Design document:
Hospital Management for rare diseases

Group 1: *Martina Galán, Celia Pérez, Elena Marco, Blanca Pueche*.

INDEX:

1. Overview:

The hospital management for rare diseases program is in charge of organizing a series of hospitals, mainly distributing the patients between them depending on many factors.

The program is composed of a user interface that is connected to a database that handles different entities and the relationship between them:

1. Hospitals
2. Machines
3. Illnesses
4. Doctors
5. Patients
6. Sorting Medics

The connections between them assure the correct workflow of information:

A hospital has patients, doctors and machines.

The patients are assigned a hospital and have one or several illnesses.

The doctors work at a hospital and treat one or several illnesses.

Each machine treats one or several illnesses.

Due to the simplicity of the program, we cannot add new hospitals, doctors, illnesses or machines, but if the user wanted to be able to do so then the changes would be minor, just using JPA or JDBC, depending on the object, and adding that option to the main menu of the application.

The database, that implements SQL, is in charge of registering all the information mentioned above in order for the program to add or retrieve information quickly and without any problems in just a few clicks.

The error handling is one of the highlights of our program, this includes not entering a number when asked or maybe selecting an incorrect ID for example. When these things happen, the user will be asked to input the option again or will be redirected to the specified menu, meaning the program does not crash and lets the user keep navigating through it.

When it comes to security, in our program, there's only three sorting medics that have access, but as mentioned earlier, if the user wishes to add more, it can be easily done with just a few lines of code.

These sorting medics have a username and a password that is kept in the database so that only if the combination is correct, the user will have access, if not, the program will ask again for the correct combination of both.

Finally, the main goal of the program is to be able to sort patients in hospitals, this is easily done thanks to the relations between the objects, as the program scans through hospitals, machines and illnesses to be able to pick the best hospitals for each patient, if an specific hospital cannot be found, or is not available to treat more patients, the patient will stay in the main hospital.

1. Requirements:

The main requirement of this program is to be run on computers that work with Windows and Mac.
Other than that, to be able to start working with it, there should be information in the database, but most importantly, there should be at least one sorting medic, if not, the program will not be able to compare the combination of password and username with anything and will not let the user enter.
As mentioned earlier, right now, we are not able to insert hospitals, illnesses, machines or doctors, so the database should have all that information installed beforehand in order to be able to perform all of its functions properly. This can be changed by simply adding a few new options in the main menu to be able to insert all those things and start with an empty database.
Having all these in mind, if used correctly, the program will be able to perform its functions to full capability.

2. System architecture:

To be able to correctly understand the workflow and structure of the program, we made an ER Diagram, a Use Case Diagram, a MockUp and a UML.
Click the following link to be redirected to a folder with all these things.
(Information here)

1. ER Diagram:
   In this diagram we can see the different relationships between entities explained earlier for a better understanding of how everything is connected.
2. Use-Case Diagram:
   Here we can see a few of the functions the sorting medic will do such as delete doctor or search hospital, to see everything in a simpler and quicker way.
3. MockUp:
   With this picture we will be able to see the menus of the program depending on the option the users choose, this will all be explained below, and can be seen in a faster way with examples and pictures in the Users Manual.
4. UML:
   In the UML we can see everything that goes on when it comes to the code, the relations, interfaces, objects and such that structure the program to make it work properly.

3. User Interface design:

A good program must have a good UI made up of different modules and menus depending on the options the user chooses. Having this in mind our program has as many as nine of those.

1. Login menu:
   In this menu, the sorting medic is able to input their username and password and will not be allowed to enter until the combination of both is correct.

2. Main menu:
   When entering, the maine menu will pop up with the following option to choose:
   1. Register new patients.
   2. Select patient.
   3. Select doctor.
   4. Show hospitals.
   5. Create XMLs.
3. Registration:
   This menu lets the user register a new patient when option 1 in the main menu is chosen. Other than the usual personal information, it will redirect the user to a different menu to input the illness information.
4. Illness information:
   Here the user inputs the illness and the severity.
5. Choose patient:
   The user will be asked to input the name of the patient and a list of patients that meet that criteria will be shown to choose one of them.
6. Patient menu:
   Here we have the following options:
   A. Update data.
   B. Show data.
   C. Search hospital.
   D. Assign new illness.
   E. Update illness severity.
   F. Create XML.
7. Points 5 and 6 are repeated with the doctor.
8. Choosing XMLs:
   In this menu the user is able to choose to do an XML of a machine, illness or hospital.

4. Data model:

In our program we have eight different pojos, including the many to many relationships.
Pojos:

1. Doctor: implements JPA and has the following attributes:
   1. Id.
   2. Name
   3. Surname.
   4. Dob (date of birth).
   5. Speciality.
   6. Salary.
   7. Hospital (the one they work at).
   8. TreatsIllness (list of the illnesses they treat).
2. Hospital: implements JDBC and has the following attributes:

1. Id.
2. Name.
3. Location.
4. Patients.
5. Doctors.
6. Machines.
7. AV (availability).

3. Illness: implements JDBC and has the following attributes:
    1. Id.
    2. Condition.
    3. Patients.
    4. Doctors.
    5. Machines.
4. Machines: implements JDBC and has the following attributes:
    1. Id.
    2. Name.
    3. Hospital.
    4. Treats (list of illnesses they treat).
5. Patient: implements JPA and has the following attributes:
    1. Id.
    2. Name.
    3. Surname.
    4. Dob.
    5. Hospital.
    6. Photo.
    7. Illness.
6. Sorting medic: implements JPA and has the following attributes:
    1. Id.
    2. Username.
    3. Password.
7. Has: implements JPA, is the relationship between patient and illness. Has the following attributes:
    1. PatientId.
    2. IllnessId.
    3. Severity.
    4. Patient.
    5. Illness.
8. HasId: manages the id of the Has relationship. Has the following attributes:
    1. PatientId.
    2. IllnessId.
9. Treats: is the relationship between machine and illness. Has the following attributes:
    1. Machine.
    2. Illness.
    3. SuccessRate.

5. Interfaces, JPA and JDBC:

Our program is managed, as mentioned in the previous chapter, with JDBC and JPA, and they also implement different interfaces with the following methods.

1. Interfaces:
    1. DoctorManager:
        a) UpdateDoctor: updates information related to the doctor.
        b) SearchByName: returns a list of doctors with a specific combination of name and surname.
        c) InsertDoctor: inserts a doctor in the database.
        d) DeleteDoctor: deletes a doctor from the database.
        e) AssignHospital: assigns the doctor to a hospital.
        f) AssignIllness: assigns an illness to a doctor.
        g) getDoctor: returns a doctor with a specific id.
        h) DocTreatsIllness: returns a list of doctors that treat a specific illness.
    2. HasManager:
        a) GetListHas: returns a list of the Has of a specific patientId.
        b) GetHas: returns a Has with specific patientId and illnessId.
    3. HospitalManager:
        a) InsertHospital: inserts a hospital into the database.
        b) UpdateHospital: updates data of a hospital.
        c) SearchByName: returns a list of hospitals that have a specific name.
        d) GetHospital: returns a hospital with a specific id.
        e) GetHospitals: returns a list of all the hospitals.
        f) Search1ByName: returns a hospital with a specific name.
    4. IllnessManager:
        a) InsertIllness: inserts an illness in the database.
        b) GetIllness: returns an illness with a specific id.
        c) GetIllnesses: returns a list of all the illnesses.
        d) RelationDoctorIllness: returns a list of illnesses being treated by a specific doctor.

    5. MachineManager:
        a) SearchByName: returns a list of machines with a specific name.
        b) AssignHospital: assigns a machine to a hospital.
        c) InsertMachine: inserts a machine to the database.
        d) GetMachine: returns a machine with a specific id.
        e) MachineTreatsIllness: returns a list of machines that treat a specific illness.
        f) GetMachines: returns a list of all the machines.
        g) MachinesInHospital: returns a list of machines in a specific hospital.
    6. PatientManager:
        a) Update: updates info of a patient.
        b) SearchByName: returns a list of patients with a specific name.
        c) InsertPatient: inserts a patient in the database.
        d) AssignHospital: assigns a hospital to a patient.
        e) AssignIllness: assigns an illness to a patient.
        f) GetPatient: returns a patient with a specific id.

7. SortingMedicManager:
    a) SearchUser: returns TRUE or FALSE if the combination of username and password is in the database.
8. XMLManager:
    a) Patient2Xml: creates an XML of a patient.
    b) Hospital2Xml: creates an XML of a hospital.
    c) Doctor2Xml: creates an XML of a doctor.
    d) Illness2Xml: creates an XML of an illness.
    e) Machine2Xml: creates an XML of a machine.
    f) xml2Patient: creates a patient out of an XML.
    g) xml2Hospital: creates a hospitalout of an XML.
    h) xml2Doctor: creates a doctor out of an XML.
    i) xml2Illness: creates an illness out of an XML.
    j) xml2Machine: creates a machine out of an XML.
    k) xml2Html: creates an HTML of a specific XML.

2. JPA: The following use JPA:
    1. Doctor.
    2. EntityManager (manages the entity manager of JPA).
    3. Has.
    4. Hospital (only a couple of methods).
    5. Patient.
    6. SortingMedic.

3. JBDC:
    1. ConnectionManager (creates tables and inserts info by default).
    2. Hospital (majority of methods).
    3. Illness.
    4. Machine.

6. XML and HTML:

As mentioned in previous chapters, our program is able to generate XMLs and HTMLs of different entities.
This is an option given in the main menu and in specific menus of patient and doctor, and when choosing that option, it lets us choose whether we want a HTML or not.
In case of choosing to create an HTML, the link will appear with the XMLs in the xml folder of the project.

7. Error and exception handling:

We already talked about this in the overview, but our program is able to manage exceptions in a way that it does not crash.
These exceptions include:

1. NumberFormatException.

2. DateTimeParseException.
3. IOException.
4. FileNotFound.
5. NoResultException.

All of which are managed either asking for a number again or selecting null for a photo file in case of not being found, depending on the case.