Patient Management System – Appointment feature

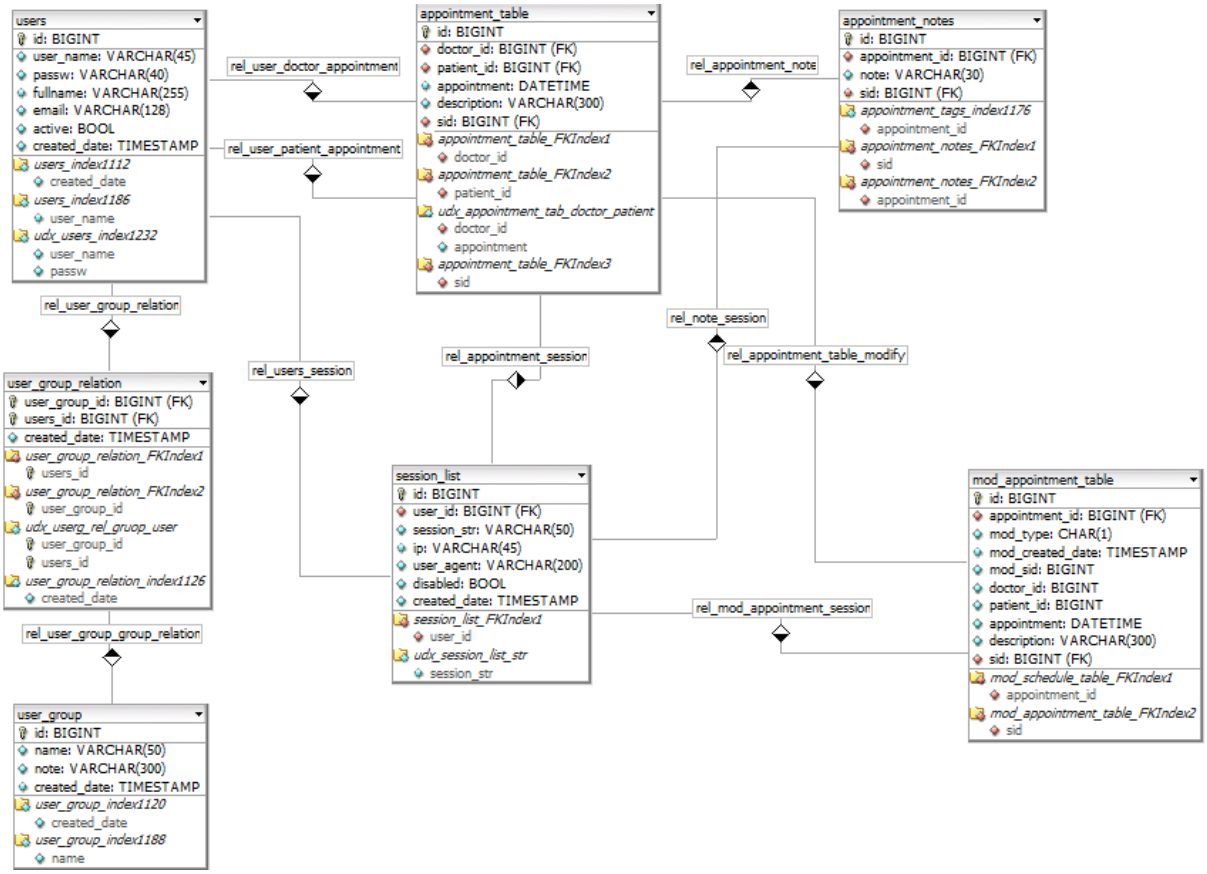
This documentation will represent the new feature. Each doctor has a calendar. The doctor has only one patient at a time. The system must be able to handle that the doctor can change his/her appointments or re-appointment an entire day.

The Database structure

In this chapter you will find the high level database design. The 1. Figure represents the full database structure.

The first column's name is id. This is the primary key of the table and it is an auto incrementing column.

More tables include a 'created_date' column. This contains creating time of rows. This column has got default value, so we don't have to manipulate manually.



1. Figure

Users table

This table contains all patients and all doctors. Currently, the table has just a few columns. Of course, this table can be expandable.

Columns:

- **user_name:** this is the user's unique name
- **passw:** this is the user's password. The content has to be encoded. e.g.: with SHA-1 or MD5 hash function
- **fullname:** first and last name
- **email:** user's email address
- **active(true):** It is possible to disable the user. false: disable, true: enable

User_group table

This table contains all of the user's groups. These groups represent the permissions and what kinds of functions are available. e.g.: User is a doctor or patient. The doctor is able to create new appointment.

Columns:

- **name:** the group name
- **note:** description of group

Appointment_table

This is an appointment table. We record here when the doctor will meet with the patient.

Columns:

- **doctor_id:** unique id of row from user table
- **patient_id:** unique id of row from user table
- **appointment:** date of appointment
- **description:** thorough details about meeting
- **sid:** this id identifies the user's process

Appointment_notes table

This table contains tags of appointments. We can say these are short descriptions.

Columns:

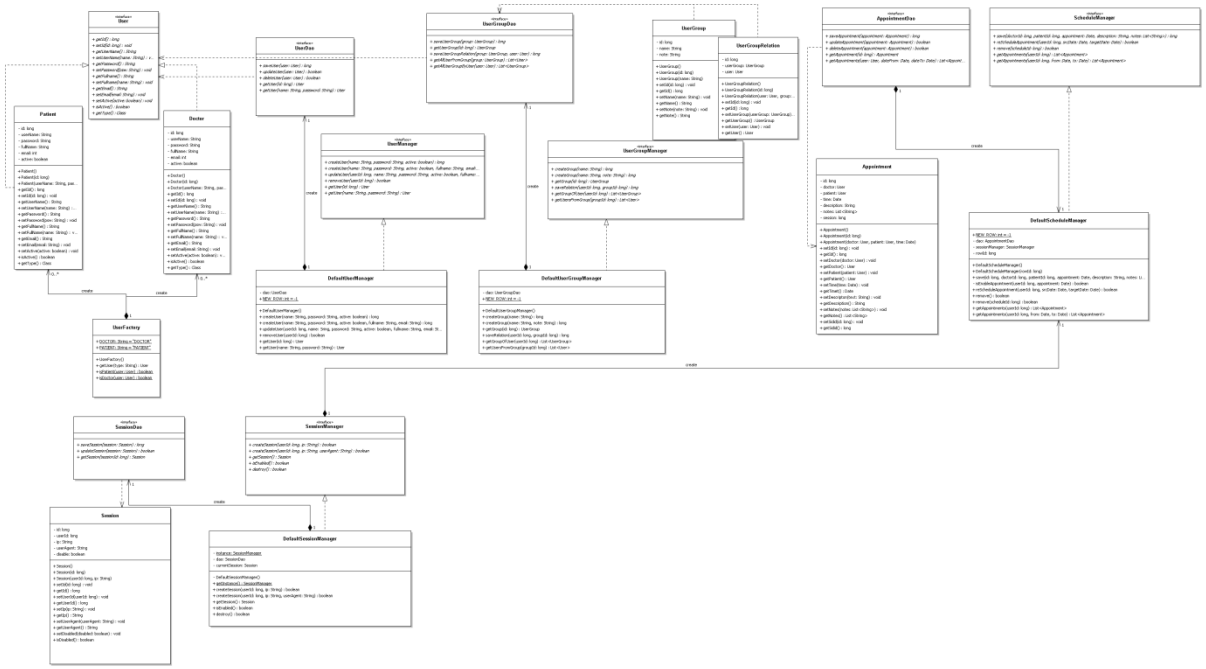
- **appointment_id:** unique id of row from appointment_table table
- **note:** this is a tag

Mod_appointment_table table

This table contains all modifications of the main table (appointment _table). So we can restore or recheck previous data that changed by doctors. We load contents with a trigger, so don't have to manipulate with manual way.

Java Class Design

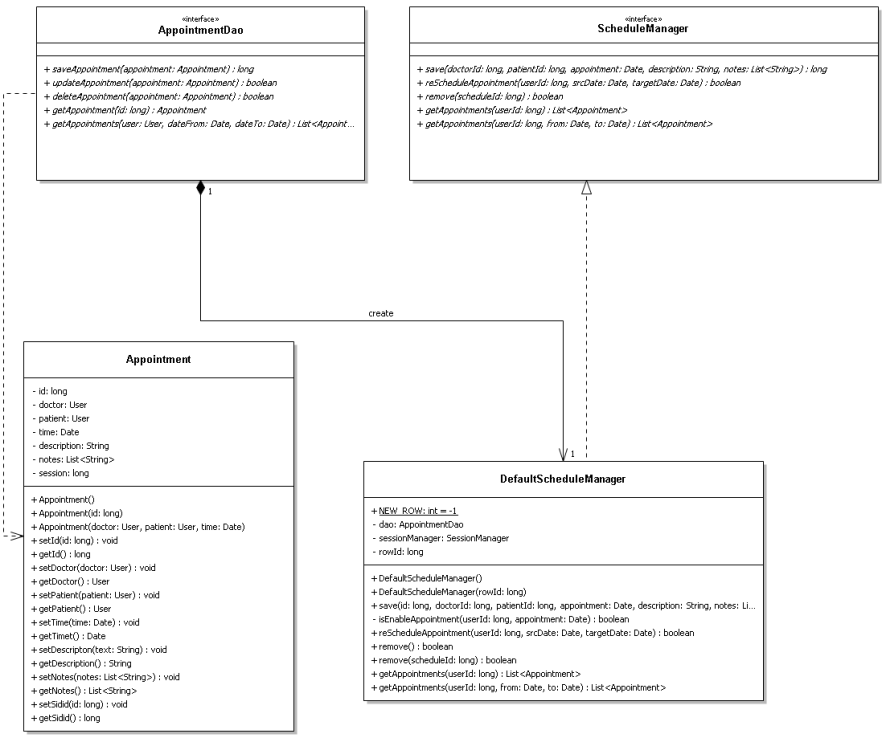
This picture represents the full java design which is important for the new feature.



2. Figure

Appointment design

This chapter deals with the appointment manager. We have an interface whose name is *AppointmentDao*. The *JDBCAppointmentDao* class implements this interface. This class uses the *Appointment* model class which will pass information to *JDBCAppointmentDao* object to get the data it needs. For managing of data we use the *DefaultAppointmentManager* class. Through the manager class we can reach the creator, the updater, controller and copy functions.



3. Figure

Appointment class

This object is a simple POJO containing get/set methods. This object is going to work as a Model.

Appointment()
Appointment(long id)
Appointment(User doctor, User patient, Date time)

Create new instance.

Parameters:

- id(long): identification of one appointment. If the value is -1 then it means that the set data will be a new data. This isn't in database, yet.
- doctor(User):: Object.User is a simple POJO object. This object contains the data of the doctor.
- patient(User):: Object.User is a simple POJO object. This object contains the data of the patient.
- time(Date): date of appointment when the user will meet her/his doctor.

AppointmentDao interface

This interface defines the standard operations to be performed on a model object(s).

saveAppointment(Appointment appointment): long

Insert new appointment into the database.

Parameters:

- appointment(Appointment): Object.Appointment is a simple POJO object.

Return:

This is a new row id.

updateAppointment(Appointment appointment): Boolean

Update one appointment row in the database.

Parameters:

- appointment(Appointment): Object.Appointment is a simple POJO object. This object contains the data which we have to modify.

Return:

It will return true for success otherwise it will return false.

deleteAppointment(Appointment appointment): Boolean

Delete one appointment row in the database.

Parameters:

- appointment(Appointment): Object.Appointment is a simple POJO object. This object contains the data which we have to delete.

Return:

It will return true for success otherwise it will return false.

getAppointment(long id): Appointment

Get one row from the database.

Parameters:

- id(long): the unique row identification of appointment

Return:

Object.Appointment is a simple POJO object.

getAppointments(User user, Date dateFrom, Date dateTo): List<Appointment>

Get more rows from database. These rows contain the appointment of the doctor between two dates.

Parameters:

- user(User): Object.User is a simple POJO object. This object contains the data of user whose appointment is necessary.
- dateFrom(Date): begin of filter date. If this is null then the begin date is open.
- dateTo(Date): end of filter date. If this is null then the end date is open.

Return:

Object.List object that it is contain Appointment objects.

ScheduleManager class

This class manages scheduling processes. Through the object we can add new appointment, change and check the appointment time that can be inserted under different conditions.

save(long id, long doctorId, long patientId, Date appointment, String description, List<String> notes): long

Add new appointment into the database or update an existing row.

Parameters:

- doctorId(long): identification of one doctor user.
- patientId(long): identification of one patient user.
- time(Date): date of appointment when the user will meet her/his doctor.
- description(String): The meeting details.
- notes(List<String>): short tags of appointment.

Return:

Inserted or updated row id.

reScheduleAppointment (long userId, Date srcDate, Date targetDate): boolean

An appointment is copied to another day.

Parameters:

- userId(long): Object.User is a simple POJO object. The user whose data we would like to copy.
- srcDate(Date): the day of copy
- targetDate(Date): target day

Return:

It will return true for success otherwise it will return false.

remove (long scheduleId): boolean

The appointment is deleted from the data source.

Parameters:

- scheduleId(long): It is a unique id of appointment in data source.

Return:

It will return true for success otherwise it will return false.

getAppointments (long userId): List<Appointment>

Get more rows from the data source. These rows contain the appointment of the doctor or the patient.

Parameters:

- userId (long): The user of id whose data we want to be queried.

Return:

Object.List object that it is contain Appointment objects.

getAppointments (long userId, Date from, Date to): List<Appointment>

Get more rows from the data source. These rows contain the appointment of the user between two dates.

Parameters:

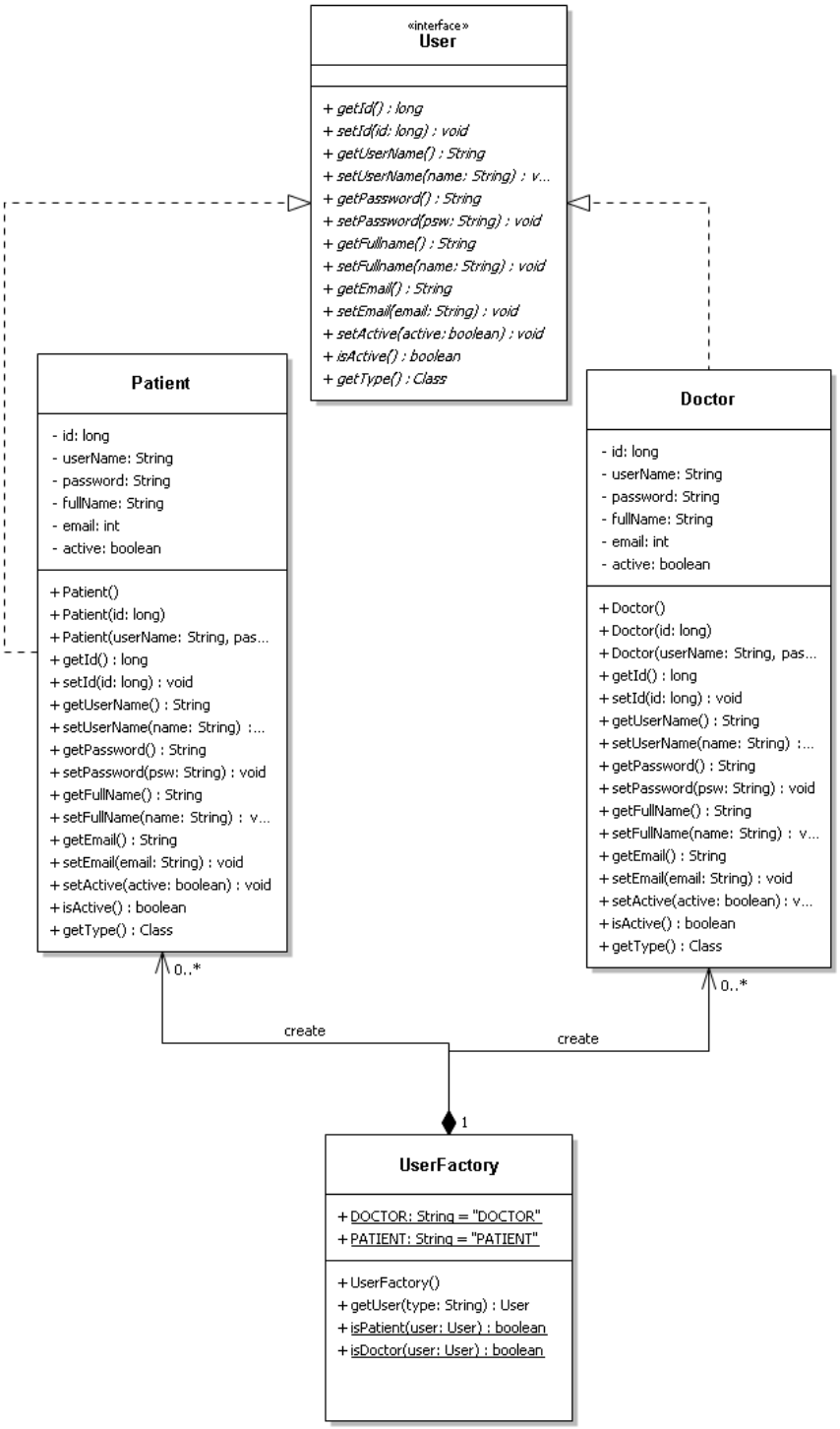
- userId (long): The user of id whose data we want to be queried.
- from(Date): begin of filter date. If this is null then the begin date is open.
- to(Date): end of filter date. If this is null then the end date is open.

Return:

Object.List object that it is contain Appointment objects.

User Factory design

We are going to create a *User* interface and the *Doctor* and *Patient* model classes implementing the *User* interface. Next step we create a factory class *UserFactory* to get a *User* object.



4. figure

User interface

This interface defines the standard operations to be performed on a Doctor or a Patient model object(s).

getType(): Class

This method returns the class type of Model object. So we can define that this user is a doctor or a patient.

Doctor class

This object is a simple POJO containing get/set methods. This object is going to work as a Model.

Doctor ()
Doctor (long id)
Doctor(String userName, String password, Boolean active)

Create new model instance.

Parameters:

- id(long): This is unique row id of user in database. If this id is -1 then this object doesn't contain data of new user. If the id is greater than -1 then this object contains data of an existing user.
- username(String): This is the user's name.
- password(String): This is the user's password.
- active(Boolean): This is a flag that the user's state is enable or disable

Patient class

This object is a simple POJO containing get/set methods. This object is going to work as a Model.

Patient ()
Patient (long id)
Patient(String userName, String password, Boolean active)

Create new model instance.

Parameters:

- id(long): Unique row id of user in database. If this id is -1 then this object doesn't contain data of new user. If the id is greater than -1 then this object contains data of an existing user.
- username(String): This is the user's name.
- password(String): This is the user's password.
- active(Boolean): This is a flag that the user's state is enable or disable

UserFactory class

This class create new instance from *User* or *Doctor* Model class without exposing the creation logic to the client.

getUser (String type): User

Create new model object from Doctor or Patient class.

Parameters:

- type(String): The type of object which we want to create. For the value we can use the static variables of class, too: UserFactory.DOCTOR or UserFactory.PATIENT

Return:

An instance of Doctor or Patient model class

isPatient (User user): boolean

Check the user is a patient.

Parameters:

- user(User): Object.User model class is a simple POJO object.

Return:

If the user is a patient then return true otherwise it will be false.

isDoctor (User user): boolean

Check the user is a doctor.

Parameters:

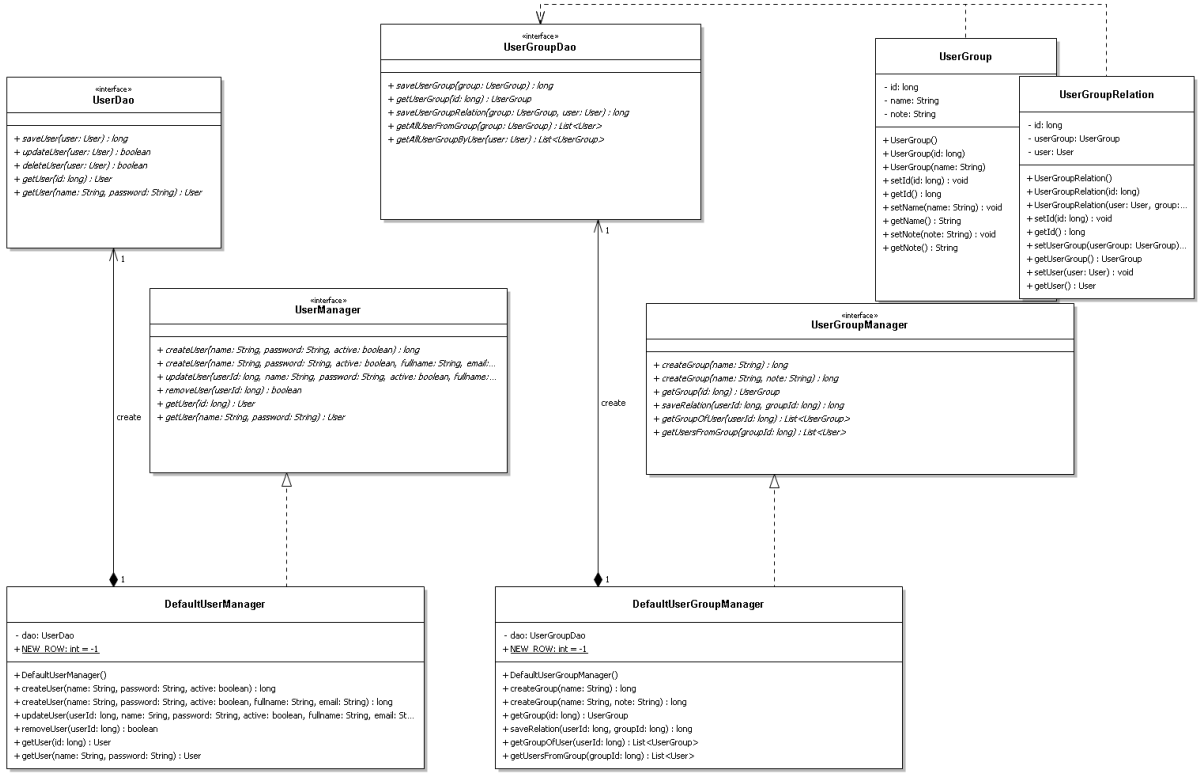
- user(User): Object.User model class is a simple POJO object.

Return:

If the user is a doctor then return true otherwise it will be false.

User and User Group data access objects

This chapter explains that how to handle the users and user groups through the objects. We are going to create a *UserDao* and *UserGroupDao* interfaces. These interfaces implement concrete classes. We are going to create two model classes (*UserGroup*, *UserGroupRelation*). Both classes will pass information to *UserGroupDao* object to get the data it needs. We have two Manager Classes (UserManager and UserGroupManager). Through these classes we can manage the user and user group data.



5. figure

UserDao interface

This interface defines the standard operations to be performed on a model object.

saveUser(User user): long

Insert new user into the database.

Parameters:

- user(User): Object.User is a simple POJO object(Patient or Doctor model object).

Return:

This is new row id.

updateUser(User user): Boolean

Update one row in the database.

Parameters:

- user(User): Object.User is a simple POJO object.

Return:

It will return true for success otherwise it will return false.

deleteUser(User user): Boolean

Delete one row in the database.

Parameters:

- user(User): Object.User is a simple POJO object.

Return:

It will return true for success otherwise it will return false.

getUser(long id): User

Get one row from the database.

Parameters:

- id(long): unique identification of the row

Return:

Object.User is a simple POJO object.

getUser(String name, String password): User

Get one row from the database. Both parameters together identify one user in the database.

Parameters:

- name(String): This is the user's name.
- password(String): This is the user's password.

Return:

Object.User is a simple POJO object.

UserGroupDao interface

This interface defines the standard operations to be performed on a model object.

saveUserGroup(UserGroup group): long

Insert new user group into the database.

Parameters:

- group(UserGroup): Object.User is a simple POJO object.

Return:

This is new row id.

getUserGroup(long id): UserGroup

Get a row from the database by id.

Parameters:

- id(Long): the unique value of a row in database.

Return:

Object.UserGroup is a simple POJO object.

saveUserGroupRelation(UserGroup group, User user): long

This method inserts a user into a new group.

Parameters:

- group(UserGroup): Object.UserGroup is a simple POJO object. This is a new group of user.
- user(User): Object.User is a simple POJO object. This user who will insert into the group.

Return:

It will return true for success otherwise it will return false.

getAllUserFromGroup(UserGroup group): List<User>

Return all users who are in this group.

Parameters:

- group(UserGroup): Object. UserGroup is a simple POJO object. This object contains the data of the group. Get a user list from this group.

Return:

Object.List object which contains User objects.

getAllUserGroupByUser(User user): List<UserGroup>

This method returns all groups of a user.

Parameters:

- user(User): Object. User is a simple POJO object. This object contains data of a user.

Return:

Object.List object which contains UserGroup objects.

UserGroup class

This object is a simple POJO containing get/set methods. This object is going to work as a Model and will pass information to other classes.

UserGroup ()
UserGroup (long id)
UserGroup (String name)

Create new model instance.

Parameters:

- id(long): This is a unique row id of the user group in database. If this id is -1 then this object doesn't contain data of new user. If the id is greater than -1 then this object contains data of an existing user.
- name(String): This is the group's name.

UserGroupRelation class

This object is a simple POJO containing get/set methods. This object is going to work as a Model and will pass information to other classes.

UserGroupRelation ()
UserGroupRelation (long id)
UserGroupRelation (User user, UserGroup userGroup)

Create new model instance.

Parameters:

- id(long): This is an unique row id of the user group relation in database. If this id is -1 then this object doesn't contain data of new user. If the id is greater than -1 then this object contains data of an existing user.
- user(User): This is a user POJO object.
- userGroup(UserGroup): This is a user group POJO object.

UserManager interface

This class manages all of the user's data. We can reach some operation functions through the class. We can create, update or delete one user's data, too.

createUser(String name, String password, boolean active)
createUser(String name, String password, boolean active, String fullname, String email)

Create new user in the database.

Parameters:

- name(String): This is the user's name.
- password(String): This is the user's password.
- active(Boolean): This is a flag that the user's state is enable or disable
- fullname(String): This is the user's full name.
- email(String): This is the user's email address.

Return:

It will return new row id for success otherwise it will return null.

updateUser(long userId, String name, String password, boolean active, String fullname, String email): Boolean

Update one row in the database

Parameters:

- userId(long): This is the unique user id. This id identifies one row in User table. We will update this row.
- name(String): This is the user's name.
- password(String): This is the user's password.
- active(Boolean): This is a flag that the user's state is enable or disable
- fullname(String): This is the user's full name.
- email(String): This is the user's email address.

Return:

It will return true for success otherwise it will return false.

removeUser(long userId): Boolean

Delete one row from the database.

Parameters:

- userId(long): This is the unique user id. This id identifies one row in User table. We will delete this row.

Return:

It will return true for success otherwise it will return false.

getUser(long id): User
getUser(String name, String password): User

Get one user's data from the database. The name and the password can identify one row, too.

Parameters:

- id(long): This is the unique user id. This id identifies one row in User table.
- name(String): This is the user's name.
- password(String): This is the user's password.

Return:

Object.User object is a simple POJO object. This method return null if the user isn't found in the database.

UserGroupManager interface

This class manages all of the data of a group. We can reach some operation functions through the class. We can create, update or delete one data of the group, too.

createGroup(String name)
createGroup(String name, String note)

Create new user group in the database.

Parameters:

- name(String): This is the name of group.
- note(String): This is a short description of group.

Return:

It will return new row id for success otherwise it will return null.

getGroup (long id): UserGroup

Get one data of group from the database.

Parameters:

- id(long): This is the unique group id. This id identifies one row in UserGroup table.

Return:

Object.UserGroup object is a simple POJO object. This method return null if the group isn't found in database.

saveRelation(long userId, long groupId): long

Create a relation between a group and a user.

Parameters:

- userId(long): This is the unique user id. This id identifies one row in User table.
- groupId(long): This is the unique group id. This id identifies one row in UserGroup table.

Return:

It will return new row id for success otherwise it will return null.

getGroupOfUser(long userId): List<UserGroup>

This method returns all groups of a user.

Parameters:

- userId(long): This is the unique user id. This id identifies one row in User table.

Return:

Object.List object which contains UserGroup objects.

geUsersFromGroup(long groupId): List<User>

Return all users who are in this group.

Parameters:

- groupId(long): This is the unique group id. This id identifies one row in UserGroup table.

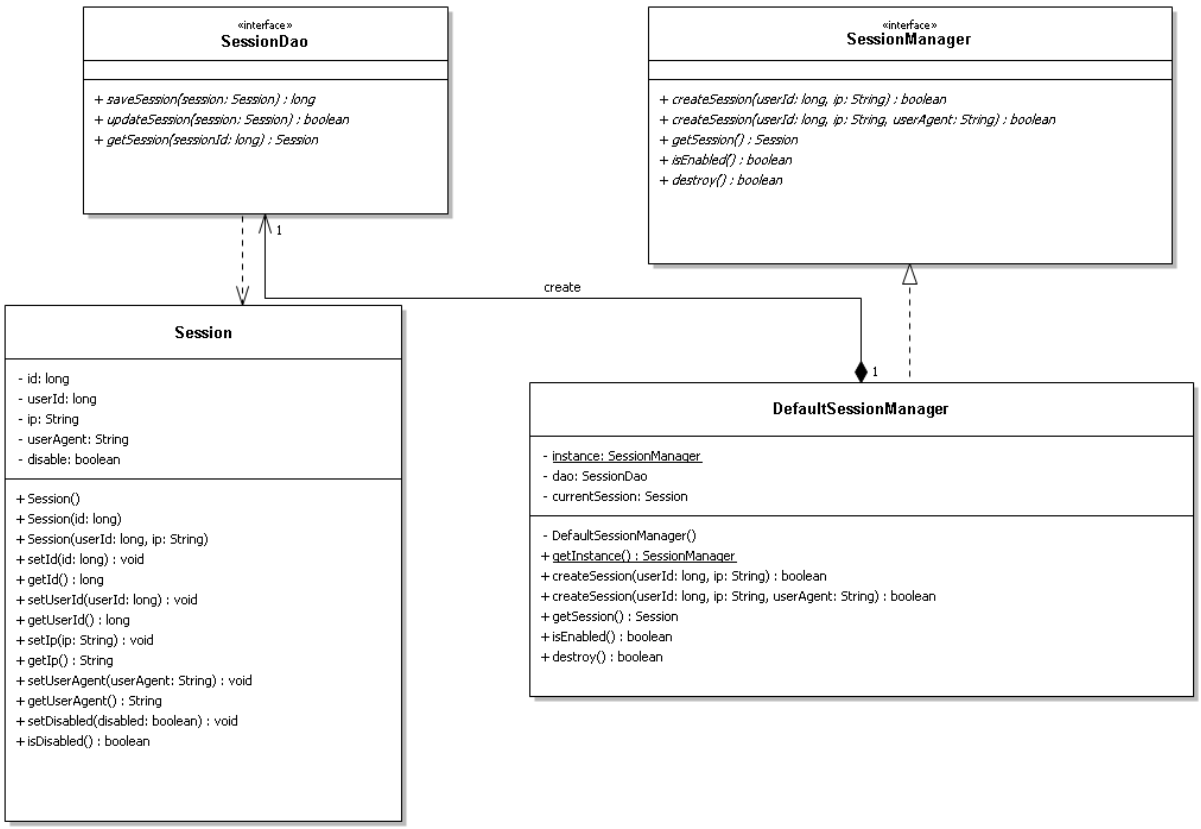
Return:

Object.List object which contains User objects.

Session manager

We are going to explain that how to handle the user's session. We create a *SessionDao* interface. This interface implements concrete classes. We are going to create a model class (*Session*). This class will pass information to *SessionDao* object to get the data it needs. We have a manager class (DefaultSessionManager).

Through this class we can manage the data of session. *DefaultSessionManager* class have a constructor as private and have a static parameter which contains an instance of itself. *DefaultSessionManager* class provides a static method to get the static instance to the outside world.



6. figure

SessionDao interface

This interface defines the standard operations to be performed on a model object.

saveSession(Session session): long

Insert new data of the session into the database.

Parameters:

- session(Session): Object.Session is a simple POJO object.

Return:

This is a new row id.

updateSession(Session session): Boolean

Update one row in the database.

Parameters:

- session(Session): Object.Session is a simple POJO object.

Return:

It will return true for success otherwise it will return false.

getSession(long sessionId): Session

Get one row from the database.

Parameters:

- sessionId(long): This is the unique identification of the row.

Return:

Object.Session is a simple POJO object.

Session class

This object is a simple POJO containing get/set methods. This object is going to work as a Model.

Session()
Session(long id)
Session(long userId, String ip)

Create new instance.

Parameters:

- id(long): identification of one session.
- userId(long): This is the unique user id. This id identifies one row in the User table.
- ip(String): ip address of user

SessinoManager interface

This class manages all of the data of the session. We can reach some operation functions through the class. We can create and check the data of the session, too.

createSession(long userId, String ip): boolean
createSession(long userId, String ip, String userAgent): boolean

Create new session data in the database.

Parameters:

- userId(long): This is the unique user id. This id identifies one row in the User table.
- ip(String): ip address of user
- userAgent(String): This is other user's data. This string has to contain data from the user pc or the browser.

Return:

It will return true for success otherwise it will return false.

getSession(): Session

Get current the data of session.

Return:

Object.Session is a simple POJO object.

isEnable(): Boolean

This method checks that the current session is active or not. If there isn't session then this method return false.

Return:

It will return true for success otherwise it will return false.

destroy(): Boolean

This method disable the current session.

Return:

It will return true for success otherwise it will return false.

Used technology

Project site: https://github.com/kuncy88/patient_management

Licenc: GNU GPL v3 (http://www.gnu.org/licenses/gpl.html)

Maven

I use Maven builder for the solution of the project. Maven is very powerful and it is simple to use. I split the project to 4 parts so later we can change the project more easily.

1. **dao:** This project contains the dao interfaces and their implementations.
2. **pojo:** This project contains the POJO classes.
3. **service-layer:** The service layer contains all of the manager classes. Through these classes the web layer and the data access layer can communicate with each other.
4. **gui:** This is the user interface.