Organizing the project of Team 2

Markus Hilpold

Burgstall/Postal (BZ), Italy, 09.12.2018

E-mail:
[1]`markus.hilpold@student.uibk.ac.at`

1. introduction

Although it is the code which gives the application its final shape, the first important step of starting off a project is without a doubt the organisation. In the following document I shall present to you my idea of how our future web-application should look like. First I shall give you a general insight of how the core of the program is structured while at the same time guiding you through my plan; I will then proceed by explaining the outer architecture of the application; and finally I will discuss how we should approach everything tomorrow, as a group.

2. design patterns

In this section I shall discuss the UML diagram I published on Friday, 7/12/2018.

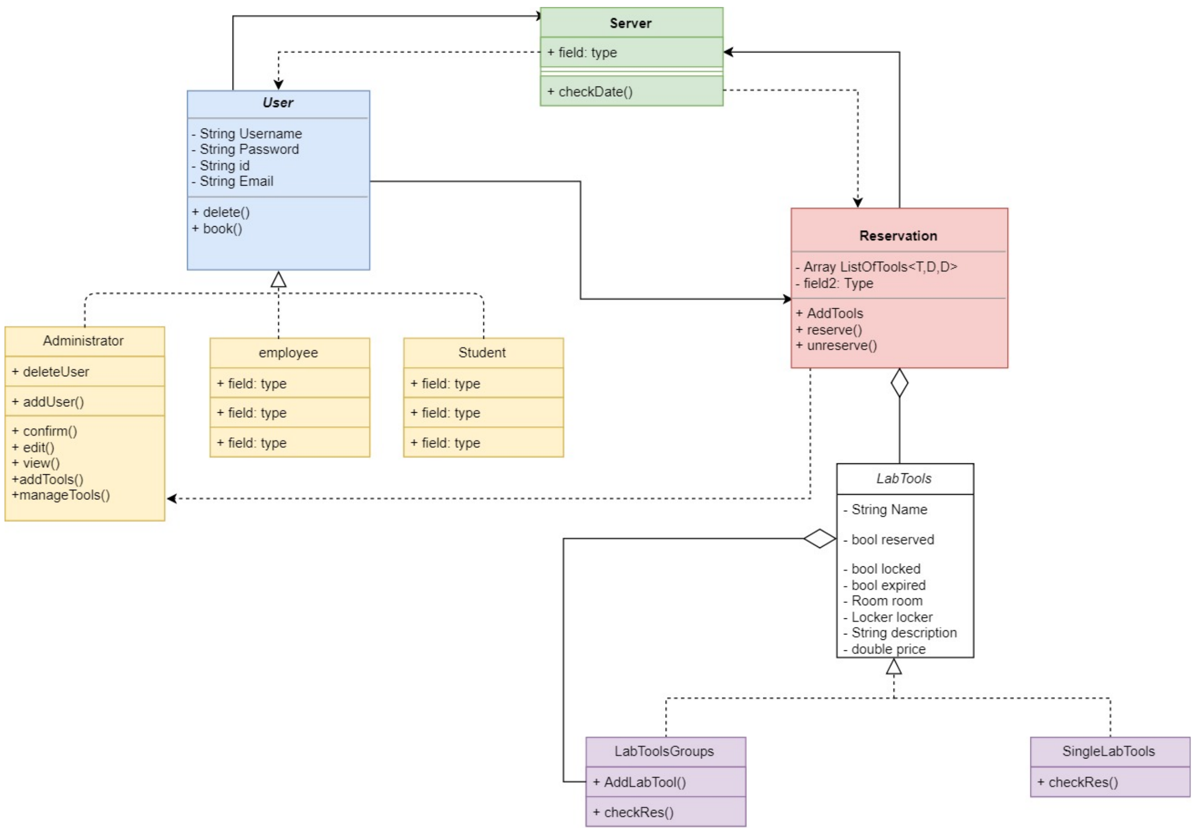

Figure 1: Design-pattern

For now, just ignore how the user interacts with the server. Instead we shall focus on the inner aspects of the application. We have three classes extending the abstract class "User". All three have the possibility to reserve tools. The student can book single tools, the employee can also book groups of tools and the administrato has also access to other functions, such as adding and deleting users and tools and to view all the reservations.

Both single tools and groups of tools can be booked. Here we shall use a composite pattern. It is important to implement a searchTree() recursive function to iterate through all elements of the tree. Note that the booking - and expiration dates need to be considered as well.

Just a quick remider of how such a composite pattern looks like.

```java
public interface CompositeInterface {

   public String getName();
   public double getPrice();
   public boolean search(String name);
   public void addFeature(CompositeInterface cI);

}
```

```java
/*Simple feature is a leaf so no list is needed to add other features*/
public class SimpleFeature implements CompositeInterface {

    private String name;
    private double price;


    public String getName() {
       return name;
    }
    public double getPrice() {
       return price;
    }

    SimpleFeature(String name, double price){
       this.name = name;
       this.price = price;
    }

    public boolean search(String name){
       return true;
    }

    public void addFeature(CompositeInterface cI){
       throw new IllegalArgumentException ("You cannot add anything to simple
          features");
    }

}
```

```java
import java.util.ArrayList; // we import the array in order to add other
   features
import java.util.List;
import java.util.Iterator;

public class FeaturePackage implements CompositeInterface {
```

```
private String name;
private double price = 0;

List<CompositeInterface> list = new ArrayList<CompositeInterface>();
    //this allows us to add both simple features and other packages to
    one package
Iterator<CompositeInterface> comparer = list.iterator();

public boolean search(String name){   //if I remember correctly this did
    not work, sorry about that guys... but you get the point
   for(CompositeInterface item : list){
      while(list.iterator().hasNext()){
         if(item.getName() == name){
            return false;
         }
         if(item.getName() != name && item.search(name) != true){
            item.search(name);
         }


      }

   }

   return true;
}

public String getName() {
   return name;
}
public double getPrice() {
   return price;
}

public void addFeature(CompositeInterface cI){
   for(CompositeInterface item : list){
      if(cI.getName() != item.getName()){
         list.add(cI);
      }
      this.price = this.price + cI.getPrice();
   }

   for(CompositeInterface item : list){
      if(search(item.getName()) == false){
         this.price = this.price - item.getPrice();
      }

   }
```

```
    }

    FeaturePackage(String name){
       this.name = name;
    }

}
```

Everything else will be discussed during our meeting and is of little relevance at moment.
Let's move on to the next section:

3. Architecture

For this project I thought it would be best to implement an MVC (Model-View-Controller) pattern to promote user-friendliness and to make sure that who is using the application does not need to know about what is happening in the background. Here is a picture to show you how it is supposed to look like:

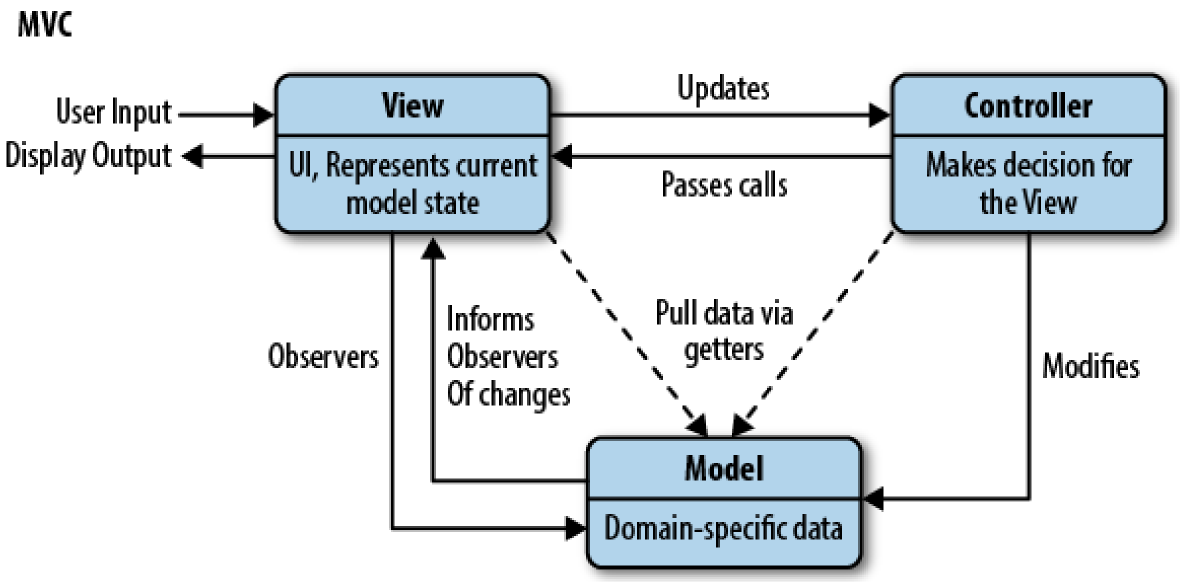

Figure 2: MVC-pattern [1]

[1]

4. Organization

Tomorrow we shall meet at 15:00 at the RR 20 in the architecture buildig. Other that splitting the work we should also make sure that everyone has what it takes to start working on the project. I hope this document is useful (it's not easy writing a coherent text at 1:30 am). Either come up with your own plans or try to read carefully through this document in case you should be called to present the work done so far. Everything else will be discussed tomorrow