Maintenance Document:

General architecture description:

Our project is built out of 2 main parts:

The logical &database server and the client-side server.

The logic and data server does what its name hits, he stores all the data, and use the logic of the system in order to create/ rewrite / transfer/ delete the data.

The client-side server is only responsible about the visualizing of the system and the creation of the Single Page Web Application in web pages, meaning the user interface of the system. The client-side server creates a basic page for users to log in with and then he transfers data to the first server, gets back data and responds by creating the right web page.

Basic understanding for extensions and future changes:

In order to change the logic of the system or the way the data is stored whoever wishes to do that shall go to the logic and data server.

If you wish to change the user interface of the system or change the way the system asks for input from the user, this is related to the client-side server.

Class description of the logic and data server:

The logic and data server are built out of layers -

First, the database, we used MySQL as our database management system. In the resources package we can find files related to the database, the create_database.sql file which creates the tables of the database, and the application.properties which is responsible to establish connection with the MySQL database. Another class related to the database is the DBAccess, by using the repositories we are going to explain below, the DBAccess has the functions that interact with the database (functions like saving experiments or fetching experiments and so on). If new database queries are added to the repositories, the DBAccess is the place to create functions that will invoke those queries.

Next, we have the Data Access Layer, in the DataAccessLayer package, for each table in the database we have a repository which uses JPARepository to get basic functions that query the database (functions like save, delete, find and so on), in those repositories, new functions that queries the database can be easily created using the JPARepository syntax.

Next, we have the BusinessLayer, this package contains several packages and classes. The Entities package which stores the objects mapped from the database tables (ORM), for each table we have its own entity (class), each entity is created using Hibernate with Java Annotations. Note that the database constraints and the annotations are strongly connected so changes on either side (database or entities must come together). The Exceptions package has some self-implemented Exceptions to have better understanding when an exception occurs. The DataCache class which maintains lists of entities to be fetched quickly when needed, saves time by causing less interactions with the database, this class also interacts with the database when needed to refresh data. Adding or removing cached data must be done here. The CreatorBusiness, ExperimenteeBuisness, GraderBuisness and their corresponding interfaces, those classes implement the use cases of our system such as creating experiments, filling in experiments and so on. Changes in the use cases and their implementation must be done in those classes.

Next, we have the ServiceLayer, in this package we have the CreatorService, ExperimenteeService, GraderService, those classes are responsible to interact with the business layer to receive data for given queries and are also responsible to return the data in the right format (the format we agreed with the client-side team). Changes in the way data is transferred from the server-side to the client-side must be done here.

Next, we have the RoutingLayer, the classes in there are the first to receive queries from the client-side, they are implemented with Rest API, and a client-side query will be mapped directly to one of the functions in those classes depending on the action preformed in the client-side. In the package we have the MangerController which is responsible for queries coming from Management Users, users that can create experiments, we have the EnviormentController which is responsible for queries coming from experimentees (like filling stages), we have the GraderController which is responsible for queries coming from the graders. Changes with the parameters passed between the client-side and the server-side for each action must be done here.

As for the test package, in this package we have the tests for each of the layers explained above, The PersistenceTests class is testing the interaction with the database for each of the entities we have. As mentioned above, new queries can be created in the future so this class is the place to test them.

The UnitTests package contains unit tests for the entity's functionality, any added functionality for the entities can be tests in there.

The BuisnessLayerTests package is testing the business classes functionality as well as the data cache functionality, any changes made to those classes can be tested here.

The ServiceLayerTests package is testing the service classes functionality, any changes made to those classes can be tested here.

The Utils class has functionality that is used in many tests like building experiments, answering them, creating new stages and so on.

Class description of the client-side server:

Before the description of the classes in the client-side server we first go throw the idea of Angular: our client-side server is built with angular framework, which is a framework for developing SPA web application. Now in Angular everything is divided to components.

Our client-side component is built from 2 different separate components, and each one is a different system:

- The management system component is related to the management experiment system.
- The experiment environment is related to the experiment environment, the actual running participating in an experiment, web pages for the experiment participants.

Afterwards every page has a component and we switch between pages, we move data as parameters between components and output of a component sometimes.

Every visual thing you would like to change in the system is in the components, find the page, find the component, and then fund the right place in the html/css/ javascipt file of the component.

If it is what you see- it is the html of the page.

If it is how you see it- it is the design, the css.

If it is the page behavior- it is the javascript.

Some examples for future improvements or changes:

Adding a permission order to users:

In a scenario where you want to add more depth into the system users and create more than just admin users that creates the experiments and the participants who preform them, you want to add let's say an observer, some one who can watch all of the experiments and their results but can not participate in one but also can not create one.

How do you do it?

- Add the entity to the data base with the data layer in the logical and data server.
- Add the logical business that each experiment has its own list of observers.
- Support the requests of viewing experiment or its result if and only if the user has observer permissions to this experiment.
- Add the route of this kind of requests (observer related) in the routing layer.
- Parse the request in the service layer, get the actual data not an object and transfer it into the right logical function that will check if the permission exists for the user and if yes will get the experiments data.
- Now the logical and data server is ready, we need to add support in the UI.
- Add another component in the app folder of the client-side server and call it observer page.
- In this component create a simple html page with login option similar to the one we already created in the management system.
- Preform login functionalities and send the login credentials the user gives as input to an observer login request and send it to the logical and data server.
- Get an answer from the server and if positive move to a new component after the login, if not present message "failed to login" similar to the one we already created in the management system.
- Create another component, a kind of homepage for observers where they can input the experiments code and if they have permissions, see the experiment and it's results.
- Send the id the user gives as input to the logical and data server as a check permission request, and if positive move to a new component, , if not present message "no permission to view this experiment" similar to the one we already created in the management system.
- If the request was successful move to a new component where the user can choose between all of the results of the experiment and the raw details of the experiment.
- If the user clicks on the results, he views the results like we presented it to the participants after they finished stages in the experiment system.
- If he clicks on the raw experiment details, he views the experiment details, just as like we did in the management system where the admin can view his own experiments.
- And that’s it.