Assignment 1

Simina Radu Alexandru

1. **Layered architecture**

For this assignment I have chosen to work under a layered architecture pattern. Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, my presentation layer is responsible for handling all user interface, whereas a business layer would be responsible for executing specific business rules associated with the request. One of the powerful features of the layered architecture pattern is the *separation of concerns* among components. Components within a specific layer deal only with logic that pertains to that layer. For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic. This type of component classification makes it easy to build effective roles and responsibility models into your architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.



In the case of my project, the presentation level consists in the **view** package. Here I have created all the GUI classes using java.swing. The only attributes of these classes are to offer a clean path of communication between the user and the software. There are multiple windows, all of them self-explanatory, with relevant labels, and pop-up messages.

The Business layer is represented in my project as the **controller**. In here, one can find all the actions performed when a certain input is received from the presentation layer (GUI). Also, inside this package, there have been made validations for the input data using pattern recognition (regex).

The persistence layer is situated in the **repository** package, where each entity has it`s specific operations. In this layer, I have established the connection with the database, created and executed queries, using ORM. Another pattern used here is the factory design pattern, in order to generate entity manager objects, that will handle the operations on the DB.

The other packages involved in the project are: **entity** which will contain the model of the objects used during the development of the application. The **service** package is used just as a wrap for the functionalities provided in **repository.** It will also contain small validations, and some null checking. The **start** package has just the main method, which will run the application. In the **utilities** package, we have two classes, one responsible for composing and sending emails, and the other one having stored some relevant text messages that are being used all over the application, especially while popping up message windows in the graphical interface. The **validators** package will have three classes, responsible for validating the input that is received from the GUI. This action is performed using regular expression pattern matching, ex:

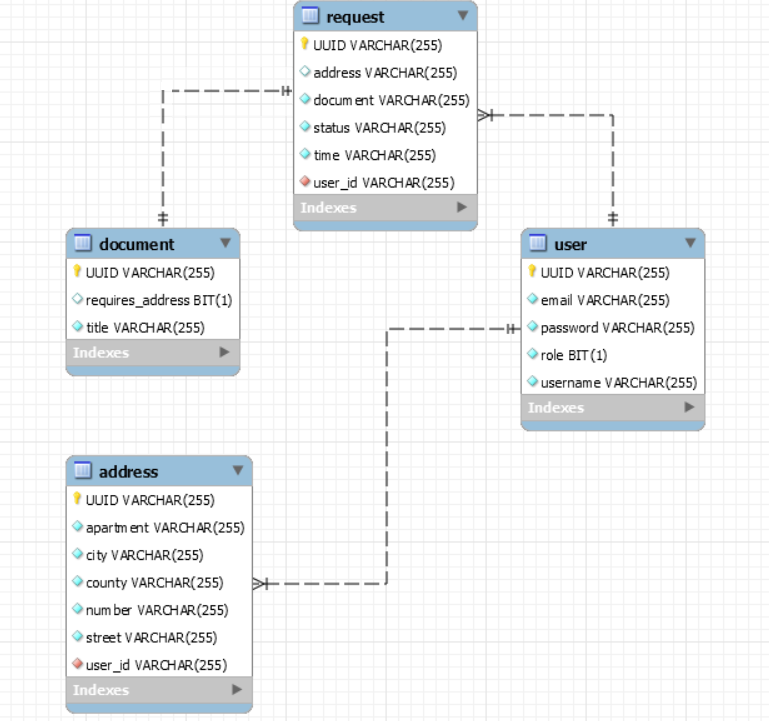
"^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[#?!@$%^&\*-]).{8,}$"

The above regex will validate only passwords that contain lower and upper case letters, at least one digit and one special character from the list: “#?!@$%^&\*-”, and also, the password must be at least 8 characters long. Resembling conditions were set for the username as well, while, for the email validation, I have used the **RFC 5322 Official Standard** which is one of the most frequently used regex for email recognition. To be noted that the success percentage is 99,99% since there is no perfect email matching regular expression.

1. **The database**

The database layer. Our database schema, was created locally, manually, using the MySQL workbench application. After that, all the modifications that are brought to the database are made through the Java code, including creating table, inserting values into the table, deleting, updating and so on. However, there is one exception. We had, indeed, to create a manual query and insert it directly into the database via MySQL workbench, and the reason for that was to add an administrator. Since we do not want to offer any user the option to activate his account as an admin, we chose not to offer at all the possibility of adding new administrators through the graphical interface.

Our database model is composed from mainly 4 individual tables, each representing a different entity. The first table, is the **user** table, which is represented by the **User** class. Each user will have its unique ID, represented as an UUID, a username, which will be again unique, an email address, a password, and a - hidden from the user – feature, his role. By default, his role will be set to Boolean.FALSE, which will mean that he is a regular user, not an admin. This table will have a relation of @OneToMany with the next 2 tables: the **address** table, and the **request** table. Also, each request will have an assigned document, which will be selected from the **document** table. As I have considered the application, there is an interesting relation between the **document** and the **address** table. Every time a user will try to submit a request, he will be asked to provide the title of the document that he would like, hence the 1:1 relationship between **request** and **document**. After that, if the document that he desires requires an address (construction authorization, property proof etc.) the user will have to provide the address that the document will be related to. There can be documents that will not require an address (Marriage officiating request etc.), for those the address related to the document will be null.



1. **Specific MySQL queries:**

Some of these queries are hand written, and others will be copy-pasted from the

console, given as output by Hibernate and the JPA framework.

1. **Creating the tables:**

-- -----------------------------------------------------

-- Table `labsd`.`user`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `labsd`.`user` (

`UUID` VARCHAR(255) NOT NULL,

`email` VARCHAR(255) NOT NULL,

`password` VARCHAR(255) NOT NULL,

`role` BIT(1) NOT NULL,

`username` VARCHAR(255) NOT NULL,

PRIMARY KEY (`UUID`),

UNIQUE INDEX `UK\_sb8bbouer5wak8vyiiy4pf2bx` (`username` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4;

-- -----------------------------------------------------

-- Table `labsd`.`address`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `labsd`.`address` (

`UUID` VARCHAR(255) NOT NULL,

`apartment` VARCHAR(255) NOT NULL,

`city` VARCHAR(255) NOT NULL,

`county` VARCHAR(255) NOT NULL,

`number` VARCHAR(255) NOT NULL,

`street` VARCHAR(255) NOT NULL,

`user\_id` VARCHAR(255) NOT NULL,

PRIMARY KEY (`UUID`),

INDEX `FKda8tuywtf0gb6sedwk7la1pgi` (`user\_id` ASC) VISIBLE,

CONSTRAINT `FKda8tuywtf0gb6sedwk7la1pgi`

FOREIGN KEY (`user\_id`)

REFERENCES `labsd`.`user` (`UUID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4;

-- -----------------------------------------------------

-- Table `labsd`.`document`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `labsd`.`document` (

`UUID` VARCHAR(255) NOT NULL,

`requires\_address` BIT(1) NULL DEFAULT NULL,

`title` VARCHAR(255) NOT NULL,

PRIMARY KEY (`UUID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4;

-- -----------------------------------------------------

-- Table `labsd`.`request`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `labsd`.`request` (

`UUID` VARCHAR(255) NOT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`document` VARCHAR(255) NOT NULL,

`status` VARCHAR(255) NOT NULL,

`time` VARCHAR(255) NOT NULL,

`user\_id` VARCHAR(255) NOT NULL,

PRIMARY KEY (`UUID`),

INDEX `FKqws2fdeknk90txm7qnm9bxd07` (`user\_id` ASC) VISIBLE,

CONSTRAINT `FKqws2fdeknk90txm7qnm9bxd07`

FOREIGN KEY (`user\_id`)

REFERENCES `labsd`.`user` (`UUID`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4;

1. **Inserting into a table:**

Hibernate: insert into address (apartment, city, county, number, street, user\_id, UUID) values (?, ?, ?, ?, ?, ?, ?)

* In this case we have “?” because Hibernate will generate the form of the query, the

Values will be added afterwards, before executing the actual query.

Hibernate: insert into request (address, document, status, time, user\_id, UUID) values (?, ?, ?, ?, ?, ?)

Hibernate: insert into document (requires\_address, title, UUID) values (?, ?, ?)

1. **Deleting from a table:**

Hibernate: delete from document where title=?

Hibernate: delete from request where time=?

1. **Updating a table entry:**

Hibernate: update request set address=?, document=?, status=?, time=?, user\_id=? where UUID=?

Hibernate: update user set email=?, password=?, role=?, username=? where UUID=?