



SOFTWARE PROJECT MANAGEMENT SYSTEM

OBJECT ORIENTED ANALYSIS AND DESIGN

Gökay Dinç | 2019510032

Emircan Tepe | 2019510073



TABLE OF CONTENTS

1 INTRODUCTION	3
2 REQUIREMENTS	5
2.1 METHODS	7
3 UML DIAGRAMS	8
3.1 CLASS DIAGRAM	8
3.2 USE CASE DIAGRAM	12
3.3 SEQUENCE DIAGRAM	13
3.4 ACTIVITY DIAGRAM	14
3.5 STATE DIAGRAM	15
4 IMPLEMENTATION	16
4.1 DATABASE	16
4.2 ENERAL VIEW OF UI	17
4.3 LOG IN SCREEN	18
4.4 CUSTOMER SCREENS	19
4.4.1 MAIN SCREEN	19
4.4.2 MESSAGE SCREEN	20
4.5 MANAGER SCREENS	21
4.5.1 MAIN SCREEN	21
4.5.2 MEET SCREEN	22
4.5.3 TASKS SCREEN	24
4.5.4 TEAM SCREEN	25
4.5.5 REPORT SCREEN	26
4.5.6 MESSAGE SCREEN	26
4.6 ITWORKER SCREENS	27
4.6.1 MAIN SCREEN	27
4.6.2 REPORT SCREEN	28
5 CONCLUSION AND FUTURE WORK	29
6 REFERENCES	30

INTRODUCTION

Software programs are nowadays everywhere where it can give a solution, quickness, easiness and even sometimes intelligence. In the first phase of software programs in history, the programs are **limited** (Throughout the report, bold texts are used to refer to keywords.) and **low** in means of complexity because of the hardware limitations and discoveries that were not made yet. It has been rapidly improving and taking place nearly everywhere. As a natural result of this improvement, the complexity of software programs and the number of people working on a project has been increasing, and it still continues to grow rapidly with a big acceleration, e.g., windows 10 has approximately 50 million lines of code and Google has 2 billion lines of code¹, these are not projects that could be managed by a few hundred people.

This complexity also leads to failures, as stated in CHAOS² report, just 17% of the software projects were finished with a success in means of time and planned financial situation and 33% of the software projects were completed, either with a planned time and financial situation in 2020.

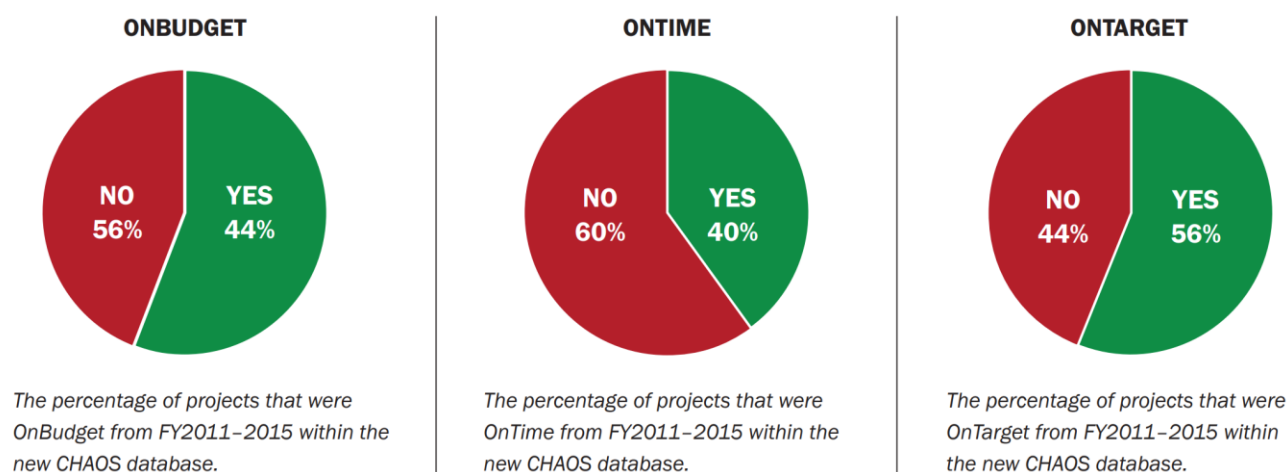


Figure 1: Pie chart from CHAOS report

There are many attempts to decrease complexity, make programs more efficient, and have less development time, called “Software Project Life Cycle Models” in the area of **Project Management**. That is exactly where the project, Software Project Management System (SPMS, Afterwards SPMS will be in use as abbreviation for *Software Project Management System*) , got inspired.

The purpose of SPMS is to **decrease the complexity** of software project management by using the power of a software program, not ironically. It is designed for the usage of **every person** carrying responsibility and authority in a project, including every level and kind of programmer and even customer.



The subject and purpose explained above, of the project, were decided on **19 March 2022**. The remaining time was 8 weeks and currently, we have a total of 4 weeks. A detailed Gantt Chartt can be found below.

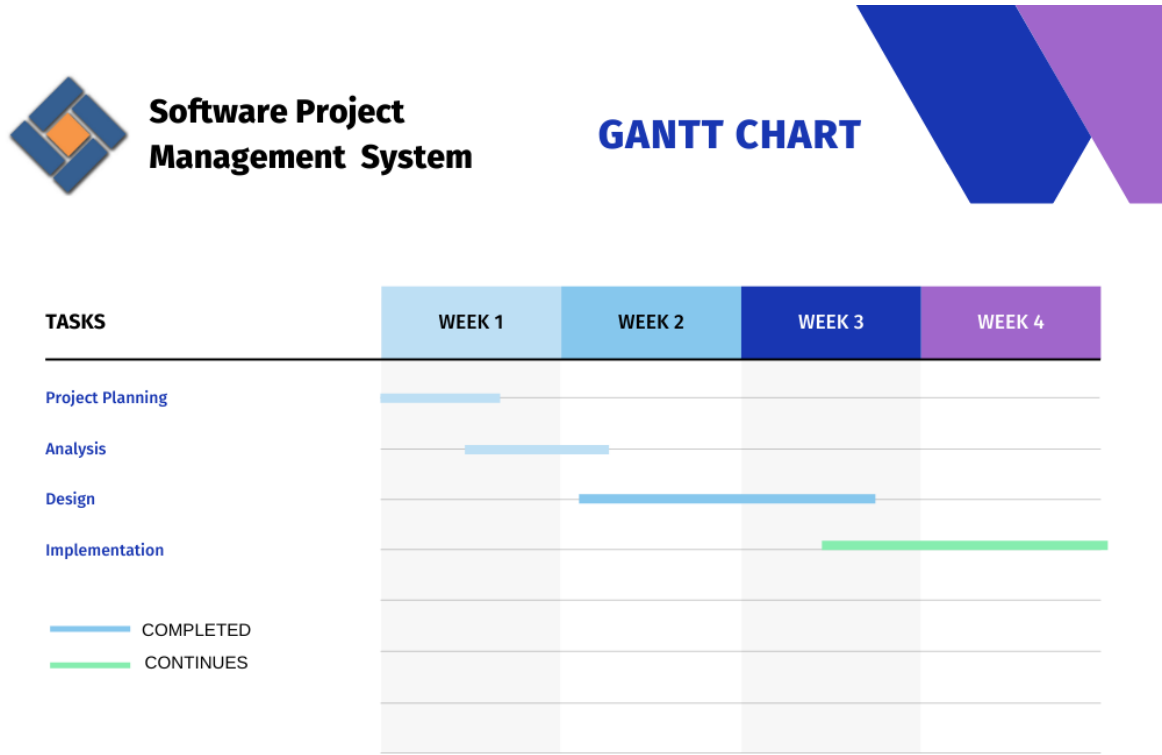


Figure 2: Gantt Chart, shows the current situation, created in [Canva](#).³

There are number of limitations resulting from the given time interval. One of them is to make it runnable also in the internet environment. The second one is to use encryption for authorization purposes. Despite these limitations, the designed program will still have the capability of showing core requirements explained in the next chapter.

The resources are as follows:

- ◆ Java is currently in use as a main programming language.
- ◆ MySql is currently in use as a database.
- ◆ Eclipse is currently in use as an IDE.
- ◆ Online tools, [Lucidchart](#)⁴ and [visual-paradigm](#)⁵ is in use for UML diagrams.



REQUIREMENTS

As stated previously, the SPMS stands out with the aim of decreasing the complexity of the management process in the projects that many people are working on, such as from different levels and kinds of programmers (senior, junior, front-end, back-end, etc.) managers and customers, which means that every person carrying responsibility and authority in the conducting project has a place in the SPMS. The designed program aims to make **communication** between every person in the project **efficient** and **faster**, which also leads to having a faster and more efficient development process, and also leads to have a better **success** at the end of the project.

The project has many user requirements, which is clearly expected when taking the capability of the designed program and the diversity of the user types using the program into account. All these user requirements are listed below in a logical order of a designer.



- ◆ All types of users have different actions and capabilities. For example, a programmer cannot accept or decline a project coming from a customer while a manager can.
- ◆ Having a login screen means that it is a need to have a database containing the informations of all users in the program, meaning **usernames** and **passwords**.
- ◆ A manager shall be able to request a report from a programmer.
- ◆ A programmer shall be able to write a report and submit it.
- ◆ A manager shall be able to see who is working on what.
- ◆ A manager shall be able to see the information of a programmer such as name-surname, experience, position, salary.
- ◆ A manager shall be able to create a new task and assign it to a programmer.
- ◆ A manager shall be able to see all information about a project, such as which programmers are included, current situation as percentage.
- ◆ A manager shall be able to create a meeting.
- ◆ A programmer in a team shall be able to choose she/he will join the meeting of the team.
- ◆ A customer shall send to a message to the manager of his/her project.
- ◆ A manager shall be able to export the report written by a programmer as a PDF.



METHODS

All methods and their return types are given below.

METHOD NAME	ARGUMENTS	RETURN TYPE
insertData	insertSql: String	Void
deleteData	deleteExe: String	Void
updateData	updateExe: String	void
selectData	selectExe: String	ResultSet:Java.Sql.ResultSet
actionPerformed	ActionEventE: java.awt.event.ActionEvent	void
getProject	-----	Project:Project
setProject	Project:Project	Void
isWorkingOnProject	-----	Boolean:Boolean
getSalary	-----	Int:Int
getExperience	-----	Int:Int
getKnowledge	-----	String[:String
getCurrentProject	-----	Project:Project
getId	-----	Int:Int
getTitle	-----	String:String
setSalary	Int:Int	-----
setExperience	Int:Int	-----
setKnowledge	String[:String	-----
setCurrentProject	Project:Project	-----
setId	Int:Int	-----
setTitle	String:String	-----

Figure 3: Method table



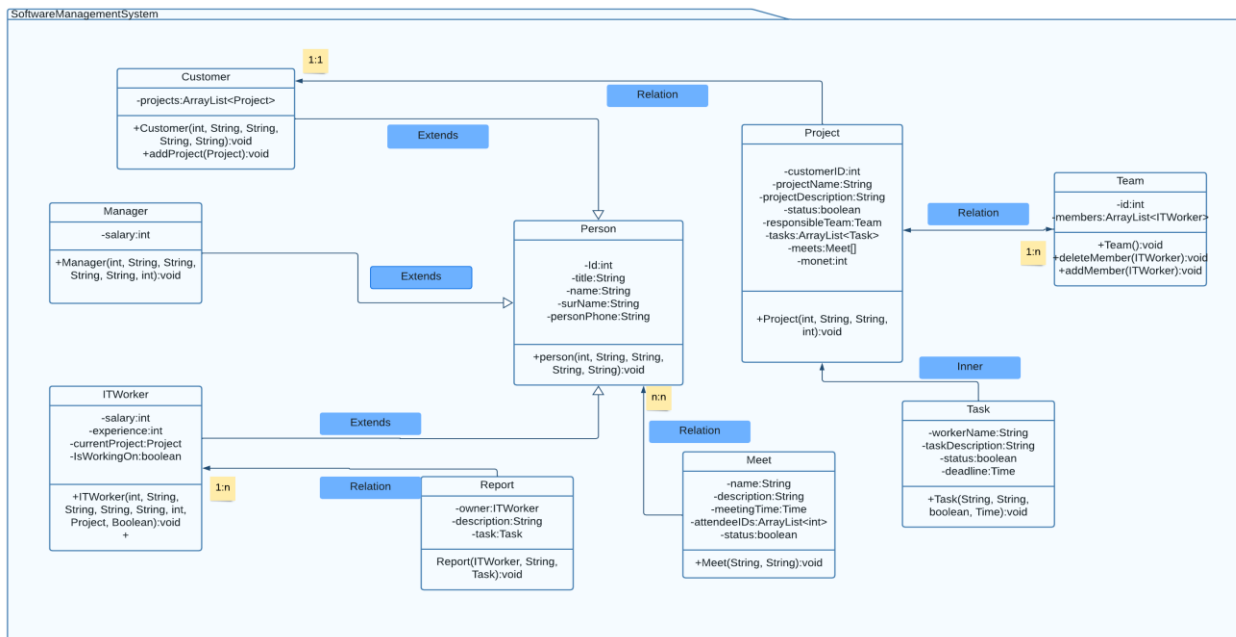


Figure 4.1: SoftwareManagementSystem package

As shown Figure 3.1, the *SoftwareManagementSystem* package consists of core classes for the **SPMS** project. Every person having an active role in the system is represented as classes. When doing so, Object Oriented Design principles are used, for instance, as **Extends** arrows refers, person **abstract** class is used for **Customer**, **Manager** and **ITWorker** classes because they share many common properties such as name, title or phone.

According to the listed [user requirements](#) in chapter 2, the objects needed to represented as classes in the program are not only about persons, so **Meet**, **Report**, **Task**, **Project** and **Team** classes also exist and they have relations with other classes, e.g., Meet class has *many to many relations* with person class, meaning that a meet can have many person while a person can also have many meet. Another example, Report class has a *one to many relations* with ITWorker class, meaning that an ITWorker can have many report while a report can have only one ITWorker.

Lastly, as can be observed from the diagram, Task class is created as **inner class** of Project class because it is impossible and needless to have a Task object outside of the Project class.



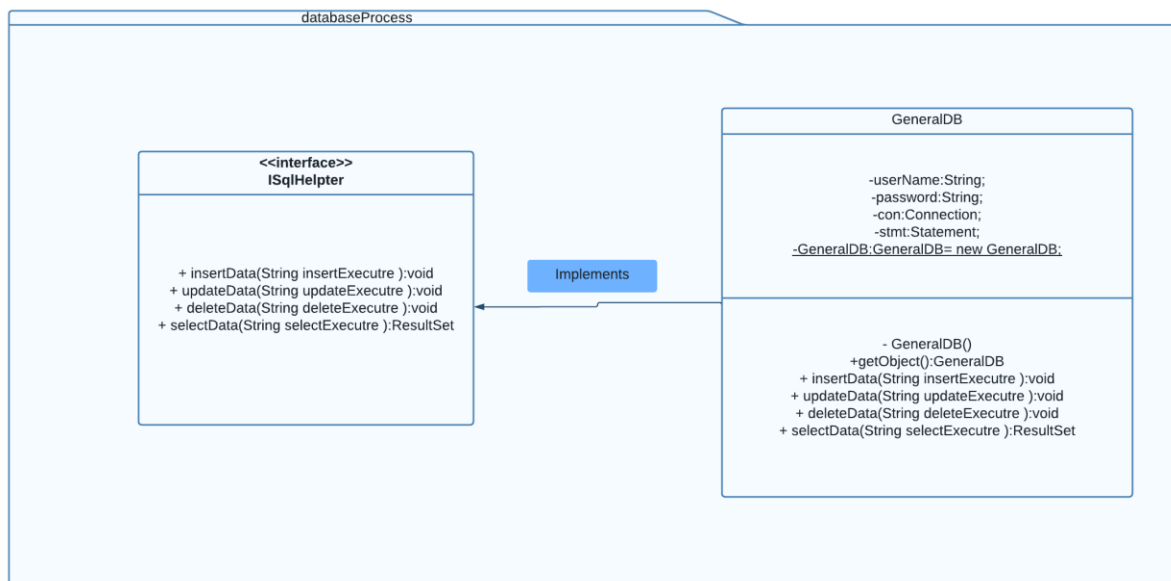


Figure 4.2: *databaseProcess* package

Figure 3.2 is the UML Class diagram package for DB (After this point, DB will be in use as abbreviation for the database) , consists of basically one interface and one class that implements the interface.

It should be highlighted that **Singleton** design pattern is used for DB operations. As can be seen from the below Figure 3.3, uiScreens package, there are needs for DB queries in nearly every class. So, instead of creating the connection with DB everytime a query is needed, only once (during compiling time) it is established and called this object when needed. It also prevented from creation of a second object of GeneralDB class by making the constructor private, meaning that it is accessible from only the class itself.

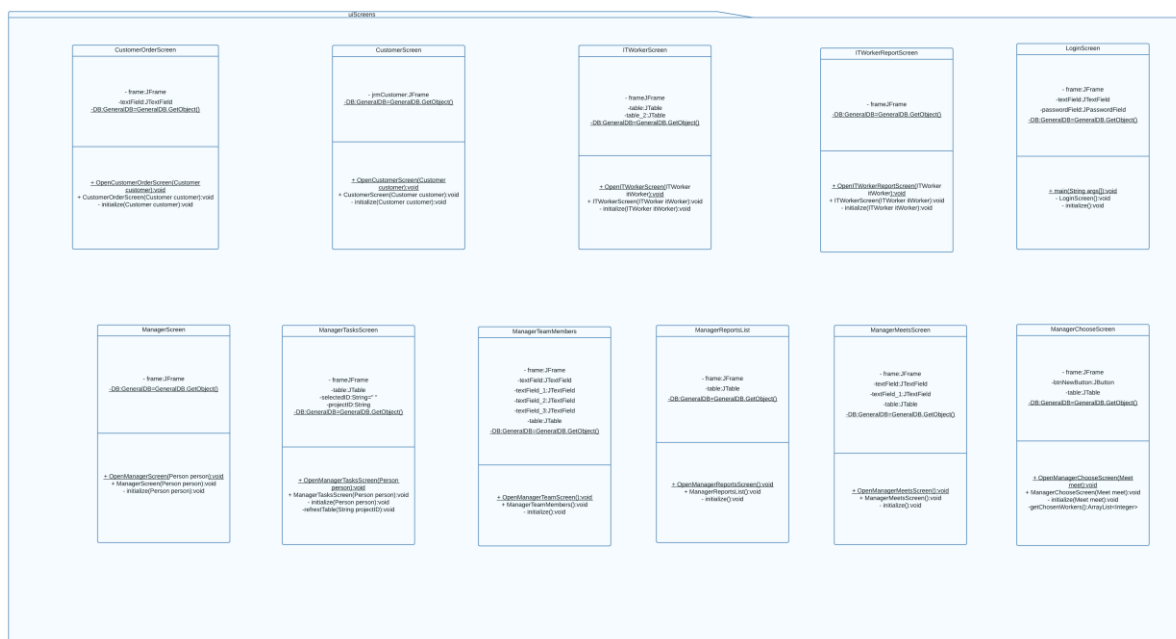


Figure 4.3: uiScreens package

All classes related with **User Interface** is inserted in the uiScreens package, also in practice too. The first observation should be done is to realize that the main function (that is the first code part will be run after compiling process) of the entire program is inserted in LoginScreen class, that is exactly where everything starts. All these classes are as created in a **flow** manner, basically following below steps

1. Login Screen → Determining who is it.
2. Opening needed screen, e.g., manager screen for a manager or customer screen for a customer.

The flow does not end at the second step, it continues but with different branches for different actors although these 2 steps are common. (What is meant here should be better understood in below *Use Case* diagrams.) For example, If a customer, after successfully logging, presses button labeled as *Meets*, the related listener will be triggered and ManagerMeetsScreen will be opened. The information (Means object(s) in the system) is carried between these UI classes by giving and taking them as an argument, and also stated before in above, when needed to have a DB query, the reference to the object (connection) created while compiling time is used in every UI class.



USE CASE DIAGRAM

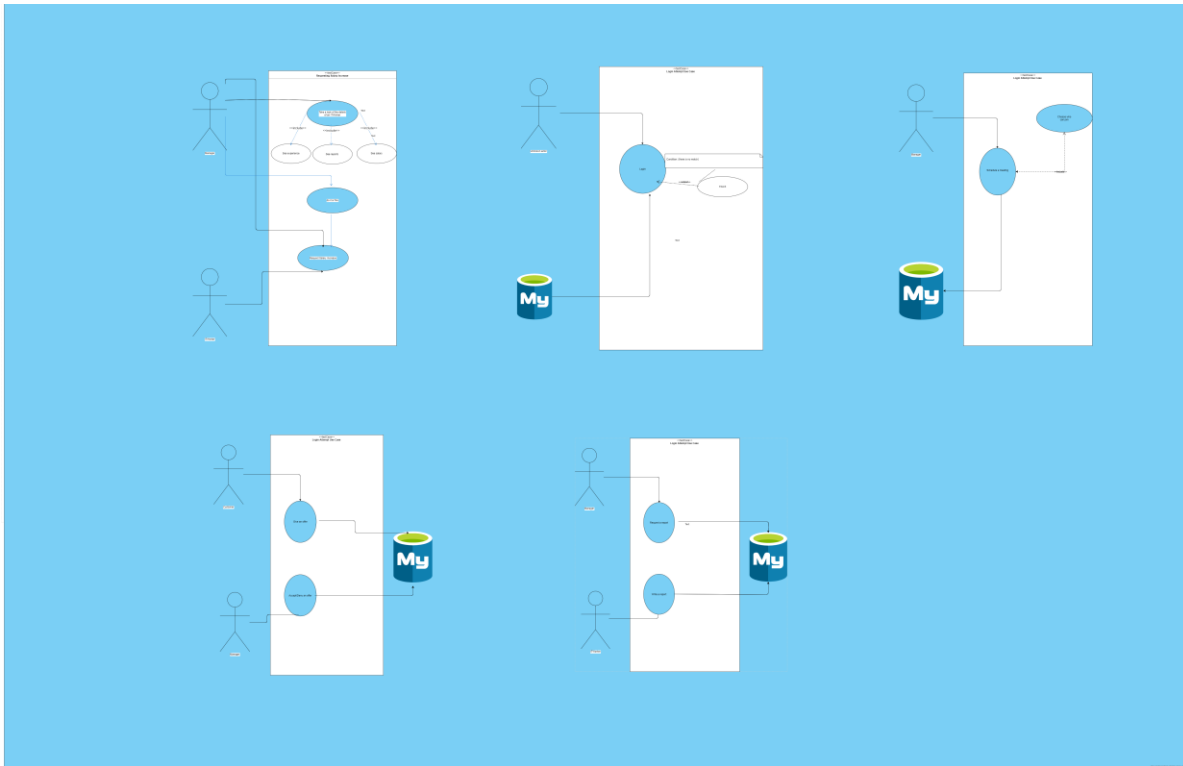


Figure 5: Use Case diagram

The user requirements listed in chapter 2 are represented as use cases in the form of five different sub-use case diagrams. As stated, many times in the entire document, the main key property of the program is login screen, whose use case can be found in the above figure. Also, all the persons (customer, Manager or ITWorker) and other **Actors** such as DB are represented in above cases.

Other use cases, such as a salary increase request (by an ITWorker) and accepting/denying it (by a manager) , requesting a report (by a manager) and submitting it after writing (by an ITWorker) or scheduling a meet and setting who is going to be able to join (by a manager) , are all represented above diagram.

SEQUENCE DIAGRAM

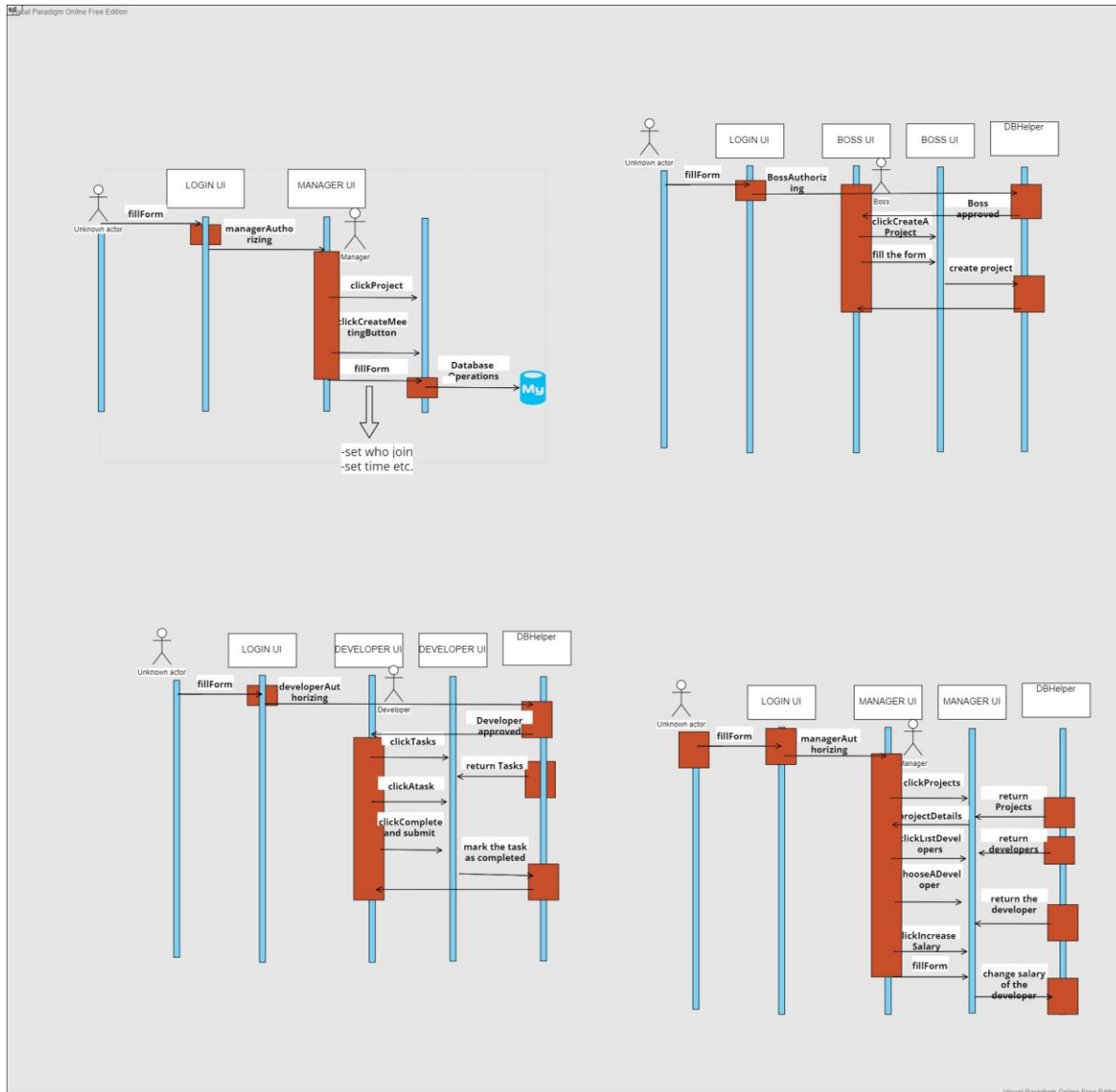


Figure 6: Sequence diagram

Four different sequence diagrams can be found above, Figure 5. Sequence diagrams are used for showing the flow of the program in a detailed manner, that can also be called as illustration of use cases in detail with objects.

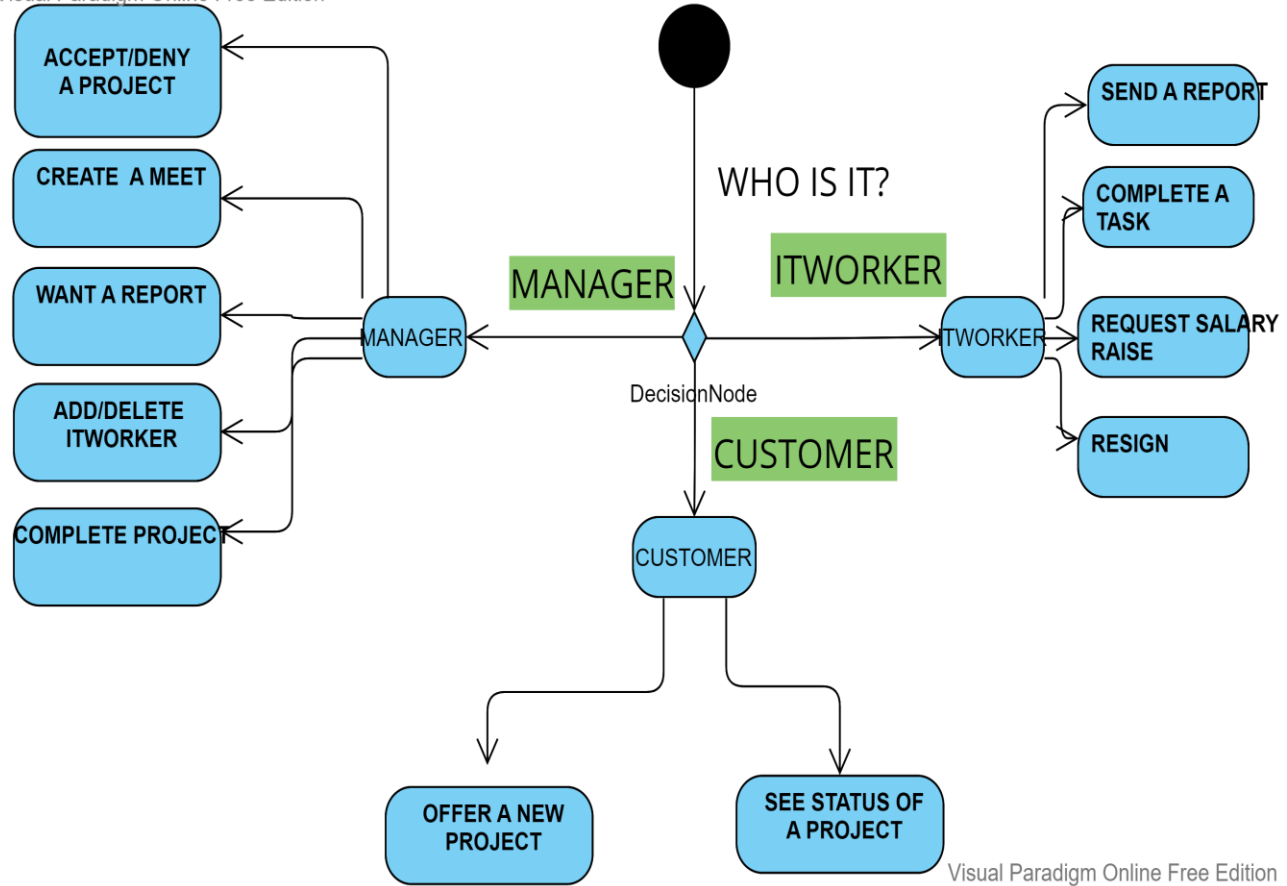
For example, logging use case ([Chapter 3.2](#)) is used at the starting point of every sequence diagram to show that it is a need for anything in the program. After successfully authorizing of the user, for example, in sequence diagram 1 in figure 5, Manager clicks Meets button and then creates a meeting by filling the form, then after completing the form, the result (new meeting data) is processed to the DB by using DBHelper object, that is the same object for entire program thanks to the **Singleton** design pattern.



Other three sequence diagrams carry the same manner with the mentioned above one. Also, the lifetimes of created objects are showed by brown rectangles.

ACTIVITY DIAGRAM

Visual Paradigm Online Free Edition



Visual Paradigm Online Free Edition

Figure 7: Activity diagram

In figure 6, a detailed activity diagram, representing the flow of the program in a **higher level** than the sequence diagram in [Figure 5](#) is given. As stated many times, it is most crucial point for the program to determine who the user is in first place, then everything takes shape according to this decide.



STATE DIAGRAM

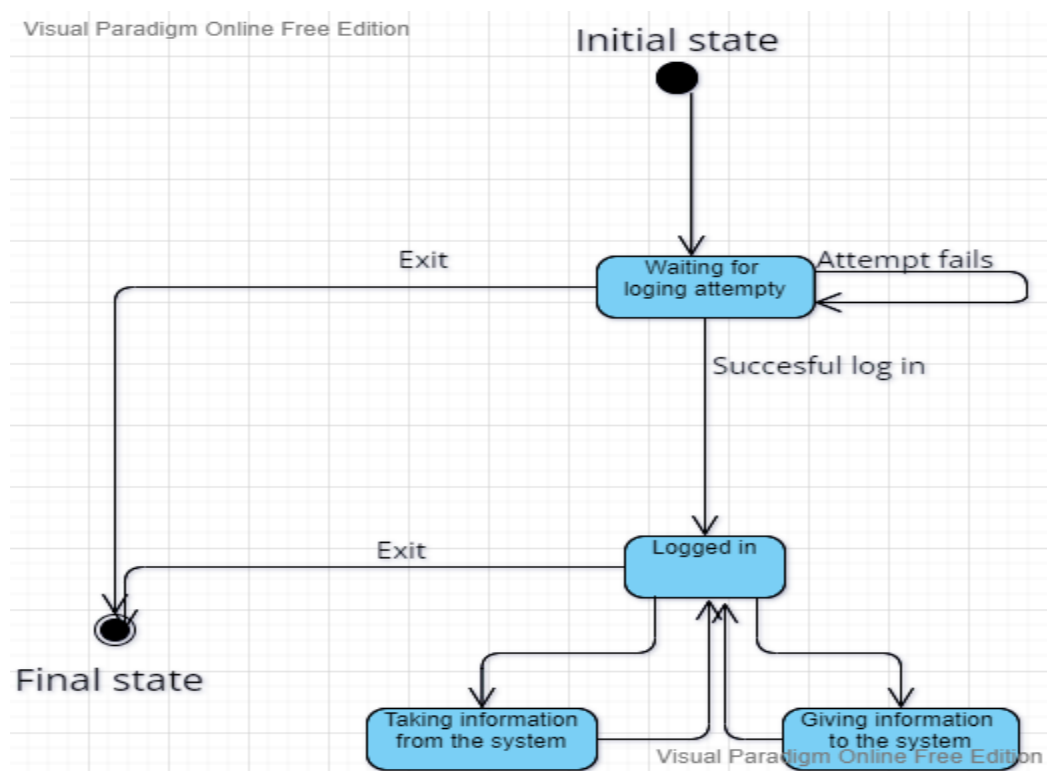


Figure 8: State diagram

The state diagram is also a behavioral UML diagram type, just like the activity diagram. They are so similar that one can be confused easily. State diagram differs from activity diagram about a few, but important points, such as state diagram depends on **actions** while activity diagram more focuses on activities.

The overall state of SPMS can be seen from Figure 7, it is currently in the very abstracted form, briefing all system states. Basically, there are 2 main states, and 2 sub-states, which are as follow.

First state is waiting for logging state. The system waits for an attempt to be logged in. While there is no successful attempt to log in, the system stays at the same state. If any successful logging occurs, the system jumps to the second main state, that is **Logged in**.

The two substates are below, as child of the Logged in step. While there is no attempt to exit from the program, the system is either giving or taking information state, such as

creating a meeting → **Giving information**

Listing ITWorkers → **Taking information**



IMPLEMENTATION

Before deep diving into UI screens, a few important paragraphs about DB is given below in first to have a better understanding of the implementation.

DATABASE

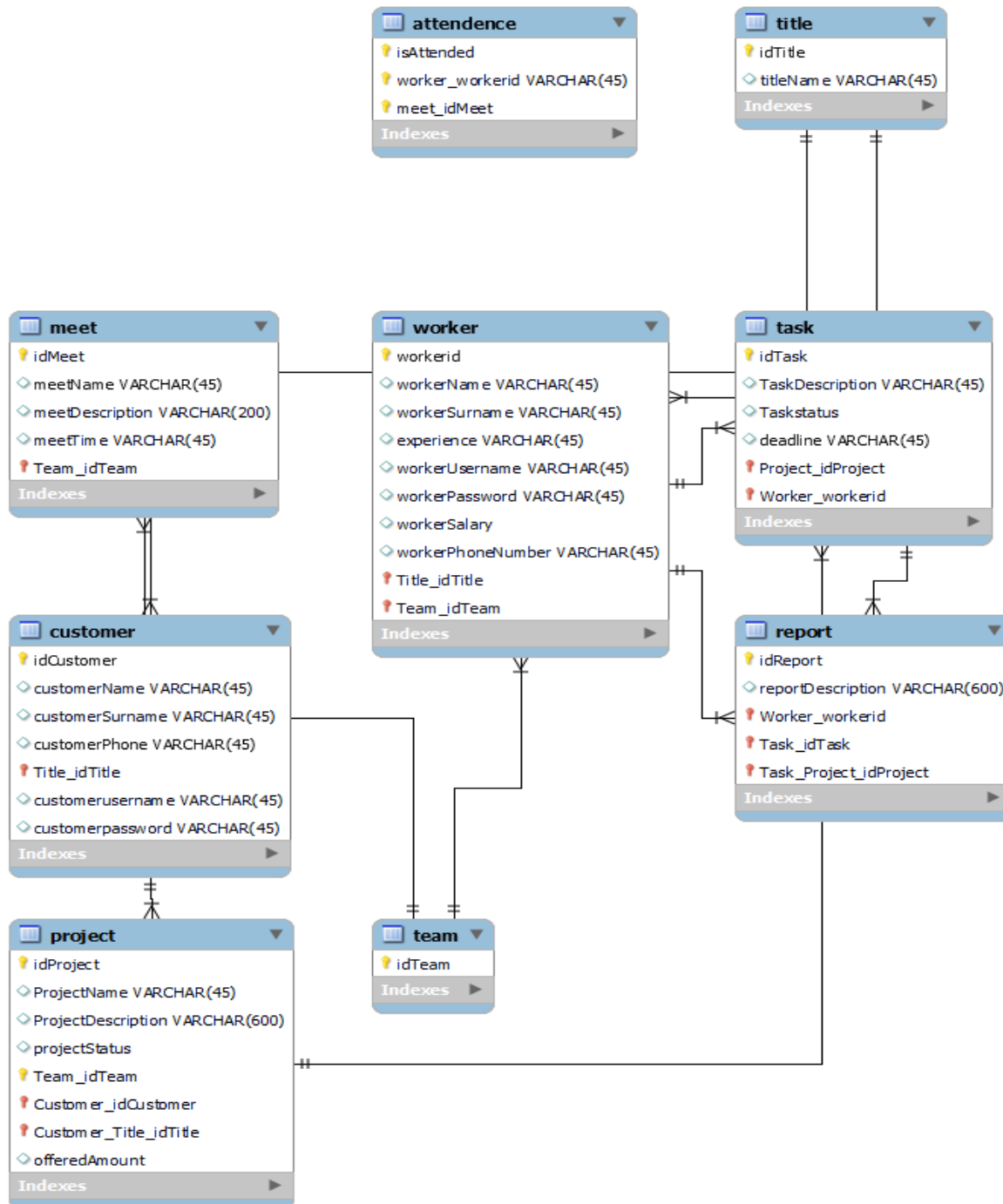


Figure 9: DB diagram, created in MySQL Workbench⁶



As can be seen from the above figure, our DB carries all the needed information to hold the system active according to the given user requirements and classes. There are many relations between different tables. These relations are set up in a logical manner, e.g., **meet** table has a **teamID** column described as **foreign key**.

Whenever a new information is added/deleted to/from the DB, it is refreshed directly by calling our **refreshTable** function that updates the necessary table(s) or label(s), so any occurring changing will immediately take place in the program in this way.

GENERAL VIEW OF UI

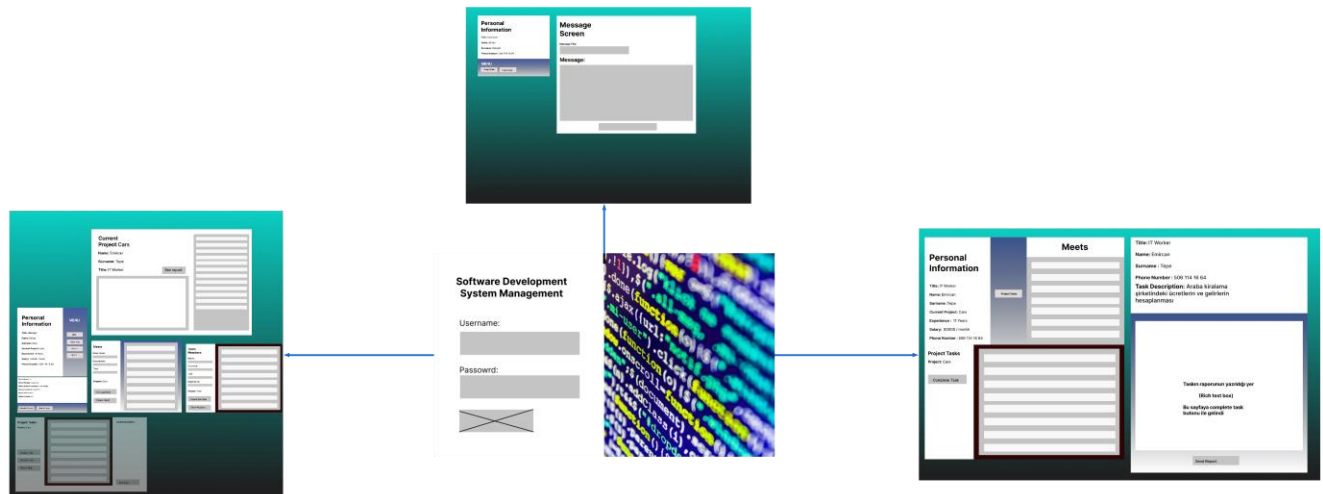


Figure 10: Design of UI, created in Figma⁷

The above figure was created in the design phase of UI screens and given here to give a general idea of the UI screens. It is divided by three 4 parts, namely as follows:

- ◆ Log in screen
- ◆ Manager's screens
- ◆ Customer's screens
- ◆ ITWorker's screens

These are the persons in the system who carries a responsibility and different kind of authority, so this logic is also implemented during UI design phase, and also practiced too as can be seen below paragraphs.



LOG IN SCREEN

Below all interfaces and their background codes is explained in detail. As stated many times throughout the document, flow of UI screens starts with the core UI, namely *Login Screen UI*.

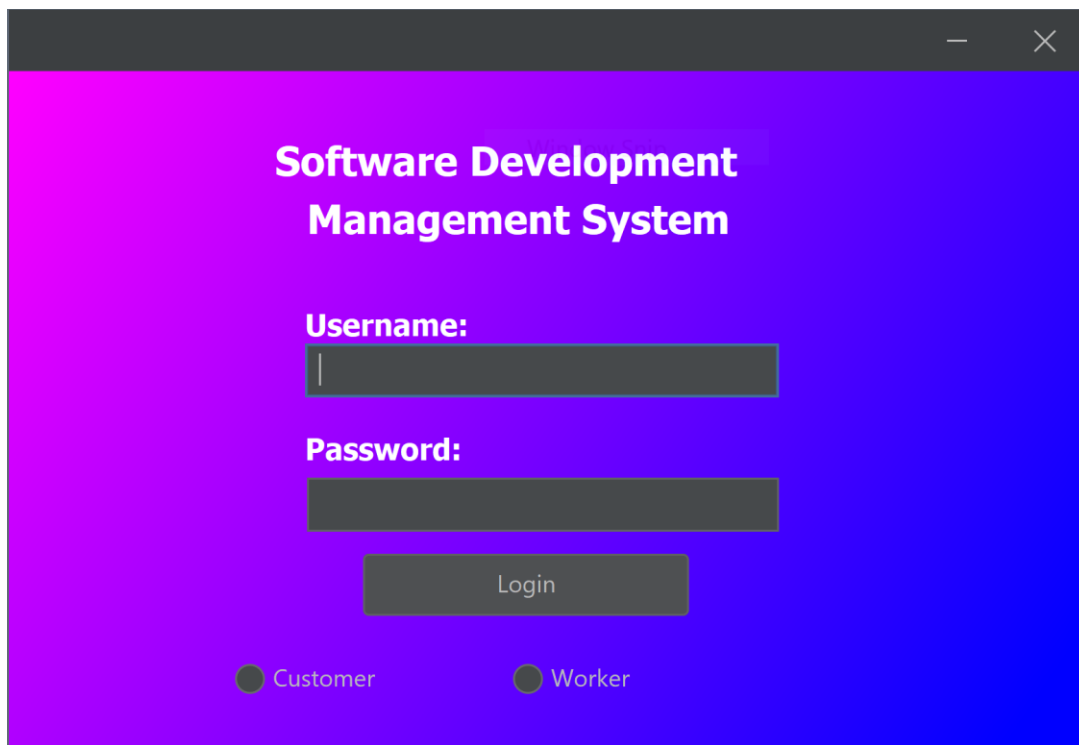


Figure 11.1: Login Screen UI

It is needed to fill *Username* and *Password* boxes correctly to log in successfully, also one of *Customer* or *Worker* box should be ticked that should also be matched with the given information. The

There is a total of 3 successful continued flow possibility; **Customer**, **ITWorker**, **Manager**. Below these 3 different flows is analyzed in detail.



CUSTOMER SCREENS

MAIN SCREEN

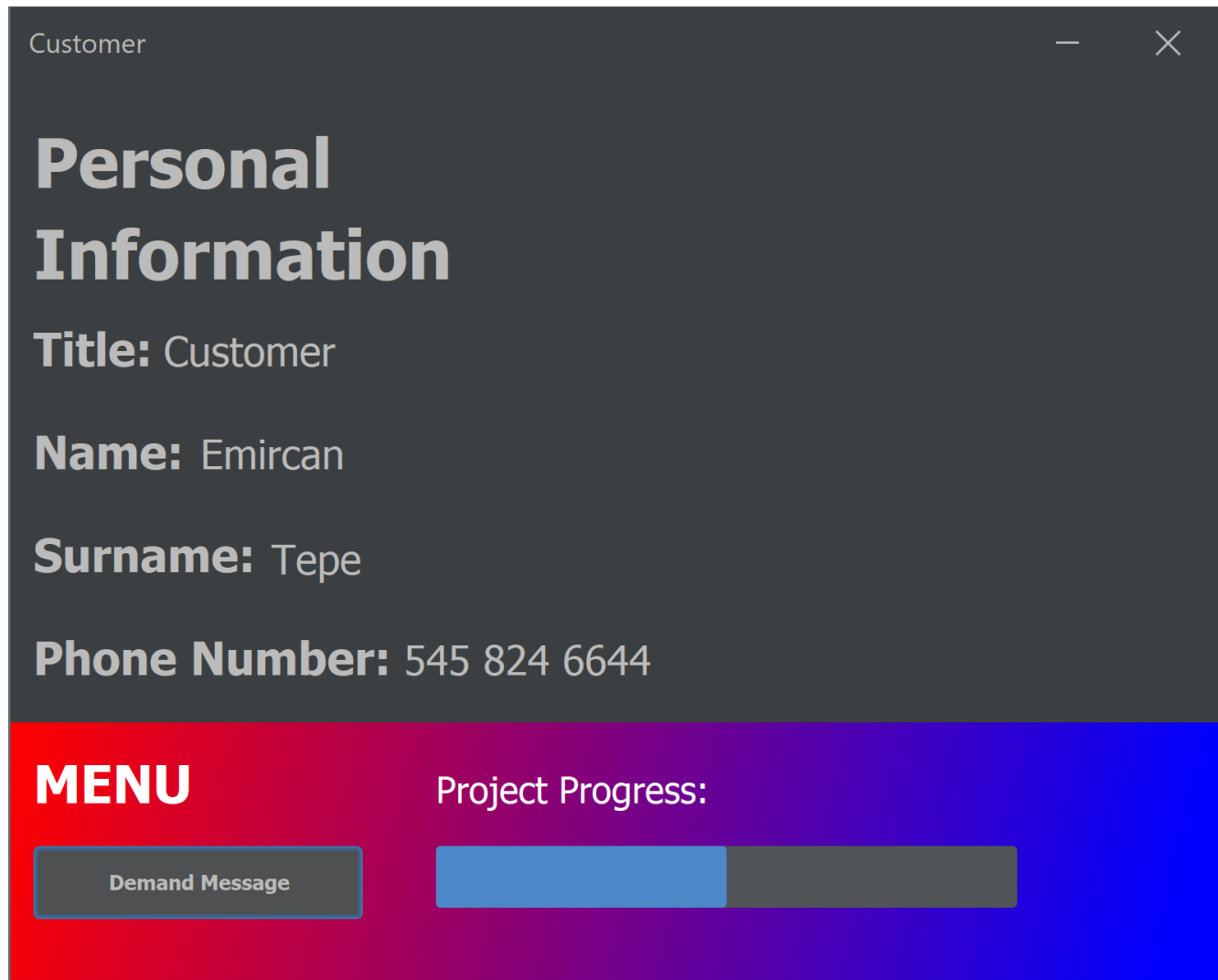
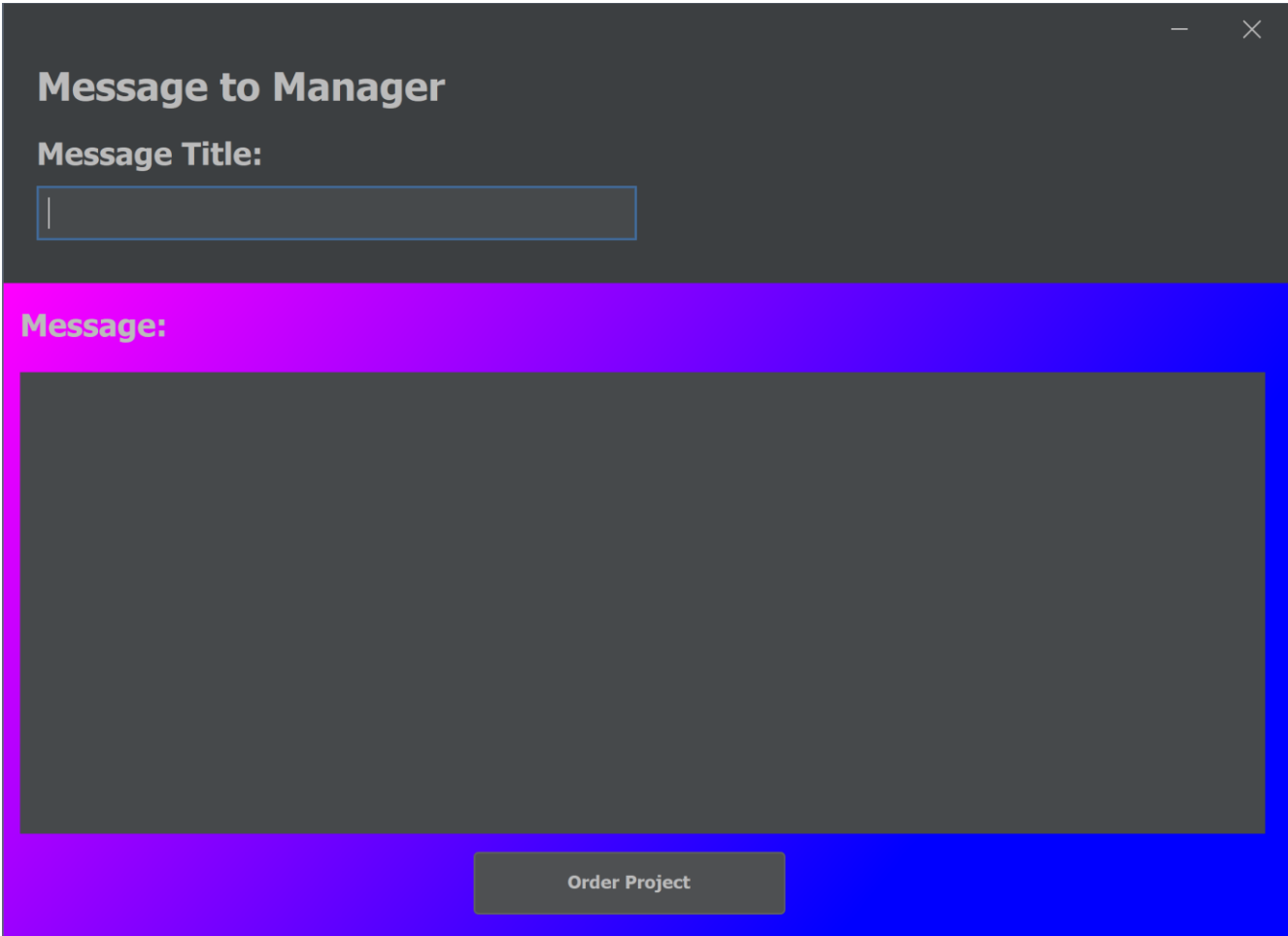


Figure 11.2: Customer Main Screen

Here is the Customer's main screen showing customer's general information with the current situation of the project (represented as $\frac{\text{Completed tasks}}{\text{Total tasks}}$) and giving the option of sending message to the manager about the project.



MESSAGE SCREEN



Message to Manager

Message Title:

Message:

Order Project

Figure 11.3: Customer's message screen

The above figure shows the opening screen named *Customer's message screen* when the customer clicks “**Demand Message**” button. It has a total of 2 text box, each of which is needed to be filled by the customer for sending it successfully to the manager of the project. When clicking the “**Order Project**” button, the written message is directly sent to the DB and the message will have seen to the [manager's main screen](#).

MANAGER SCREENS

MAIN SCREEN

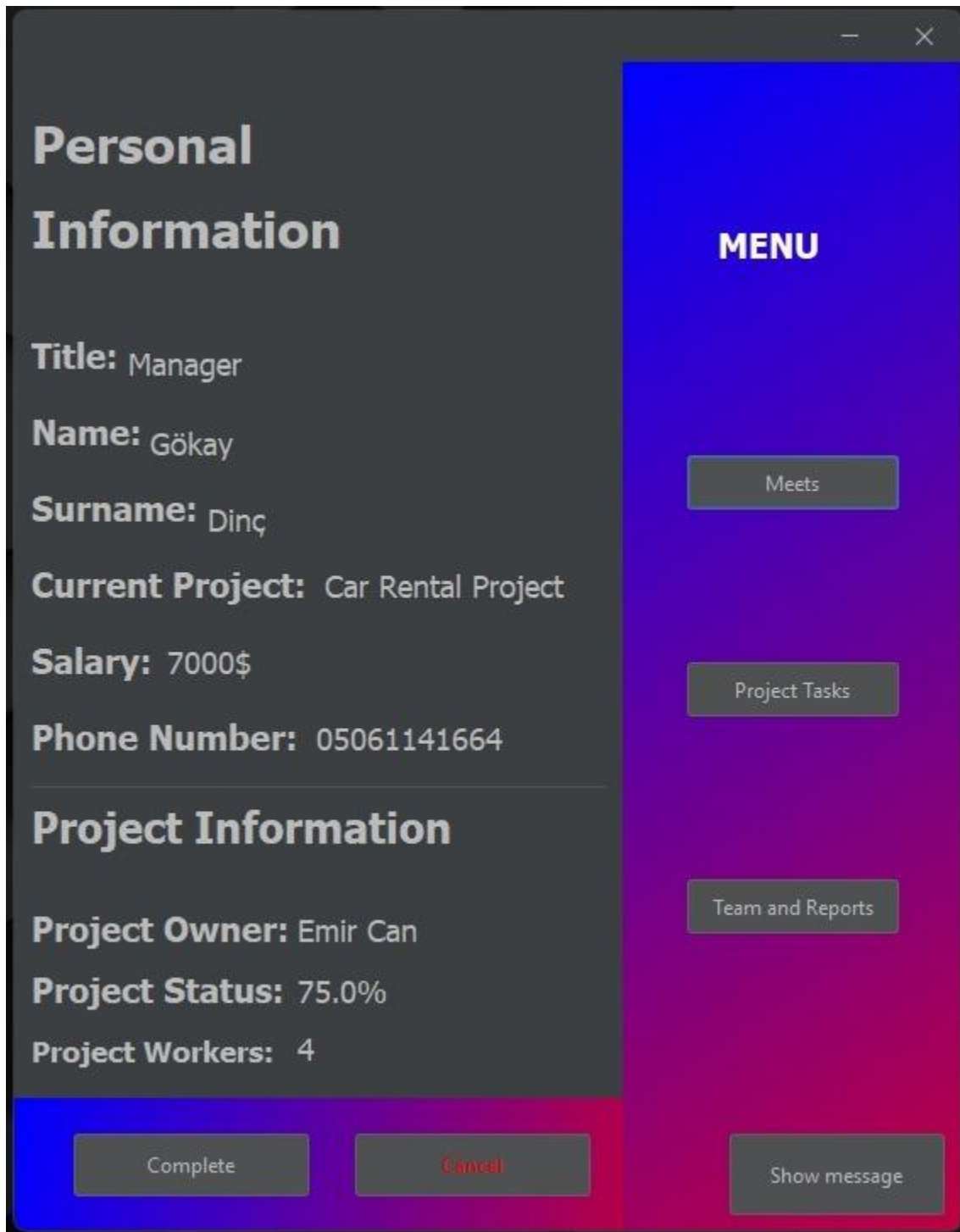


Figure 11.4: Manager main screen

After successfully logging to the system as a manager, the screen the manager will face is the above figure that is the main screen of the Manager. The left side is left for showing the information about the manager and, above side of the left panel is left for the project information. The right side has four buttons, each of which has different purposes and actions explained below separately.



MEET SCREEN

Meets

Meet Name:

Description:

Time:

1

1

2022

16

28

Project: Project Name

Arrange Meet

Cancel Meet

MeetId	Meet Name	Description	Time
17	Çeşitli yazılımların T...	Çeşitli yazılımlar ha...	5/8/19-17:8
19	Senkronizasyon alg...	Senkronizasyon alg...	8/7/26-16:30
21	Alınan karar	Alınan kararlar baş...	7/11/2026-05:22
23	Araba Yazılımı	Araba Yazılımının b...	6/6/2026-05:22
24	Ağır koşullar	Pandmei nedeni ile ...	1/1/2022-03:0
37	asdasdasdsa	sadasdasdasdasd	1/1/2022-00:0
43	gfdsgfd	fsdfsdfsdfs	1/1/2022-00:0
44	fdgfdgfd	gfdgfdgfd	1/1/2022-00:0
45	asdsadas	sdsadasdasda	8/7/2029-07:7
46	emircan	tepe	1/1/2022-00:0

Figure 11.5: Manager's meet screen

If the manager clicks the “**Meets**” button, the figure 8.5 is the screen she/he will see. The left side is left for adding/cancelling a meet while right side is left for a table, showing the created meets and their information.

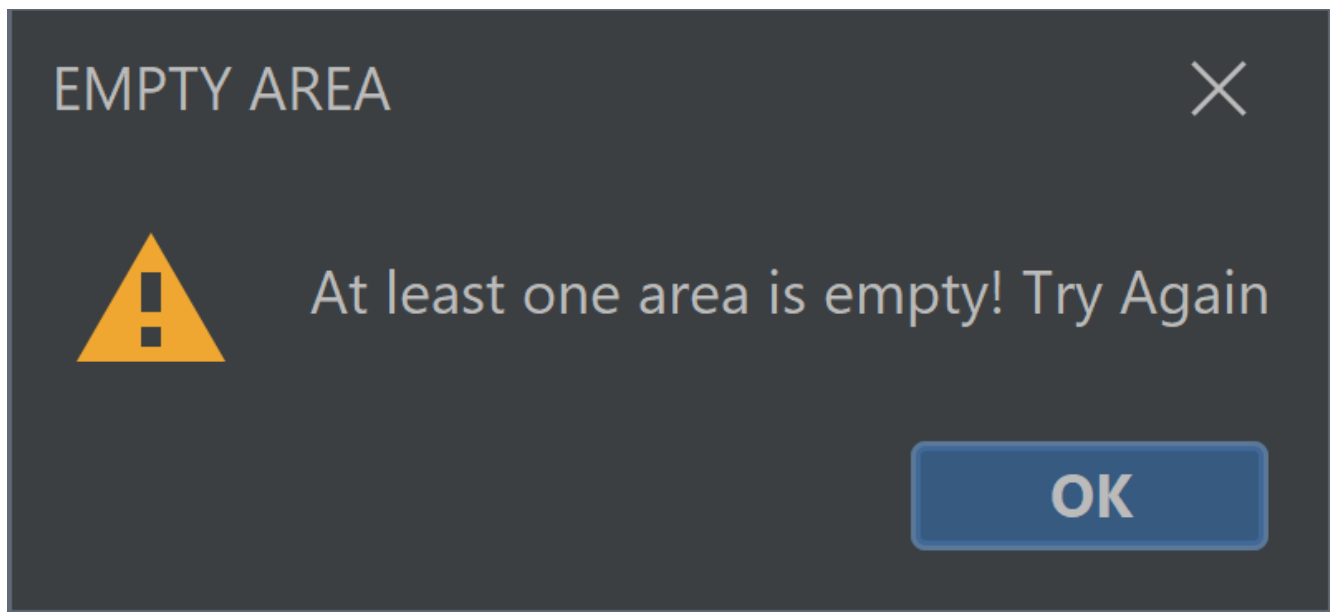


Figure 11.5.1: Error Box, when arranging a meet with empty areas

The above figure shows an example of the arranging a meet action, if one of the needed areas (name, *Time* or *Description*) is empty, the program will detect it before trying to create a meet and raise an error box.

TASKS SCREEN

Project Tasks

Project Name:
Car Rental Project

Accept Task

Decline Task

Delete Task

Task Description:

Deadline:

1 1 2022

Worker Id:
1 Gökay Dinç

Add Task

Task Id	Task Description	Deadline	Worker	Task Status
8	Zamanlama fonksiyonları	15.04.2022	Selami Ak	Completed
15	Web Scrapping Kodlan...	13/7/2029	Ahmet Velioglu	Not Completed
20	Breadth First Search Fon...	13-3-2025	Selami Ak	Completed
23	Path finding yazılması	5-5-2027	Selami Ak	Completed

Figure 11.6: Manager's tasks screen

If the manager would like to see information about tasks, add a new task or process some information about a created task such as accepting, declining or deleting it, the UI that user needs to go is above figure. As can be seen from the above screenshot, the manner for designing the UI page is so similar with the above ones, left side carries some functionality while right side is left to show information. In this UI, the right side is nothing more than a table, showing information about meets column by column.



TEAM SCREEN

Team Members

Name:

Surname:

Title:

Experience

Project Name:

Car Rental Project

Show Reports

Delete Member

Worker Id	Name	Surname	Title	Experience
1	Gökay	Dinç	Manager	4 years
2	Ahmet	Velioglu	Programmer	2 years
5	Selami	Ak	Programmer	2 years
6	Hülya	Uzun	Developer	3 years

Figure 11.7: Manager's team screen

As can be seen from the above figure, the team screen triggered from the main manager screen by clicking "Team and Reports" button, gives the user, namely manager to see the detailed information about workers as a table manner again.

After selected one worker from the table, if the manager presses the "Show Reports" button, a new UI, explained below, will have opened in a second and show the reports belongs to the selected worker.

REPORT SCREEN

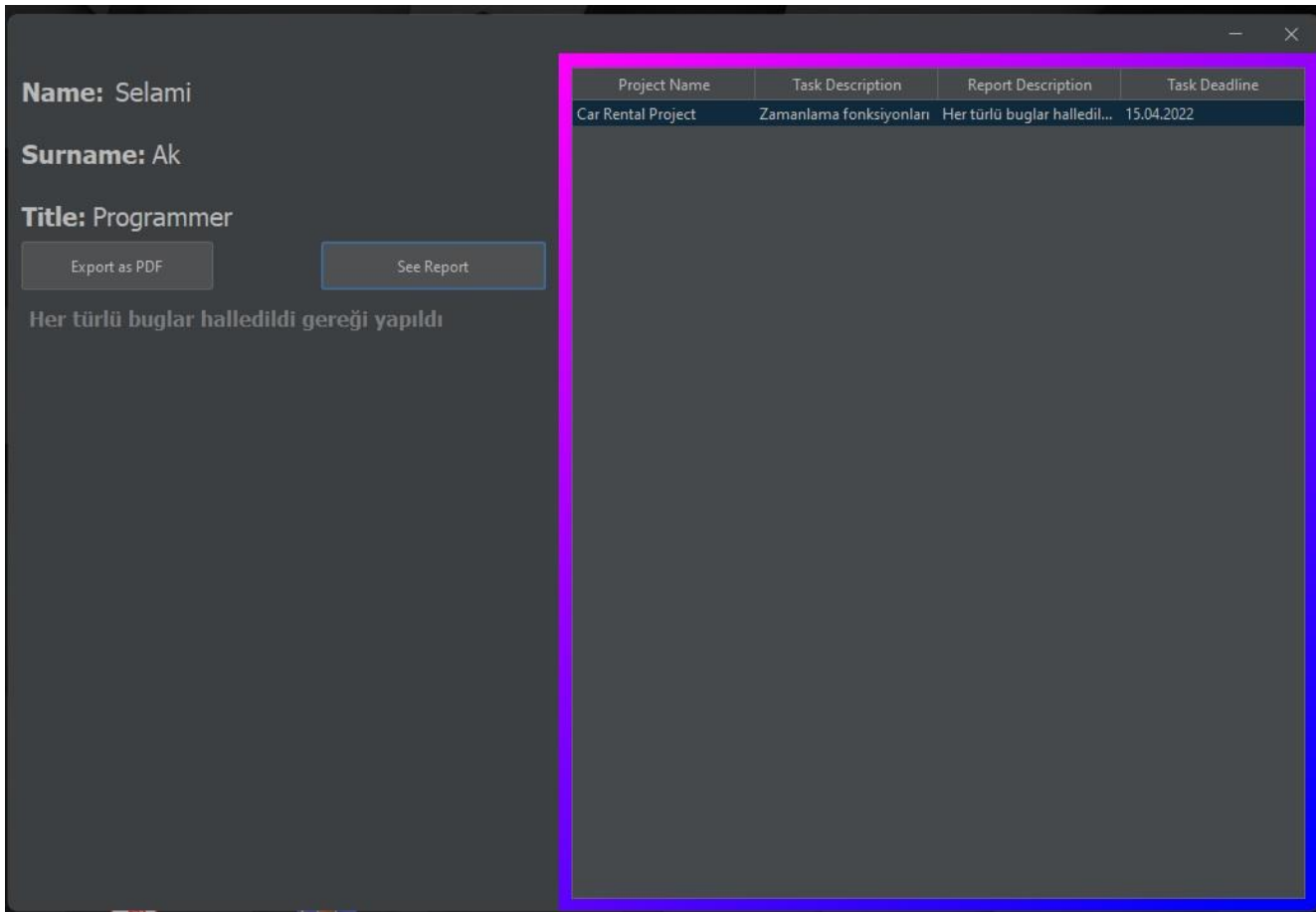


Figure 11.8: Manager's report screen

The above screenshot is the screen explained above, opened when the user would like to see all reports written by a specific worker who is elected by the manager from the table. The see report button carries the report to the a larger area as can be seen. Also, there is a “Export as PDF” button, as name states, exports the report to the outside in the PDF format.

MESSAGE SCREEN

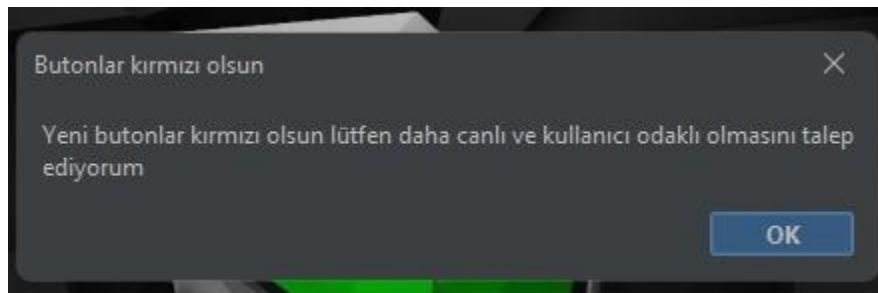


Figure 11.8: Manager's message screen

If the manager clicks the button named “Show Message” then a pop-up will have opened and showed the message as above example. This feature aims to make the communication between customer and manager faster.

ITWORKER SCREENS

MAIN SCREEN

Personal Information

Title: New label

Name: New label

Surname: New label

Current Project: New label

Experience: New label

Salary: New label

Phone Number: New label

Complete Task

MEETS

ID	Meet Name	Description	Time
17	Çeşitli yazılımların Tekra...	Çeşitli yazılımlar hakkınd...	5/8/19-17:8
19	Senkronizasyon algorit...	Senkronizasyon algoritm...	8/7/26-16:30
21	Alınan karar	Alınan kararlar baştan de...	7/11/2026-05:22
23	Araba Yazılımı	Araba Yazılımının baştan...	6/6/2026-05:22
24	Ağır koşullar	Pandmei nedeni ile topl...	1/1/2022-03:0
37	asdasdasda	sadasdasdasd	1/1/2022-00:0
43	gfdsgfd	fsdfsdfsdfs	1/1/2022-00:0
44	fdgfdgfd	gfdgfdgfd	1/1/2022-00:0
45	asdsadas	sdsadasdasda	8/7/2029-07:7
46	emircan	tepe	1/1/2022-00:0

Accept Meet

Cancel Meet

ID	Task Description	Status	Deadline
15	Web Scrapping Kodlanması	False	13/7/2029

Figure 11.10: Worker main screen

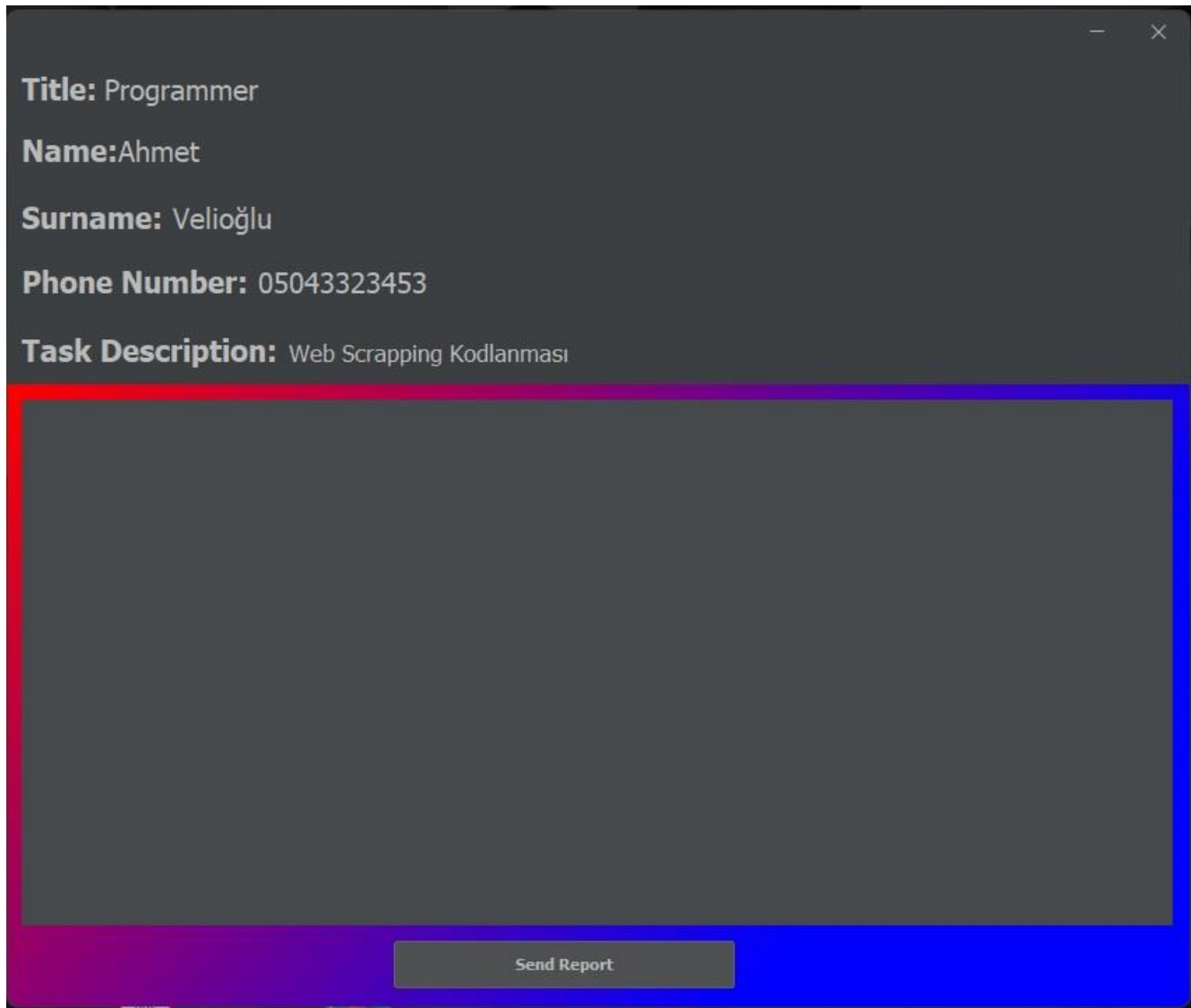
The above UI has basically 3 different component showing different kind of information and 3 different button each of which has different actions. As can be seen from the left side, it is left for the personal information of the ITWorker as labels. The right table shows the information of the meetings while bottom table shows the information of tasks.

The **Accept/Decline Meet** buttons changes the ITWorker's status, showing that she/he will attend or not on a specific meet. The other button, namely complete task, located to the left side will trigger opening of another screen, explained above, for the ITWorker to write a new report about a completed task. Of course, all the functionalities of these three buttons depend on the selection made by



the ITWorker and it is checked, and anything is wrong, an error box triggered. (Please look at the example given before by clicking [here](#).)

REPORT SCREEN



Title: Programmer

Name: Ahmet

Surname: Velioğlu

Phone Number: 05043323453

Task Description: Web Scrapping Kodlanması

Send Report

Figure 11.11: ITWorker's report screen

This UI makes the ITWorker able to write a report on a specific task given by the manager. After clicking “Send Report” button, first, the area is checked if it is empty or not, if not empty, then it will be processed to the DB to carry it to the [manager's report screen](#).



CONCLUSION AND FUTURE WORK

Software projects take place nearly everywhere in our world, from refrigerator to self-driving cars, home automation, etc. Also, the technologies used get more complicated day by day. As a natural result of this increment of the complication, the overall **complexity** and **management** of a software project gets harder. There are many researchers working on that issue and big companies creating programs to **decrease** the complexity and making easier to **manage** the project with a better success rate, and clearly speaking, we, as a team, have to state that the future is not very promising as the CHAOS report shows.

Our approach is based on taking help from the **software** to solve software complexity. It is not anything more than a representation of these attempts explained in above paragraph. As a conclusion, the project is **completed** with not only success and in time, but with also **additional features** such as exporting a report to the outside as a PDF.

The project can be expanded to have a range scale of functionality, such as adding other roles from a real-world company, e.g., CTO, CEO, Financial manager, etc. in means of future work. Adding more roles gets new authorities, also leads to the need of adding new features. Of course, to present the project to the real world, making the program runnable on the internet environment is another key future work.

BY
Gökay DİNÇ
Emircan TEPE
23/05/2022



REFERENCES

- [1] **Utku S.**, *Lecture Slide*, CME 2210, 2022 spring semester
- [2] **Standish's group**, CHAOS report, 2015, available [link](#)
- [3] **Canva**, Drawing app, online free tool, available [link](#)
- [4] **Lucid**, Drawing app, online free tool, available [link](#)
- [5] **Visual Paradigm**, Drawing app, online free tool, available [link](#)
- [6] **MySQL workbench**, software tool of MySql DB
- [7] **Figma**, UI design app, online free tool, available [link](#)

