DMD I Project. Phase III

Team:

Alfiya Musabekova

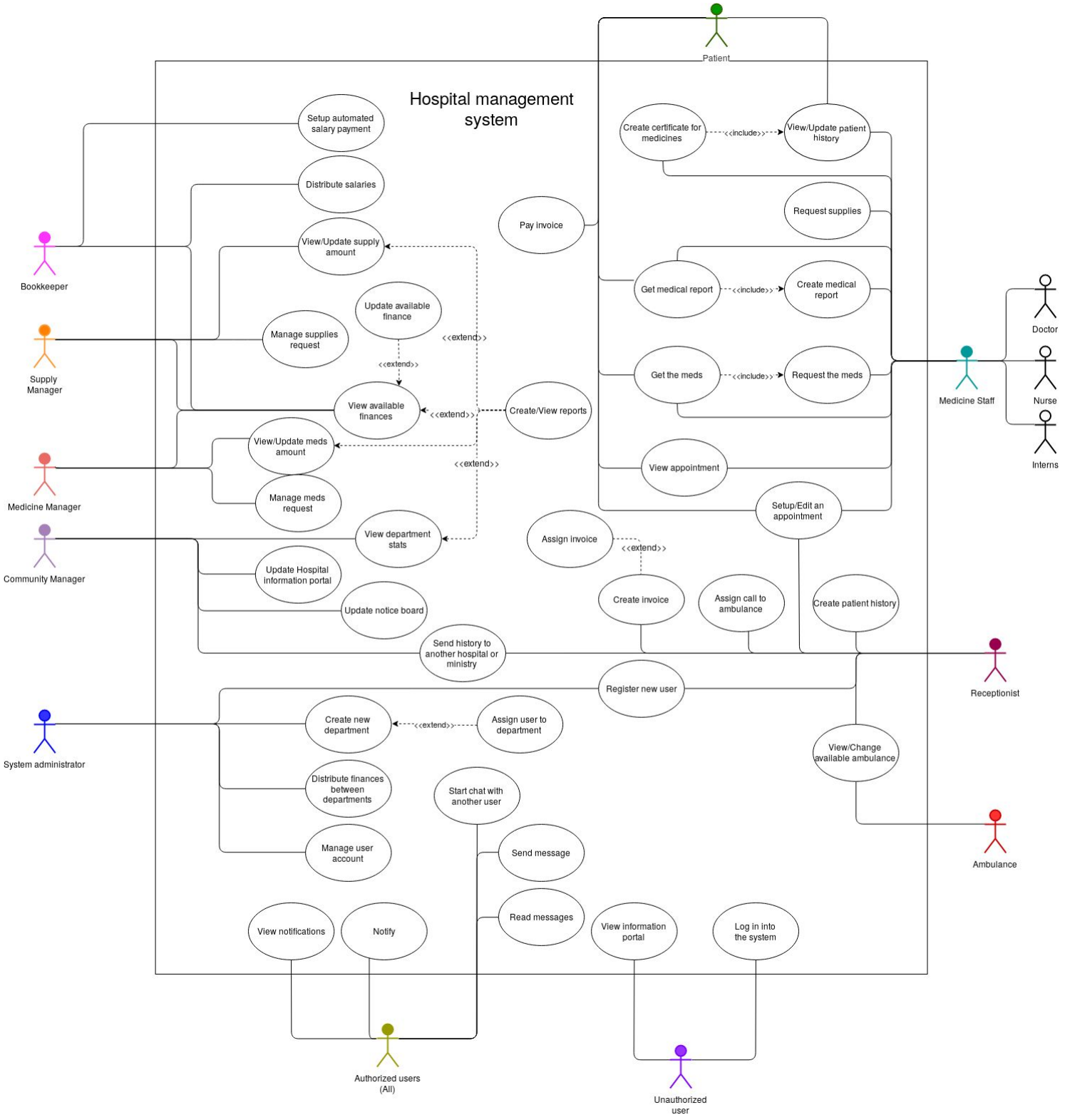
Jameel Mukhutdinov

Mikhail Kuskov

Ruslan Muravev

Improved Phase I and Phase II:

Use-case diagram:



[LINK]

Requirements:

Requirement ID	1
Title	Simultaneous access
Type	Non-Functional
Description	Simultaneous access to all resources available for all users
Priority	1
Risk	C

Requirement ID	2
Title	Management of users' accounts
Type	Functional
Description	Ability to create, update or delete user accounts, change their roles, attachment them to departments, also show list of all users
Priority	1
Risk	C

Requirement ID	3
Title	Security (Several levels of access)
Type	Non-functional
Description	Several levels of protection for user data and the system itself
Priority	1
Risk	C

Requirement ID	4
Title	Performance
Type	Non-Functional
Description	Maximum response time should not exceed 15 seconds
Priority	1
Risk	C

Requirement ID	5.1
Title	Management of medicines
Type	Functional
Description	Allow users to store and view a list of available medicines, change it and keep a history of these changes. System should accept requests on medicines and confirm it
Priority	1
Risk	H

Requirement ID	5.2
Title	Capacity and resources
Type	Non-functional
Description	Amount of computational and memory resources are linearly dependent on amount of system users. At most, one user requires 1 TiB memory
Priority	1
Risk	H

Requirement ID	5.3
Title	Management of supply
Type	Functional
Description	Allow users to store and view a list of available supplies, change it and keep a history of these changes. System should accept requests on supply and confirm it
Priority	1
Risk	H

Requirement ID	5.4
Title	Management of patient history
Type	Functional
Description	Allow users to create, store and view patients' medical histories and medical reports, change it and keep a history of these changes System should also provide way to send this histories and reports to ministry of healthcare or another hospital
Priority	1
Risk	H

Requirement ID	5.5
Title	Management of finances
Type	Functional
Description	Allow users to store and view data of the finances, change it and keep a history of changes. Distribute budget between departments, salaries between employees
Priority	1
Risk	H

Requirement ID	6
Title	Ambulance management
Type	Functional
Description	Provide convenient way to manage the ambulance - Show free and busy ambulances - Ability to assign incoming calls for first-aid lorry
Priority	1
Risk	H

Requirement ID	7
Title	Appointment management
Type	Functional
Description	Provide an easy way to create, view, edit and delete appointments for medical stuff, patients and receptionists
Priority	1
Risk	H

Requirement ID	8
Title	Notifications
Type	Functional
Description	System allows to create notifications and also should notify all the involved actors about the related events and have ability to define the notification policy
Priority	1
Risk	M

Requirement ID	9
Title	Automated report generation
Type	Functional
Description	Ability to create and save all types of reports and automatically fill them out based on available information
Priority	1
Risk	M

Requirement ID	10
Title	Hospital system monitoring
Type	Functional
Description	Provide convenient way to collect, process and represent statistical data or information about hospital departments and all subsystems
Priority	1
Risk	M

Requirement ID	11
Title	Invoice Management
Type	Functional
Description	Creation, distribution, and payment of invoices
Priority	1
Risk	M

Requirement ID	12
Title	Responsive user interface
Type	Non-functional
Description	User interface should be friendly and intuitive for inexperienced users

Priority	2
Risk	H

Requirement ID	13
Title	Notice board
Type	Functional
Description	Publicly accessible board with all information related to the hospital and relevant to specific user
Priority	2
Risk	M

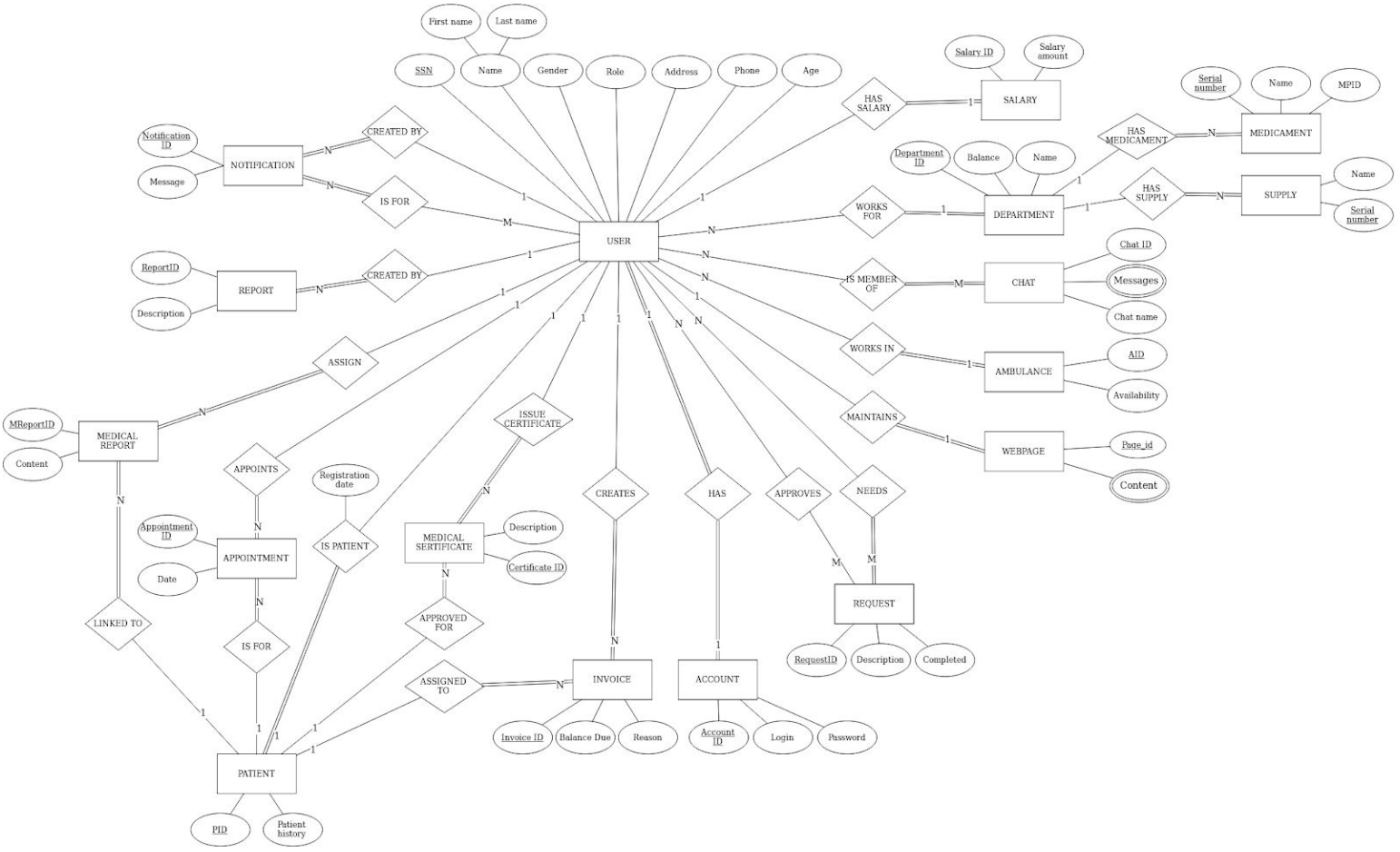
Requirement ID	14
Title	Access from different devices
Type	Non-Functional
Description	Provide all the described functionality for users on personal computers (under OS Windows, MacOS and Linux) and mobile devices (Android, IOS)
Priority	2
Risk	M

Requirement ID	15
Title	Internal Communication
Type	Functional
Description	Provide internal chat for users to communicate within the system
Priority	2
Risk	M

Requirement ID	16
Title	Information portal
Type	Functional
Description	Information portal accessible from the Internet, where visitors will be able to see general information about the hospital. Authorized patients will be able to use appointment subsystem
Priority	2
Risk	L

Requirement ID	17
Title	Automated salary payment
Type	Functional
Description	Ability to setup a frequency and amount of salary, automatically paid for all hospital employees
Priority	3
Risk	L

Entity Relation Diagram:



[LINK]

Explanation of design decisions in ERD:

Instead of inheritance that is made by many *IS A* relations we decided to use additional attribute "*Role*" for user. Patient and ambulance is represented by another entity because it simplifies the user entity, and also they have additional proper relations .

Diagram contain only strong entities. Such solution design is simpler and prevents creation of weak entities and partial keys for them.

Unambiguity about attributes:

- Balance of department is current state of balance sheet of department.
- MPID of medicaments represents - identification of type of medicine.
- Availability of ambulance can be either true if ambulance is free or false otherwise.
- Completed of request can be either false if request was not approved, true if request was approved, unknown if there is no decision yet.
- Role of user is not multi-valued attribute because we assume that one user can have only 1 role.

Unambiguity about entities:

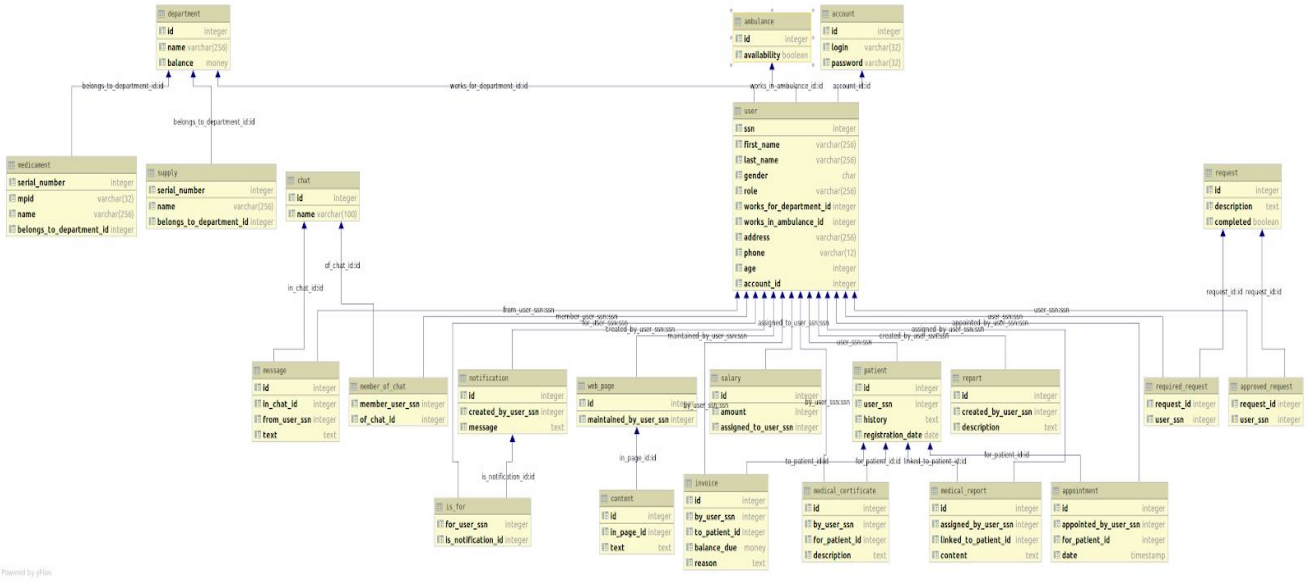
- User represent any existing human in our system.
- Report entity represent description of transactions of user.
- Pages of hospital information portal are connected to one Hospital Portal represented as a website.
- Medical certificate represent access of patient to get meds.

Unambiguity about relations:

- User *WORKS FOR* department is not total participation because user can be patient.(While department *HAS WORKERS* as users is total because there should be at least one worker in department)
- Request can be disapproved; therefore, we do not have total participation in relation *APPROVED*.
- Sometimes to request something expensive hospital is needed requests from not only one doctor and, the same situation from other hand, to buy something expensive, requests should be approved by not only one doctor; therefore, we have many-to-many cardinality in relationships *NEEDS* and *APPROVES*.
- Patient is guaranteed to be user.(*IS PATIENT* is total from patient side). Same situation with ambulance and *WORKS_IN* relation.

Phase III:

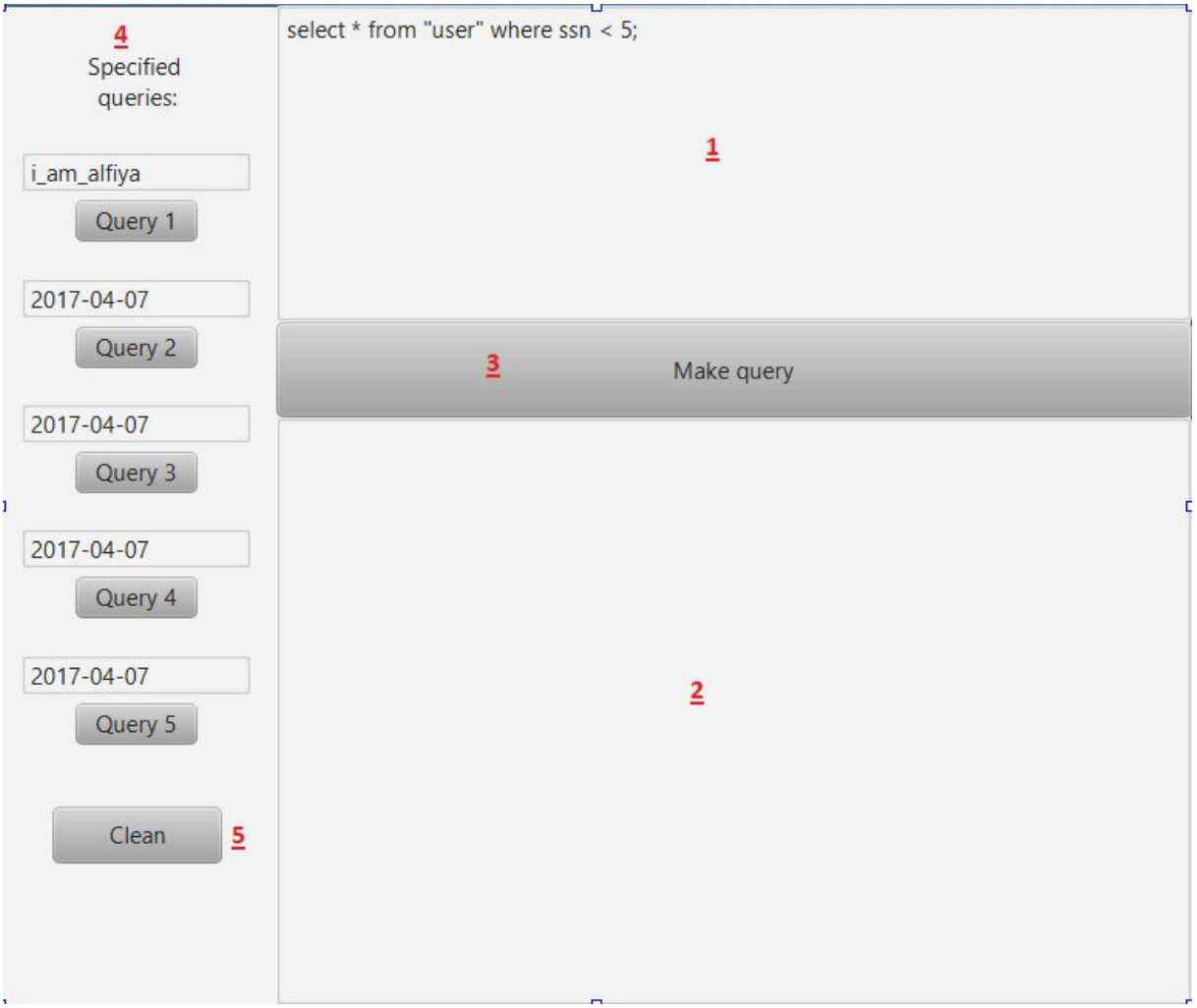
Relational Schema:



https://github.com/GamingJam/DMD-Phase-3-JavaFx/blob/master/images/Relational_Scheme.png

This project phase is implemented using Java language and Java FX for database population program and GUI. Postgres JDBC driver were used to connect GUI to database server. PostgreSQL and MySQL were used as the RDBMS.

GUI Description:



1. Text area for the specification of the query for the database.
2. Text area for output of query results.
3. When clicked, it executes the query described in field 1 and displays the result in field 2.
4. Specified queries, with text field and button for each one. Text fields allow to vary needed parameters. When the button is pressed the result will be show in the 2.
5. Clean button allows to clear 2 field.

Specified SQL queries:

Query 1

```
WITH appointment_for_current AS
         (
             SELECT *
             FROM (SELECT id as pid
                   FROM patient
                   WHERE user_ssn in (
                       SELECT ssn
                       FROM "user" JOIN account on account_id = account.id
                       WHERE login = 'i_am_alfiya' AND gender = 'F')
                  ) as current_patient
                      JOIN appointment on for_patient_id = pid
         )
SELECT first_name, last_name
FROM "user"
WHERE ssn in (SELECT appointed_by_user_ssn
              FROM appointment_for_current
              WHERE date in (SELECT max(date) FROM appointment_for_current)
)
  AND ((left(first_name, 1) = 'M' AND left(last_name, 1) <> 'M') OR
       (left(first_name, 1) = 'L' AND left(last_name, 1) <> 'L') OR
       (left(first_name, 1) <> 'M' AND left(last_name, 1) = 'M') OR
       (left(first_name, 1) <> 'L' AND left(last_name, 1) = 'L'));
```

1. find all appointments for patient, who forget her bag, using her login and the fact that she is women
2. find latest date when patient was in hospital
3. find all doctors, who have name starting with *'L'* or *'M'* (first or last but not both)

*Login, which in this case is 'i_am_alphiya', is taken from text field in GUI.

Query 2

```
SELECT ssn,
       first_name,
       last_name,
       to_char(DATE '2019-11-17'+ INTERVAL '1 day' * day_of_week, 'dy') as weekday,
       total,
       statistic_about_appointments.total / count_of_workweeks.amount_of_work_weeks :: double precision as average
FROM (
         SELECT appointed_by_user_ssn,
                day_of_week,
                count(date) as total
         FROM (
                  SELECT appointed_by_user_ssn, EXTRACT(dow FROM date) as day_of_week, date
                  FROM APPOINTMENT
                  WHERE date >= ('2017-03-07'::date - INTERVAL '1 year')
              ) as doctor_ssn_dow_and_date
         GROUP BY appointed_by_user_ssn, day_of_week
     ) as statistic_about_appointments
         JOIN
     (
         SELECT distinct appointed_by_user_ssn, count(week) as amount_of_work_weeks
         FROM (
                  SELECT appointed_by_user_ssn, EXTRACT(week FROM date) as week
                  FROM appointment
                  WHERE date >= ('2017-03-07'::date - INTERVAL '1 year')
              ) as doctor_works_in_week
         GROUP BY appointed_by_user_ssn
     ) as count_of_workweeks
     on count_of_workweeks.appointed_by_user_ssn = statistic_about_appointments.appointed_by_user_ssn
         JOIN "user" on count_of_workweeks.appointed_by_user_ssn = ssn
WHERE role = 'doctor'
ORDER BY ssn, day_of_week;
```

1. find all appointments in required period
2. find total amount of appointments in each time slot for each doctor
3. find amount of workweeks for each doctor in required period
4. find average amount of appointments in each time slot based on total amount of appointments and amount of workweeks

*Date, which in this case is ‘2017-03-07’, can be changed by text field in GUI.

Query 3

```
SELECT first_name || ' ' || last_name as twice_a_week
FROM "user" JOIN
     (SELECT count(date) as num, user_ssn
      FROM patient JOIN appointment a on patient.id = a.for_patient_id
      WHERE EXTRACT('week' from date) = EXTRACT('week' from '2017-04-07 00:00:01'::timestamp)
      GROUP BY patient.id) as week1 on ssn = week1.user_ssn
      JOIN
     (SELECT count(date) as num, user_ssn
      FROM patient JOIN appointment a on patient.id = a.for_patient_id
      WHERE EXTRACT('week' from date) = EXTRACT('week' from '2017-04-07 00:00:01'::timestamp) - 1
      GROUP BY patient.id) as week2 on ssn = week2.user_ssn
      JOIN
     (SELECT count(date) as num, user_ssn
      FROM patient JOIN appointment a on patient.id = a.for_patient_id
      WHERE EXTRACT('week' from date) = EXTRACT('week' from '2017-04-07 00:00:01'::timestamp) - 2
      GROUP BY patient.id) as week3 on ssn = week3.user_ssn
      JOIN
     (SELECT count(date) as num, user_ssn
      FROM patient JOIN appointment a on patient.id = a.for_patient_id
      WHERE EXTRACT('week' from date) = EXTRACT('week' from '2017-04-07 00:00:01'::timestamp) - 3
      GROUP BY patient.id) as week4 on ssn = week4.user_ssn
WHERE week1.num >= 2 AND week1.num >= 2 AND week3.num >= 2 AND week4.num >= 2;
```

1. find number of appointments for each patient in every week
2. check this number to be greater or equal than 2, patients, for which this is correct, visited hospital at least twice a week

*Date, which in this case is ‘2017-03-07 00:00:01’, can be changed by text field in GUI.

Query 4

```
WITH possible_charge AS (
    SELECT *
    FROM (
             SELECT 200
             UNION ALL
             SELECT 250
             UNION ALL
             SELECT 400
             UNION ALL
             SELECT 500
         ) as possible_charge(charge)
)
SELECT SUM(charge * amount_of_appointments) as income_in_rubles
FROM (
         SELECT age, amount_of_appointments
         FROM (
                  SELECT id as patient_id, age
                  FROM patient
                           JOIN "user" on patient.user_ssn = "user".ssn
              ) as age_of_patient
                  JOIN
              (
                  SELECT for_patient_id, count(app_id) as amount_of_appointments
                  FROM (
                           SELECT id as app_id, for_patient_id
                           FROM appointment
                           WHERE date >= ('2017-04-07'::date - interval '1 month')
                       ) as appoinments_in_previous_month
                  GROUP BY for_patient_id
              ) as amount_of_appoinments_in_previous_month
              on patient_id = for_patient_id
     ) as age_and_amount_of_appointments_in_previous_month_of_patient
         CROSS JOIN possible_charge
WHERE (age < 50 AND amount_of_appointments < 3 AND charge = 200)
   OR (age < 50 AND amount_of_appointments >= 3 AND charge = 250)
   OR (age >= 50 AND amount_of_appointments < 3 AND charge = 400)
   OR (age >= 50 AND amount_of_appointments >= 3 AND charge = 500);
```

1. find amount of appointments for each patient in required period
2. find charge for each appointment for each patient based on patient age and amount of appointments
3. find income by multiplying amount of appointments for each patient on corresponding charge value

*Date, which in this case is ‘2017-03-07’, can be changed by text field in GUI.

Query 5

```
SELECT ssn, first_name, last_name
FROM(
    SELECT ssn, first_name, last_name
    FROM(
        SELECT ssn, first_name, last_name, count(year) as amount_of_hard_working_years
        FROM(
            SELECT ssn, first_name, last_name, year
            FROM(
                SELECT ssn, first_name, last_name, year, count(for_patient_id) as patients_in_year
                FROM(
                    SELECT distinct ssn, first_name, last_name, EXTRACT(year from date) as year, for_patient_id
                    FROM "user" join appointment on appointed_by_user_ssn = ssn
                    WHERE role = 'doctor' AND EXTRACT(year from date) > EXTRACT(year from CURRENT_DATE) - 10
                ) as patients_visited_by_doctor_in_year
                GROUP BY ssn, first_name, last_name, year
            ) as amount_of_patient_visited_by_doctor_in_year
            WHERE patients_in_year >= 5
        ) as hard_working_years_of_doctor
        GROUP BY ssn, last_name, first_name
    ) as amount_of_ard_working_years_of_doctor
    WHERE amount_of_hard_working_years = 10
) as doctors_with_not_less_than_5_patients_per_year
JOIN
(
    SELECT ssn as ssn_of_doctor_with_100_patients
    FROM(
        SELECT ssn, count(for_patient_id) as total_patients
        FROM(
            SELECT distinct ssn, for_patient_id
            FROM "user" join appointment on appointed_by_user_ssn = ssn
            WHERE role = 'doctor' AND EXTRACT(year from date) > EXTRACT(year from CURRENT_DATE) - 10
        ) as patients_visited_by_doctor_ssn
        GROUP BY ssn
    ) as amount_of_patients_visited_by_doctor_ssn
    WHERE total_patients >= 100
) as doctors_with_not_less_than_100_patients
on ssn = ssn_of_doctor_with_100_patients
```

1. find appointments in required period
2. find doctors that have such appointments
3. find amount of patients visited by each doctor each year in required period
4. find doctors that visited at least 5 patients every year each year in required period
5. find amount of patients visited by each doctor in required period
6. find doctors that visited at least 100 patients in required period
7. find common entries between doctors that visit 100 patients in 10 years and 5 patients every year

*CURRENT_DATE can be changed by text field in GUI.

Navigation in .zip:

1. **images** folder contains ERD, use-case diagram and relational schema as png files.
2. **sql** folder contains sql dump files for database manipulation, in particular, it contains files for creation, deletion , population on PostgreSQL and MySQL and described in specification queries on postgres.
3. **src/test** folder contains all java files responsible for data generation and GUI for queries.
4. **libs** folder java library for operating on postgres.

File	Function
postgres_create.sql, postgres_drop.sql,	A SQL dump files for creation and population/dropping all entries of the database on *PostgreSQL*
mysql_create.sql, mysql_drop.sql	A SQL dump files for creation and population/dropping all entries of the database on *MySQL*
query_n.sql	SQL query for the specified n^{th} statement
mysql_insert.sql, postgres_insert.sql	Examples of pseudo-random population of the database on *MySQL* and *PostgreSQL*
DataGenerator.java	Class which is used for random data generation, data is taken from folder *txts*
PostgresGen.java, GenScript.java	Classes with script for an automatic and pseudo-random generation of textual sequence of INSERT instructions
DBConnector.java	Class for connection with Database
MainFormController.java, mainForm.fxml, GUIMain.java	Classes for working with GUI: read data from text fields, execute queries
config.cfg	Settings file to specify setting for connection to Postgres Server

How to run the application:

Unzip the contents of the archive to any suitable place.

1) Install JRE version 11 and JavaFX library
2) Using command line type $\$java\ -jar\ GenScript.jar$ to start generating script
3) To start GUI you may use $start\ gui.sh$

In case when .jar is not working, you may import this folder into IntelliJ IDEA. Then setup launch for GUIMain java class, install openjfx library and add it to the project. Additionally, all necessary libs and files may be found in $libs$ folder

Link to the project on GitHub

https://github.com/GamingJam/DMD-Phase-3-JavaFx