Server Documentation

Entities: (Tables)

User {

String email;, String name; String userName; String phoneNumber; int notificationFlag; int sysId; String hashPassword; Account account; String type; }

User class has 3 children which extend it:

-Admin, UserAccount, Support

Device{

int id; long imei; String name; int accountId; DeviceType type; Timestamp lastUpdate; String logitude; String latitude; boolean isRegistered; boolean isFaulty; Timestamp faultyTime; String phoneNumber; String password;}

This class has all the data that the device has including its phonenumber, password and when last time it sent updates.

DeviceData{

long ID; long imei; int gpsType; char gpsValid; Timestamp dateAndTime; Time updateTime; double lat; char latIndicator; double lon; char lonIndicator;Float speed; int orientation; String mileage; int satelites; int hdop;int gsmSignal; boolean externalPowerOn; boolean externalPowerLow; boolean internalBatteryLow; boolean internalBatteryCharching;
boolean input1Activated; // for private companies ATM
boolean switch1Activated; // for private companies ATM
boolean switch2Activated; // for private companies ATM
boolean sesmoActivated; // for private companies ATM
boolean customInputBit0; boolean customInputBit1; boolean customInputBit2; boolean customInputBit3; boolean powerCut; boolean fuelCut; boolean doorLocked; boolean doorUnlocked; boolean moveAlertActive; boolean speedingAlertActive; boolean outOfGeoFenceActive; boolean intoGeoFenceActive; boolean customAlertBit0; boolean customAlertBit1; boolean customAlertBit2; boolean customAlertBit3; boolean workingMode1; boolean workingMode2; boolean workingMode3; boolean workingMode4; boolean harshBrake; boolean harshAccelerate; boolean harshTurnRight; boolean harshTurnLeft; Float externalPo

wer; Float internalBattery; Float internalBatteryPower; Float temperatureInsideDevice; Float temperatureExternal; Float fuelVoltage; Float humidity; Float distance; Float analog1; Float analog2; Integer check_interval_seconds; }

This class contains every piece of data that the device can transmit. Using this data we can monitor how the device is working.

public class Notification {

private int id; private Device device; private int userId; private Timestamp dateTime; private Severity severity; private boolean readed; private int errorCode; private String message;

}

Account{

String name; int Id; List<Device> devices

}

This is the entity of an Account, the list is linked to each one of the devices in the devices table.

Error{

int Code; private String message; private Severity severity; }

Business Logic:

This package as its name implements is where we manipulate and handle the data like fetching data from data base and sending it back to the client side.

Classes:

AccountBL, AdminBL, DeviceBL, DeviceDataBL, NotificationBL, ErrorBL, SupportBL, UserAccountBL, UserBL.

Controllers:

The controllers classes are the classes which received the parameters from the client and called for the right classes of the business logic to handle the query. Each functionality is mapped by a special name In the right controller for it.

Classes:

AccountController, DashboradInfoController, DeviceController, DeviceDataController, NotificationController, UserController.

Repositories:

This package is an interface package, it contains interface classes that have built in functions from the spring API and overridden functions that perform SQL queries from and to the database.

AccountRepository, AdminRepository, DeviceRepository, DeviceDataRepository, ErrorRepository, NotificationRepository, SupportRepository, UserAccountRepository, UserRepository.

Adding New query:

To add new functionality to the app, we followed these steps: (add new user for example)

1- **The client builds the API, explains the function; what to return and what parameters you are getting. For example:**

```
/*this function return true if addition of new User is accepted, else return false
*
*@param     username          String username
*@param     emailAddress      String email address of the user
*@param     userType          String usertype of the user {Admin, Support, Account}
*@param     accountName       String account name (name of the company), if the userType is not
"Account", then it will be a null value
*@return    boolean
*/
```

The client specified the type they wanted to be returned (Boolean), the function name(addNewUser) and the parameters they are sending:

```
boolean addNewUser(String username, String email, String userType, String accountName)
```

2- **Following the MVC(model-view-control) design pattern, the entity (or model) that we are handling has a class in the beans package, which translates as a table in the database also the entity has another class which is the entity controller . For example:** (User.java, UserController.java)

In the controller class we added we created the following function:

```
@GetMapping("addNewUser") ←( mapping should be the exact name used in client.)
        public boolean AddToDb(@RequestParam String username, @RequestParam String email,
@RequestParam String userType, @RequestParam String accountName) throws NoSuchAlgorithmException
{
                return userBL.addNewUser(username, email, userType, accountName);
        }
```

-as noted in the document above, the controller only receives the parameters and directs them to the right class to compute and perform the query.

3- **The BL package is where all of the computing and execution of queries is done.**

```
public boolean addNewUser(String username, String email, String userType, String accountName) throws
NoSuchAlgorithmException {
Account acc = accountRepository.findByName(accountName);
if (userType.contentEquals("Admin")) {
        Admin admin = new Admin(email,username,username,null,acc);
        adminRepository.save(admin);   }
else if(userType.contentEquals("Account")) {
        UserAccount user_acc= new UserAccount(email ,username, username, null, acc);
        UseraccountRepository.save(user_acc);     }
else if(userType.contentEquals("Support")) {
        Support sup = new Support();
        supportRepository.save(sup);     }
return true;
}
```

*as we see above we created new instance of the entity required to be added, and the with the use of the suitable interface class in the repository package we can add the new user to the database.

4- the repo (repository) package has interface classes which extend the CRUD repository. For example the UserRepository.java class has a set of built in functions that are implemented in the crud interface which the spring boot provided us, and also a set of custom made quires.

```
public interface UserRepository<T extends User>  extends CrudRepository<T, Integer> {
        User findByUserName(String UserName);
        User findBysysId(int id);
        ArrayList<T> findByNameContaining(String name);
        ArrayList<T> findAll();
        ArrayList<T> findAllByOrderBySysIdDesc();

        @Query(nativeQuery = true, value =
                        "select * " +
                        "from public.users " +
                        "where account_id = ?1 ;" )
        List<User> getAllUsersForAccountID(int accountId);

        @Query(nativeQuery = true, value =
                        "select account_id " +
                        "from public.users " +
                        "where sys_id = ?1 ;" )
        public int getAccountIDForUser(int Id);

}
```

Summary

The server was developed with the help of spring boot, which was a very helpful framework. Here is a link of the spring documentation which has more specified explanation about the whole process of building a web server using this tool:

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/