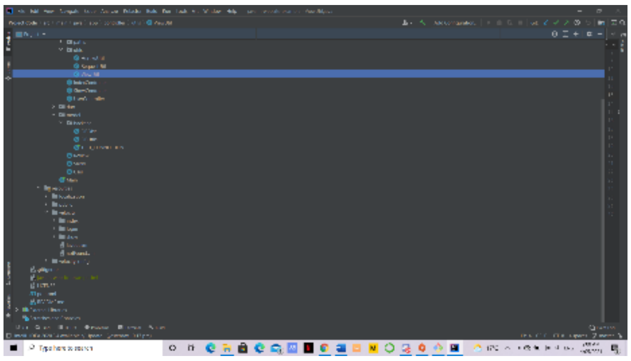
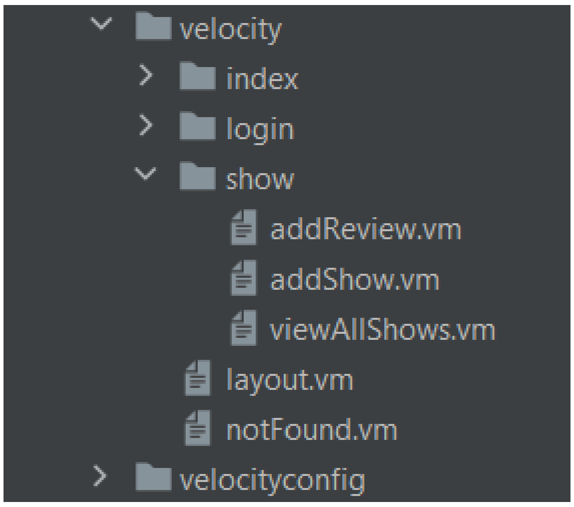
Task 2.3B

The architectural patterns that we implemented are n-Tiers -> 3- Tier Architecture, DAO & DTO and MVC: Model-View-Controller.

The implementation of our code is a perfect example of 3- Tier Architecture because it has got 3 layers- data, application and the presentation layers. The front-end package represents the presentation layer which is connected to the database (data layer) via the back-end programs (application layer). This architecture helps with much greater flexibility since we, as a developer, have the freedom to update any part of an application without affecting the other as they are independent of each other.

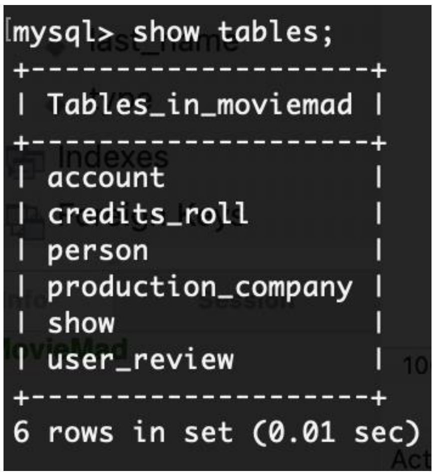


This is the overview of all the classes.



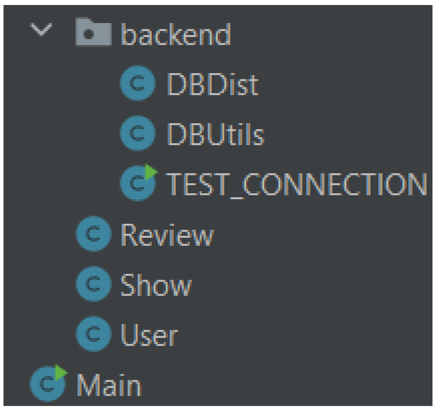
```
velocity
    index
    login
    show
        addReview.vm
        addShow.vm
        viewAllShows.vm
    layout.vm
    notFound.vm
velocityconfig
```

This is the presentation layer.



```
mysql> show tables;
+--------------------+
| Tables_in_moviemad |
+--------------------+
| account            |
| credits_roll       |
| person             |
| production_company |
| show               |
| user_review        |
+--------------------+
6 rows in set (0.01 sec)
```

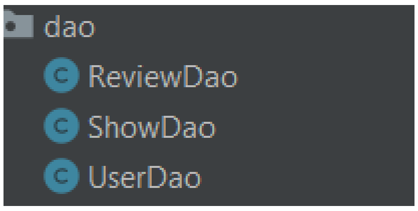
This is the Data Layer and shows all the tables in the MOVIEMAD database.



```
backend
    DBDist
    DBUtils
    TEST_CONNECTION
  Review
  Show
  User
Main
```

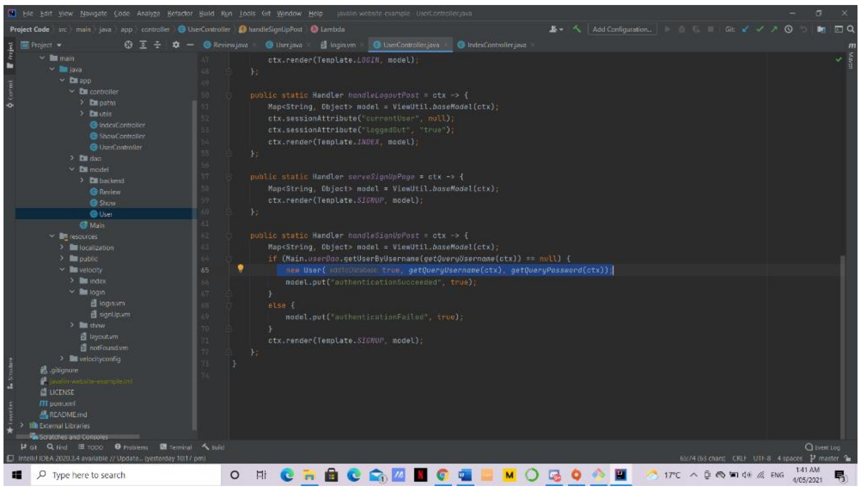
These are the objects in the application layer.

Our project follows a pattern similar to DAO & DTO to access and move data between application and view classes and the database. Data from database tables are fetched/manipulated using DAO's like the DBUtils object and moved around in the system using DTO/DAO's like Show, User and Review objects and controller objects. Data from the tables in the database like the details of the movies, users and user specific data are connected and encapsulated by the aforementioned classes and objects which move data back and forth between the presentation, application and data layers. The contents of the below classes are the certain cases of the scenario occurring as stated.



```
dao
  ReviewDao
  ShowDao
  UserDao
```

It is also seen that our project has all the functionality of MVC model. Our project implements a Model, View & Controller which are all interconnected as they are supposed to be. Any user actions are conveyed to the controller. The Controller is then in charge of updating the Model, & any requests are handled both ways between the Model & Controller. Next, the Model makes requests in a two-way connection with the View. Notifications are shown in the View with a two-way connection with the Controller. Finally, the View displays any updates to the User.



```java
        ctx.render(Template.LOGIN, model);
    };

    public static Handler handleLogoutPost = ctx -> {
        Map<String, Object> model = ViewUtil.baseModel(ctx);
        ctx.sessionAttribute("currentUser", null);
        ctx.sessionAttribute("loggedOut", "true");
        ctx.render(Template.INDEX, model);
    };

    public static Handler serveSignUpPage = ctx -> {
        Map<String, Object> model = ViewUtil.baseModel(ctx);
        ctx.render(Template.SIGNUP, model);
    };

    public static Handler handleSignUpPost = ctx -> {
        Map<String, Object> model = ViewUtil.baseModel(ctx);
        if (Main.userDao.getUserByUsername(getQueryUsername(ctx)) == null) {
            new User( addToDatabase: true, getQueryUsername(ctx), getQueryPassword(ctx));
            model.put("authenticationSucceeded", true);
        }
        else {
            model.put("authenticationFailed", true);
        }
        ctx.render(Template.SIGNUP, model);
    };
}
```

For example, UserController.java is, as the name suggests a Controller in our application. In the above screenshot, during a user registration/signup process, this controller fetches user details like username and password from a View component (user registration page), checks if a user with the same username already exists, with a DAO. If a duplicate is not found in the database, it then proceeds to create an instance of a Model (User class) to hold the user information from the view. The model or the User object then updates/inserts information in the database which will again be displayed using another view (login.vm).

We did not choose 2-tiers architecture because in terms of functionality, the performance tends to decrease with the increment of users and is more costly to operate. But the main problem is that multiple requests cannot be responded by the server at a same time which can be done with 3 trier architecture and with the help of 3 tier architecture, we can update any part without affecting the other.