

      // Log this Request Locally
        Request request = DataAccess.addRequest(
            dataHolder.getRequestorId(),
            dataHolder.getBranchId(),
            dataHolder.getCarModel(),
            dataHolder.getVehiclePreffered(),
            //dataHolder.getRequestDate()
            new Date()
            );


        return ResponseEntity.ok(null);
    }
```

```
    @GetMapping("requests")
    public @ResponseBody Iterable<Request> getRequests(){
       return DataAccess.getAllRequests();
    }

     @GetMapping("request/{id}")
     public Request getRequest(@PathVariable Long id){
         return DataAccess.getRequestById(id);
     }

     @DeleteMapping("_request/{id}")
     public ResponseEntity<String> deleteRequest(@PathVariable("id") Lon
         DataAccess.deleteRequest(id);
         return ResponseEntity.ok("Removed");
     }

}
```

A request service method for adding a request only

```
    public Request addRequest(Long requestorId,Long branchId,String carM
                 String vehiclePreffered,Date requestDate) {
      User user = userRepository.findById(requestorId)
              .orElseThrow(() -> new IllegalArgumentException("Invalid r
      Office office = officeRepository.findById(branchId)
              .orElseThrow(() -> new IllegalArgumentException("Invalid b
         Request request = Request.builder()
                         .requestorId(user)
                         .branchId(branchId)
                         .carModel(carModel)
                         .vehiclePreffered(vehiclePreffered)
                         .requestDate(requestDate)
                         .status(false)
                         .build();
         requestRepository.save(request);
         return request;
 }
```

If successfully, the requests is transfered to the hq for further processes.

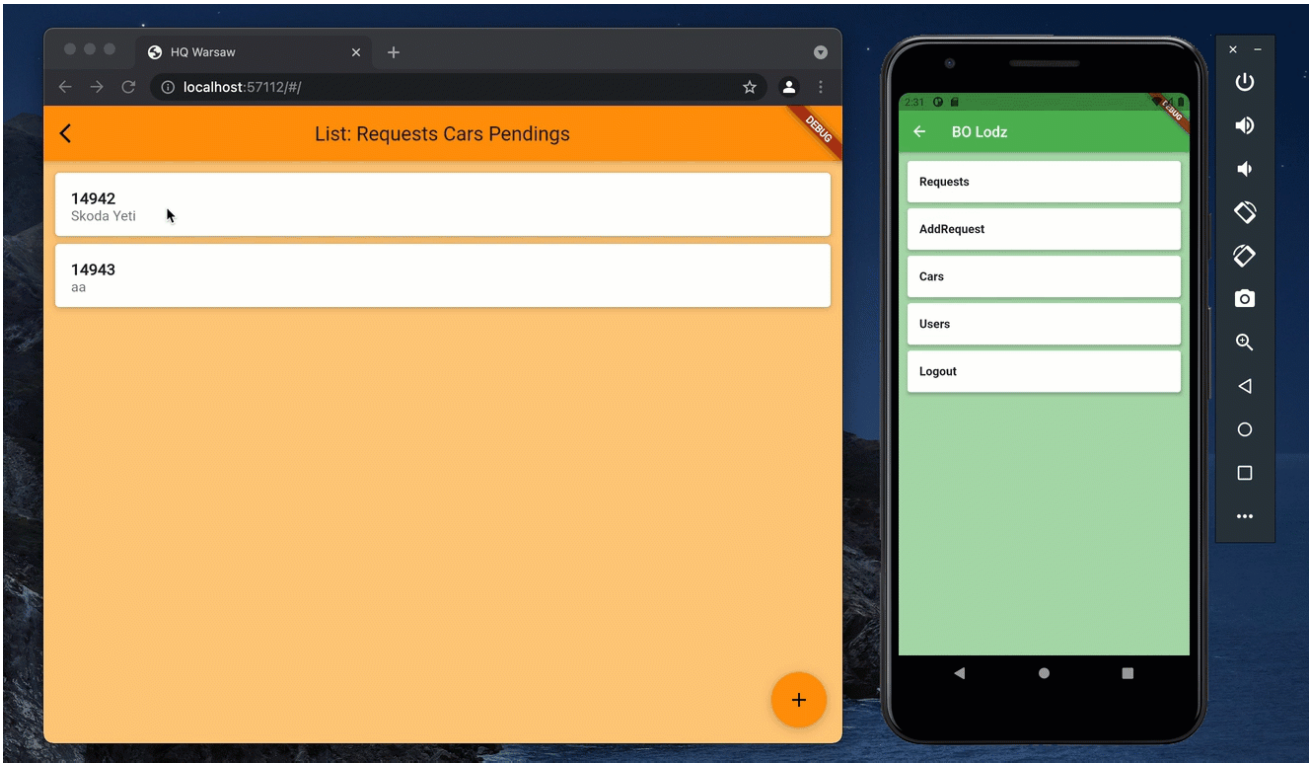
3. The hq manager will manage the requests, via the AssignCarController service

```
@RestController
public class AssignCarController {

 @Autowired
 Querries DataAccess;


 @GetMapping("pendingrequests")
 public @ResponseBody Iterable<Request> getPendingRequest(){
```

```
        return DataAccess.getPendingRequests();
    }

    @GetMapping("filter/{model}/{type}")
    public @ResponseBody Iterable<Car> getCarByFilter(@PathVariable String
        return DataAccess.getCarByModelType(model, type);
    }

    @PostMapping("assign/{requestId}")
        public @ResponseBody
        ResponseEntity<String> assign(@PathVariable Long requestId){
        Response response = DataAccess.assign(requestId);
        return ResponseEntity.ok(null);
        }
```

> At the end, the synchronization service will take of the rest, including sending back the response once the request has been attended.

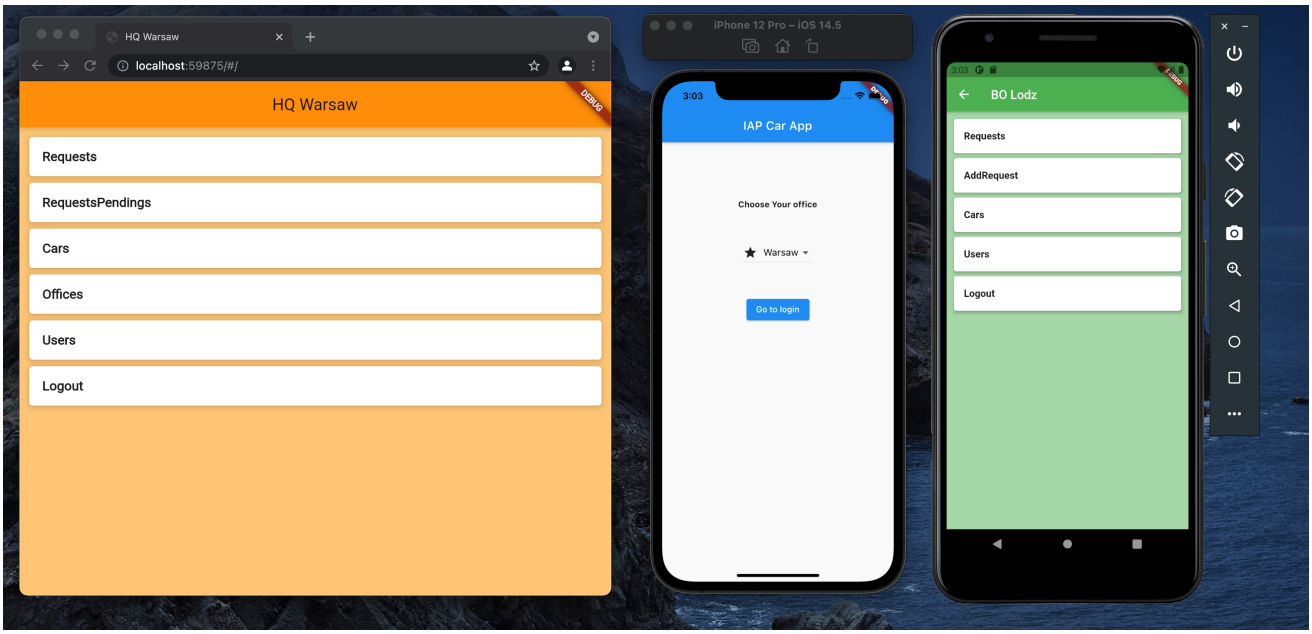
Application - Frontend



Application was prepared for multiplatform usages. During tests, we use 3 deployment types:

- iOS application
- Android application
- Web application

Application was prepared for different screen sizes and proportions (landscape, horizontal, browser window)



Application can be used as Headquater or Branch office, and can automaticaly adapt to type of office (HQ/BO). Main color theme and operations depend on office type.

Diffrent office types

At the beginning of application, user can select backend application, where he wants do activities. API client is prepared at the next screen (Page Login).

Moreover, at login login screen, we initialize the most important libraries.

```
import 'package:apiconsument/data/servers.dart';
import 'package:apiconsument/page_menu.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'data/api_service.dart';

class MaterialAppHome extends StatelessWidget {
  final Server choosenServer;
  MaterialAppHome(this.choosenServer);

  @override
  Widget build(BuildContext context) {
    return Provider(
      create: (_) => ApiService.create(choosenServer.address),
      dispose: (context, ApiService service) => service.client.dispose(),
      child: MaterialApp(
        theme: ThemeData.from(
          colorScheme: choosenServer.colorScheme,
        ),
        title: choosenServer.type + ' ' + choosenServer.city,
        home: PageMenu(
          choosenServer: choosenServer,
        ),
      ),
    );
```

```
  }
}
```

Important libraries:

- Provider - library to manage communication between screens
- Chopper (here called ApiService) - library for managing API service
- Material - library with Material Design style widgets

Dynamic build

Some views are used to build both application (for example PageListRequestsCars - but this page has a lot of differences between different office types. First - if this is HQ, list can show only pending request, and if it is HQ, user have extra button "Assign" on this page)

```
import 'dart:convert';
import 'data/api_service.dart';
import 'data/requestCar.dart';
import 'data/servers.dart';
import 'package:apiconsument/page_one_requestCar.dart';
import 'package:chopper/chopper.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

class PageListRequestsCars extends StatelessWidget {
  final Server choosenServer;
  final bool isPendings;

  const PageListRequestsCars({
    Key? key,
    required this.isPendings,
    required this.choosenServer,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(

      appBar: AppBar(
        title: Text('List: Requests Cars' + (isPendings ? ' Pendings' : ' ')
      ),
      body: _buildList(context),
      floatingActionButton: FloatingActionButton(
        onPressed: () {},
        child: const Icon(Icons.add),
      ),
    );
  }

  FutureBuilder<Response> _buildList(BuildContext context) {
    return FutureBuilder<Response>(
      future: isPendings
          ? Provider.of<ApiService>(context).requestsPendingsAll()
```

```
        : Provider.of<ApiService>(context).requestsAll(),
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.done) {
        final List requestsCarsCollection =
            json.decode(snapshot.data!.bodyString);
        final List<RequestCar> requestCar = [];
        for (var item in requestsCarsCollection) {
          requestCar.add(RequestCar.fromJson(item));
        }
        if (requestCar.length == 0) {
          if (isPendings) {
            return ListView(
              children: [
                Center(
                  child: Text(
                    "No pendings requests!",
                    style:
                        TextStyle(fontSize: 30, fontWeight: FontWeight.bol
                  ),
                ),
              ],
            );
          } else {
            return ListView(
              children: [
                Center(
                  child: Text(
                    "No requests!",
                    style:
                        TextStyle(fontSize: 30, fontWeight: FontWeight.bol
                  ),
                ),
              ],
            );
          }
        }
        return _buildRequestCarList(context, requestCar);
      } else {
        return Center(
          child: CircularProgressIndicator(),
        );
      }
    },
  );
}

ListView _buildRequestCarList(BuildContext context, List<RequestCar> posts
  return ListView.builder(
    itemCount: posts.length,
    padding: EdgeInsets.all(8),
    itemBuilder: (context, index) {
      return InkWell(
        child: posts[index].showAsCard(),
        onTap: () =>
            _navigateToRequestDetails(context, posts[index].requestId),
```

```
        );
      },
    );
  }

  _navigateToRequestDetails(BuildContext context, int requestId) {
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => PageOneRequesCar(
          choosenServer: choosenServer,
          requestId: requestId,
          isPending: isPendings,
        ),
      ),
    );
  }
}
```

API consumption

API is consumed using Chopper library. This library can generate class for http API using simple abstract class. "To define a client, use the @ ChopperApi annotation on an abstract class that extends the ChopperService class." More information: As we see, service address is declared during initialization, so we can use one API instance to prepare a lot of API services.

Chopper API class before generate

```
import 'package:chopper/chopper.dart';

part 'api_service.chopper.dart';

@ChopperApi()
abstract class ApiService extends ChopperService {
  @Get(path: '/user/{id}')
  Future<Response> userById(@Path('id') int id);

  @Post(path: '/assign/{requestCarId}')
  Future<Response> assignRequestCar(@Path('requestCarId') int requestCarId);

  @Get(path: '/users')
  Future<Response> usersAll();

  @Get(path: '/office/{id}')
  Future<Response> officeById(@Path('id') int id);

  @Get(path: '/offices')
  Future<Response> officeAll();

  @Get(path: '/requests')
  Future<Response> requestsAll();

  @Post(path: '/request')
```

```
  Future<Response> postNewRequestCar(@Body() Map<String, dynamic> myRequest)

  @Get(path: '/pendingrequests')
  Future<Response> requestsPendingsAll();

  @Get(path: '/request/{id}')
  Future<Response> requestById(@Path('id') int id);

  @Get(path: '/car/{plateNumber}')
  Future<Response> carByPlateNumber(@Path('plateNumber') String plateNumber)

  @Delete(path: '/_car/{plateNumber}')
  Future<Response> deleteCarByPlateNumber(
      @Path('plateNumber') String plateNumber);

  @Delete(path: '/_user/{userId}')
  Future<Response> deleteUserById(@Path('userId') String userId);

  @Post(path: '/user')
  Future<Response> postNewUser(@Body() Map<String, dynamic> myUser);

  @Get(path: '/cars')
  Future<Response> carAll();

  @Get(path: '/posts')
  Future<Response> getPosts();


  @Get(path: '/posts/{id}')
  Future<Response> getPost(@Path('id') int id);

  @Post()
  Future<Response> postPost(
    @Body() Map<String, dynamic> body,
  );

  static ApiService create(String address) {
    final client = ChopperClient(
      baseUrl: address,
      services: [
        _$ApiService(),
      ],
      interceptors: [HttpLoggingInterceptor(), CurlInterceptor()],
      converter: JsonConverter(),
    );

    return _$ApiService(client);
  }
}
```

To generate file, in console user must type `flutter packages pub run build_runner watch` - this command observed abstract API class and generate new class with endpoints at every file saved.

Chopper API class after generate

```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'api_service.dart';

// **************************************************************************
// ChopperGenerator
// **************************************************************************

// ignore_for_file: always_put_control_body_on_new_line, always_specify_type
class _$ApiService extends ApiService {
  _$ApiService([ChopperClient? client]) {
    if (client == null) return;
    this.client = client;
  }

  @override
  final definitionType = ApiService;

  @override
  Future<Response<dynamic>> userById(int id) {
    final $url = '/user/$id';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> assignRequestCar(int requestCarId) {
    final $url = '/assign/$requestCarId';
    final $request = Request('POST', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> usersAll() {
    final $url = '/users';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> officeById(int id) {
    final $url = '/office/$id';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> officeAll() {
    final $url = '/offices';
    final $request = Request('GET', $url, client.baseUrl);

    return client.send<dynamic, dynamic>($request);
```

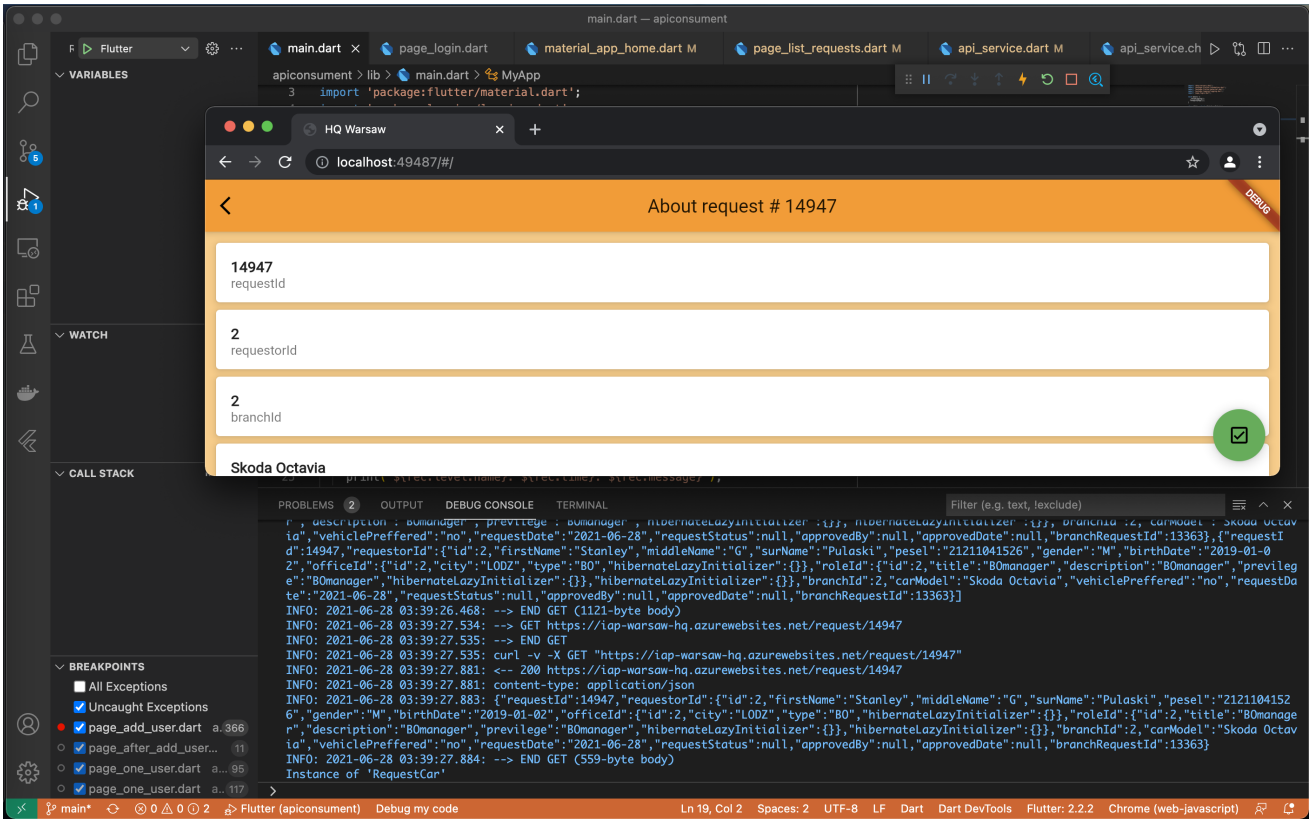
```
}

@override
Future<Response<dynamic>> requestsAll() {
  final $url = '/requests';
  final $request = Request('GET', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> postNewRequestCar(Map<String, dynamic> myRequest
  final $url = '/request';
  final $body = myRequest;
  final $request = Request('POST', $url, client.baseUrl, body: $body);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> requestsPendingsAll() {
  final $url = '/pendingrequests';
  final $request = Request('GET', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> requestById(int id) {
  final $url = '/request/$id';
  final $request = Request('GET', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> carByPlateNumber(String plateNumber) {
  final $url = '/car/$plateNumber';
  final $request = Request('GET', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> deleteCarByPlateNumber(String plateNumber) {
  final $url = '/_car/$plateNumber';
  final $request = Request('DELETE', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> deleteUserById(String userId) {
  final $url = '/_user/$userId';
  final $request = Request('DELETE', $url, client.baseUrl);
  return client.send<dynamic, dynamic>($request);
}

@override
Future<Response<dynamic>> postNewUser(Map<String, dynamic> myUser) {
  final $url = '/user';
```

```
    final $body = myUser;
    final $request = Request('POST', $url, client.baseUrl, body: $body);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> carAll() {
    final $url = '/cars';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> getPosts() {
    final $url = '/posts';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> getPost(int id) {
    final $url = '/posts/$id';
    final $request = Request('GET', $url, client.baseUrl);
    return client.send<dynamic, dynamic>($request);
  }

  @override
  Future<Response<dynamic>> postPost(Map<String, dynamic> body) {
    final $url = '';
    final $body = body;
    final $request = Request('POST', $url, client.baseUrl, body: $body);
    return client.send<dynamic, dynamic>($request);
  }
}
```

API debugging

Library can show every used request as cURL request, it can help to debug or error reproduction. It is easy to compare request from Flutter with request from Postman (both tools have option export

request to cURL)



New page, new request

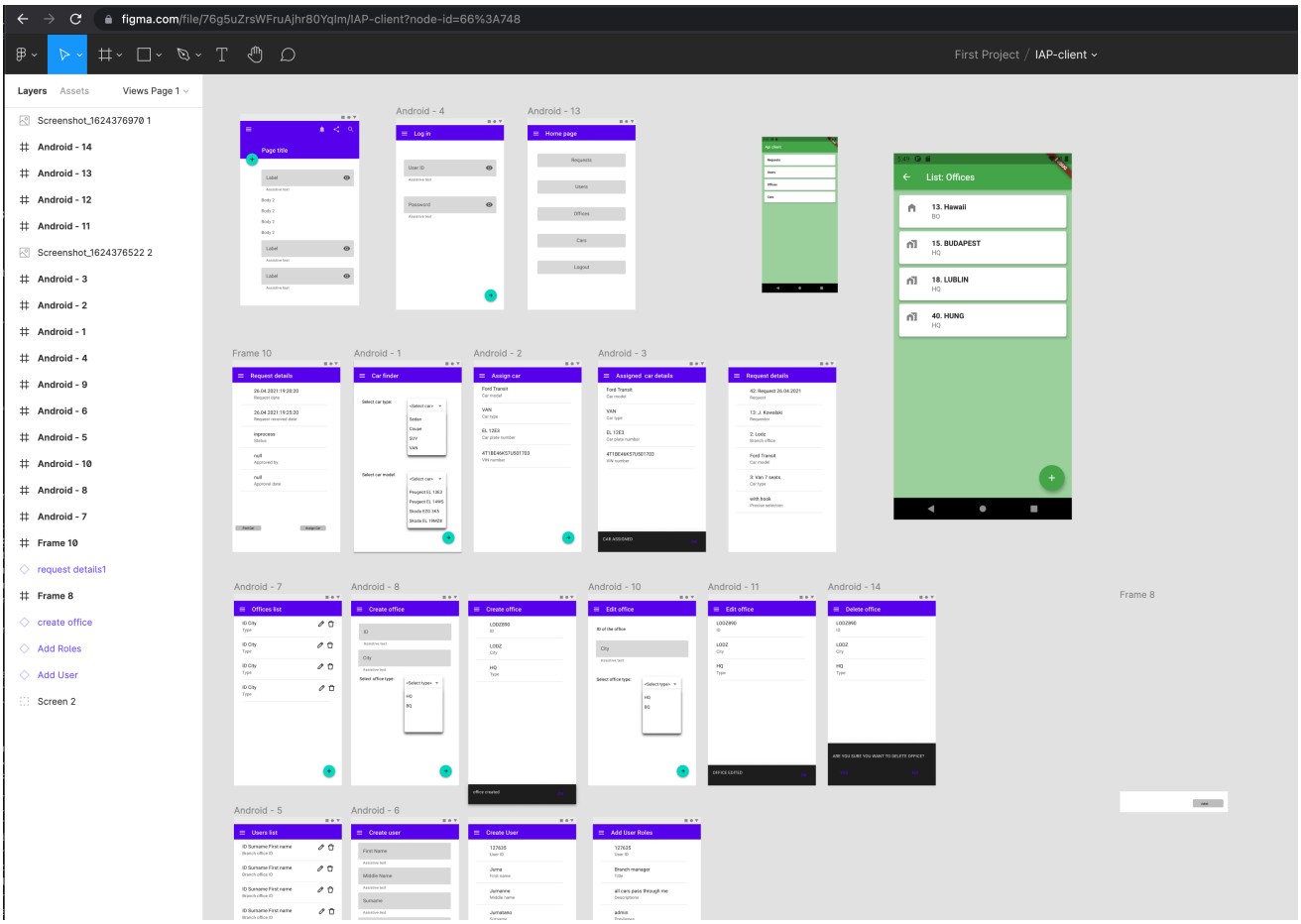
Requests are send when user open new page. When view is not ready, builder show Circular indicator. Application wait for respond or timeout.

```
import 'data/api_service.dart';
import 'data/servers.dart';
import 'data/user.dart';
import 'package:chopper/chopper.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'page_add_offices.dart';
import 'page_menu.dart';
import 'page_list_users.dart';

class PageAfterAddUser extends StatelessWidget {
  final User user;
  final Server choosenServer;

  const PageAfterAddUser(
      {Key? key, required this.user, required this.choosenServer})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('User added: ' + user.id.toString()),
      ),
      body: _buildList(context),
```

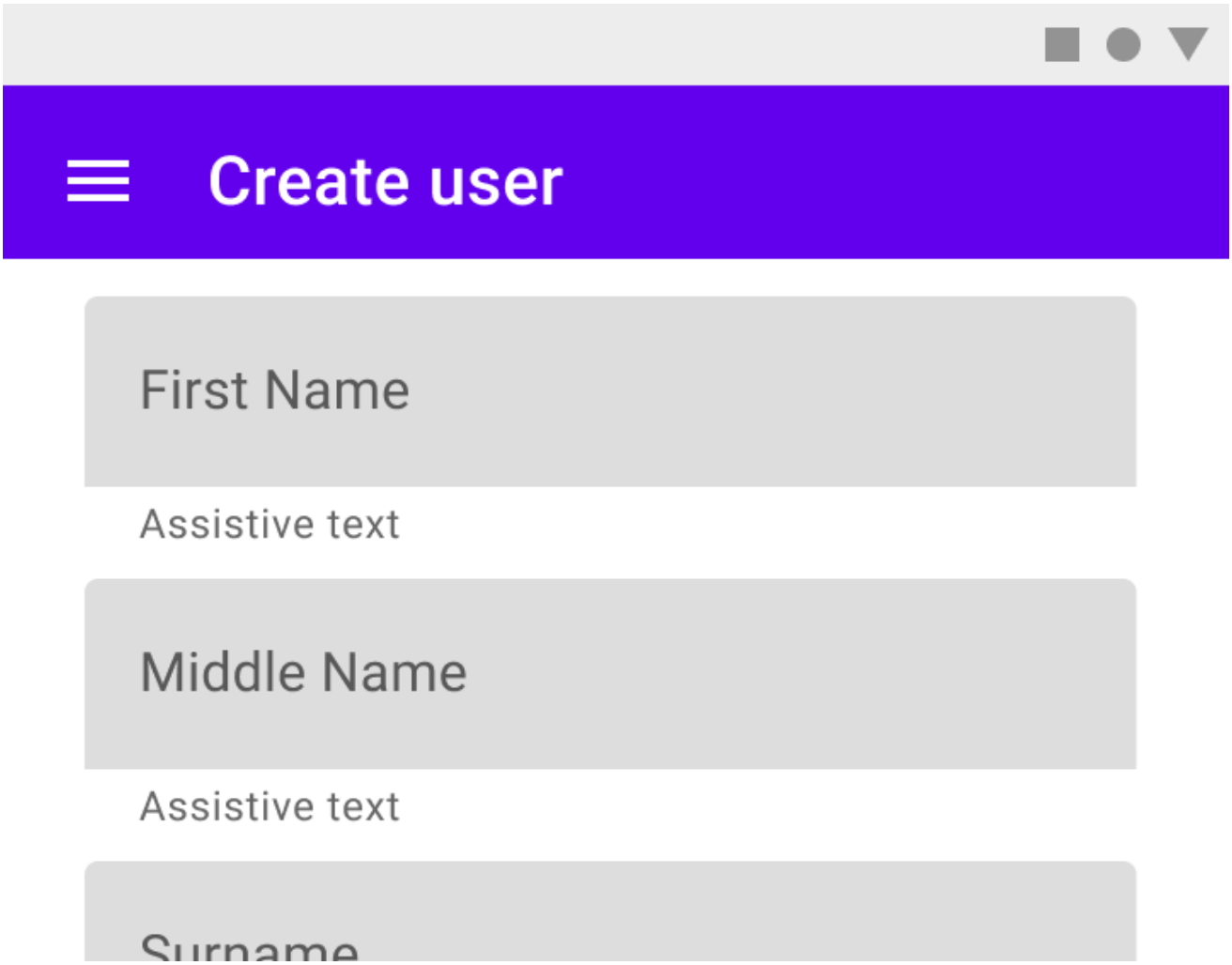
```
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.backpack_outlined),
        onPressed: () => _navigateToAddOffice(context),
      ),
    );
  }

  FutureBuilder<Response> _buildList(BuildContext context) {
    return FutureBuilder<Response>(
      future: Provider.of<ApiService>(context).postNewUser(user.toJson()),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done) {
          final String myRespond = snapshot.data!.bodyString;
          return _buildPosts(context, myRespond);
        } else {
          return Center(
            child: CircularProgressIndicator(),
          );
        }
      },
    );
  }


Widget _buildPosts(BuildContext context, String myRespond) {
...

void _navigateToAddOffice(BuildContext context) {
...
```

Views design - project

First project was prepared in Figma application - tool to UI design.



This table shows samples of views from figma and their Flutter implementations (Violet - figma design, orange - Flutter implementation)



Surname

Assistive text

PESEL

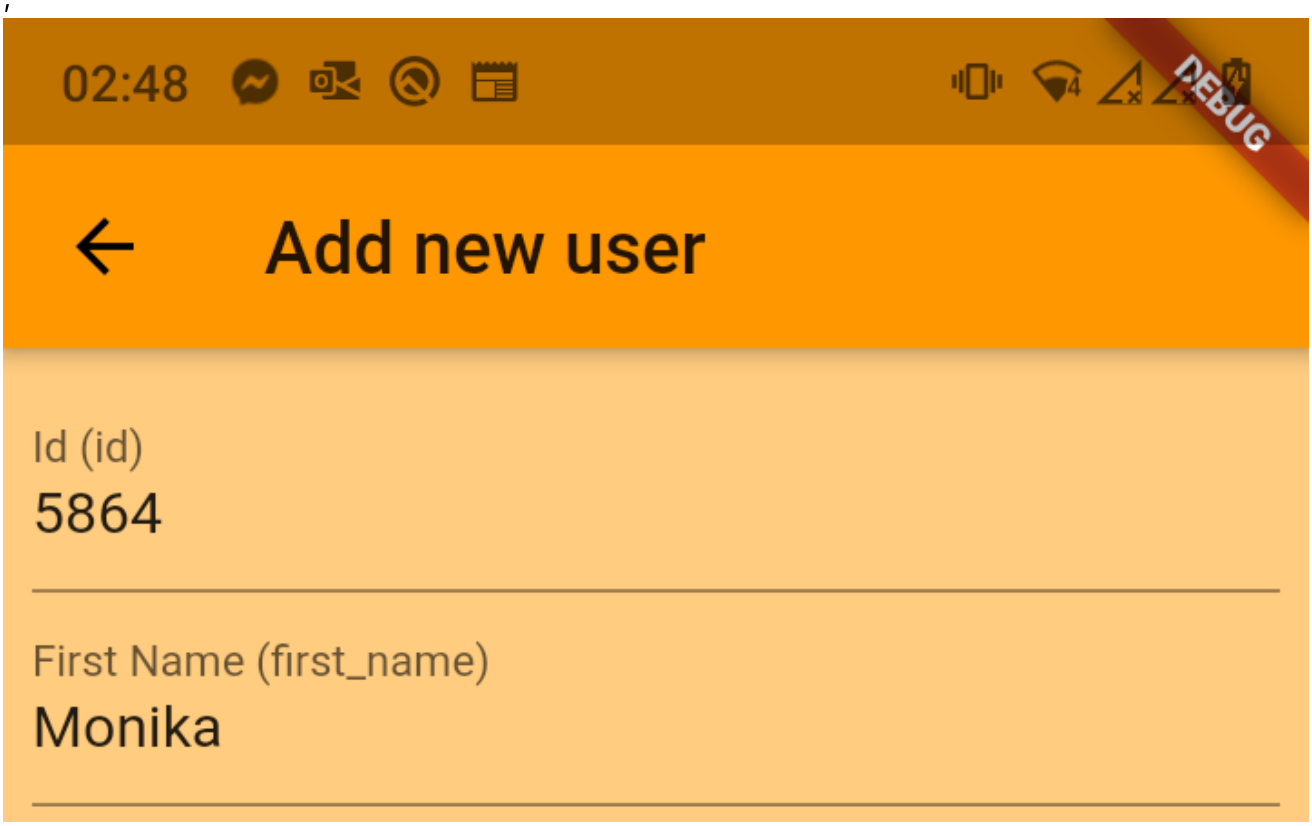
Assistive text

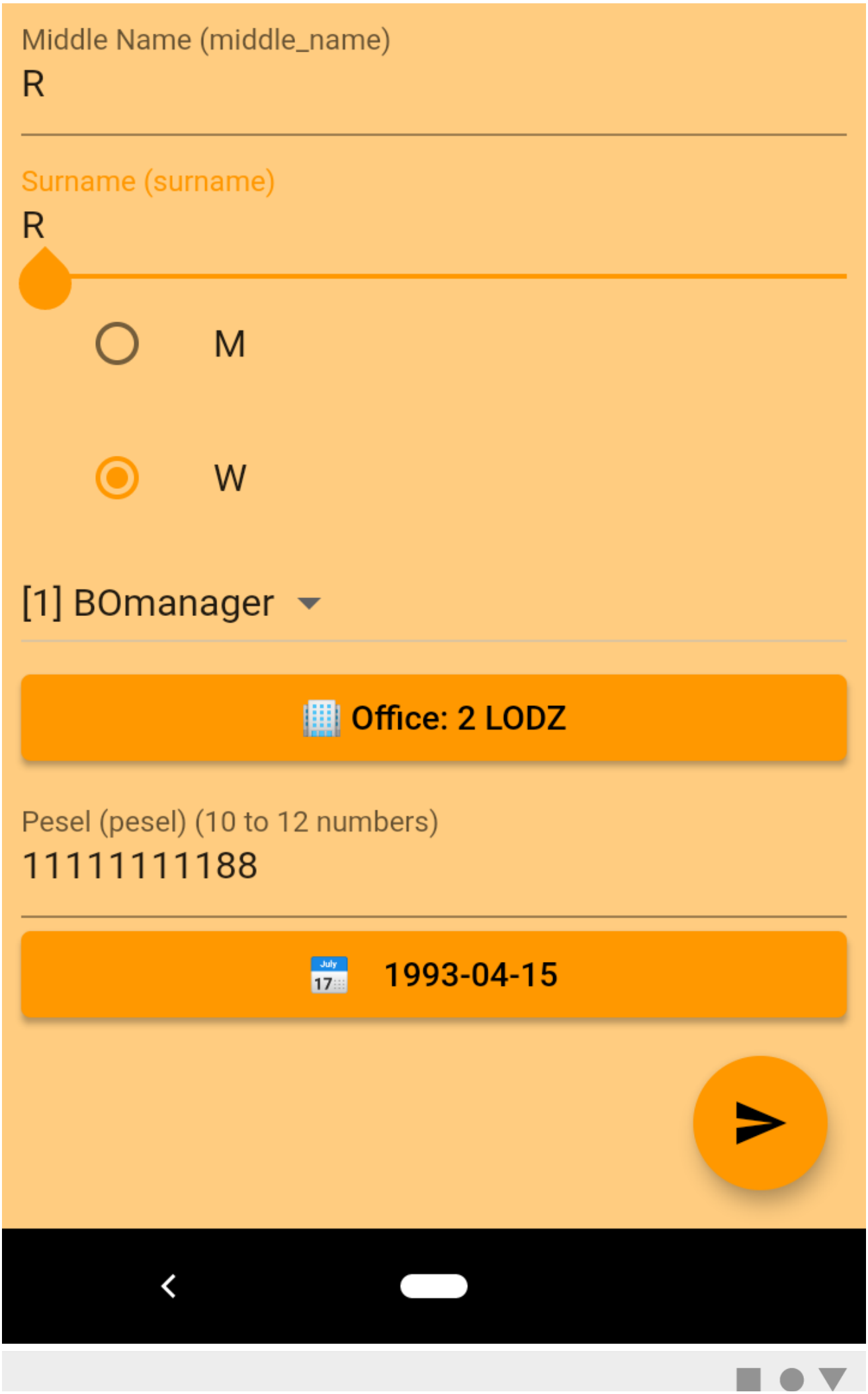
Gender

Assistive text

Birth date

Assistive text




Middle Name (middle_name)
R
Surname (surname)
R
M
W
[1] BOmanager
Office: 2 LODZ
Pesel (pesel) (10 to 12 numbers)
11111111188
1993-04-15

Requests list

Request 26.04.2021

Status: in progress

Request 22.04.2021

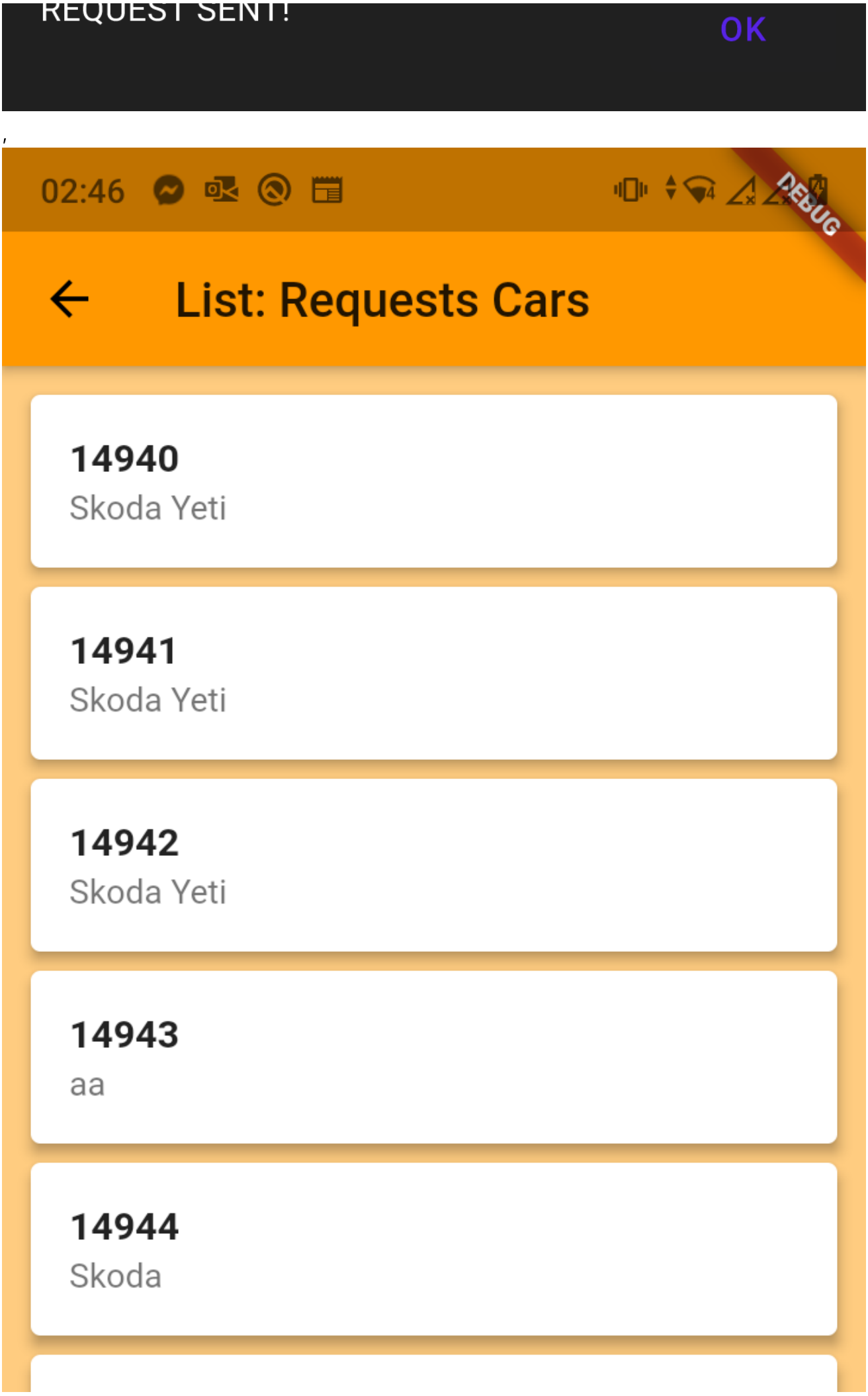
Status: approved

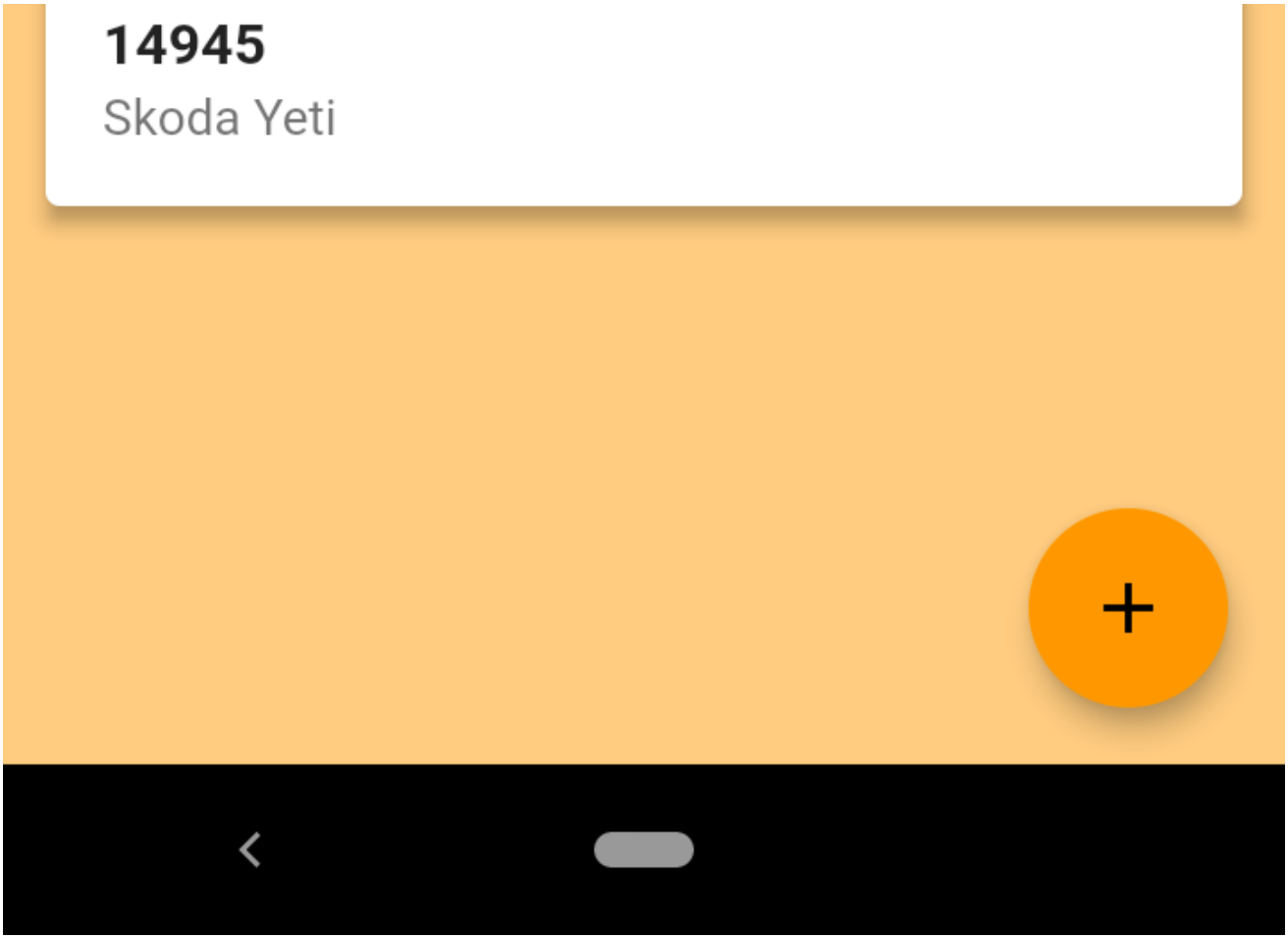
Request 22.04.2021

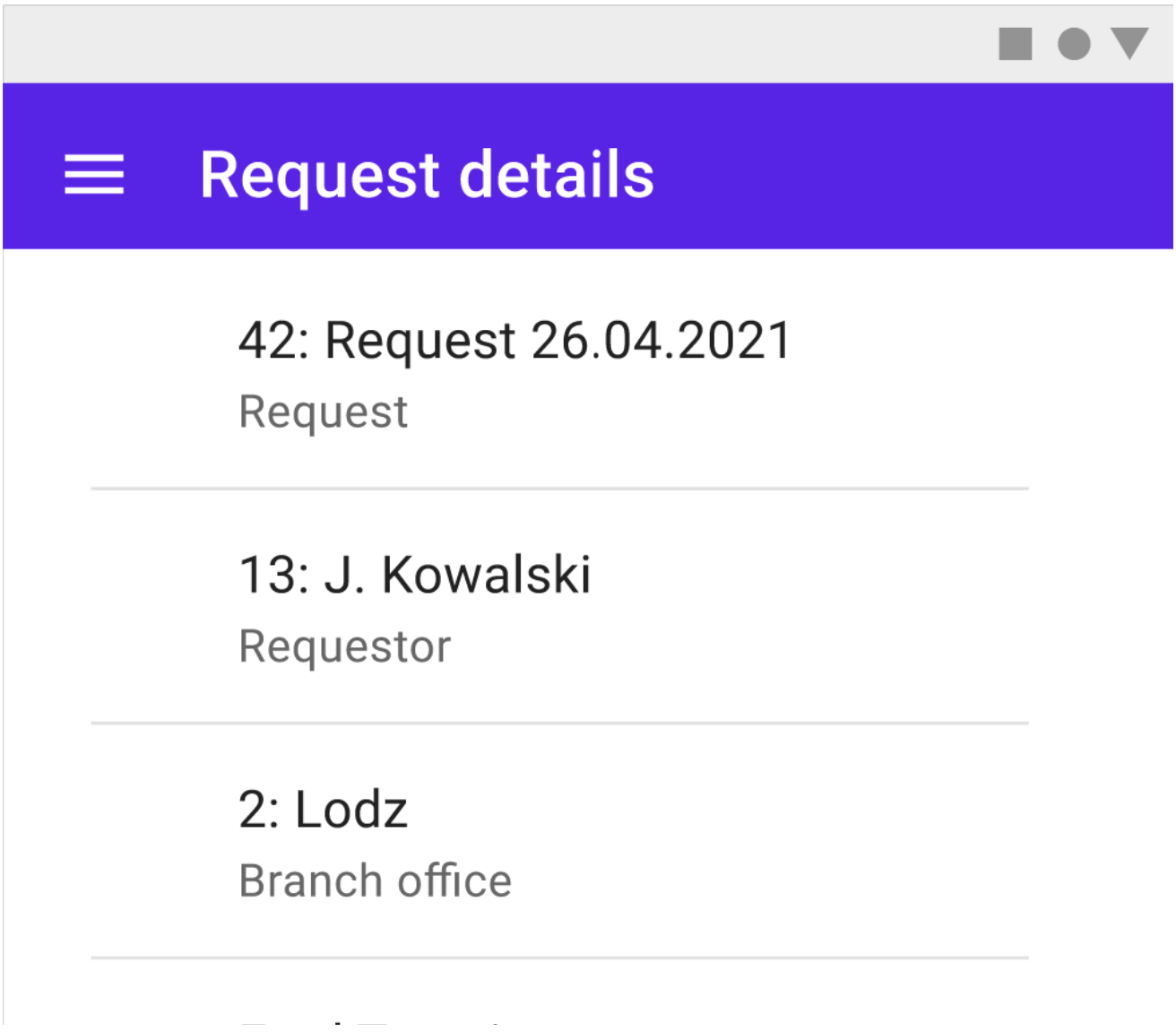
Status: rejected

Request 25.04.2021

Status: assigned


REQUEST SENT!
OK
02:46
DEBUG
List: Requests Cars
14940
Skoda Yeti
14941
Skoda Yeti
14942
Skoda Yeti
14943
aa
14944
Skoda


14945
Skoda Yeti


Request details
42: Request 26.04.2021
Request
13: J. Kowalski
Requestor
2: Lodz
Branch office

Ford Transit

Car model

3: Van 7 seats

Car type

with hook

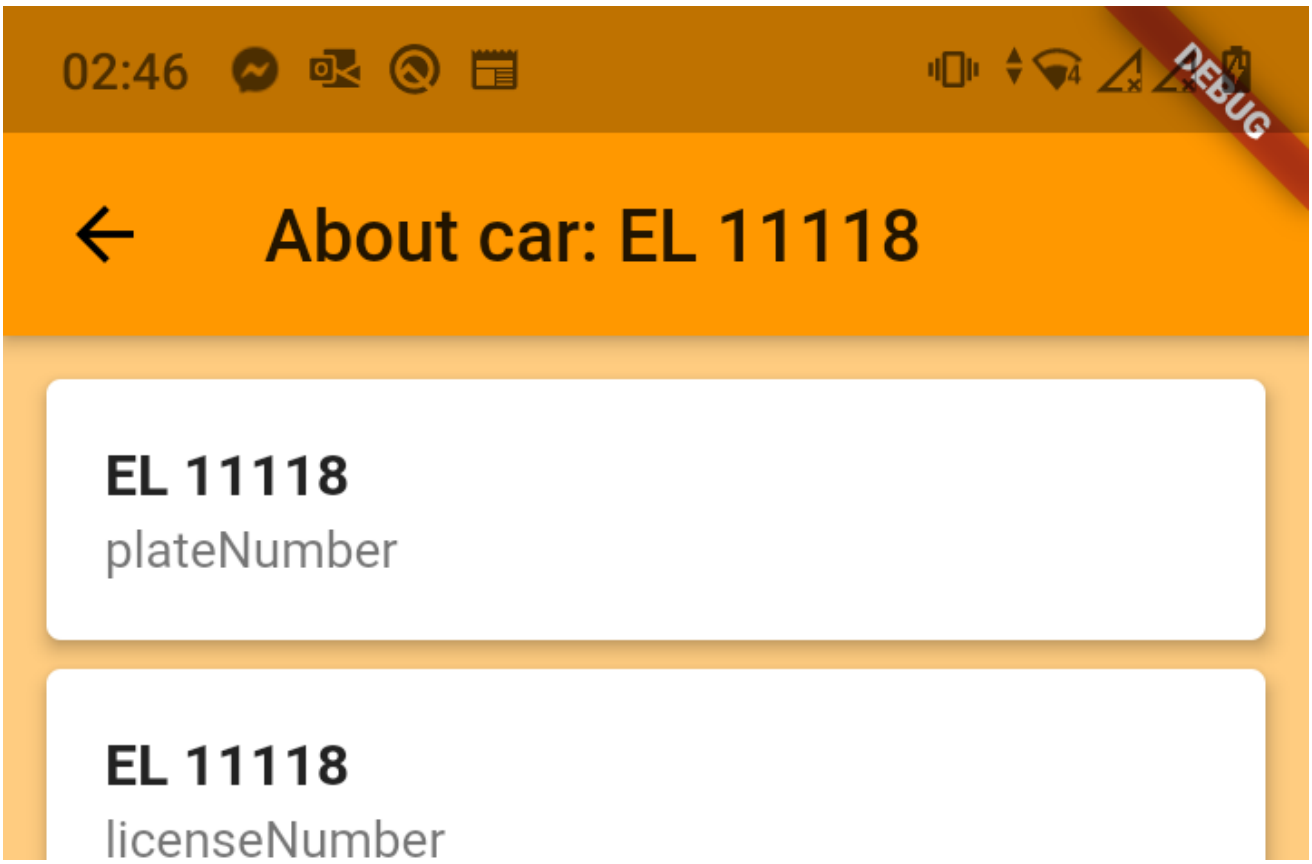
Precise selection

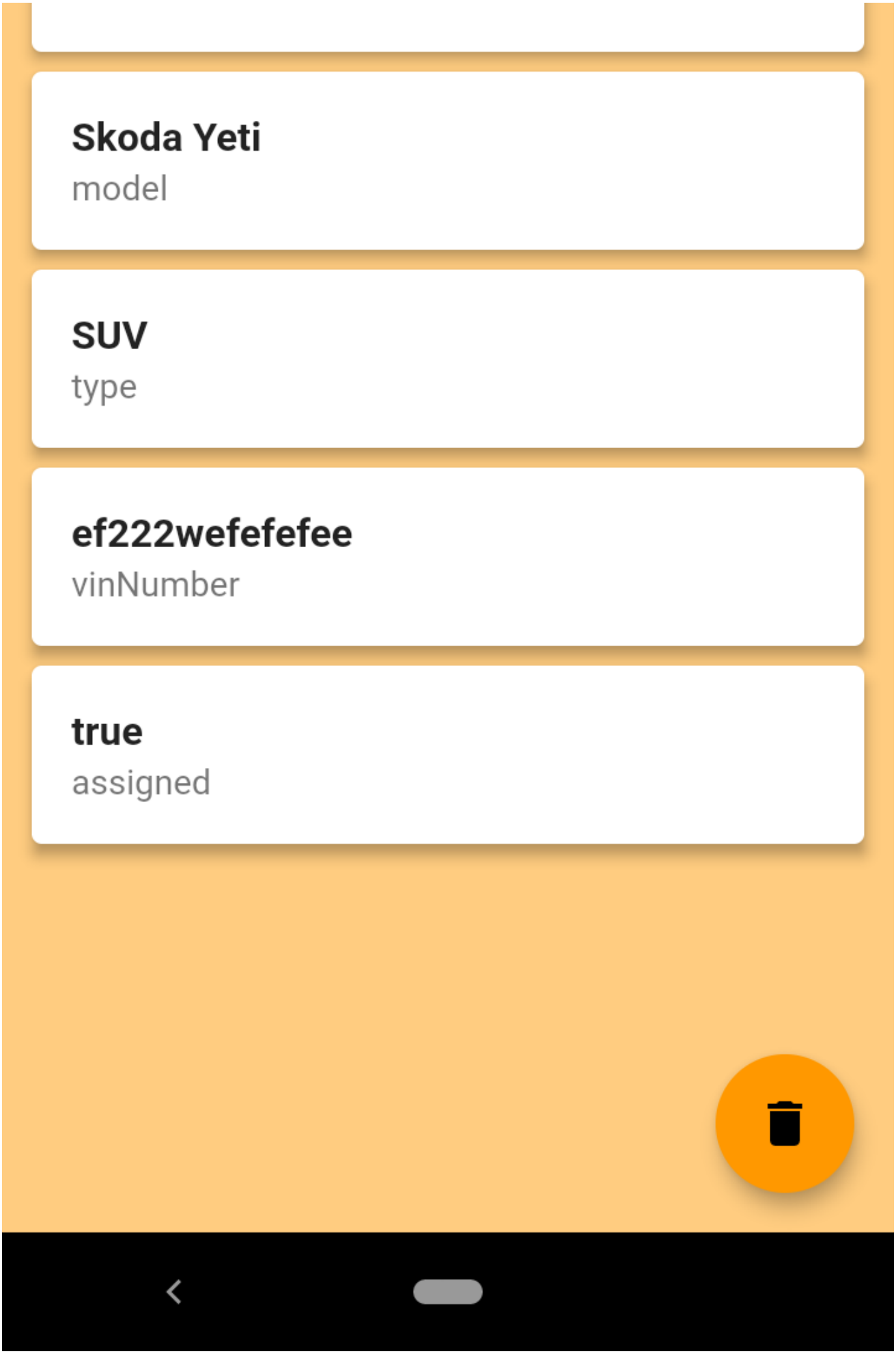
1

Quantity

15:32:11 26.04.2021

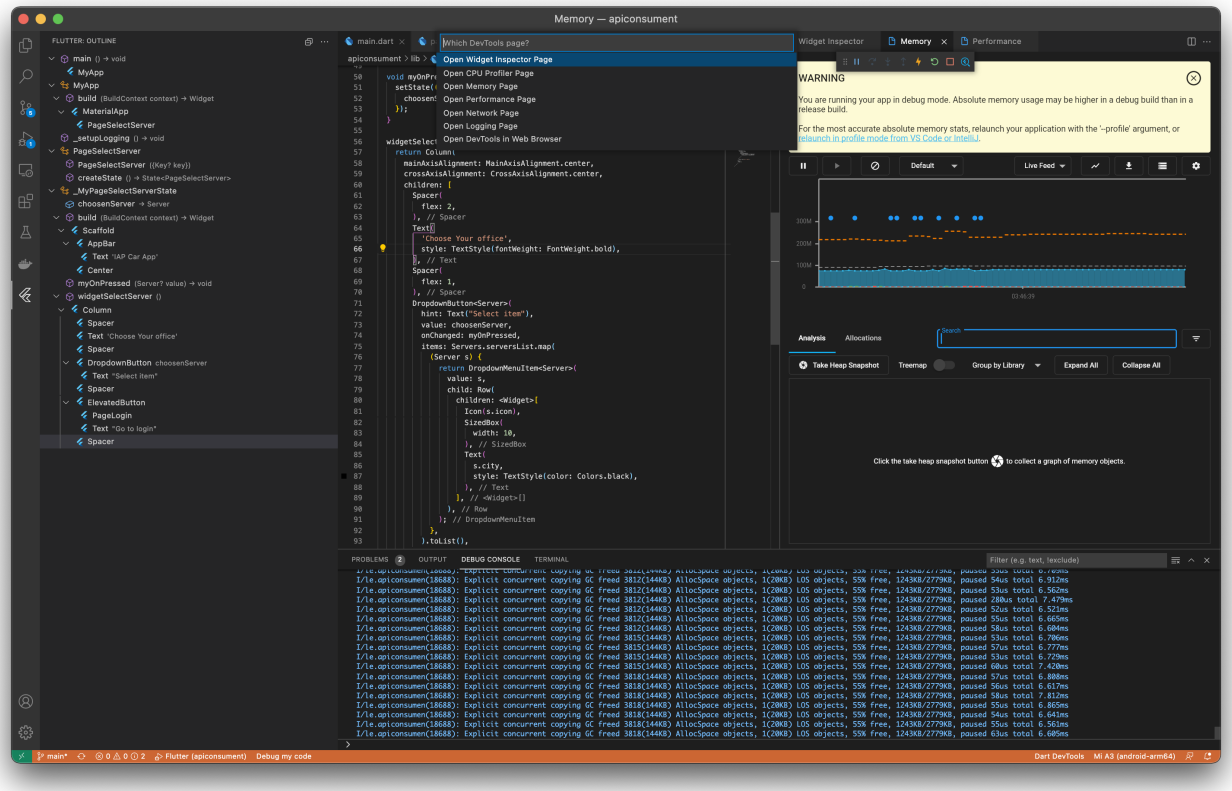
Request date





Flutter, Dart, Debug

Dart have a lot of tools (called DartDevTools) which can support develop process.



RESOURCES for Report 3

1. REST Template for Data Exchange
2. Task Schedular interfaces for Data Synchronization
3. Setting Up Swagger 2 with a Spring REST API
4. How does Chopper work?

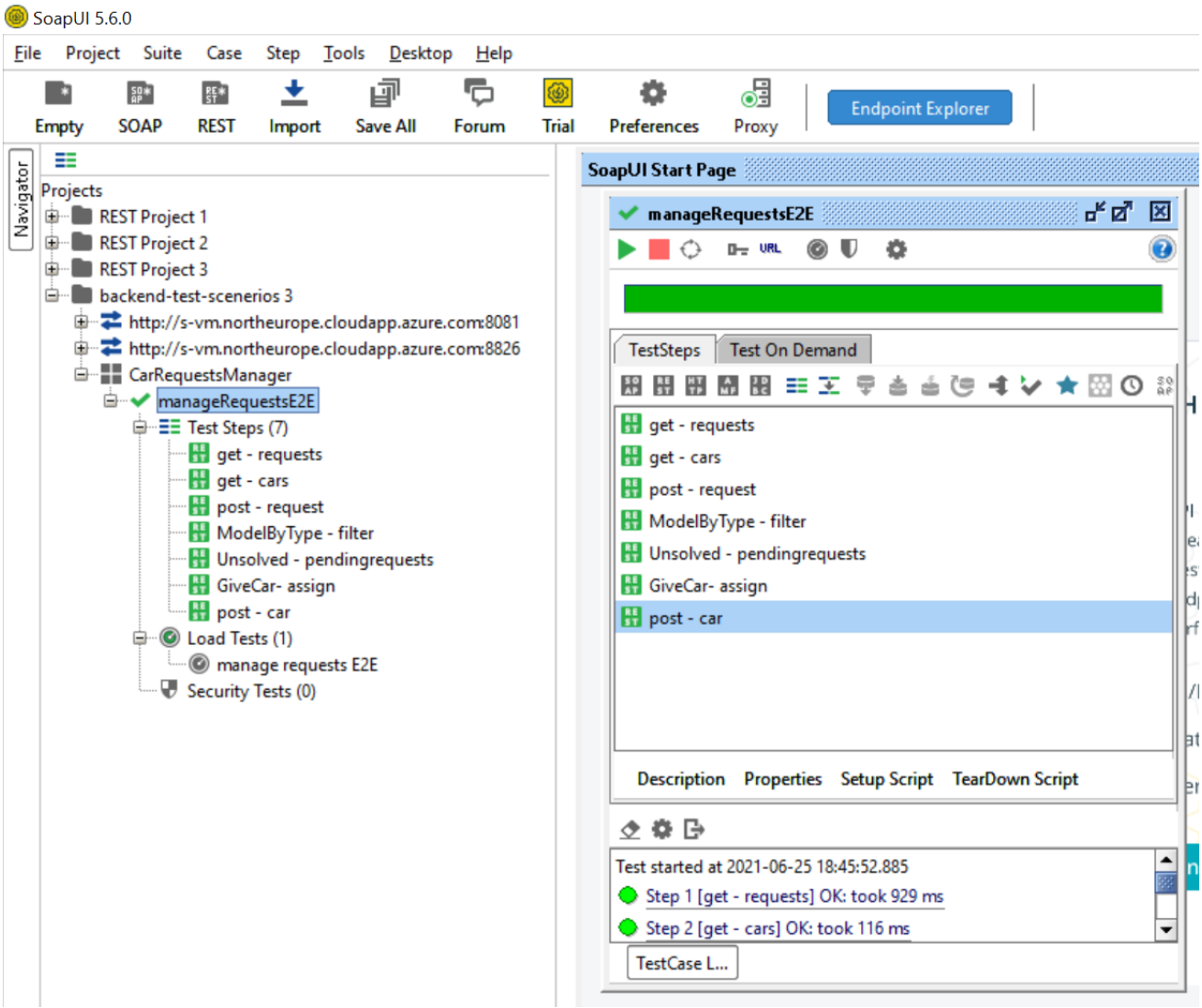
Report 4 - Performance analysis, summary and conclusions of the project

a). Introduction

In this part, we are trying to simulate the business logic (as one testsuite having seven steps as listed below) in an end to end manner where actors will be :-

1. Creating car requests,
2. Browsing the submitted requests,
3. Browsing the unassigned requests,
4. Viewing a list of available car stock,
5. Processing/assigning the cars requested,
6. Adding cars to the stock, as new cars have just arrived to our warehouse, and
7. Filtering the requested car by model and car type.

The figure below shows the testsuite setup in soapUI



- Performance testing tool

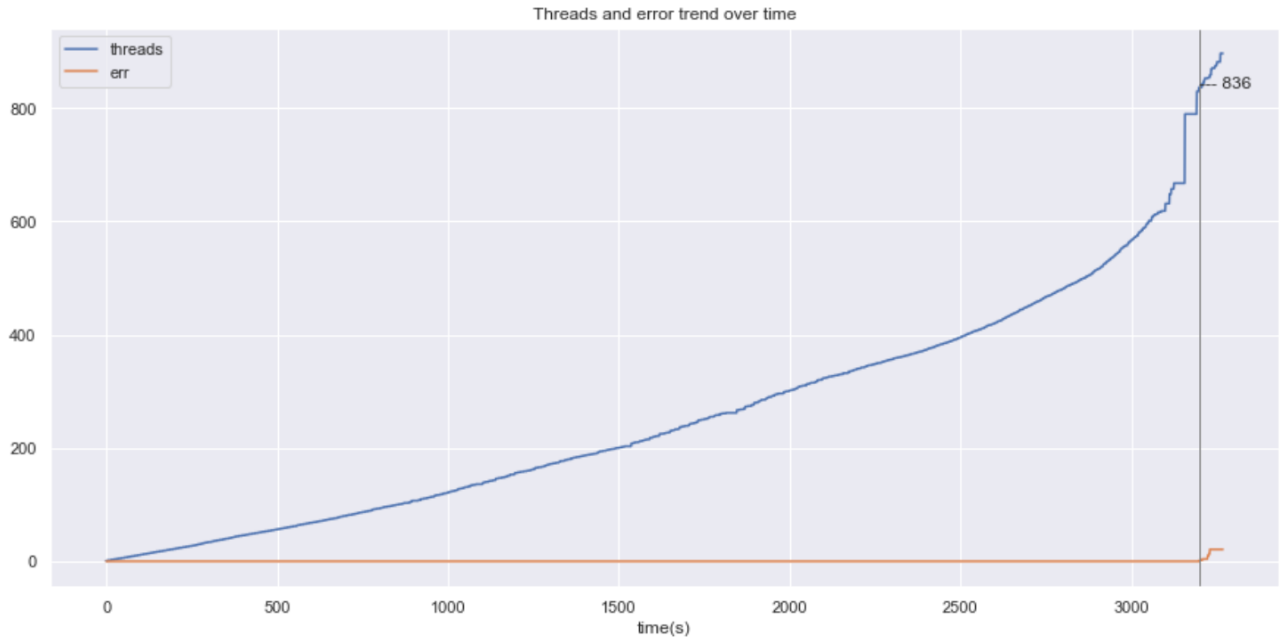
  We use SoapUI `5.6.0` (open-source) to perform set and run the performance test for our REST API. The open-source version of SoapUI can handle a limited number of concurrent threads (i.e 200) at its basic configuration, therefore we altered these basic configurations inorder to support up to 2000 threads. However, increasing the number of threads triggers another key problem, the memory limitations which leads to not able to run a load test continuously over a long period of time. Yet, there is a workaround for improving memory usage as described here and here. Improving memory usage by changing the basic settings is limited on the hardware capabability of the testing machine, for our case the testing computer had a 8GHz RAM and Core i5-4258U CPU @2.4GHz 2.4GHz. Due to those limitations, the simulation was performed for approximately 60 minutes.

- Testing strategies

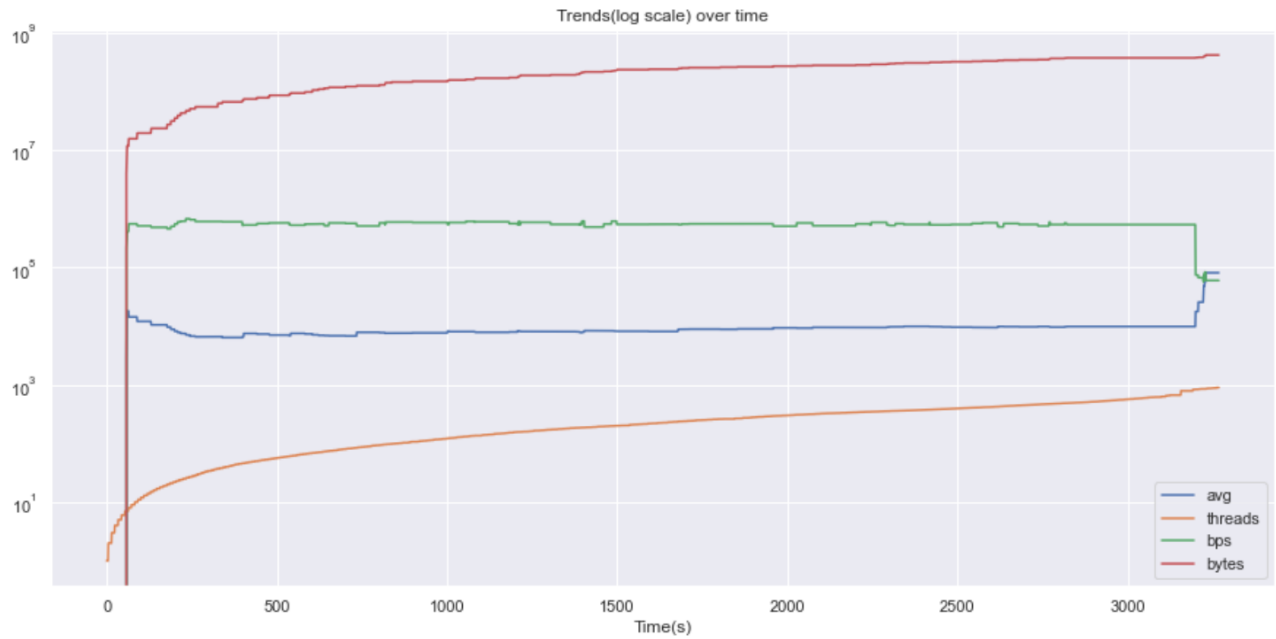
  - The Thread strategy - We linearly change the number of threads/virtual users from one level to another over the run of load test. We aim at establishing the threads baseline above which the application will start flooding with errors. We then, determine the average data volume which our application can handle just before errors occur, and finally establish the average response time.

b). Performance analysis

The test was set and run as described in part (a), and logs were collected for analysis. The figure below shows the trend between threads and errors. It can be observed that after **836** threads, the total error just started to increase. The gray line depicts the bound above which the total errors are above zero.



- We also observed on how much volume of data can be processed over the test period.It can be stated that as the number of threads were increasing so do the data volume (bytes) which were being sent to the server. Just before the errors a total volume of **360 GB** were transacted at a rate of 72 mbps. The plot below shows the data volume (bytes), response time (avg in millisecond), speed (bps), and threads over the test duration. Note that, the vertical scale is logarithmic.



- The average response time in millisecond (avg) were gradually increasing as the number of threads were increasing. Just before the errors, the average response time was around **9.6 seconds**.

c). Summary

Below table depicts the overall statistical summary of the performance testing over the test duration. It can be seen that, our application had a total of 21 errors, about 400 GB data volume were transacted.

	avg	bytes	bps	err
count	3269.000000	3.269000e+03	3269.000000	3269.000000
mean	9655.900459	2.234492e+08	533603.888345	0.297033
std	8417.725502	1.123400e+08	105790.969231	2.349417
min	0.000000	0.000000e+00	0.000000	0.000000
25%	7852.870000	1.298119e+08	538938.000000	0.000000
50%	8910.750000	2.404053e+08	546343.000000	0.000000
75%	9749.220000	3.156464e+08	579134.000000	0.000000
max	81503.400000	4.205256e+08	678458.000000	21.000000

d). Conclusion

This marks the end of the project, but before the dead end each team member would like to highlight some thoughts on the challenges, skills gained, etc during the course of project realization.

- First, I would like to thank God for keeping me safe during the semester. Second, I extend my heartfelt gratitude to our lecturer Mr. Wiktor Wandachowicz for his tireless guidance and support during the project realization. Technically, It was my first time to realize a project in an end-to-end manner from the designing and testing stage. The following were the technical skills gained during the semester for this course. First, developing a web API using the JAVA spring boot framework. It was a headache at first since I did not program with java before, but thanks to team members for their support and thoughts on some resource links. Second, the understanding of the basic architecture of the REST API, some new keywords like endpoints, resources, different layers from data access up to presentation layer, and how to differentiate them. Third, working with JPA and SQL interchangeably. Fourth, realizing synchronization between two APIs and observing that the setup works as it is supposed to work was a joyous moment. Fifthly, it was my first time working with performance testing tools like SoapUI, and I know some ins and outs of this tool, at least the open-source one, from setting up and running the test using various strategies, environment properties expansion, adjusting memory settings in a bin folder in case you want to test the heavy load, and many more. Sixth, docker basic commands knowledge, for example, commands for building and running container images. I also gained knowledge on online tools for fast-creating database table models. Working with the support team was also a great advantage of this project as we always helped each other. THANKS, TEAM 💯 -- ~**GODFREY MGHASE**
- Participation in the project gave me an opportunity to amplify my interests and was an outstanding chance to expand my knowledge of developing web applications. Among the skills

and knowledge acquired during the project is the understanding of the architecture of REST API. My knowledge about software engineering and User Interface designing was significantly extended. During project realisation I had used Figma application in order to design views, which was a new experience for me. Due to Figma's useful prototyping features, I plan to use this tool in the future as well. Furthermore, I expanded my knowledge of software engineering such as creating diagrams (deployment diagram, activity diagram) with usage of online tools and describing use cases. Moreover, I have improved my organisational, interpersonal and managerial skills. Working in the team, was a beneficial experience and allowed us to exchange our knowledge. All team members were supportive, reliable and engaged in the work, therefore it has been a pleasure working with all of you. – ~**MONIKA ROSA**

- First of all, Thank to Mr. Wiktor Wandachowicz for his lectures and laboratory meetings - this lessons help us solve some problems with project. Second, thank to IAP group8 team for cooperation and development process. Thanks to this project, I develop my soft and hard skills. Below I write some insights:
    1. Swagger is amazing tool, and this is more than "swagger-ui.html" what I know before project. Swagger hub can help manage Our API for different applications and different version of API. Using Swagger we can design API in YAML BEFORE development stage. Swagger can generate not only client code, moreover can generate server side code.
    2. Befor project I have some CI/CD projects with Azure DevOps, Jenkins and other tools, but now I know that it is very easy to set trigger on ACR and deploy image from ACR stright to WebApp Container, it is simple and powerfull tool, amazing for test a lot of instances, and not necessary duplicate whole pipelines, it just webhook. Every `docker push` is new deployment, without any advanced CI/CD tools.
    3. Figma is amazing tool for prototyping. Here is a lot of plugins to generate code (I try `FigmaToFlutter` and `Figma to Code` plugins). But plugins not every time working correct, so at this project we prepare views manually based on Figma project, not generate them automatically. In the future projects we must take care about it, select the best plugin and generate views directly from Figma.
    4. Flutter 2.x is now `null-safety`, thank to this code is better, but development process last longer that I expected after working with Flutter 1.x.
    5. When frontend application do not need acces to hardware (Camera, Bluetooth, Local Storage, etc.), it is very simple to develop multiplatform code. Our project was tested on `Android`, `iOS`, and `Web`. One code for a lot of platforms, and only one, little problem - for Web project we need only add CORS on server side. So Flutter is for me the best framework for rapid development application for API consumption.
    6. Chopper is very interesting library and easy to use.
    7. First time we test web application during load test, and we know that Our server can working with about `800` users at the same time per Office instance. It is great score, because we acheve it on ASP server 3.5GB RAM 2vCPU where we had a lot of other applications hosted on same ASP!
    8. It was amazing challenge and interesting course, thank You one more time, ~**STANISŁAW PUŁAWSKI**
- Spring boot is nice framework for API development and backend applications as it has made easier to fast create a web application. Using swagger-ui.html endpoint we can rapid test backend service inside browser - we do not need postman, curl or SoapUI for it. Taking part in this project was a great opportunity for me to expand not only technical abilities but also my multi-tasking skills, managerial as well as organizational skills. ~**WIKTOR MURASZKO**

e) RESOURCES for report 4

1. Concurrent threads limitation in SoapUI
2. Memory management in SoapUI part 1
3. Memory management in SoapUI part 2
4. Simulating different types of load
5. Azure Container Register webhooks
6. Swagger editor online
7. FigmaToFlutter plugin