Experimentation system – Design Document

Gilad Chen 207168568, Gili Chiko 209296326, Sean Naor 206911315, Shachar Morad 206781080

Abstract

This document will specify the design of our experimentation system. We will expand on the requirements document and specify the design of the system in detail.

Contents

Chapter 1 - Use Cases

Use case 1 - Creating an experiment

Use case name: Create an experiment.
Actors: User (experiment creator).
Parameters: Experiment's data.
Preconditions: The user is registered to the experiment management.
Postconditions: The new experiment is made and saved in the database.
Description:

- The user will log in to the management system and choose to create new experiment
- The user will create the experiments stages(UC1.1)

now experiment was added but not yet published, to publish the experiment user need to do the following steps:

- The user will create grading tasks and assign graders for them(UC1.2)
- The user will add experimentees to the grading tasks, and will receive an access code for the experimentees to use
- The user can add allies (users with access to the experiment) and defines their permissions (UC1.3)

Acceptance testing:
Positive:

- Init: -
- Actions: user will enter his username and password, and log in
- Inspection: will be transferred to the user experiments page

Positive:

- Init: - user logged in and choose to create new experiment
- Actions: user will add experimentee by Email
- Inspection: will receive an access code

Negative:

- Init: -
- Actions: user enter wrong username/password
- Inspection: user will get an error message

Negative:

- Init: user logged in and choose to create new experiment
- Actions: user will add experimentee by invalid Email
- Inspection: user will get an error message

Use case 1.1 – create the experiment's stages

Use case name: Create the experiment's stages.
Actors: User (experiment creator).
Parameters: Experiment's stages.

Preconditions: The user is registered to the experiment management.
Postconditions: The new experiment stages are made and saved in the database.
Description:

- The user will start creating the stages from a template and can add stages as desire
- For each stage the user will enter the needed data
- For each additional stage the user will choose its type and create it. (UC1.1.*)
- When done, the user can choose to finish the creation of the experiment stages

Acceptance testing:

Positive:

- Init: -
- Actions: user will create all of the template stages, and finish the creation
- Inspection: all stages was add to the experiment

Negative:

- Init: -
- Actions: user will not create all of the template stages, and finish the creation
- Inspection: user will get an error message

Use case 1.2 – create a grading task

Use case name: Create a grading task.
Actors: User (experiment creator).
Parameters: personal's stages, evaluation's stages, graders, experimentees.
Preconditions: The user is registered to the experiment management.
Postconditions: The new grading task is made and saved in the database.
Description:

- The user will start creating stages for a personal experiment(UC1.1)
- The user will start creating stages for an evaluating experiment(UC1.1)
- The user will add experimentees to the grading task
- The user will add graders to the grading task, and will receive an access code for the graders usage
- For each grader, the user assigns experimentees manually

Acceptance testing:

Positive:

- Init: -
- Actions: user will enter personal's stages, evaluation's stages, graders, experimentees and finish the creation
- Inspection: the grading task was added to the experiment

Negative:

- Init: -
- Actions: user will not enter evaluation's stages and finish the creation
- Inspection: user will get an error message

Use case 1.3 – add allies

Use case name: Add allies
Actors: User (experiment creator).

Parameters: ally, permissions.
Preconditions: The user is registered to the experiment management.
Postconditions: The ally can now access the experiment and use his permissions.
Description:

- The user will enter the Email of the ally
- The system will display a set of permissions to choose for the ally and a set of his existing permissions if exist
- the user will add/remove permissions for the ally

Acceptance testing:

Positive:

- Init: -
- Actions: user will enter the ally's Email and choose permissions to add/remove
- Inspection: the ally's permissions are updated

Negative:

- Init: -
- Actions: user will not enter the ally's Email
- Inspection: user will get an error message

Use case 1.1.1 - Create a questionnaire stage

Use case name: Create a questionnaire stage
Actors: User (experiment creator)
Parameters: questions
Preconditions: The user is registered to the experiment management.
Postconditions: the stage is generated and added to the experiment.
Description:

- The system will present types of questions that can be added to the questionnaire.
- For each question in the stage:
  - The user will choose the question's type, and will input the appropriate data for said question.
- After finishing the creation of the stage, the user will save the new stage and will be directed to the stages creation

Acceptance testing:

Positive:

- Init: -
- Actions: user will enter questions and finish the creation
- Inspection: the new stage was added to the experiment's stages

Negative:

- Init: -
- Actions: user will not enter questions and finish the creation
- Inspection: user will get an error message

Use case 1.1.2 - Create an information stage

Use case name: Create an information stage
Actors: User (experiment creator)
Parameters: Information
Preconditions: The user is registered to the experiment management.
Postconditions: the stage is generated and added to the experiment.
Description:

- The user provides text, information and explanations to be displayed on the information page.

- After finishing the creation of the stage, the user will save the new stage and will be directed to the stages creation

Acceptance testing:

Positive:

- Init: -
- Actions: user will enter the info and finish the creation
- Inspection: the new stage was added to the experiment's stages

Negative:

- Init: -
- Actions: user will not enter info and finish the creation
- Inspection: user will get an error message

Use case 1.1.3 - Create a coding stage

Use case name: Create a coding stage
Actors: User (experiment creator)
Parameters: description, requirements, programming language, base code
Preconditions: The user is registered to the experiment management.
Postconditions: the stage is generated and added to the experiment.
Description:

- The user will insert task description
- The user will insert a list of requirements
- The user chooses a programming language from a given set of languages the system supports and optionally base code for the experimentee to start with.
- After finishing the creation of the stage, the user will save the new stage and will be directed to the stages creation

Acceptance testing:

Positive:

- Init: -
- Actions: user will enter description, requirements, programming language and base code and finish the creation
- Inspection: the new stage was added to the experiment's stages

Negative:

- Init: -
- Actions: user will not enter requirements/programming language and finish the creation
- Inspection: user will get an error message

Use case 1.1.4 - Create a statistics stage

Use case name: Create a statistics stage
Actors: User (experiment creator)
Parameters: statistics
Preconditions: The user is registered to the experiment management and had created
a coding stage.
Postconditions: the stage is generated and added to the experiment.
Description:

- The user chooses statistics criterias from a given set of statistics the system supports.
- After finishing the creation of the stage, the user will save the new stage and will be directed back to the stages creation

Acceptance testing:

Positive:

- Init: -

- Actions: user will choose statistics and finish the creation
- Inspection: the new stage was added to the experiment's stages

  Negative:
- Init: -
- Actions: user will choose no statistics and finish the creation
- Inspection: user will get an error message

Use case 2 - participation in an experiment

Use case name: participation in an experiment
Actors: User (experimentee)
Parameters: Access code
Preconditions:
Postconditions:
Description:

- The user will begin participation in an experiment(2.1)
- The user will fill in the experiment(2.2)
- The system will transfer the user to thank you page

Use case 2.1 - Beginning participation in an experiment

Use case name: Begin participation in an experiment
Actors: User
Parameters: Access code
Preconditions:
Postconditions:
Description:

- The user will insert his access code
- If the access code is valid, the system will begin participation in the experiment for said user
- Else, the system will alert that the code is invalid

Acceptance testing:

Positive:

- Init: There's an experiment in the system and the user got a valid access code.
- Actions: The user would enter the access code.
- Inspection: User was transferred to the appropriate experiment.

Negative:

- Init: There's an experiment in the system / no experiment in the system
- Actions: The user would enter a random string.
- Inspection: The user should get an error message.

Use case 2.2- Fill in Experiment

Use case name: fill in experiment
Actors: user (experimentee)
Parameters:
Preconditions: The user successfully logged in
Postconditions:
Description:

- for each stage of the experiment:

- the user will fill in the stage(2.2.*)
- when no more stages left, the system will display a thank you page and the user can now exit the system

Acceptance testing (also covers 2.2.*):

Positive:

- Init: a user was transferred to an experiment.
- Actions: for each stage (that required input), the user will enter some data, and move to the net stage.
- Inspection: system would display a thank you page.

Negative:

- Init: a user was transferred to an experiment.
- Actions: for each stage (that required input), the user will try moving to the next stage without entering any data (or some data).
- Inspection: system would ask him to enter data to the missing spots.

Use case 2.2.1- Fill in Questionnaire Stage

Use case name: fill in questionnaire stage
Actors: user (experimentee)
Parameters: answers
Preconditions: The user has completed all previous stages
Postconditions: the user's data has been transferred to the server, and current stage is no longer accessible for him
Description:

- the system will display the stage questions
- for each question in the stage:
- the user will answer the question to his understanding
- when no more questions left, the user will press next and will finish the stage

Use case 2.2.2- Fill in Coding Stage

Use case name: fill in coding stage
Actors: user (experimentee)
Parameters: coding stage
Preconditions: The user has completed all previous stages
Postconditions: the user's data has been transferred to the server, and current stage is no longer accessible for him
Description:

- the system will display a programming task which can be viewed at any moment in the stage
- the user will complete the task to his understanding
- the user will press next and will finish the stage

Use case 2.2.3- Fill in Tagging Stage

Use case name: fill in tagging stage
Actors: user (experimentee)
Parameters: tagging stage, code
Preconditions: the user has completed the programming task
Postconditions: the user's data has been transferred to the server, and current stage is no longer accessible for him
Descriptions:

- the system will display the programming task and the user's code
- for each requirement in the programming task:
  - the system will ask the user to tag the requirement to a segment in the code
  - The user will tag the appropriate code segment, to his understanding
- when no more requirements left, the user will press next and will finish the stage

Use Case 3 fill in grading task

Use case name: fill in grading task
Actors: Grader
Parameters: Access code
Preconditions:
Postconditions:
Description:

- The grader will login to grading task (UC 3.1)
- The grader will fill in the personal experiment, only once(UC2.2)
- For any experimentee result, the grader will fill in the evaluation experiment(UC2.2)
- at any point the grader can choose to save all the results he graded and leave the system

Use Case 3.1-

Use case name: login to grading task
Actors: Grader
Parameters: Access code
Preconditions:
Postconditions:
Description:

- The grader will insert his access code of a specific grading task
- If the access code is valid, the system will begin participation in the experiment page as a grader.
- Else, the system will alert that the code is invalid

Acceptance testing:

Positive:

- Init: There's an experiment in the system and the grader got a valid access code.
- Actions: The grader would enter the access code.
- Inspection: Grader was transferred to the appropriate experiment page for graders.

Negative:

- Init: The grader was transferred to the wrong experiment / there are no experiments in the system
- Actions: The grader entered an invalid access code.
- Inspection: The grader should get an error message.

Chapter 2 - System Architecture

As described in the ARD, the system consists of two sub systems: the experiment management and the experiment environment. Therefore, there are two types of

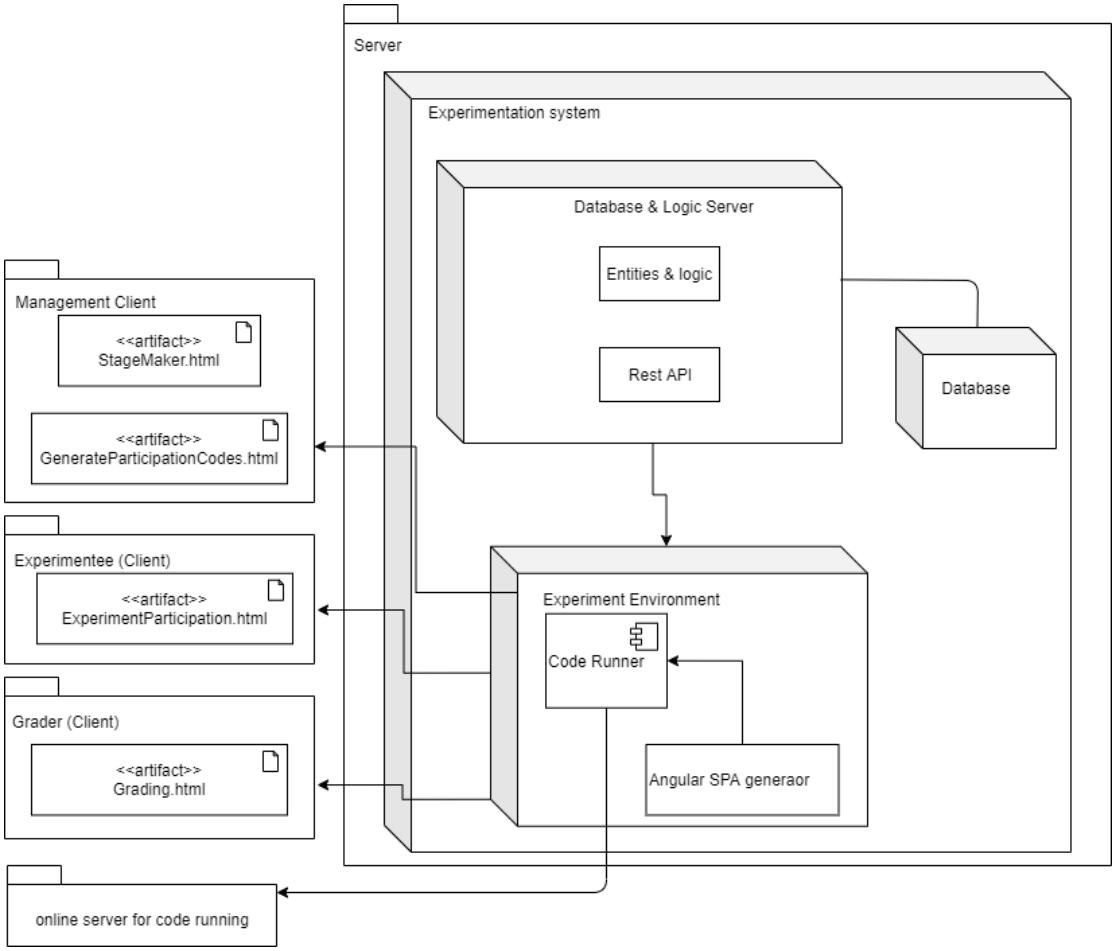
clients in our client-server system. The environment's client is split into two clients: the experimentee and the grader. These two sub-systems share access to the local database.

The management's client, the management user, logs in to the system using the BGU username and password. He can then manage existing experiments or make a new experiment, interfacing with the experiment management component. The experiment management processes every stage created by the client, merges the stages into the whole experiment and updates the database accordingly. For an existing experiment, the client can get access codes for experimentees using the ID code generator component. This component generates access codes for experimentees and makes it available in said experiment.

The environment's first client, the experimentee, logs in using his access code and begins participation in an experiment. Upon reaching the coding stage, he interfaces with the code runner. This component is responsible for compiling the code in the given language using outsourced compilers, running it and returning the output to the experimentee.

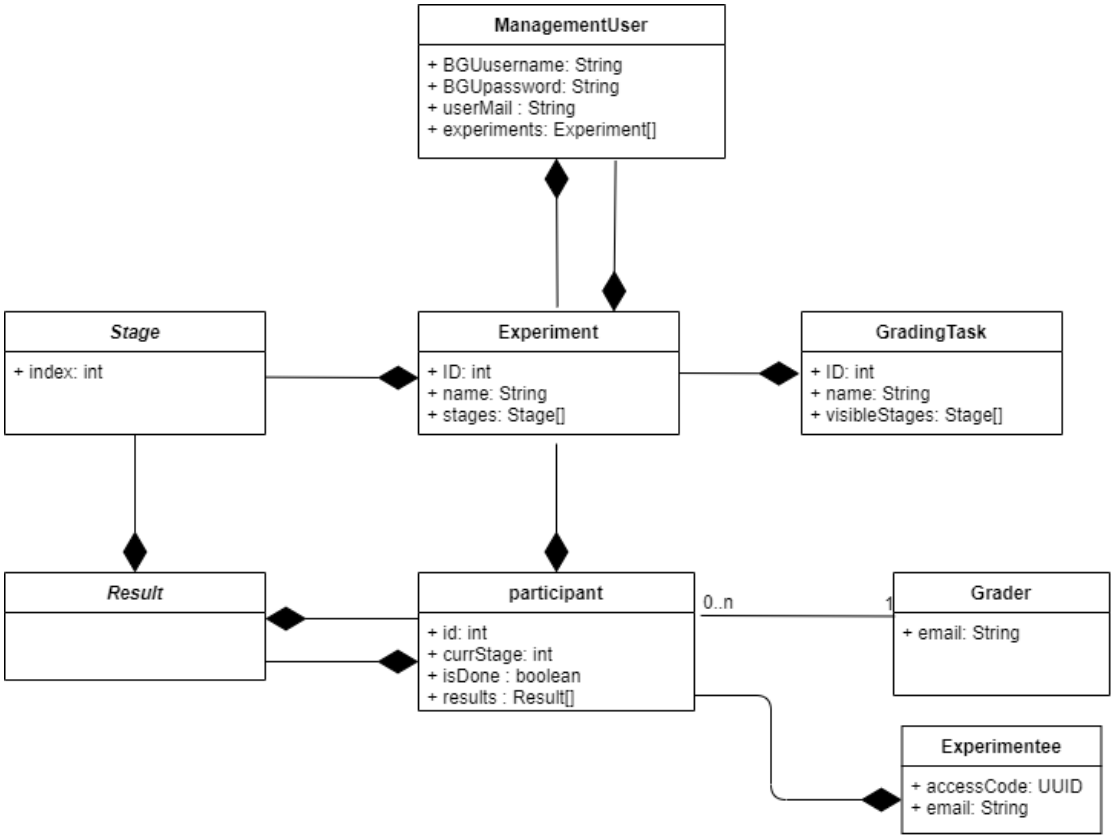
The environment's second client, the grader, logs in using his access code and can then grade results for the given experiment. The section for graders in the experiment environment displays to the grader the ungraded results. Upon selecting an experimentee that participated and completed an experiment, the grader can input his grade by participating in an "evaluation experiment", which is the grading of the experimentee's results. The grader also participates in a general experiment, which contains general instructions and questions for a grader to be answered once per grading task. The grader can also view and edit evaluation experiments until he submits them for the reviewing of the experiment managers.

Deployment diagram for the architecture:

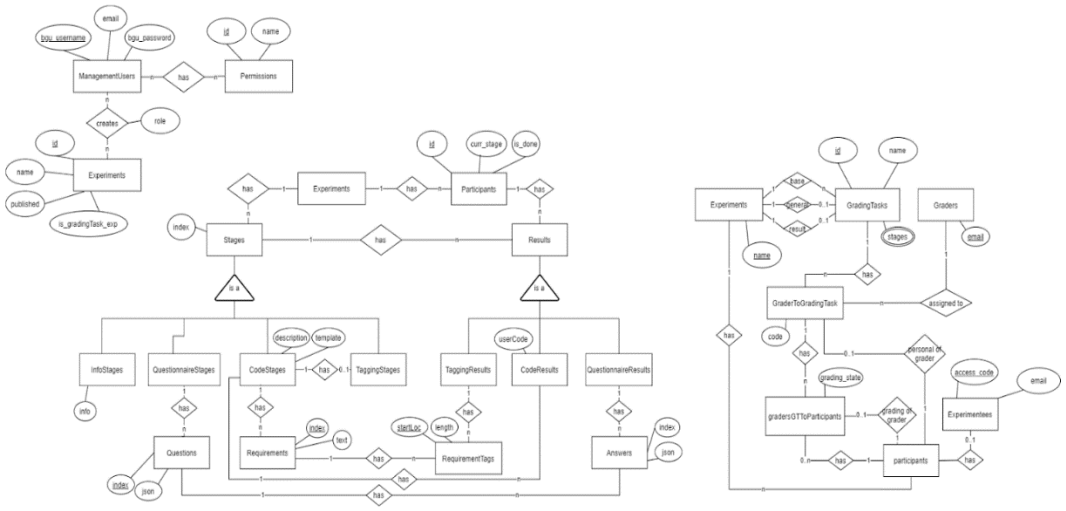


Chapter 3 - Data Model

3.1, 3.2 - Description of the main data objects and relationships



3.3 Databases



Main Transactions:

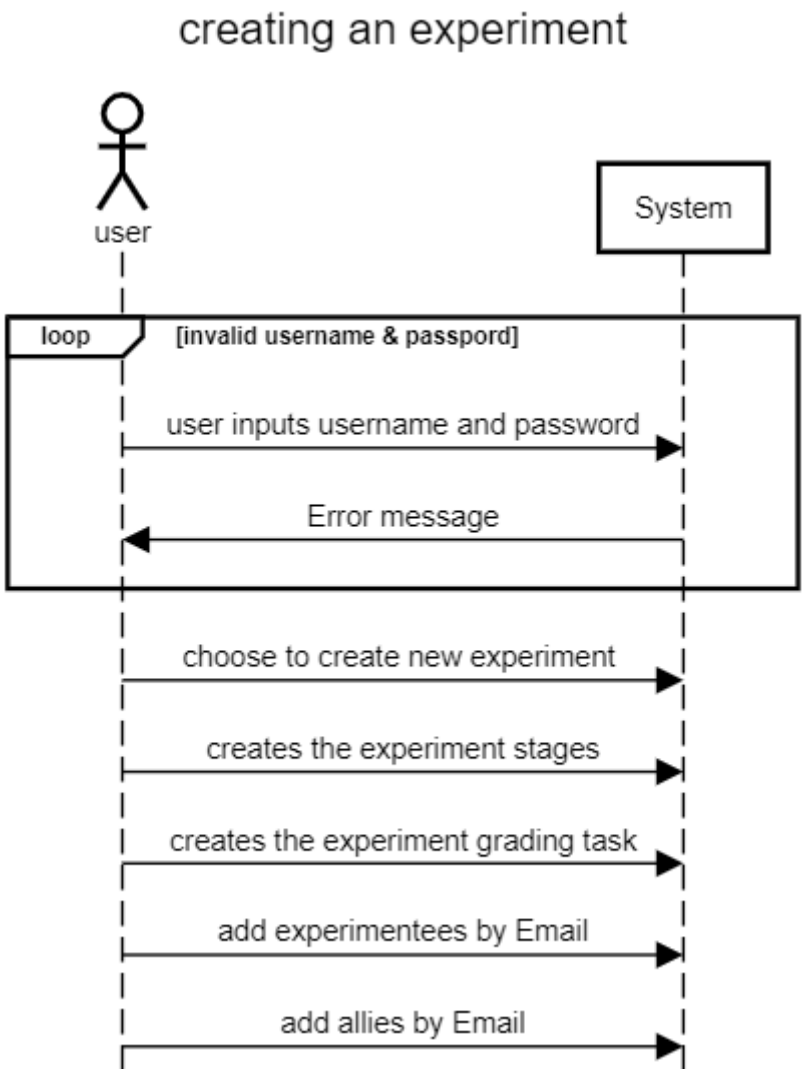
- Creating new experiments, this transaction will modify the experiments table and the stages table by adding data to them. It will also add new experimentees and graders to their tables.
- Participating in an experiment, this transaction will affect the results tables (such as Answers, CodeResult, RequirementTags) by adding data to them.
- Grading experimentees, this transaction will mainly affect the GradingTask table since that is the table that connects between the experimentees and the graders

Each one of the data objects will be mapped directly to a database entity.

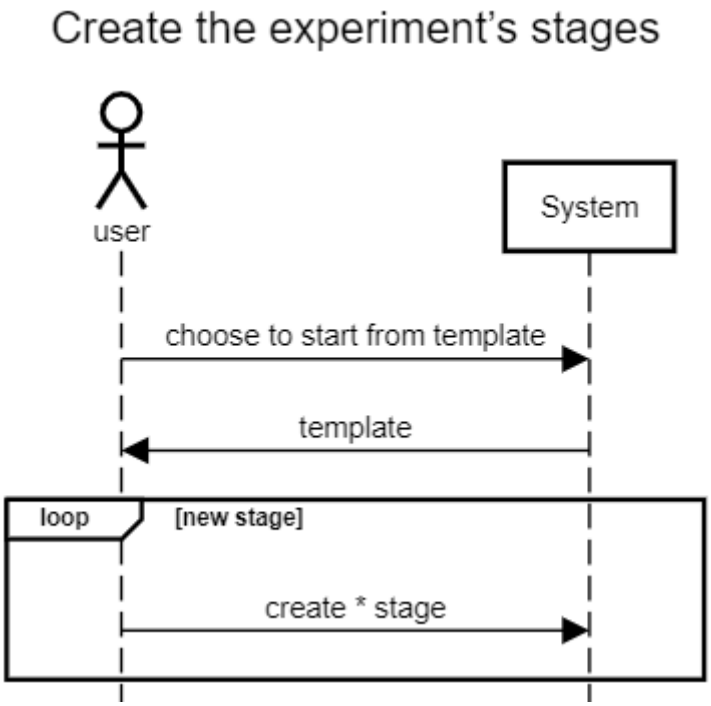
Chapter 4 - Behavioral Analysis

4.1 Sequence Diagram

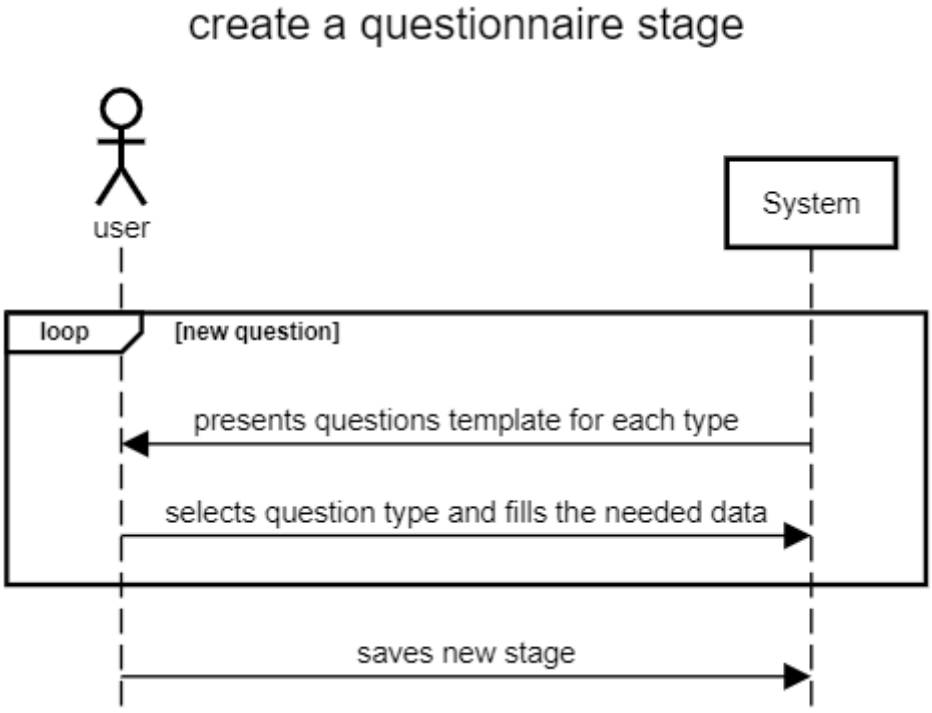
use case 1:


creating an experiment
user
System
loop
[invalid username & passpord]
user inputs username and password
Error message
choose to create new experiment
creates the experiment stages
creates the experiment grading task
add experimentees by Email
add allies by Email

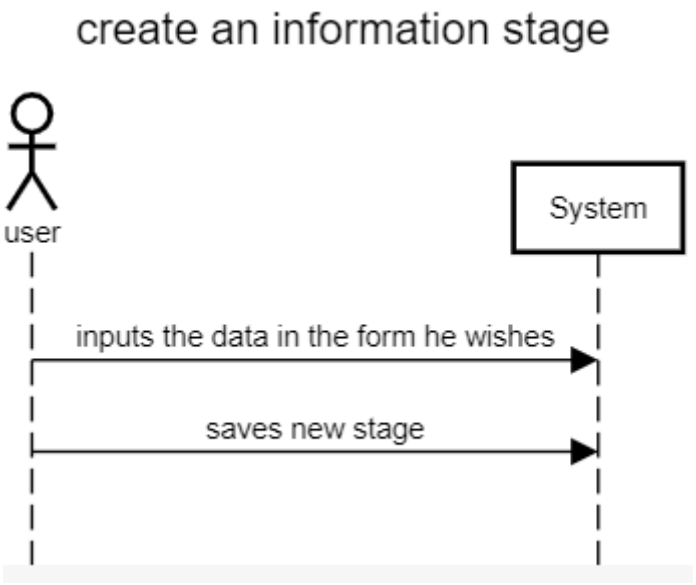
use case 1.1:



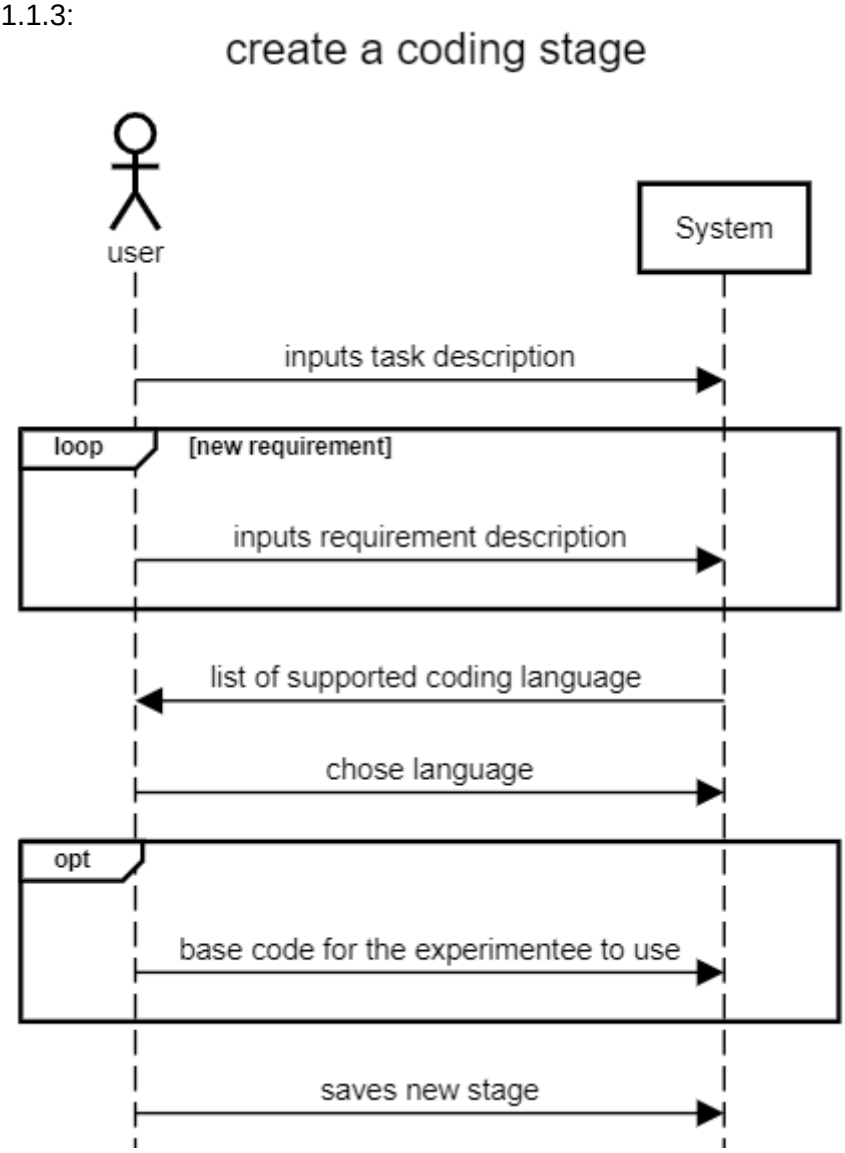
use case 1.1.1:



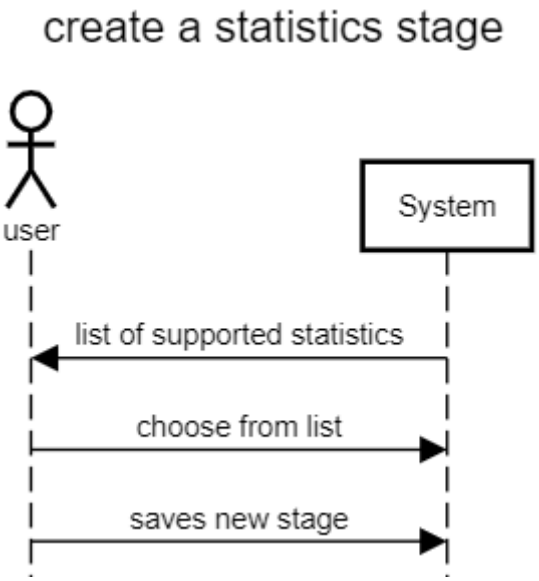
use case 1.1.2:



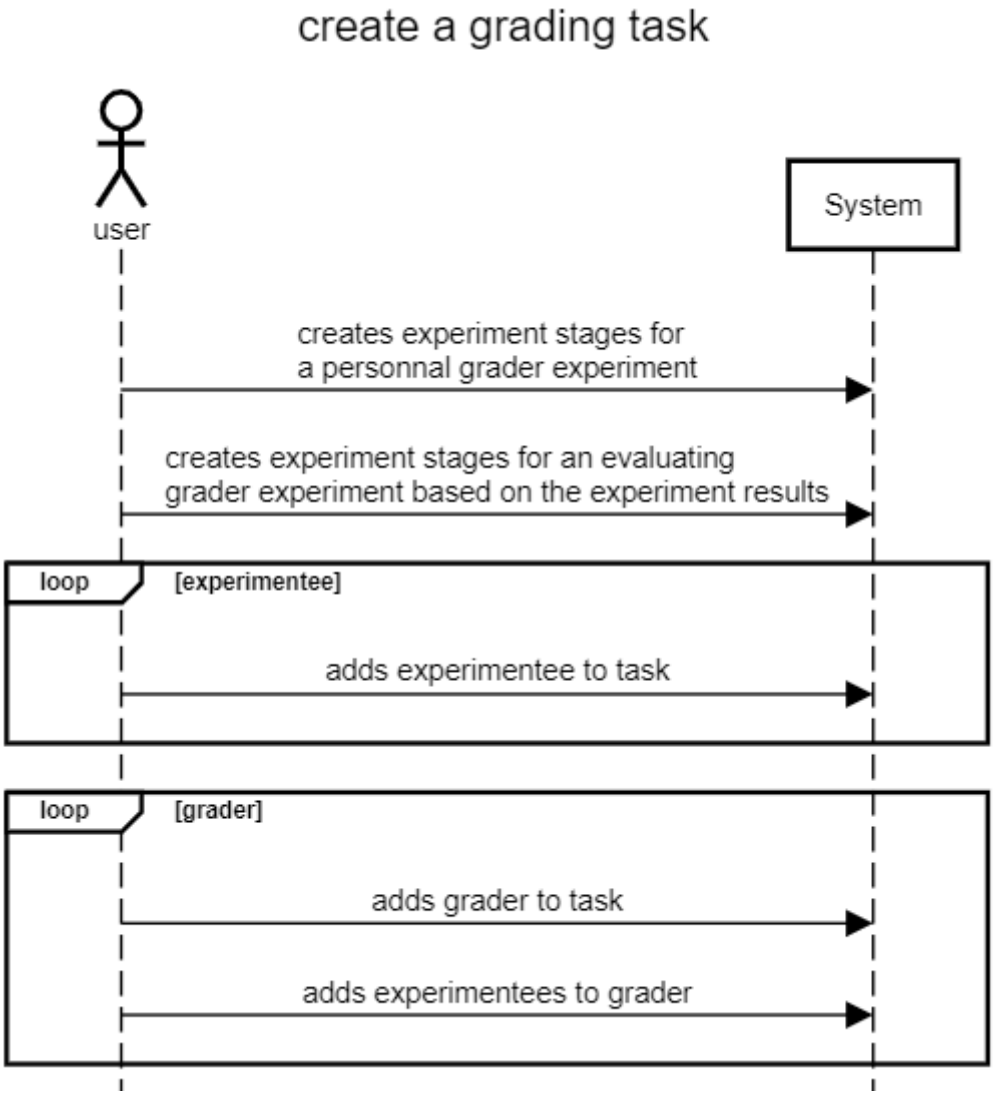
use case 1.1.3:



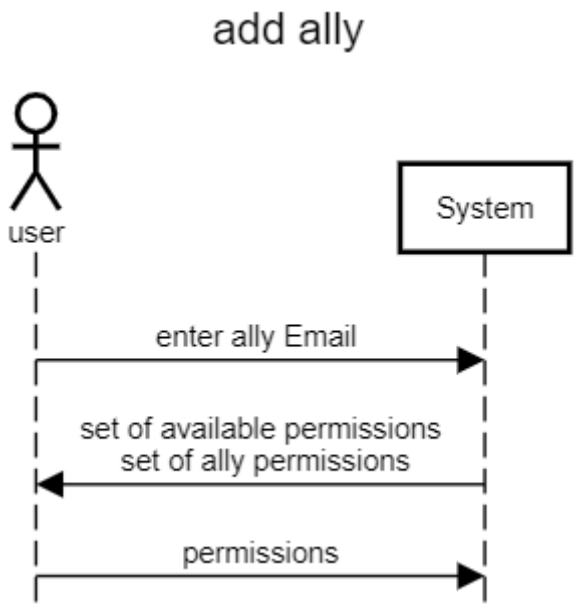
use case 1.1.4:



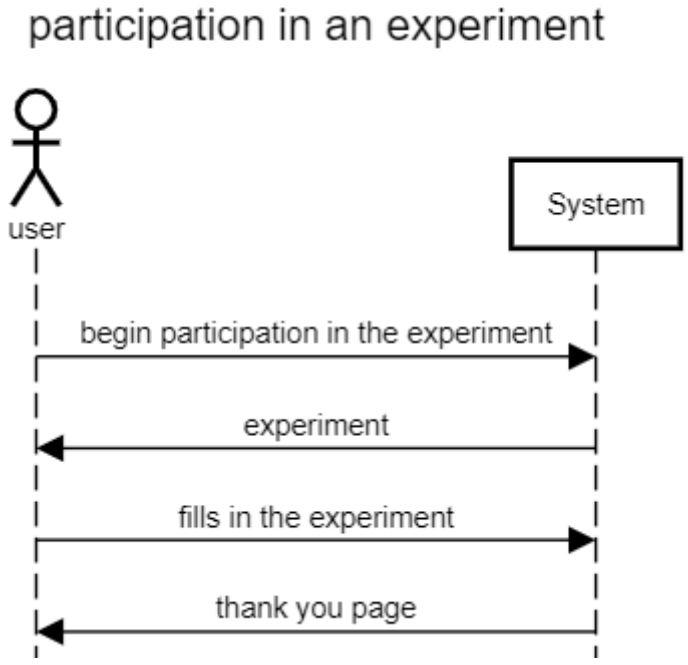
use case 1.2:



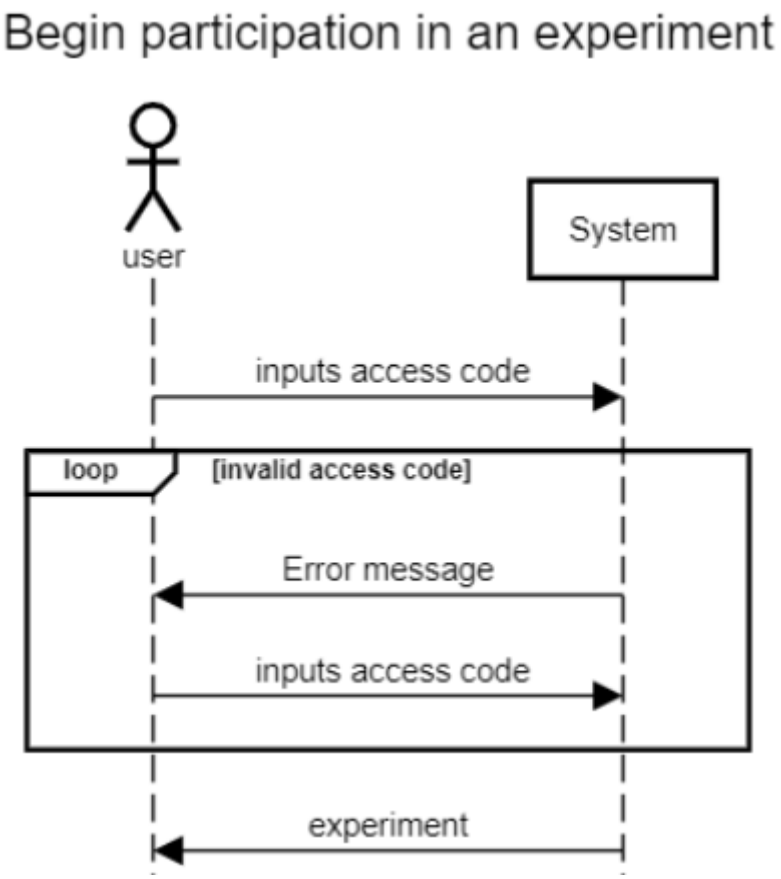
use case 1.3:



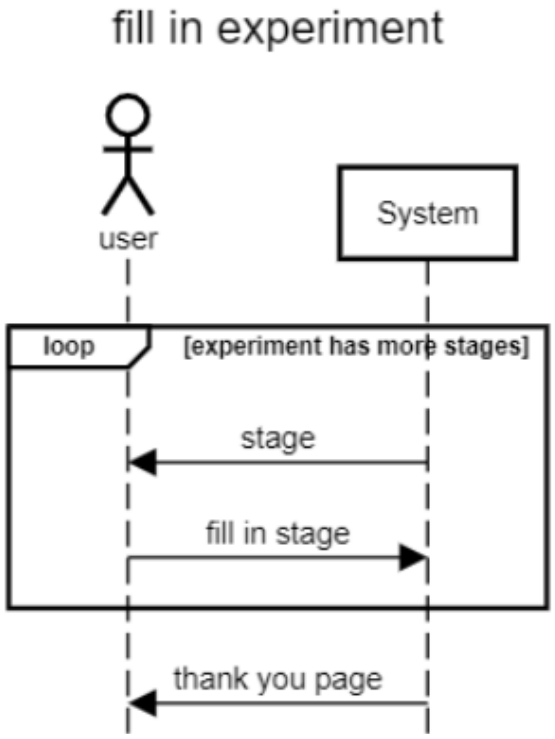
use case 2:



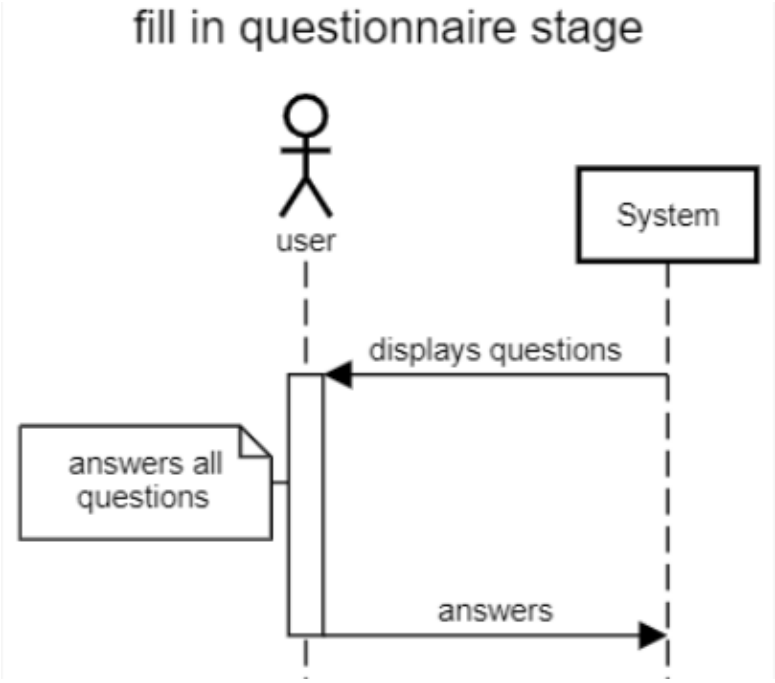
use case 2.1:



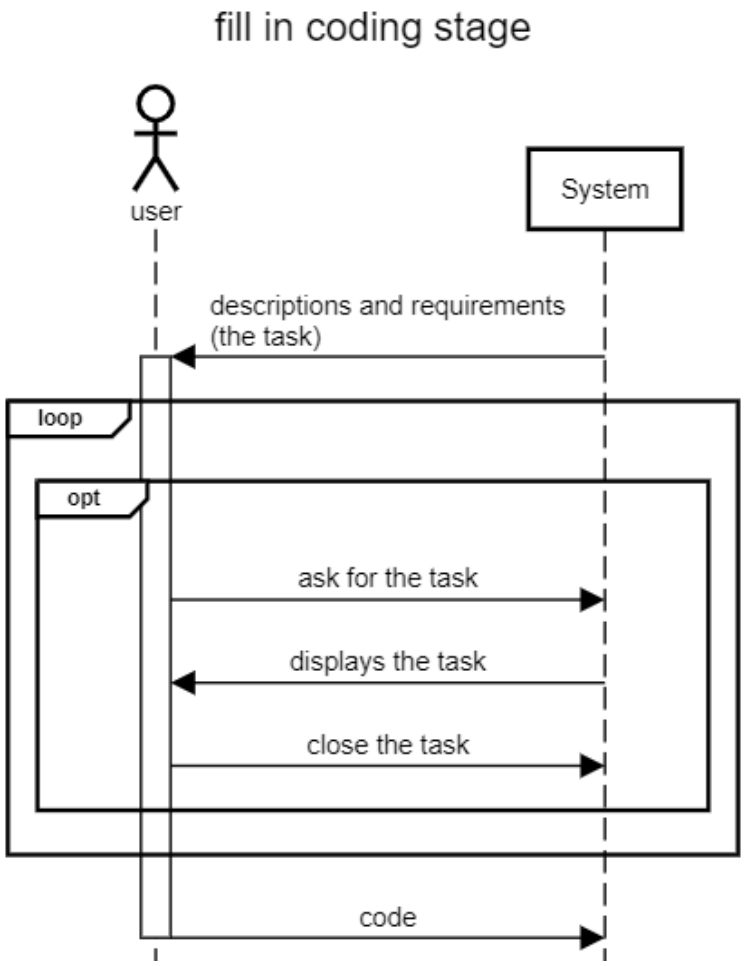
use case 2.2:



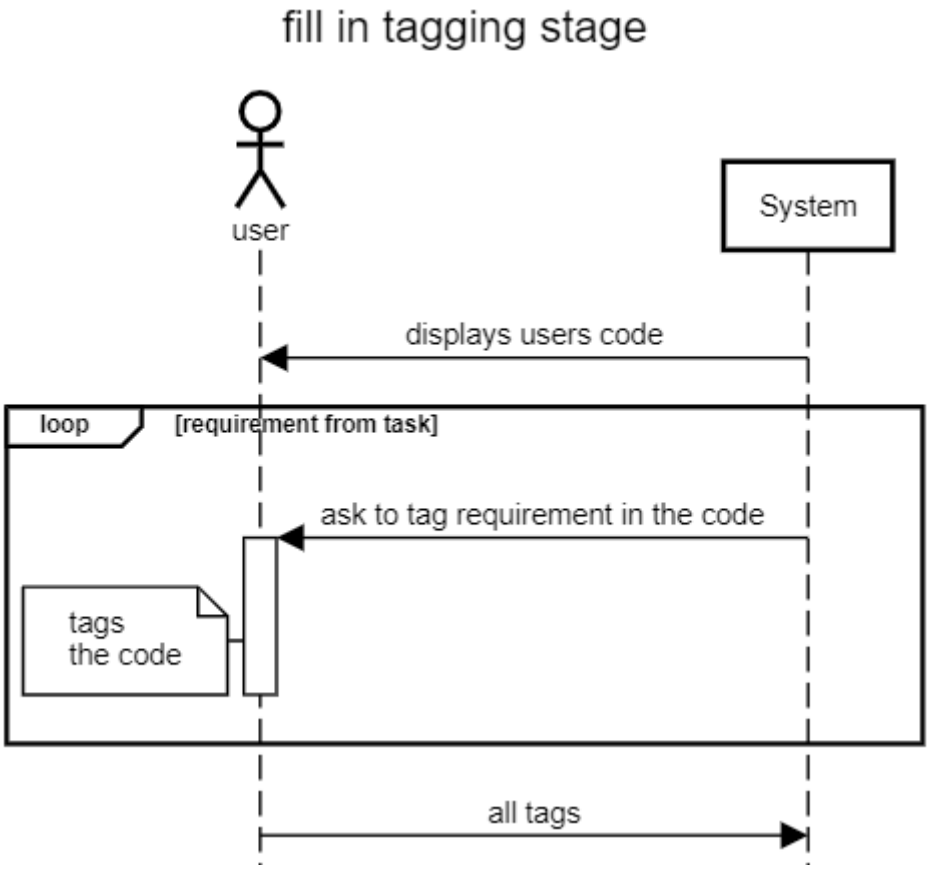
use case 2.2.1:


fill in questionnaire stage
user
System
displays questions
answers all questions
answers

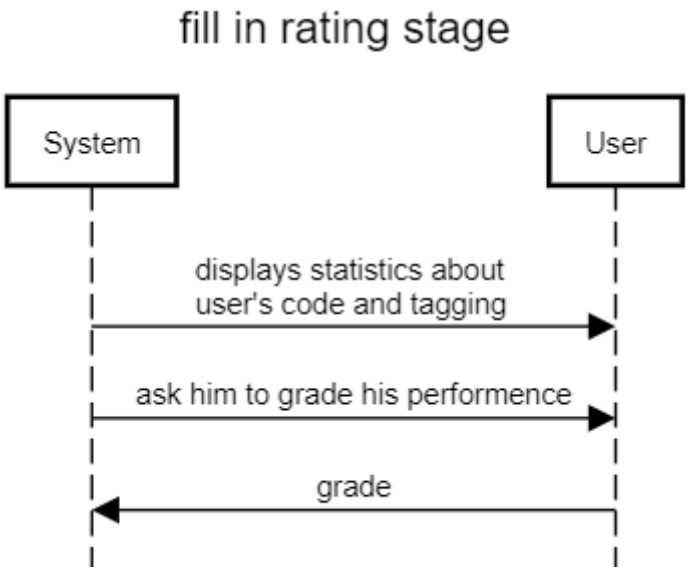
use case 2.2.2:


fill in coding stage
user
System
descriptions and requirements
(the task)
loop
opt
ask for the task
displays the task
close the task
code

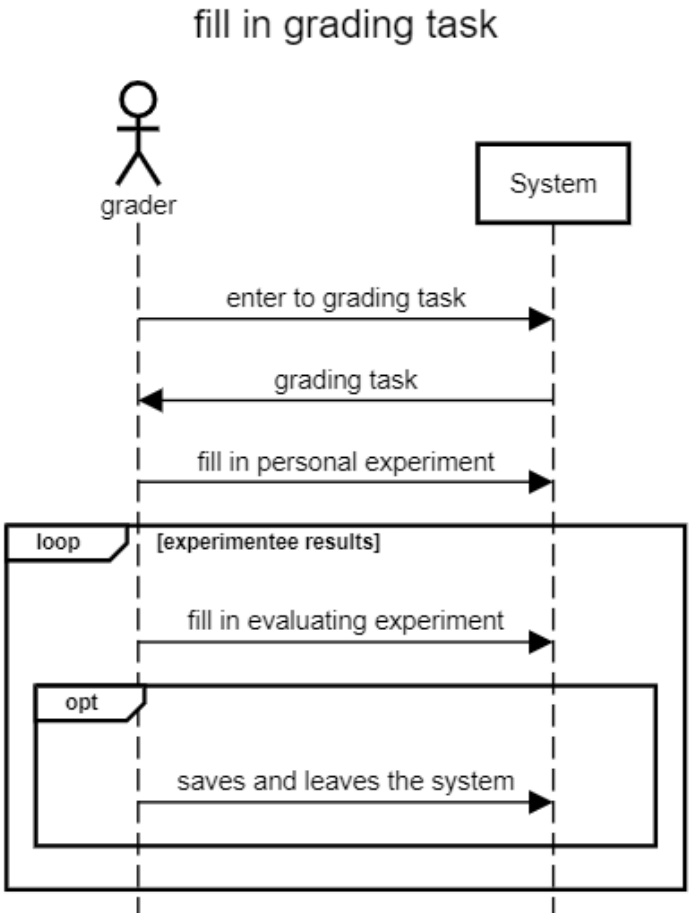
use case 2.2.3:



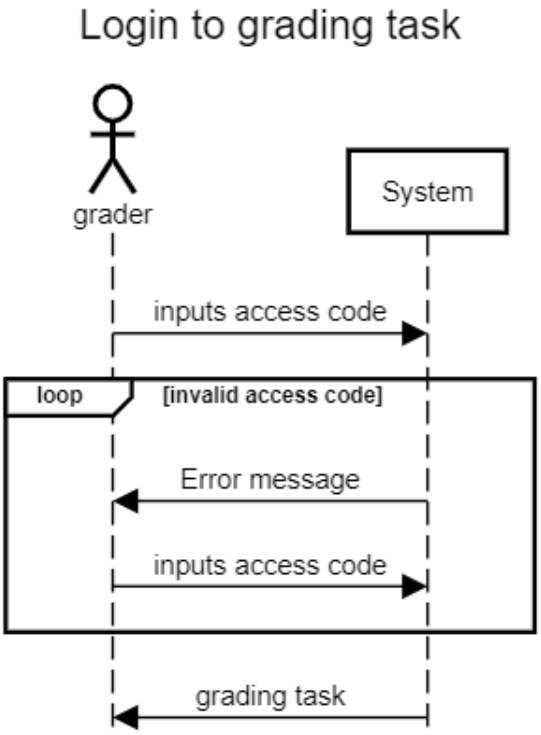
use case 2.2.4:



use case 3:



use case 3.1:



4.2 Events

As we mentioned before our system is based on 3 parts, experiment management system, grading system, and answer systems for the experimentees.

The experiment management system will handle the following events:
Log In- every researcher will have a bgu username and password, and he will have to submit them as input and then try to log in with in, this will initiate the event of logging in.
Log Out- simply logging out of the system and going back to the default page, at any time the logged in researcher would be able to initiate the event of logging out.
Creating a New Experiment - after logging in, every researcher would be able, with a press of the bottom to initiate the event of creating an experiment and moving to a state where he chooses to add stages from given types and by that creating the experiment.
Creating a New Stage in an Experiment- after starting creating the experiment, every researcher would be able, with a press of a bottom to initiate the event of creating a new stage and moving to state where he chooses to input the data from given types according to the type of the stage, and by that creating the new stage and to add it to the experiment.
Ending the creation of an Experiment- after stating to create an experiment and if at least one coding task stage was created, the experiment is valid and therefore the researcher can end it's creation and set it up by creating access codes, by creating the codes he finishes the creation of the experiment.
Viewing an Experiment- after logging in, every researcher would be able, while viewing a list of all of his previous published researchers , to click on one of them and by that to initiate the event of viewing an experiment, and afterwards to view all the details about the experiment.

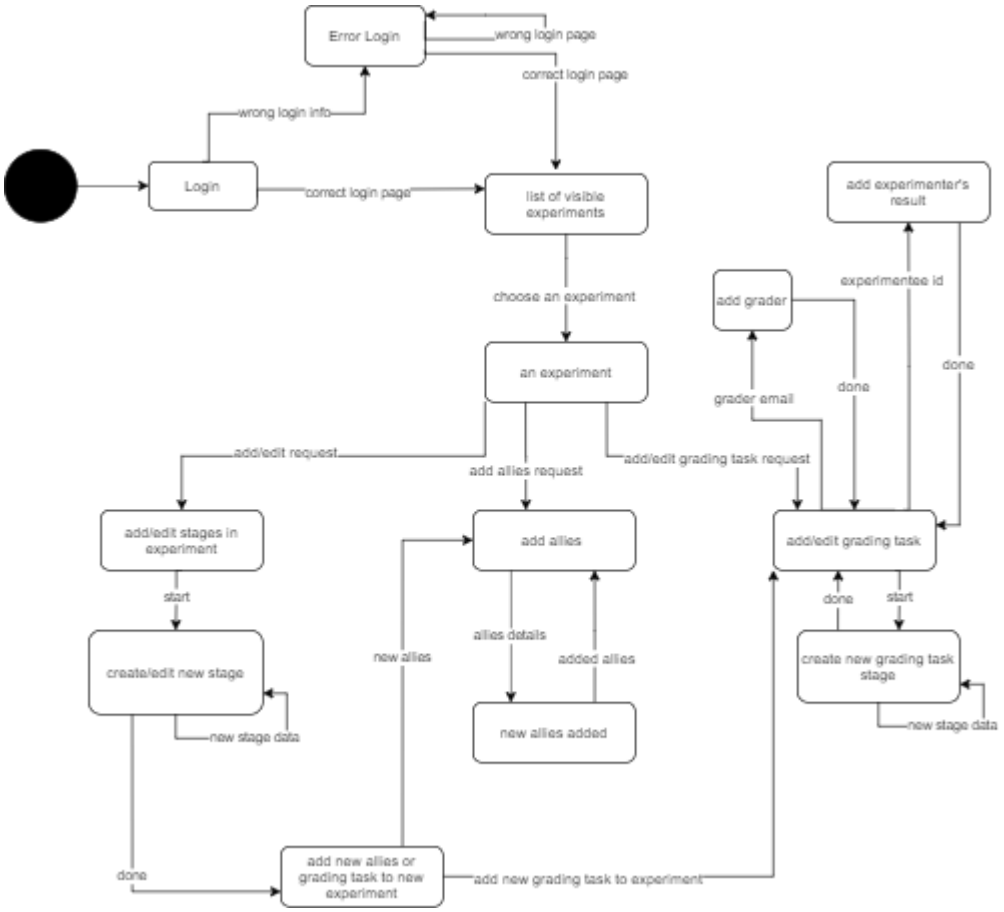
The grading system will handle the following events:
The Log In, Log Out, are very similar.
View results- the user wants to view all the results, both graded and not graded.
Grade result- the user can select a result from the pull with a click on it and see the specific result in detail thus starting the check and grade it.
Fill personal experiment- the user is a grader and the experiment creator wants to know some personal information about the person who graded the results, therefore before submitting any of the results, the personal experiment must be filled.
Submit results- the user submits some results for the experiments creator to see, and after submitting the results the grading of those results according to a specific grading task is not editable.

The answer systems for the experimentees will handle the following events:
The Log In, Log Out, are very similar.
Filling a stage- the user is doing an experiment, each state of the experiment is requesting data from the user, to move on the user must supply data of his choice, thus filling the stage and moving on to the next stage.
End experiment- the user finishes the experiment, gets a "thank you" and exits the site.

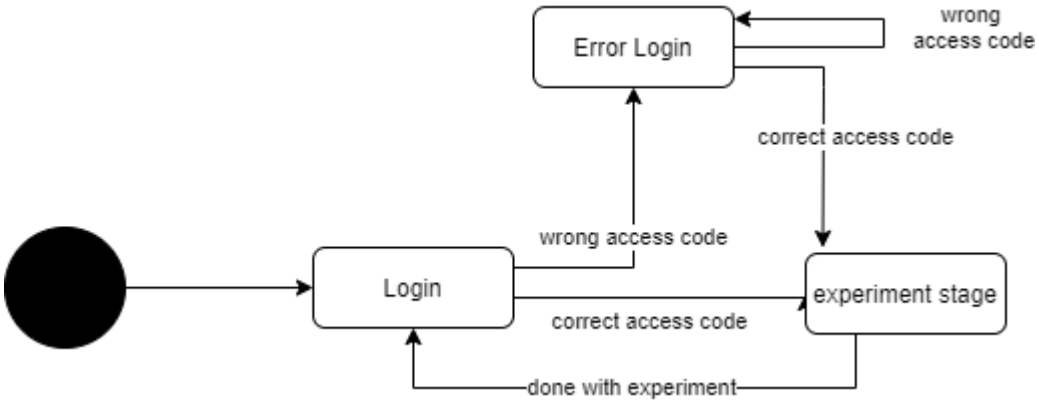
4.3 States

As we mentioned before our system is based on 3 parts, experiment management system, grading system, and answer systems for the experimentees.

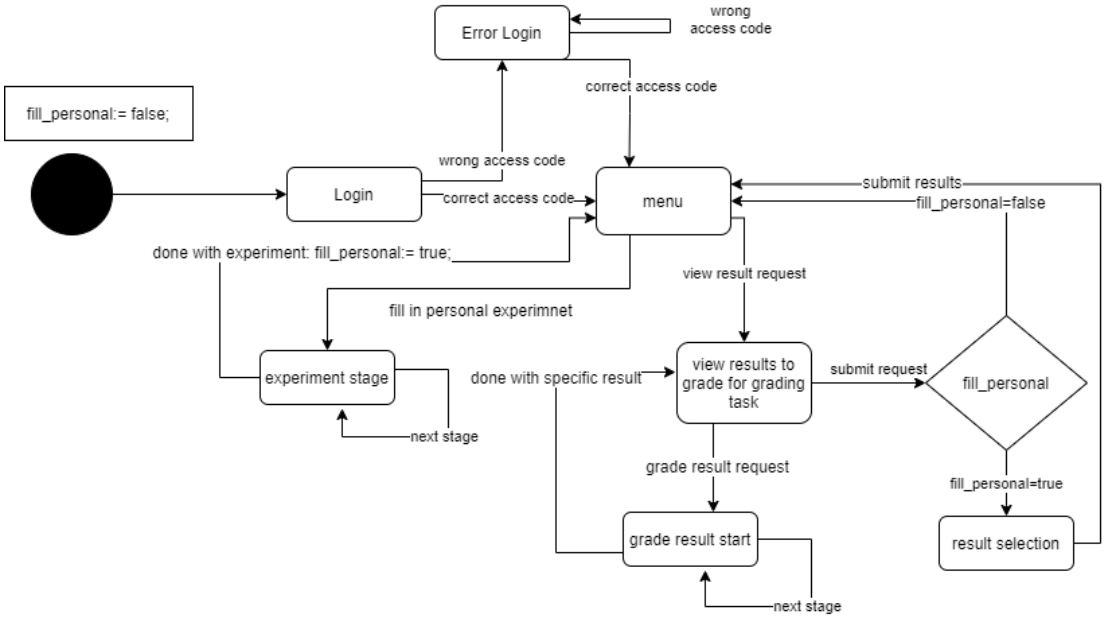
The state machine for the experiment management system:



The state machine for the answer systems for the experimentees:



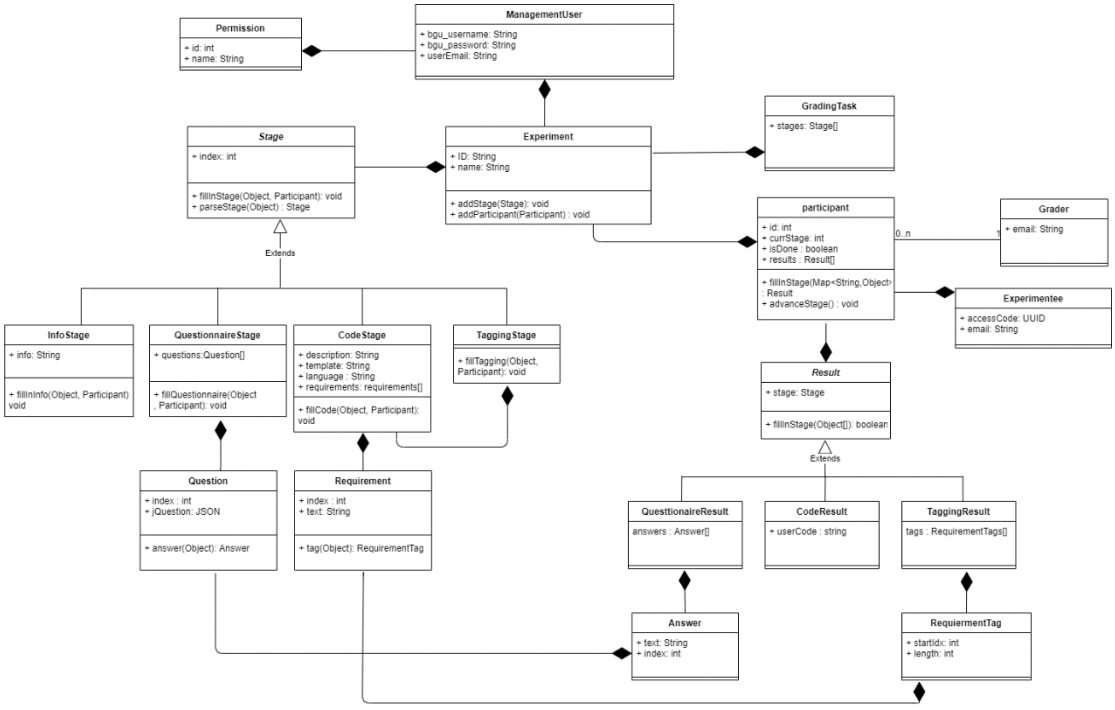
The state machine for the grading system:



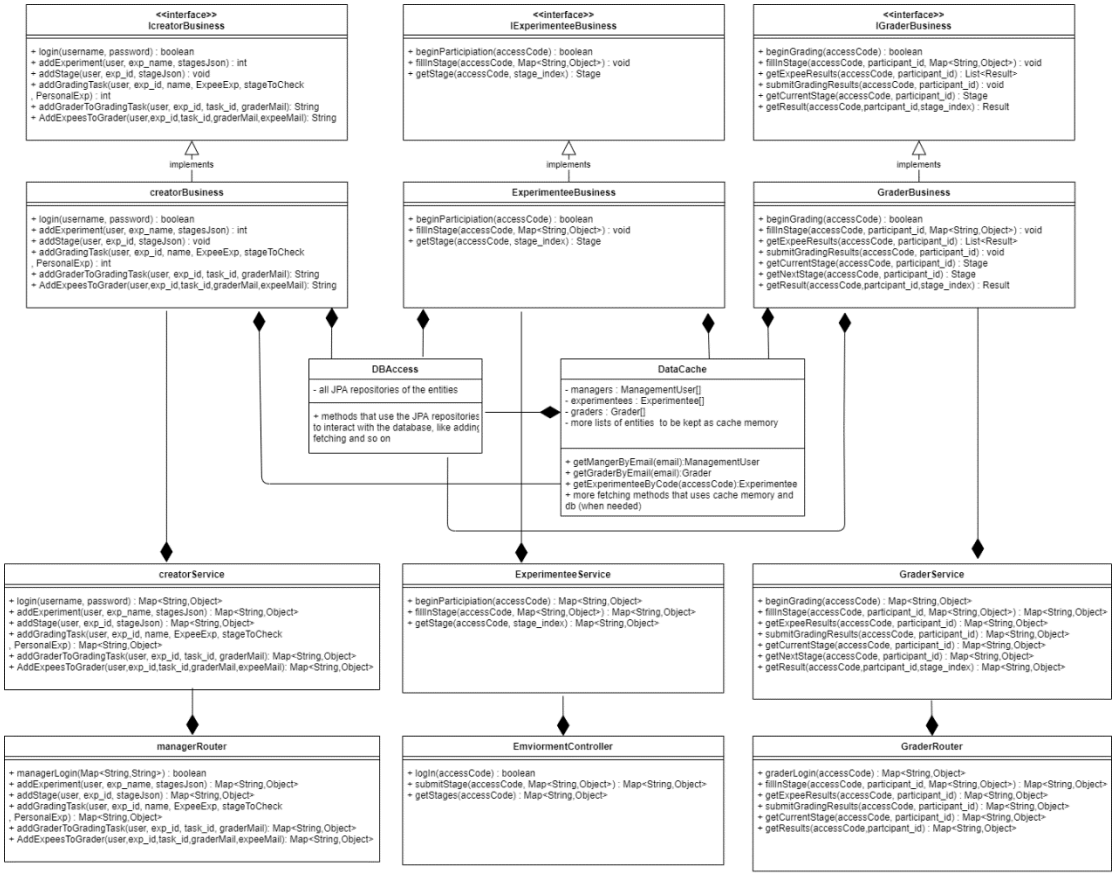
Chapter 5 - Object-Oriented Analysis

5.1 Class Diagrams

Entities Diagram:



Business to Routing Layers:



5.2 Class Description

The “Entity” classes: These classes are our representation of the entities we derived in chapter 3 - for every Entity in the ERD there is an entity class. They also contain the important functionality of said entity in the business logic.
The “Repository” interfaces: These interfaces are our persistence layer’s communication means and are not presented in the class diagram, for they are without any code and the methods in them are made automatically thanks to Spring Boot’s JPA technology. The methods from these interfaces perform various operations on the database. The interfaces all extend the JPARepository that creates automatic methods for communication with the database.
The “Controller” interfaces and classes: These interfaces are facades and describe the main functionality that is needed to communicate through our pipeline - requests are sent from the appropriate sub-system (management or environment (experimentee’s or grader’s)) and the communications with the business layer is neatly done through the facade.

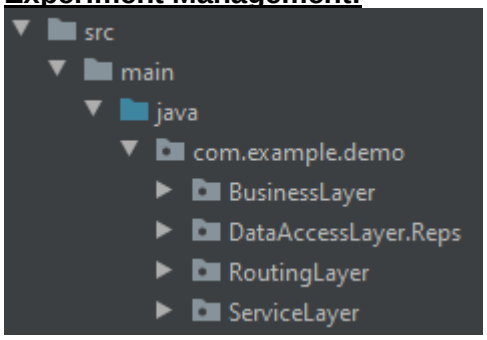
The “Service” interfaces and classes: These interfaces are responsible for the service layer’s communication with the upper and lower layers, the routing and business logic layers respectively.

The “Routing” interfaces and classes: These interfaces are responsible for the routing layer. They handle different requests and sends them down the layer pipeline, to the business service layer, for processing.

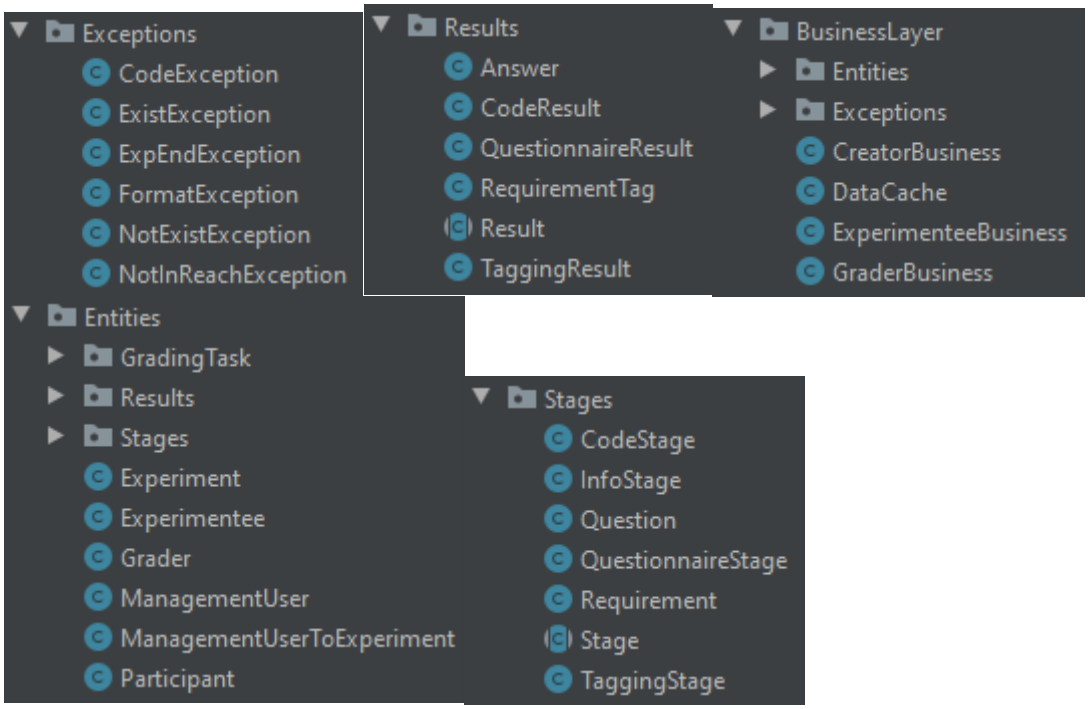
5.3 Packages

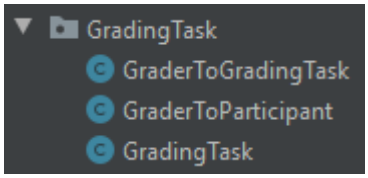
We will describe our division into packages in our modules:

Experiment Management:



The Management module is divided into packages using the multilayer architecture.
Business Logic Layer:



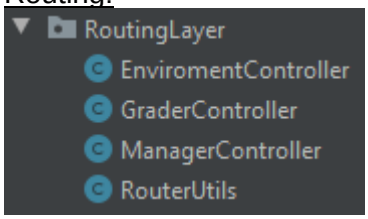


The business logic layer contains all of the entities and the business logic.

Service Layer:

The service layer contains three service classes for communicating through layers

Routing:



The routing layer contains three classes to route messages in and out the server.

5.4 Unit Testing

Our classes are divided into groups. For example, we have our entity classes, repository classes and so on. Because of this, we can design a generic test for every group and further describe it for each class.

The repository classes can be tested by calling various queries in the repository and checking the result to be as expected. Queries like adding a field that already exists or trying to remove a field that doesn't exist should raise an error.

The entity classes will be tested by performing the main functionality of every entity. For example - for a stage, we can call fillInStage() and check for correctness.

The controllers can be tested by simulating messages from the service layer through a driver and checking for correct responses.

The service layer can be tested by giving it a simulated request from the routing layer through a driver. It will also be tested by sending a simulated response from the business logic layer.

The routing classes can be tested by sending requests to them and checking that the behaviour is as expected.

Chapter 6 - User Interface Draft

SEE POWERPOINT

Chapter 7 - Testing

Our system has two main modules, the first one is the experiments environment, this module contains the participation in an experiment and the process of grading experimentees which are two of our main use cases. Each one of these main use cases will have sub use cases which will be helper methods for the main use case, we are planning to create unit tests for those sub use cases, the unit tests will cover valid inputs as well as inputs that might fail the system and check the system response on those inputs. after ensuring the correctness of the sub use cases, we will create acceptance tests for the main use cases, we will create both negative and positive tests on different groups of inputs, tests will be done according to the use cases description.
The second module is the experiments management, this module mainly contains the creation of new experiments which is the one of the main use cases. This use case also has sub use cases so the tests for this use case will be the same as those in the other module.

After testing the use cases, we wish to create integration tests which will contain testing use cases from different modules together and check the behavior of the system when the modules work together.
Each test will be added to the regression tests as we move on with the implementation.

As for the non functional constraints:

1. When testing the creation of experiments, the tests will focus on creating different kinds of experiments and not just the same experiments with different values. This will help us ensure that the system is generic.
2. For the requirement we will use special tools for load testing which will help us simulate events where multiple users are participating in the same experiment.
3. For the requirement we will use special tools for load testing which will help us simulate events where users are participating in several experiments in parallel.
4. Every test that contains sending data to the database will also test the correctness of the database after the data was sent.
5. To check this requirement we will create special tests that will add some information for a coding stage for example, wait a certain time to make sure the system performed the auto save and then we will check the database and verify it’s correctness.