**Final Project Report**

P-CB-S04



Group: 3

Joran van de Moosdijk - 4220923

Matija Mandarelo – 4279425

Shanessa Kostaman – 450080

Tobias Halomoan - 4252861

**Fontys University of Applied Sciences**

**Eindhoven – Netherland**

**2020-2021**

**Introduction**

1. **Problem**

Student Housing BV is a company that rents multiple housing complexes around the Netherlands and market them towards students staying in the country. These buildings consist of multiple personal rooms and a public shared living space that contain the essential items/ utensils for living (kitchen, laundry, storage, etc.). In this simulated environment, our group were to become a part of the consultant team appointed to implement software solutions for the company’s problems. In this simulation, Student Housing BV will be the client judging the finished product, represented by our instructors.

The company has been receiving multiple complaints from the clients regarding the use of the public shared living space. These range from students in the building not properly maintaining the facilities provided, to unannounced student gatherings that does not fit current quarantine regulations, and other minor inconveniences. Therefore, an application that allows for easy management between the students and building owners (tenants) to maintain a safe and clean living space for would be a good solution for these quandaries.

1. **Implementation**

For this problem, we envisioned an application that would allow students the ability to relay public announcements and send complaints / input to the tenants. As building managers, the application would assist the tenants by giving them the ability to manage student accounts inside the application, monitor the announcements, and receive the complaints as to be acted upon later. Since different roles require different functionalities in the application, we had decided to divide the application into two pages, one for the tenant, one for the student.

Graphical user interface, application

Description automatically generated

**Figure 1**. Mock-up of the application

As shown in our mock-up **Figure 1**, the application starts at the homepage, where the user can choose their role in the housing facility (tenant / student). They would also choose the tower where they live in or where they would choose to post/ view the events/ complaints on.

When the user chooses the tenant button, it will direct them towards a sign in page, where they can only access the tenant page through a username and a password. In the student page, the user can add events and post anonymous complaints to the tenants on that specific tower. They can also view the current rules. In the tenant page, the user can view the events and receive the complaints on that specific tower, as well as updating the rules as they see fit.

Here is the flow for our application.

Diagram

Description automatically generated

**Figure 2**. Flow of the application

1. **Project Planning and Execution**

Since the project will need to be done by a specific deadline, an efficient division of work was devised. Through the version control application Github, every partial implementation by each group member will be readily available and everyone will be held accountable to their part of work.

The devised schedule and planning map is presented below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Date** | **Time** | **Activities / Minutes** |
| Week 13 | Tue, 1 Dec 2020 | 13.00 - 16.00 | Have small meeting with our group to discuss about project analysis |
| (Online) Discussing initial vision, design mockup and flowchart |
| Wed, 2 Dec 2020 | 09.00 - 12.00 | Have small meeting with the teacher to share about our idea and show |
| our project analysis (Online) |
| Fri, 4 Dec 2020 | 13.00 - 16.00 | Discussed how we should divided the work (R 10 - OIL 0.30) |
| Week 14 | Tue, 8 Dec 2020 | 13.00 - 16.00 | Start working on our own division (Online) |
| Wed, 9 Dec 2020 | 09.00 - 12.00 | Have small meeting with our group to show each other’s progress (Online) |
| Fri, 11 Dec 2020 | 13.00 - 16.00 | Show the teacher about our progress and do the project based on work division (R10 OIL 0.30) |
|  |
| Week 15 | Tue, 15 Dec 2020 | 13.00 - 16.00 | Have small meeting with the teacher to show the progress that we've made, deliver interim version (Online) |
|  |
| Wed, 16 Dec 2020 | 09.00 - 12.00 | Working on our own division (Online) |
| Week 16 | Tue, 5 Jan 2021 | 13.00 - 16.00 | Working on our own division (Online) |
| Wed, 6 Jan 2021 | 09.00 - 12.00 | Working on our own division and push all the final code to github (Online) |
| Fri, 8 Jan 2021 | 13.00 - 16.00 | Have small meeting with our group to collect all the code together from github (Online) |
|  |
| Week 17 | Tue, 12 Jan 2021 | 13.00 - 16.00 | Do the connecting form for student page and admin page and start page |
| (Online) Creating final presentation |
| Wed, 13 Jan 2021 | 09.00 - 12.00 | Do the connecting form for login page with student page and admin page |
| (Online) Creating final report |
| Fri, 15 Jan 2021 | 13.00 - 16.00 | Final finishes with work, submission of deliverables, preparation |
| Week 18 | Mon, 18 Jan 2021 | 10.00 - 11.15 | Project Final Presentation (Online) |

**Table 1**. Scheduling table for the allotted project time

The work division was divided by having each group member work on a specific page’s functionality of our application. Joran focuses his work on the tenant page, Shanessa on the user log-in functionality, Tobias on the student page, and Matija on the submission forms.

Of course, working in a team on a project this scale would bring some risks and constraints. One possible constraint is the different ways each group member would implement their work, from using different variable names, methods, events, and so much more. Therefore, constant communication as well as the use of UML diagrams would be used as to ensure that variables, methods, and classes are consistently and properly used by everyone.

Due to the coronavirus pandemic, the amount of times we can meet up in person has become severely mitigating, impeding our productivity in implementing the software. This problem can be addressed by continually meeting up online and hold every group member accountable.

Of course, these trying times may allay several risks as well. Some of our group members were reportedly sick, one has their laptop broken, and others received some corrupted data. However, the constant use of Github allowed us to safely store back-ups of our builds and in the end our productivity weren’t fully hampered by these circumstances.

**Software**

1. **Finished Prototype**

Graphical user interface

Description automatically generated

**Figure 3**. Starting page (user login)

Graphical user interface, text, application

Description automatically generated

**Figure 4**. Student page

![Graphical user interface, application

Description automatically generated]()Graphical user interface

Description automatically generated

**Figure 5**. Tenant page

![Graphical user interface, application

Description automatically generated]()

**Figure 7**. Submission form

**Figure 6**. Form to add and remove users for the application

As seen on **Figures 3-7,** our application consists of multiple forms. The user login page (**Figure 3**) is the start of the application, where user input in the username and password fields would be checked by a set of user credentials already stored beforehand through our use of an offline database.

This database consists of two text files, one for the students and one for the tenants, located in the \bin\Debug\Database\Users directory inside the project folder. When checked, the application will browse the text file and compare the user input with the stored credentials; if both are the same by comparison, the user will successfully log-in to the page of their choice.

By using the AddUser form **(Figure 6**), a form put inside the tenant page, the tenant / admin can add more users and tenant accounts for the application. They can remove existing accounts as well.

By using this account system, hopefully the client can create informative user accounts that include their housing information, such as “TobiasA237”, which is an example where the student named Tobias lives in tower A and in room 237. By having this kind of user account naming format, it will make the user experience more easier to manage.

The student page (**Figure 4**) contains the non-editable rules board, while the user can add anonymous complaints and events/announcements through using the submission form (**Figure 7**) that also features a calendar where you can pick dates for the event, as well as automated username addition to the announcement. Obviously, these username additions are not included in the anonymous complaints submission.

The tenant page (**Figure 5**) contains the editable rules, complaints, and events board where their access also allow them to remove posts from these boards. The AddUser form can only be accessed through this page, since it is an important application control functionality. The complaints gathered from the students will be shown with the identity of the submitter only through this page as well, so the tenant could properly assess the complaint / input.

1. Methodology

Diagram

Description automatically generated

1

AddUser

**Figure 8**. UML Diagram

**Figure 8** is the UML diagram for our software. It describes the connectivity of our classes with the multiple forms, as well as the notable functions and constructors used throughout the program’s execution. Lines with arrows denote the forms that uses some parts of a class pointed by the arrow, while straight lines indicate the connectivity of the forms and the classes that were used to transfer data with each other.

The main classes used for the application is the Tenant and Student classes. They store data in the same way : through the use of 3 string lists where each correspond to the 3 boards interacted by the user (rules, complaints, and events board). For example, through the submission form, the student adds event strings to the events list, and complaints to the complaints list. When the form closes, the Student class containing these lists were transferred into the StudentPageForm through a constructor that receives this Student class. Therefore, the data will stay intact, and can be further manipulated.

Through these constant transfers of data along multiple forms, the application will always have the data synchronized throughout its runtime. A notable form is the Login form; because it is the starting page for both the tenant page and student page, it acts as an “intermediary” hub that transfers both Tenant class and Student class through both pages.

The database functionality uses the System.IO namespace, which allows for the reading and writing of files. It reads and edits the text files containing the user credentials.

**Conclusion**

These past few weeks working on the project have been an instructive look towards the collaborative aspect of software development. Learning to divide work, receive constructive input, and adding / removing from other people’s work allowed us to improve upon the dynamics of teamwork that is necessary when entering the workplace.

Of course, there were many problems we have encountered along the way. At first, we struggled on finding a solution to connect forms with each other, but we eventually found the solution from further lessons with our instructor. We also had difficulties in working with github, but we managed to also use other ways as to back-up and update each other’s work.

The application, despite working properly, were still rife with problems that can be solved in the future. One problem is that since the application only works through a single thread, some forms are dependent on each other and cannot be closed; these forms were then merely hidden instead of being closed, and these hidden forms would lead to memory problems in the future. By using multiple threads, this problem can be assuaged and the forms can be closed safely, but this topic is outside of the scope of our current learning.

Another problem revolves with the inconsistencies of some of the variables named throughout the application. This may lead to some confusion regarding the reading of the code. This is due to working in teams without a complete UML diagram, and so everyone used different names for similar functionalities/variables. A more comprehensive UML would be recommended the next time so work can be done much more efficiently.

Finally, the use of multiple constructors on different forms to transfer data may be a good method, but there might be better methods to use in the future that are more efficient. But, the software as it stands currently is workable, and it is the with the hopes of us software implementors that it provides the means to manage these housing complexes properly for Student Housing BV.