Almaty, 2020, fall semester

«University network»
PROJECT REPORT

PROJECT TEAM*:*
Khamraev Alim (captain)
Ulan Orazgaliev

PROJECT GOAL:
Create an internal KBTU system

CONTENT OF THE REPORT:

INTRODUCTION. MVC

In our work, we tried to implement the project structure in such a way that it conforms to the standards of the MVC project architecture. We have divided our project into several components:

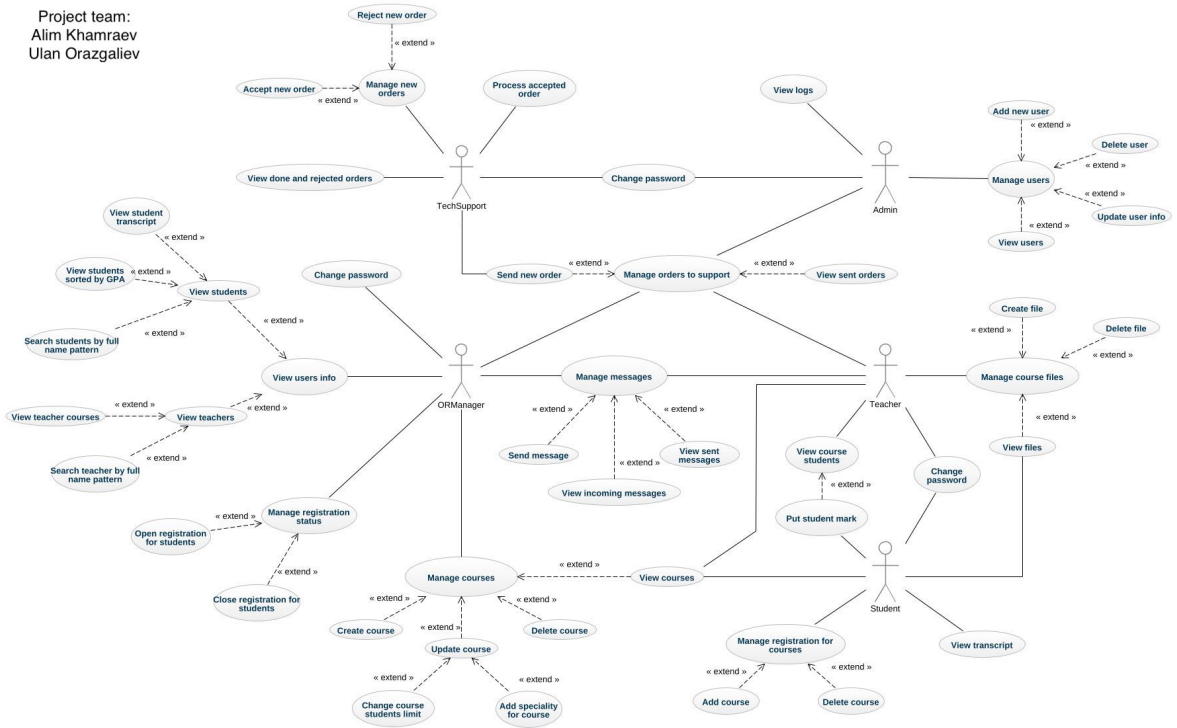
1. Models (classes) - are responsible for the business logic of our system
2. Controllers - provide interaction of models with views
3. Views - render and display all necessary information to the user

USE-CASE DIAGRAM. CLASS DIAGRAM

First of all, we started by creating a use-case diagram in order to clearly understand which functions should be present and to what type of user each of them should correspond.

This is our use-case diagram. Of course, in the process of subsequent work, we had been changing its appearance many times, eventually acquiring the final version.

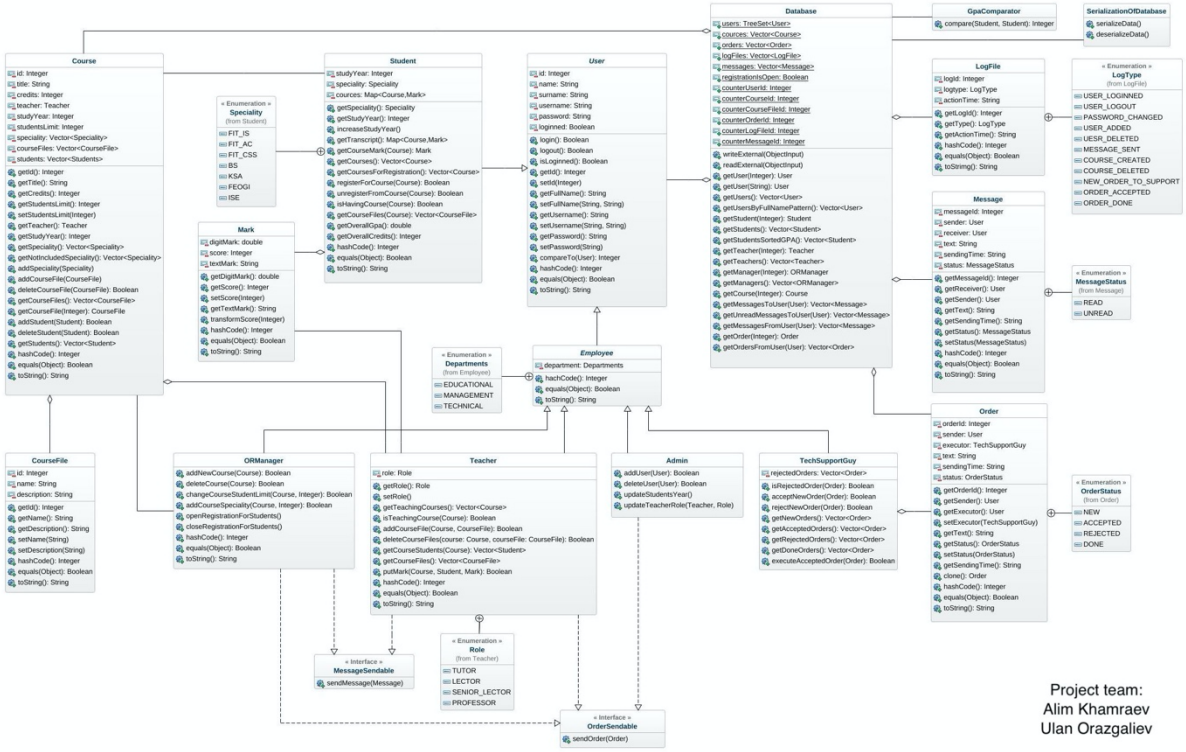
It turned out to be large, so we recommend you increase the scale of the document so that all the details will be clearly visible.



According to the diagram, we have 5 types of users, each of which has its own functionality.

So, when we already had the initial version of the use-case diagram, we started creating a class diagram, on which we displayed what fields and methods each model should have.

In the course of work, we also had been changing its appearance many times. And at the moment it looks like this. It also turned out to be really large, so we recommend you increase the scale of the document so that all the details will be clearly visible.



This is the final format of the business logic of our project. Now let's take a closer look at some system models.

MODELS INFO

The project consists of many classes, but the most important project model is the Database, where we store all the information necessary for the operation of the system as a whole.

```
public static TreeSet<User> users = new TreeSet<User>();
public static Vector<Course> courses = new Vector<Course>();
public static Vector<Order> orders = new Vector<Order>();
public static Vector<LogFile> logFiles = new Vector<LogFile>();
public static Vector<Message> messages = new Vector<Message>();

public static boolean registrationIsOpen = true;
public static int counterUserId;
public static int counterCourseId;
public static int counterCourseFileId;
public static int counterOrderId;
public static int counterLogFileId;
public static int counterMessageId;
```

Variables are static because the database class for all models must store common information. However, we decided to apply the Singleton pattern to this model to ensure that its object is unique. Although with this further we will experience many more problems associated with saving data to a file.

```
private static Database INSTANCE = null;

public Database() {}
public static Database getInstance() {
    if (INSTANCE == null)
        INSTANCE = new Database();
    return INSTANCE;
}
```

In addition to the fact that the database serves as a repository of information for us, we must be able to process and receive it back in a form convenient for us, depending on the situation.

To do this, we have created many static utility methods that we use in other models and controllers to ensure that the entire system works.

These are just a few examples of them:

A method for getting users from a set by searching a person's name

```
public static <T> Vector<T> getUsersByFullnamePattern(String pattern, Vector<T> users) {
    Vector<T> usersList = new Vector<T>();
    for (T user: users)
        if (((User)user).getFullName().toLowerCase().startsWith(pattern.toLowerCase()))
            usersList.add((T)user);
    return usersList;
}
```

getStudents () method for getting students from many other users
getStudentsSortedGPA () method to get students sorted by GPA

```
public static Vector<Student> getStudents() {
    Vector<Student> students = new Vector<Student>();
    for (User user: users)
        if (user instanceof Student)
            students.add((Student)user);
    return students;
}


public static Vector<Student> getStudentsSortedGPA() {
    Vector<Student> students = new Vector<Student>();
    for (Student student: getStudents())
        students.add(student);
    students.sort(new GpaComparator());
    return students;
}
```

Methods for getting all received, all unread messages, and all messages sent by the user

```
public static Vector<Message> getMessagesToUser(User user) {
    Vector<Message> messagesList = new Vector<Message>();
    for (Message message: messages)
        if (message.getReceiver().equals(user)) {
            messagesList.add(message);
        }
    Collections.reverse(messagesList);
    return messagesList;
}

public static Vector<Message> getUnreadMessagesToUser(User user) {
    Vector<Message> messagesList = new Vector<Message>();
    for (Message message: messages)
        if (message.getReceiver().equals(user) &&
                message.getStatus().equals(MessageStatus.UNREAD)) {
            messagesList.add(message);
        }
    Collections.reverse(messagesList);
    return messagesList;
}

public static Vector<Message> getMessagesFromUser(User user) {
    Vector<Message> messagesList = new Vector<Message>();
    for (Message message: messages)
        if (message.getSender().equals(user)) {
            messagesList.add(message);
        }
    Collections.reverse(messagesList);
    return messagesList;
}
```

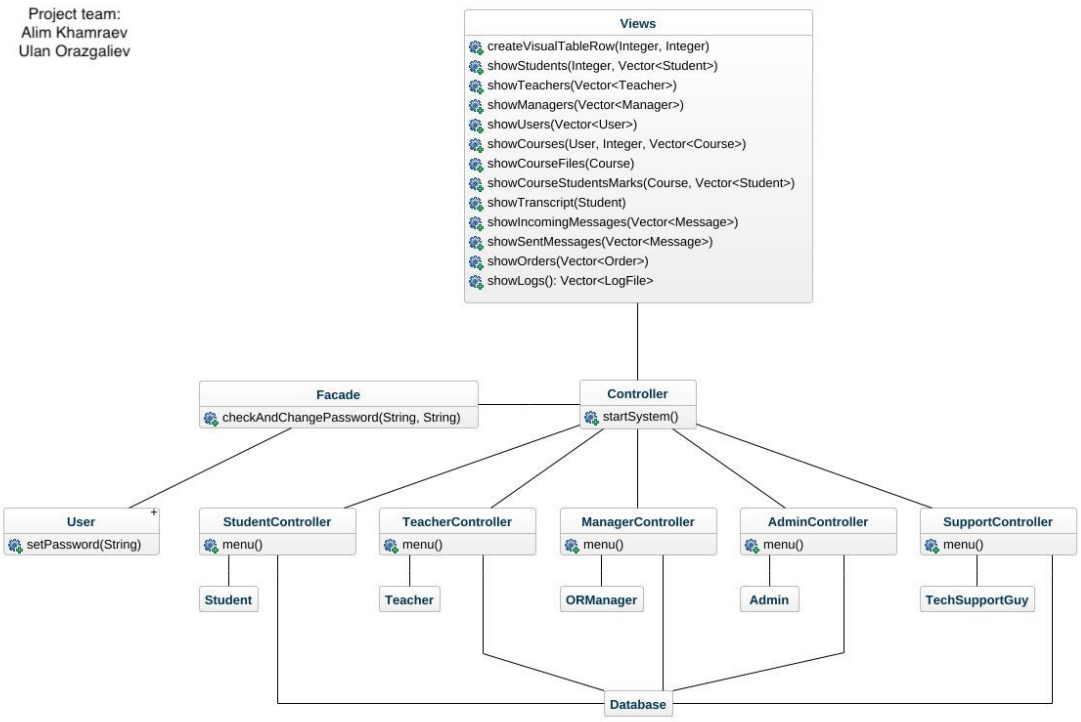
By similar methods, we get all the necessary information in the primary sorting and processing so that the other models can carry out the necessary actions related to this information.

In addition, we should be able to save all the current information so that the next time you log in, it will be restored. But more about this in chapter №6 "Data serialization".

CONTROLLERS INFO

Our controllers act as an intermediary between models and views, while interacting with which options the user chooses.

We decided to make several separate controllers that are connected to one main controller, while each of them has a connection to those models that implement the necessary business logic for a particular type of user.



At the same time, we also used a facade pattern that performs the method of checking and changing the old password to a new one.

Also, inside each controller there are internal methods that perform the same functional duties, but at the same time visually unload the main method.

VIEWS INFO

We have made special methods, the main function of which is to output data to the console to the user. Further, in all controllers, we simply use them, passing the necessary data for display.

We wanted to somehow creatively approach the output of information to the console, so we created a special method that creates visual table rows, with which you can create tables of varying complexity.

```java
public static String createVisualTableRow(int numOfColumn, int[] columnSizes) {
    String result = "|";
    for (int i = 0; i < numOfColumn; i++) {
        result += " %-" + (columnSizes[i]-2) + "s |";
    }

    result += "\n+";
    for (int i = 0; i < numOfColumn; i++) {
        for (int j = 0; j < columnSizes[i]; j++) {
            result += "-";
        }
        result += "+";
    }
    return result;
}
```

The method accepts the number of columns and their sizes as input. Next, a table row is created with special characters for further formatting and entering data into cells.

As an example we created one table row with 5 columns, so it this format, we receive a response from this method.

```
| %-10s | %-20s | %-30s | %-20s | %-40s |
+----------+--------------------+------------------------------+--------------------+----------------------------------------+
```

And then we just format the string using the command: (table row).formatted (...)
Here are some examples of tables that we can create.

Displaying all users for the Administrator

ID	Name	User type
1	Admin	Admin
4	Aibek Bolathan	Student
3	Dave Jones	Student
7	Kate Katrin	ORManager
6	Mark Randell	TechSupportGuy
5	Ramesh Kini	Teacher

Displaying sent requests to technical support

ID	Sending time	Sender	Order message	Status
1	21-12-2020 19:11:28	Admin	Please help me. My computer doesn't work	NEW
2	21-12-2020 19:12:09	Admin	I need some help with monitors	NEW

View courses

ID	Discipline title	Speciality	Credits	Study year	Teacher	Students
1	Basics of Information Systems	[FIT_IS, FIT_AC, FIT_CSS, BS, KSA]	3	2	Ramesh Kini [PROFESSOR]	0/150

Displaying sent messages to user

Sending time	Receiver	Message text	Status
21-12-2020 19:18:00	Ramesh Kini	Also i need to know when you can come here	UNREAD
21-12-2020 19:17:05	Ramesh Kini	Hello teacher, i sent some documnets to you, please check	UNREAD

Displaying course registration for student

ID	Discipline title	Speciality	Credits	Study year	Teacher	Students	Registered
1	Basics of Information Systems	[FIT_IS, FIT_AC, FIT_CSS, BS, KSA]	3	2	Ramesh Kini [PROFESSOR]	1/150	true

Displaying teacher student info before/after putting course mark

ID	Student name	Speciality	Study year	Score	Digit mark	Letter mark
4	Aibek Bolathan	FIT_IS	2	87	3.33	B+

Displaying student transcript

Transcript of student: Aibek Bolathan [ID:4] [Speciality: FIT_IS]

ID	Discipline title	Credits	Study year	Score	Digit mark	Letter mark
1	Basics of Information Systems	3	2	87	3.33	B+

Number of registered disciplines: 1
Overall Credits: 3
Overall GPA: 3,33

Thus, we can create tables of any complexity to visually display information in a form convenient for us, depending on the situation. Views also support the ability to select the number of columns that will be displayed to the user.

DATA SERIALIZATION

So, we needed to create the functionality of saving and restoring database information. To do this, we have created a separate class SerializationOfDatabase, using the methods of which we can serialize and deserialize information.

```
// Process of data deserialization from file
public static boolean deserializeData() {
    try {
        FileInputStream fileInputStream = new FileInputStream("uninetData.ser");
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

        objectInputStream.readObject();

        fileInputStream.close();
        objectInputStream.close();
        return true;

    } catch (ClassNotFoundException exception) {
        return false;
    } catch (FileNotFoundException exception) {
        return false;
    } catch (IOException exception) {
        return false;
    }
}

// Process of data serialization to file
public static void serializeData() {

    try {
        FileOutputStream fileOutputStream = new FileOutputStream("uninetData.ser");
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);

        objectOutputStream.writeObject(Database.getInstance());

        fileOutputStream.close();
        objectOutputStream.close();
    }
    catch (IOException exception) {
        System.exit(0);
    }
}
```

Also, our Database class implements the Externalizable interface, since our variables in which we store information are static. In addition, when we tried to implement the ordinary Serializable interface, we faced a certain problem related to the fact that static variables were not serialized. Thus, with the help of Externalizable, we were able to manage this process ourselves.

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeObject(users);
    out.writeObject(courses);
    out.writeObject(orders);
    out.writeObject(messages);
    out.writeObject(logFiles);
    out.writeObject(registrationIsOpen);
    out.writeObject(counterUserId);
    out.writeObject(counterCourseId);
    out.writeObject(counterCourseFileId);
    out.writeObject(counterOrderId);
    out.writeObject(counterMessageId);
    out.writeObject(counterLogFileId);
}
```

For restoring info

```
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    users = (TreeSet<User>) in.readObject();
    courses = (Vector<Course>) in.readObject();
    orders = (Vector<Order>) in.readObject();
    messages = (Vector<Message>) in.readObject();
    logFiles = (Vector<LogFile>) in.readObject();
    registrationIsOpen = (Boolean) in.readObject();
    counterUserId = (int) in.readObject();
    counterCourseId = (int) in.readObject();
    counterCourseFileId = (int) in.readObject();
    counterOrderId = (int) in.readObject();
    counterMessageId = (int) in.readObject();
    counterLogFileId = (int) in.readObject();

}
```

However, here we have another problem, namely, that the Externalizable interface requires an empty constructor for the class, in order to restore information later. And this violates the Singleton principle. We did not understand how to solve this problem.

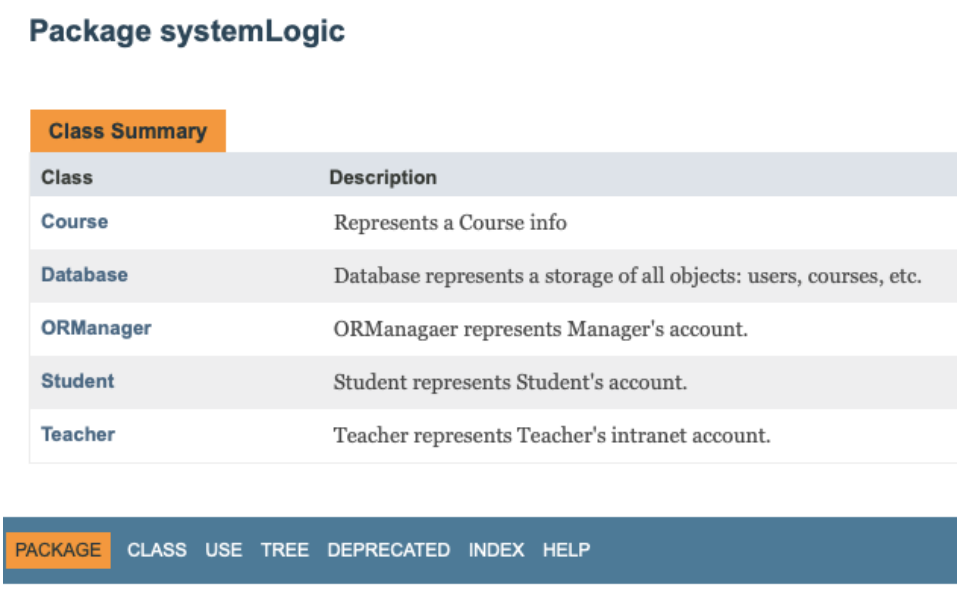
Further, in the main controller, before starting the program, we try to restore the saved information. If this cannot be done, then we create a new user with the Admin type in order to be able to create a user of other types and later continue working from their accounts.

```
if (!SerializationOfDatabase.deserializeData())
    Database.users.add(new Admin("Admin", "", Departments.TECHNICAL));

startSystem();
```

DOCUMENTATION

We also wrote a little documentation for some of our classes (Student, Teacher, Database, Course, ORManager). For this we used comments in Eclipse and then generated HTML documentation. In it, we described the purpose of the available methods, added an explanation to their arguments and return value. This documentation is available in our project folder.



Package systemLogic

Class Summary

Class	Description
Course	Represents a Course info
Database	Database represents a storage of all objects: users, courses, etc.
ORManager	ORManagaer represents Manager's account.
Student	Student represents Student's account.
Teacher	Teacher represents Teacher's intranet account.

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

TEAMWORK PROCESS

Our team consists of two people (Khamraev Alim (captain), Ulan Orazgaliev)

In order to work together on the project, we used Microsoft Teams. We planned a meetings and during them discussed ideas, design stages, ways of implementing certain functions in the system. If necessary, the participant turned on the screen demonstration in order to show and correct the code online. Also, we used the Telegram messenger with same purposes.

Ulan Orazgaliev:
Full development of functionality for a user with the TechSupportGuy type, as well as maintaining the class diagram up to date, according to the stages of our development, and writing documentation for the project.

Alim Khamraev:
Defining tasks and coordinating the actions of the team, developing other components of the system and checking the work of teammates.

CONCLUSION

As a result of joint actions, we have created a network of KBTU, which provides functionality for various types of users. During development we honed our Java programming skills using all object-oriented programming instruments. We learned how to plan projects of this scale, using all the tools, such as use-case diagram and class diagram. We have developed teamwork and responsibility sharing skills to achieve the final result faster and better.

We express our deep gratitude to our teacher. Pakita, you are a wonderful teacher who can teach people how to program and do it with great love. Thank you for this unforgettable Java programming course and for such an interesting final project that allowed us to consolidate and improve our soft and hard skills.