ICT Engineering

**SEP1Y**

**Process Report**

Submitted to:

Steffen Vissing Andersen

Birgitte von FyrenBalsløv

Prepared by:

Sebastian-Dan Basca

Martin Kostadinov

Octavian Grozman

Mihai Timotin

Submitted on : INSERT DATE HERE



**Contents**

1. Group policy

2. SWOT Analysis

2.1. Group SWOT Analysis

2.2. Individual SWOT Analysis

3. Considerations

4. Daily Project Log

5. Task Log

6. Meeting reports

7. Bloom’ forms

**Group Policy**

The following group rules were established and approved by the entire group before starting working on the semester project.

* Always contact all group members when we are going to miss a meeting.
* Work on group assignments collaboratively and cohesively.
* Switch roles for each assignment.
* Show up on time at meetings.
* All members are notified about the future meetings within at least 12 hour notice.
* Complete the assignments during group meetings (in case we do not manage to complete the tasks during the meetings, we should finish everything at home).
* Assist other members when they encounter problems with their assignment.
* Show interest and passion for the project and be serious as much as possible.
* Communicate in English only.
* Communication between members can be done through Facebook group and also on Skype.
* Take into consideration everyone’s opinion.
* Be open-minded to new suggestions.
* Prioritize SEP over other activities in your schedule.
* Do not disrespect other members of the group, try to behave professionally.
* There will be a weekly discussion about the progress of the project.
* All members will have something to work on.

**SWOT Analysis**

1. Group SWOT Analysis:

**Strengths:** Enthusiasm, positive group members, motivation to succeed, hard-working, good team chemistry, open-minded members

**Weaknesses**: Little experience with group projects, many contradicting opinions, limited creativity, easily distracted.

**Opportunities**: Improve programming and team working skills (proper communication), learn to take responsibility, learn to pay more attention to details.

**Threats**: Splitting the amount of work properly, time management issues (setting up team meetings).

1. Individual SWOT Analysis:

**Martin Kostadinov:**

**Strengths**: Motivated to succeed, positive behavior, practical, reliable.

**Weaknesses**: Easily stressed, selfish, worrying about small details.

**Opportunities**: Develop teamworking and social skills, learn more about programming, get a lot of practical experience.

**Threats**: No specific threats.

**Mihai Timotin:**

**Strengths**: Punctual, creative, optimistic.

**Weaknesses**: Easily distracted and bored.

**Opportunities**: Getting experience in group programming, expand IT knowledge, learning to work in a team environment.

**Threats**: No specific threats.

**Octavian Grozman:**

**Strengths**: Experienced in programming, good communication skills, cooperative, friendly, resourceful and creative.

**Weaknesses**: Not very punctual, inflexible, not very efficient with time spending.

**Opportunities**: Improve programming and communication skills, become better organized, learn new things, improve presentation skills.

**Threats**: No specific threats.

**Sebastian-DanBasca:**

**Strengths**: Reliable, passionate, hard-working.

**Weaknesses**: Laziness, postponing work until last moment.

**Opportunities**: Improve group work and IT skills, gain wider understanding of working with other people.

**Threats**: No specific threats.

**Considerations**

Sebastian-Dan Basca

As the semester project period was getting closer and closer I felt more and more excited about the project our group will have to work on. I was thinking that this project would take no more than a week but the complexity of the interview it is based on changed my mind drastically. This was the first time in my life that I got the opportunity to experience a real group project.

I was pretty confident in my programming skills and almost straight away I started out making a plan on how to code this project which ended up being extremely helpful for the entire team. I felt relieved when I discovered all of the other member’s talents and skills and the fact that we split the workload fairly. We all to came up with team work tactics and efficient ways to communicate with the others. We spent a lot of time working on the code and in parallel with this we made the documentation.

Martin Kostadinov

When the project period started, I was very excited about this new challenge and was highly motivated to succeed. It really helped that everyone in our group shared the desire to make the project as good as possible. As we started, I was confident in our group, but I was aware of our inexperience. However, everyone remained positive which was important for our team mentality. In the very first week, we were meeting a lot and making sure that we are all one the same page and if we weren’t – we would discuss our ideas and choose the optimal path. Everyone was open-minded and we were all working towards the same goal. As time went by, we continued to meet on a regular basis and we often split the workload as to provide maximum efficiency. Very often we also went over the work we had done so far where everyone had the chance to explain how they approached their tasks. This way of keeping everyone updated about everything was very important to us, because we were all so eager to learn and felt that the everyone should know how all the parts of the project were made. And even if it meant spending more time talking about their particular tasks, it was very much worth it. Since this was our first group project, we were making a lot of mistakes, which was humbling and helped us learn more about the process of software development. The middle and the end of the project period was with most challenges, as this is when we had to do most of the work. However, with a lot of cooperation by everyone, and clear communication, we were able complete the project. I was very happy with the group I chose, because everyone was working hard and making sure we all succeed. I learned a lot during this first project period which I’m sure will be helpful in the future and I can gladly say it was a very educating experience.

Octavian Grozman

I was very passionate about our final SEP project. During the entire 1st semester I had been learning and absorbing everything in order to be ready for the project. After the 1st presentation about our coming project I was quite thrilled and even alarmed because it seemed to me that we had a huge amount of work to do. Actually, I was right about the big amount of efforts and work. Before starting working on the project, I was not very fond of group projects, I considered that individual work is much better and more efficient, but eventually, I found out that I was wrong, and that group work may be an extremely productive and exciting.

Even though we were not so hard-working, active and interested during the first period, we were a great team. After clarifying some main issues together with the teacher, we dived into the project. We planned every meeting, every assignment in order to see a clear progress, which , undoubtedly, had a role of motivator for the whole team. The fact that we were approximately at the same level of knowledge also was a favorable factor. We treated each other equally and with respect. Our meeting were full of enthusiasm and passion, so we were definitely improving our knowledge and skills. It was only a short period when we got a little distracted and lazy, but after realizing that time passes by very quickly, we pushed laziness out of our minds and kept on working on SEP. Atmosphere inside the team was a great one, so I got a lot of pleasure spending time with my team members. I am glad that everyone could contribute to the project improvements, so everyone was fully involved in this exciting process. Additionally, I have never done an analysis for a project, so it was quite confusing and insignificant for me, but now I realize how essential it is if you intend to build a qualitative piece of software.

Personally, I consider that it was an extremely useful and pleasant experience for me. I learned a lot of new things and certainly improved my communication skills, so now I can be much more productive working together with other team members.

Mihai Timotin

When my semester project group was formed I barely knew the people I was going to work with and I was looking forward to our cooperation and a great SEP experience. At our first meetings we set up all the requirements and I found out that we got the same goal, to take a 12.

The most important things about our group where: if someone didn’t understand something then we always helped him with that, everyone was positive so all of us felt confident, we did our best to split the tasks equally and to keep everyone updated with the changes in the project because we felt like everyone of us should know each part of the project. We did a lot of mistakes since the very beginning starting with the requirements until the very end with the sequence diagram and the code but we were always updating our work, there is no part of our project which we didn’t change at least 3 times.

This project was an unforgetable educational experience being the first one in my life and I am glad that I had a great group for this project. I learned a lot of knew stuff and gained real group work experience which I am sure will be helpful in the future.

|  |  |
| --- | --- |
| **Daily Project Log** | |
| **PROJECT NAME** | **SEP – VIA BUS** |
| **GROUP MEMBERS** | Octavian Grozman, Martin Kostadinov, Sebastian-DanBasca, Mihai Timotin |
| **DATE** | **TASK** |
| 11 October | For the first we met in order to create and sign the group policy. |
| 5 November | Discussion of the interview, determining the main points about the future system. |
| 7 November | Writing down some requirements |
| 10 November | Continuing to write down the requirements for the system |
| 12 November | General discussion about the system and fixing the requirements. |
| 15 November | Starting work on Use Case Diagrams |
| 17 November | Working on Use Case Diagrams and also Use Case Description |
| 20 November | Finishing the Use Case diagrams and descriptions, and starting work on Activity Diagrams |
| 23 November | Working on Activity Diagrams, fixing the Use Case elements |
| 25 November | Starting work with Class diagram, working on Activity Diagrams |
| 27 November | Finishing the Activity diagrams, planning the model classes. |
| 29 November | Group discussion about the project structure: MVC components, building up the model, File system. |
| 1 December | Finishing the Class Diagram (Condensed version) |
| 2 December | Complex programming on creating the base model classes and Lists classes. |
| 5 December | Creating the File System and connecting it with Model. |
| 6 December | Making the complex Class diagram and fixing the bugs with file system. |
| 7 December | Group discussion about GUI design and drawing the sketch for the system. |
| 8 December | Splitting up the work on doing the User Guide, Bloom forms, UML, Sequence diagram, SWOT analysis and updating the old Introduction with Group Policy. |
| 9 December | Programming together on GUI elements |
| 10 December | Fixing some problems with GUI, creating the main GUI elements (lists, tabs, buttons, labels, textfields) |
| 11 December | Finishing the GUI (creating and binding of event listeners) Connecting the model with GUI properly. |
| 12 December | Making various tests together and checking the system functionality, fixing the found bugs. |
| 13 December | Final testing of program and creating the layout for Documentation. System is ready. |
| 14 December | Writing the bloom forms, condensation essays, SWOT analysis, personal process reports and writing the documentation. |

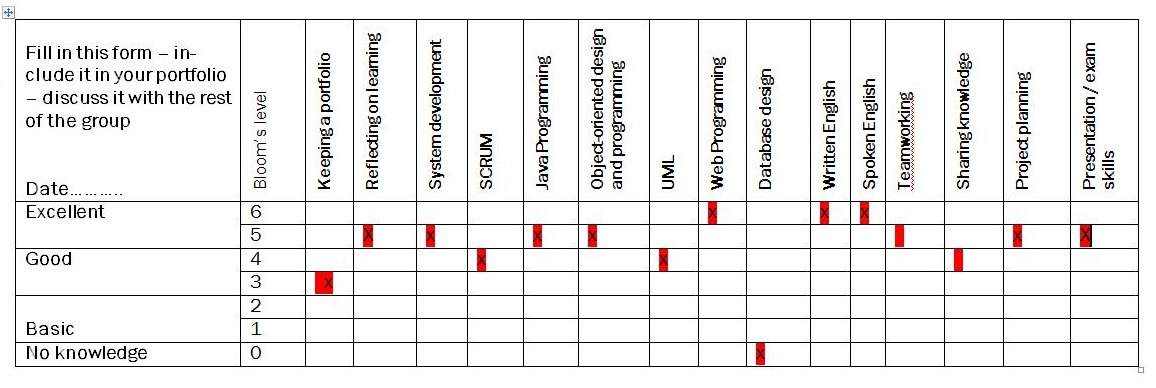
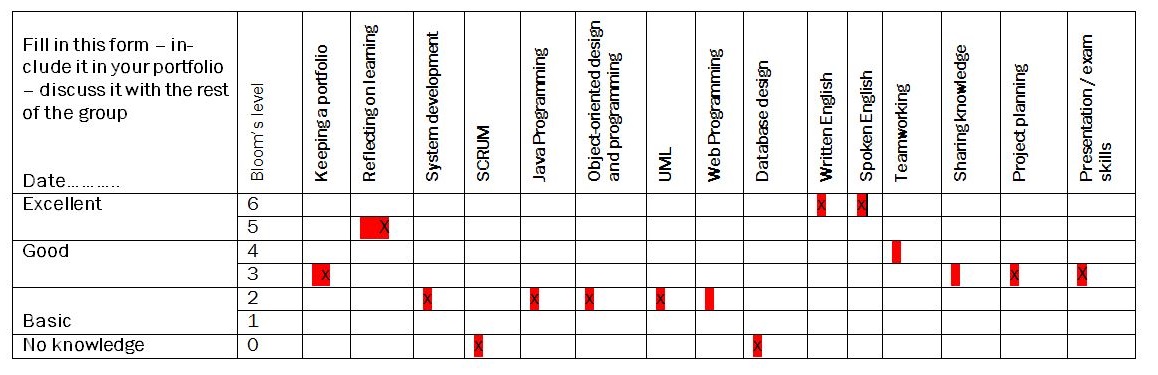
|  |  |  |  |
| --- | --- | --- | --- |
| Group Task Log | | | |
| **Project Name** | SEP – VIA BUS | | |
| **Group members** | Octavian Grozman, Martin Kostadinov, Mihai Timotin, Sebastian Basca | | |
| **Task** | **Assigned To** | **Status** | **Completed** |
| Requirements | Mihai Timotin | Approved by Group | Yes |
| Use Case Diagram | Octavian Grozman | Approved by Group | Yes |
| Use Case Description | Whole group | Approved by Group | Yes |
| Activity Diagrams | Whole group | Approved by Group | Yes |
| Daily Log | Octavian Grozman | Approved by Group | Yes |
| SWOT | Whole group | Approved by Group | Yes |
| UML Diagram(condensed) | Sebastian-DanBasca | Approved by Group | Yes |
| UML Diagram (extended) | Sebastian-DanBasca | Approved by Group | Yes |
| User Guide | Martin Kostadinov | Approved by Group | Yes |
| Sequence Diagram | Octavian Grozman | Approved by Group | Yes |
| Task Log | Mihai Timotin | Approved by Group | Yes |
| Contents | Sebastian-DanBasca | Approved by Group | Yes |
| Person + MyDate Class | Martin Kostadinov | Tested and Working | Yes |
| Reservation Class | Octavian Grozman | Tested and Working | Yes |
| Tour Class | Sebastian-DanBasca | Tested and Working | Yes |
| Customer Class | Martin Kostadinov | Tested and Working | Yes |
| Chauffeur Class | Octavian Grozman | Tested and Working | Yes |
| Passenger Class | Mihai Timotin | Tested and Working | Yes |
| BusList Class | Martin Kostadinov | Tested and Working | Yes |
| Data Class (File System) | Sebastian-DanBasca | Tested and Working | Yes |
| GUI Design | Whole Group | Approved by Group | Yes |
| GUI Implementation | Whole group | Tested and Working | Yes |
| Blooms Profile | Whole group | Approved by Group | Yes |
| Introduction | Mihai Timotin | Approved by Group | Yes |
| Abstract | Martin Kostadinov | Approved by Group | Yes |
| GUI breakdown | Sebastian and Octavian | Approved by Group | Yes |
| Meetings Report | Octavian Grozman | Approved by Group | Yes |
| UML breakdown | Martin and Sebastian | Approved by Group | Yes |

**Bloom’s Forms**

**Sebastian-Dan Basca:**



**Martin Kostadinov:**



**Octavian Grozman:**



**Mihai Timotin:**



ICT Engineering

**SEP1Y**

**Project Report**

Submitted to:

Steffen Vissing Andersen

Birgitte von FyrenBalsløv

Prepared by:

Sebastian-Dan Basca

Martin Kostadinov

Octavian Grozman

Mihai Timotin

Submitted on : INSERT DATE HERE



**Contents**

1. Abstract

2. Introduction

3. Analysis

3.1. System Requirements

3.2. Use Case Diagram

3.3. Use Case Description – Manage reservations data

3.4. Activity Diagram – Manage reservations data

4. Design

4.1. UML Class Diagram (Relations)

4.2. UML Class Diagram (Expanded)

4.3. GUI Design Breakdown

4.4. Sequence Diagram

5. Implementation

6. Test

7. Results

8. Discussion

9. Conclusion

10. Reference

**Abstract**

*The reason we are creating a Single User System is to help a company that deals with a large amount of data, as well as a lot of customers, to be able to modernize and automate a lot of their daily tasks. The idea is to create a simple and useful application, for the company VIA Bus, that will be capable of dealing with the adding/editing/removing of reservations, tours, chauffeurs, customers and passengers. The approach we took rewarded simplicity and straightforwardness. By making our GUI as user-friendly as possible, we were making sure that intensive training wasn’t required. Our code is well structured and easy to maintain if needed. We’ve made a very easy to get into system that even though lacks immense depth, it still handles all the necessary requirements well. Overall, the system is not something revolutionary, but instead our approach of taking several tasks and putting them in an environment where they are easy to perform and can save a lot of manual work.*

**Introduction**

Our team has been given the task to develop a booking system for the company VIA Bus. An interview was conducted with the owners of the company in order to decide what were their requirements. It was decided that the system must allow employees to handle reservations for - seats for trips and journeys to different destinations or a whole bus with personal preferences (food, party guide, additional stops) to desired destination. The system contains information about fixed tours(trips and journeys),and non-fixed tours(bus-and-chauffeurs) as well as chauffeurs, customers, and passengers data. The company decided that only employees will be able to access and work with the system(add/edit/remove tours, reservations, chauffeurs passenger and customer data).

The system has a simple user interface with interactive elements.  
It is developed using Java programming language.

The purpose of this system is to make it easier and more efficient for VIA Bus to manage all the reservations and keep track of all the data that the company works with. In the following pages of the report, we will go in greater detail about the things we’ve done to make all of this possible and create a simple, yet efficient system.

**System Analysis**

**Requirements**

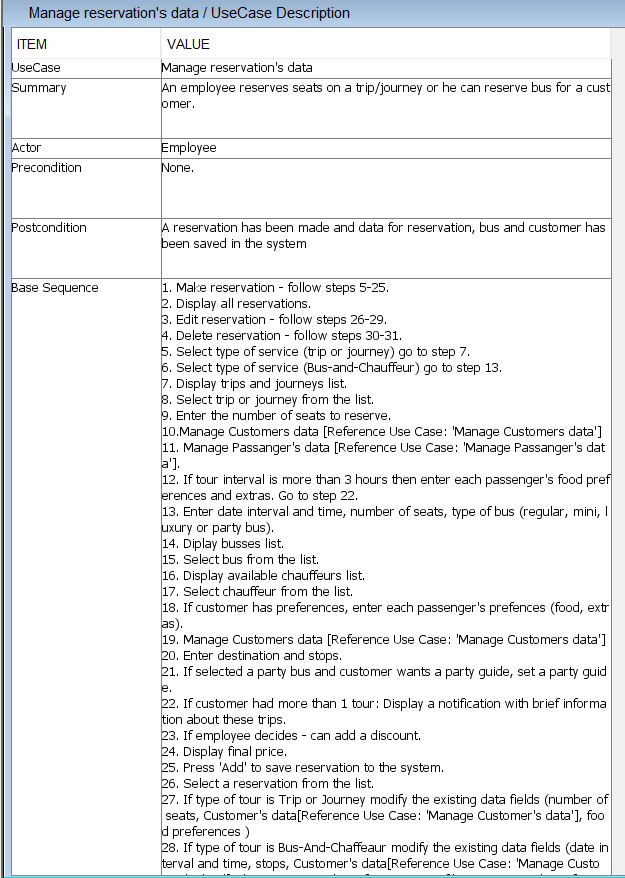
1. An employee should be able to make a reservation for a customer (trips, journeys and Bus-and-Chauffeur).
2. Only employees of VIA BUS have access to the system.
3. An employee should be able to add extras.
4. Journeys and Bus-and-Chauffeur services can have up to 2 chauffeurs.
5. An employee can create a tour by specifying departure date and time, return date and time, departure and arrival location, number of seats, add up to 2 available chauffeurs, select a bus, destination(s), stops.
6. An employee should be able to register information about customers and all the passengers (only for trips and journeys).
7. An employee should include the customer’s or company’s name if it is a bus-chauffeur service.
8. Each customer’s information should contain: name, address, phone number and e-mail address if he or she wants to get newsletter and company name if applicable.
9. Each passenger should contain: name, address and age only for trips and journeys.
10. An employee should be able to specify the number of busses of a specific type.
11. An employee should be able to delete a reservation.
12. An employee should be able to edit a reservation, customer’s info, chauffeurs’ info.
13. An employee should be able to see the Chauffeur’s profile.
14. Each Chauffeur’s profile contains : name, phone number, form of employment (part-time or full-time), preferences, ID.
15. An employee should be able to add, edit or remove a Chauffeur.
16. A customer should be able to reserve a party guide for the bus-and-chauffeur services that have a party bus.
17. An employee should be able to add a new bus to the current list of busses.
18. Employee chooses either to set or not a discount for the customer (e.g family, frequency discount).
19. The system should display if the customer made reservations in the past so he may get a discount from the employee when he will make a new reservation.
20. The system should have a list of tours (trips and journeys) with fixed destinations.

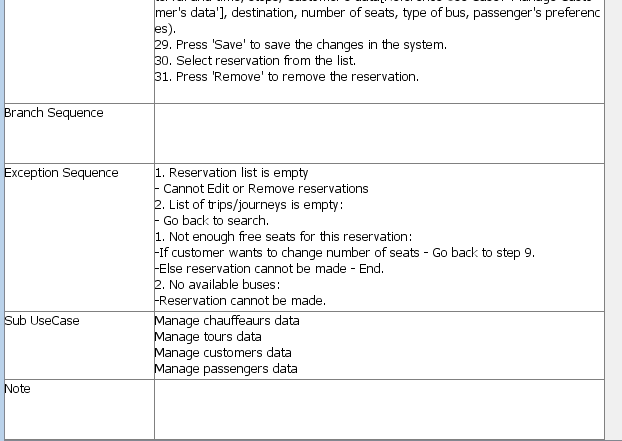
**Use Case Diagram**



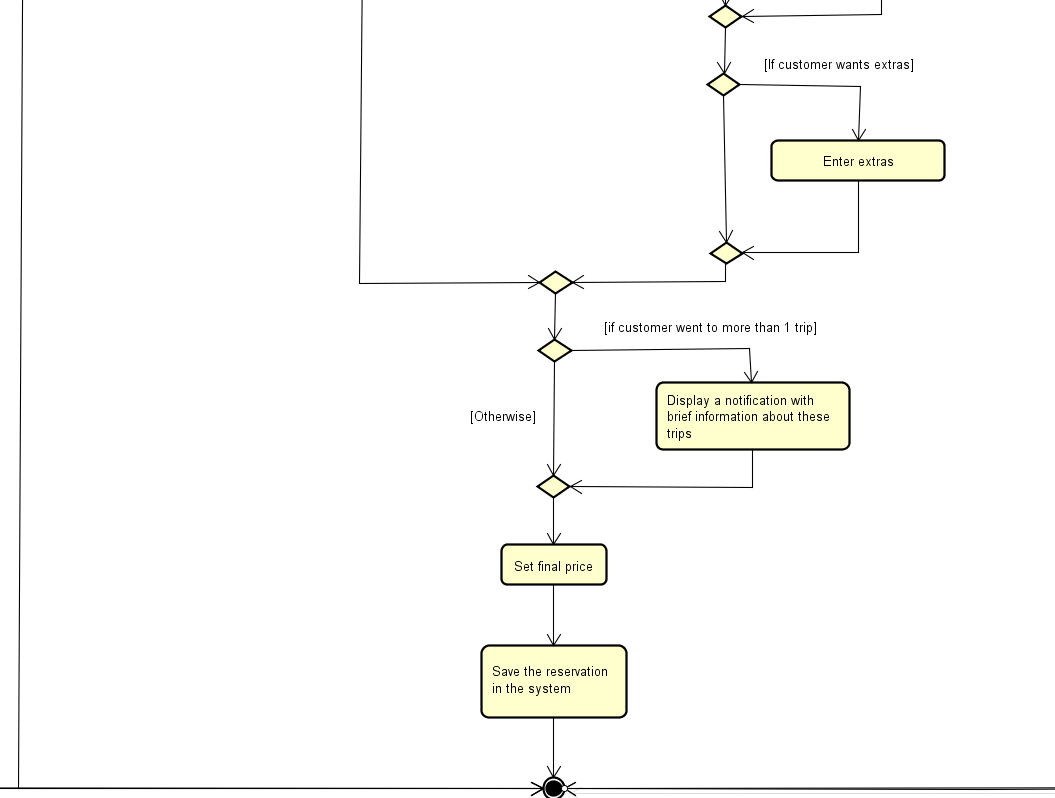
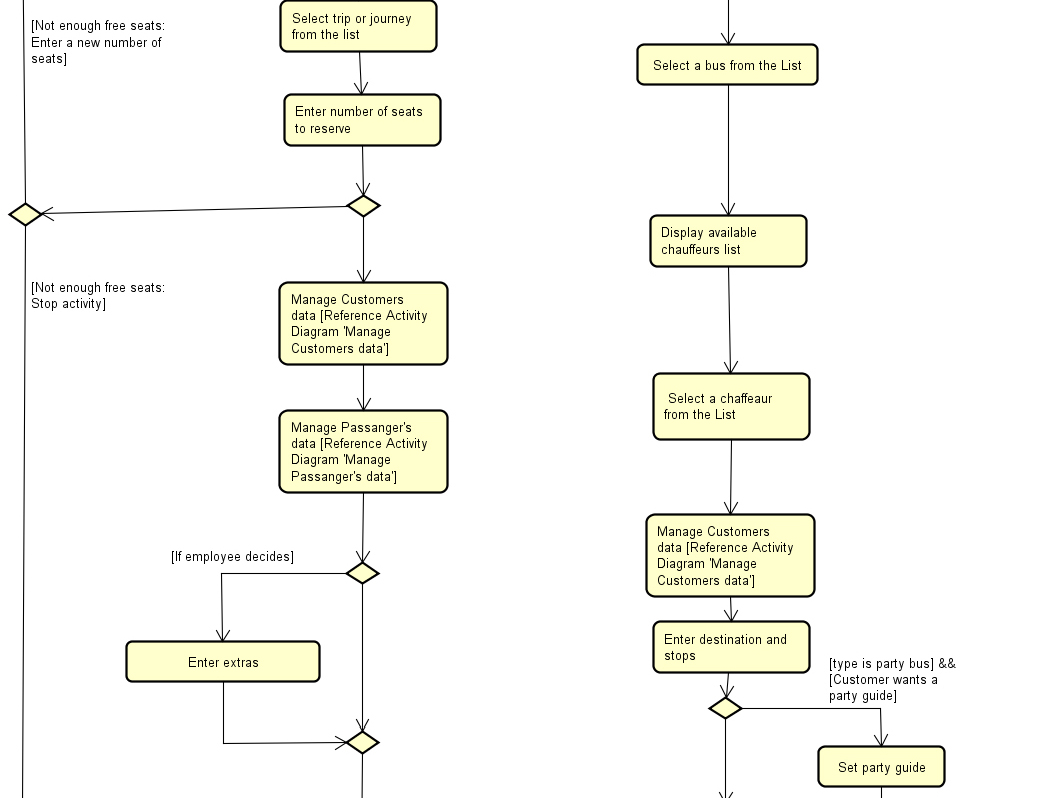
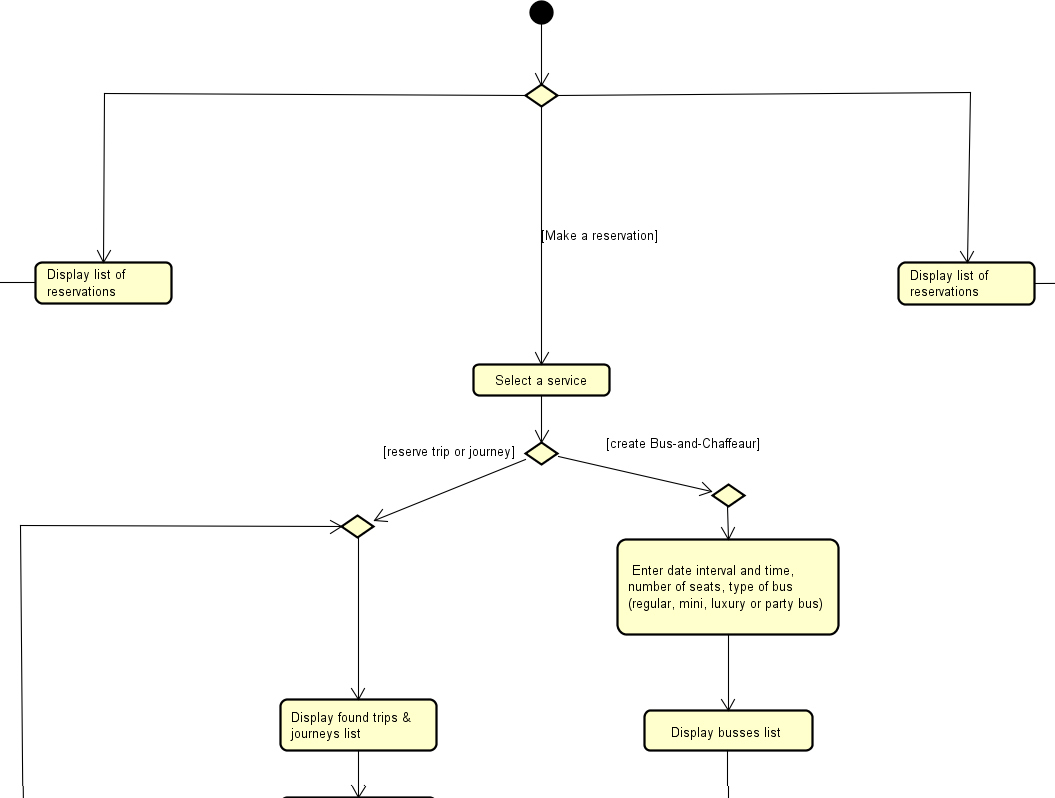
Illustrates the priority use cases performed by the employee of the bus company. It is the essential use cases required to add ,edit and remove a tour, chauffeur, customer, reservation or passenger.

**Use Case Description – Manage Reservations Data**



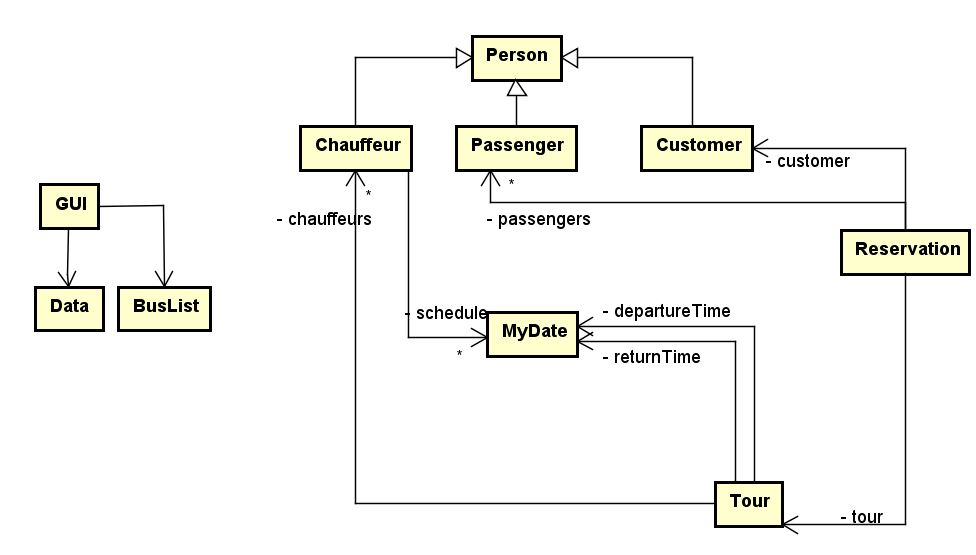


**Activity Diagram – Manage Reservations Data**



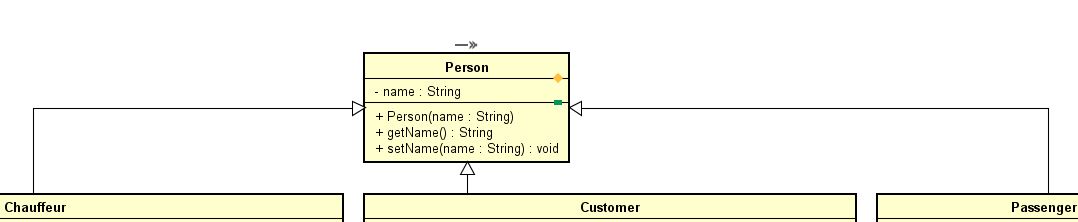
This activity diagram is for ‘Manage Reservations data’ adding a new reservation to the system.

**UML Class Diagram (Relations)**

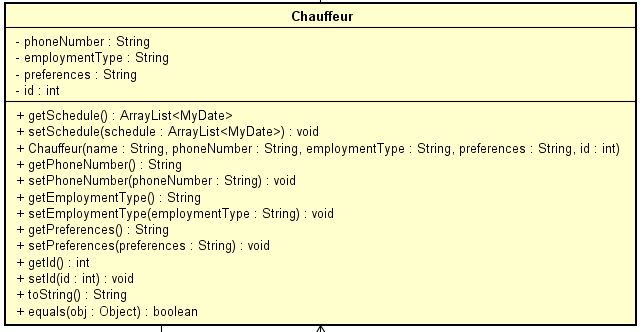


**UML Class Diagram (Complex)**

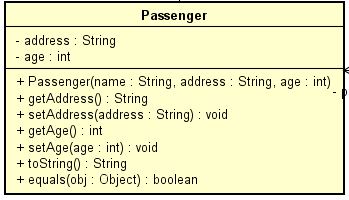
**Breakdown**



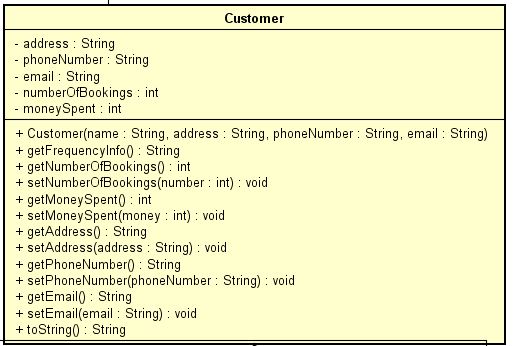
The Person class is the class inherited by three classes (Chauffeur, Customer and Passenger). It contains only one instance variable of type String (name) and has a constructor that sets that instance variable with a getter and a setter.



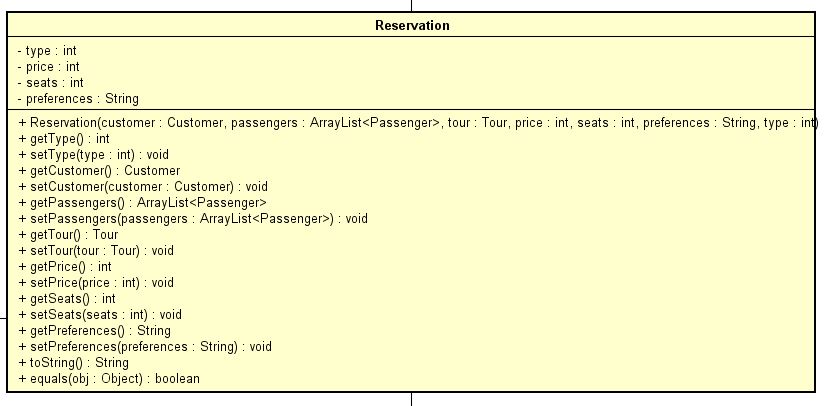
The Chauffeur class contains as instance variables: name (String), phoneNumber (String), employmentType (String), preferences (String), schedule (ArrayList of MyDate objects) and id (Integer). The constructor takes all instance variables as arguments apart from ‘schedule’ which will be populated when the chauffeur is assigned to a tour. Apart from this method there are setters and getters for the instance variables and also a ‘toString()’ method and an ‘equals()’ method.



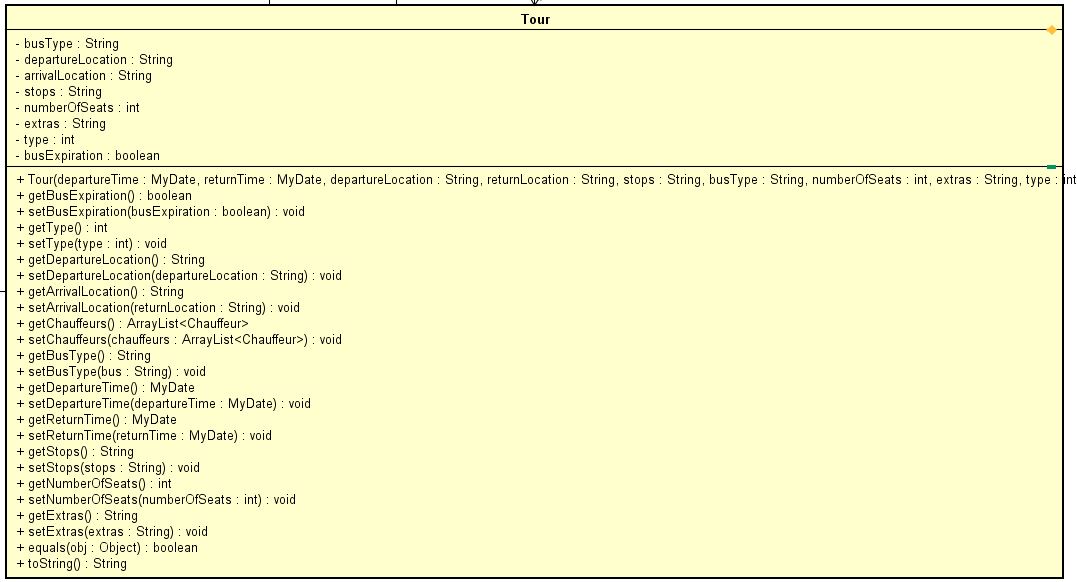
The Passenger class contains three instance variables: name (of type String), address (of type String) and age (of type Integer). The constructor sets these two instance variables. The class also contains setters and getters for the instance variables and also has ‘toString()’ and ‘equals()’ methods.



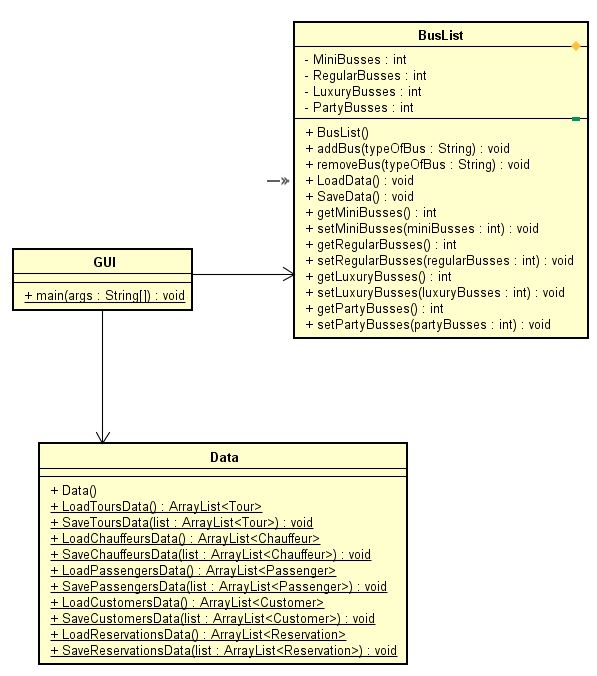
The Customer class contains six instance variables, four of type String (name, address, phoneNumber and email) and two of type Integer (numberOfBookings and moneySpent). The last two instance variables are used to store how many reservations a customer has made in the past and how much did he pay in total in order to offer him a discount. The constructor sets only the address, phoneNumber and email variables. There are setters and getters for all instance variables and also a ‘toString()’ method.



The Reservation class has association relations with the following classes: Chauffeur, Customer, Passenger and Tour. This class has seven instance variables: customer (of type Customer), passengers (ArrayList of type Passenger), tour (of type Tour), type,price,seats (all of type Integer) and preferences (of type String). The constructor sets all instance variables and also increases the number of bookings of the customer and the money he spent on all past reservations. Apart from this there are setters, getters, ‘toString()’ and ‘equals()’ methods. The variable type refers to what type of reservation it is (when type is equal to 1 then it’s a reservation for a trip or journey and if the type is 2 it’s a reservation for bus-and-chauffeur service.



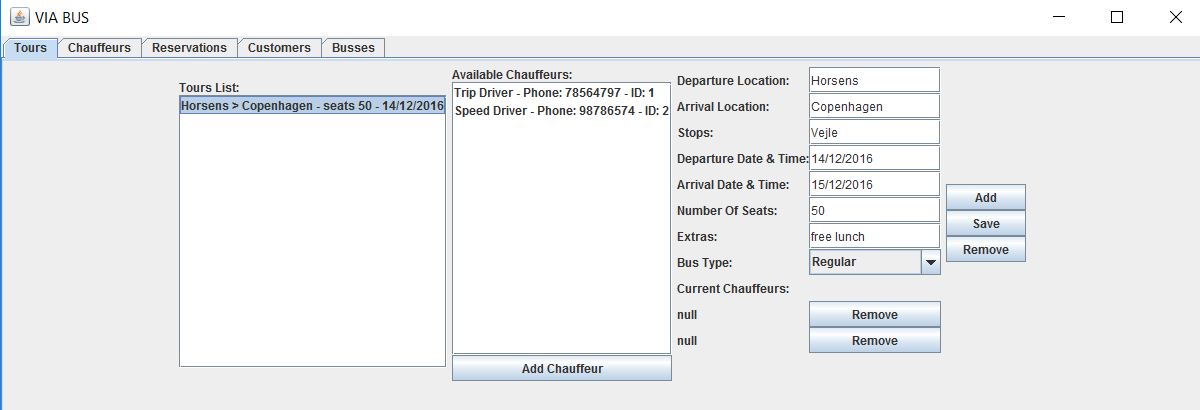
The Tour class has ten instance variables: two MyDate objects (departureTime and returnTime), Strings (departureLocation, returnLocation, stops, busType and extras), Integers (numberOfSeats and type) and a Boolean (busExpiration). The constructor sets all instance variables and sets the ‘busExpiration’ Boolean to true. The class has setters and getters for all instance variables and also a ‘toString()’ and ‘equals()’ methods. The type variable refers to the type of tour (type 1 for trips & journeys and type 2 for bus-and-chauffeur).



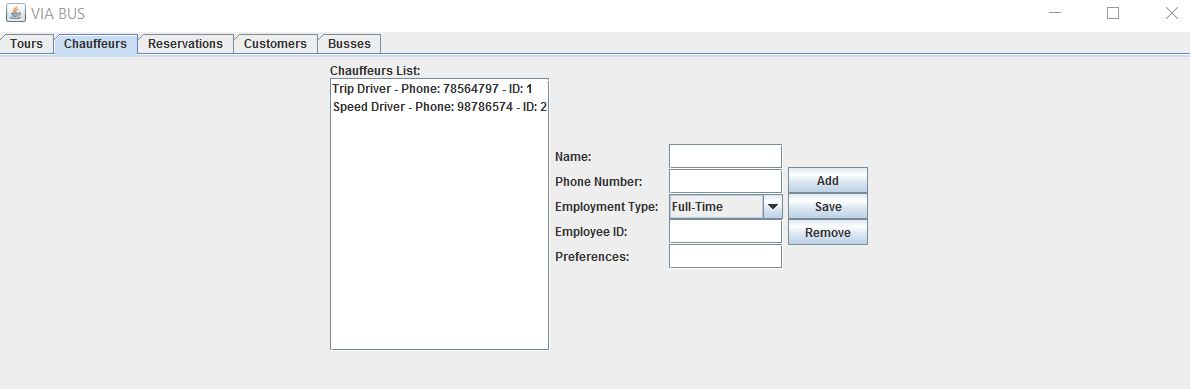
The GUI class is the main class from where the program is started. It has association relation with two classes (Data and BusList). The GUI class contains the GUI and all the functionality for it. It loads and saves all the data about tours, customers, reservations, chauffeurs and passengers using the Data class and also loads and saves the busses information using the BusList class (LoadData and SaveData methods).

**GUI Design**

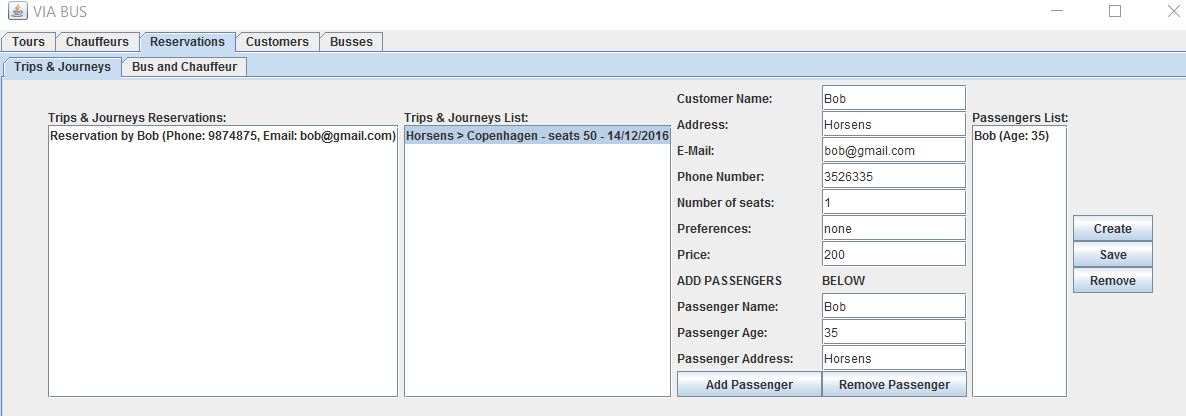
The GUI we decided on implementing is easy to understand and straightforward to use and learn by almost anyone with little computer skills. There are five different main tabs in our program, they contain information about all the data related to tours, chauffeurs, reservations, customers and busses. The data is displayed clearly and the buttons are named in such way the employee will understand their actions.



The “Tours” tab contains on the left side a list with all existing trips and journeys displaying minimal and useful information about each tour (such as departure location, arrival location, the total of seats and the departure date). On the middle there is another list called “Available Chauffeurs” that will update after inserting the dates into the Departure Date and Arrival Date fields in the form or it will update when a tour is clicked (in the tours list). Under this list there is a button (“Add Chauffeur”) used to assign the selected chauffeur to the selected tour. The Tours tab also features a form that can be filled in with information needed to create a Tour. Under the form there are two slots on which each chauffeur assigned to the selected tour is displayed giving the user the option to remove them. On the far right side the tab has three buttons: Add (for creating a new tour), Save (for saving the editing done to a tour) and Remove (for permanently removing a tour).

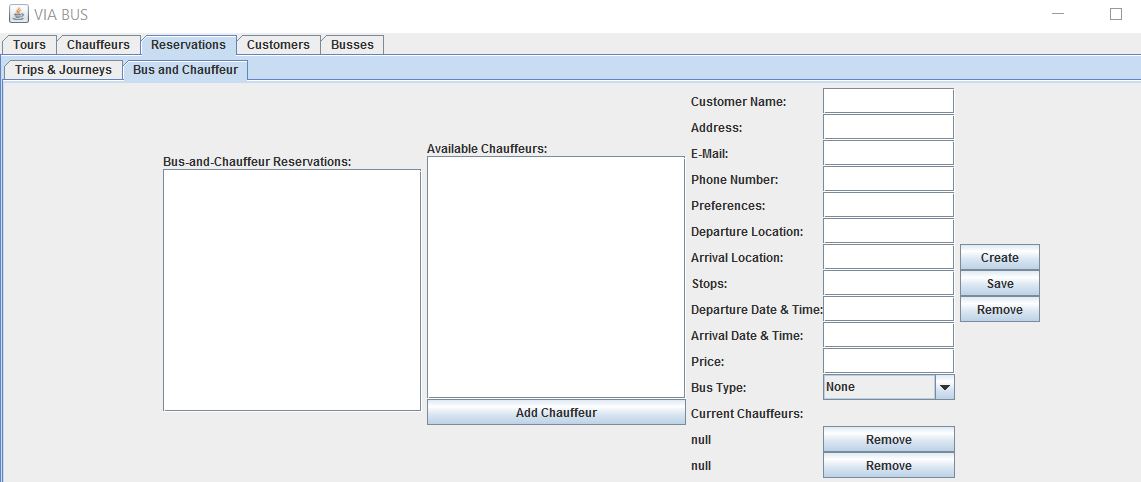


The “Chauffeurs” tab contains on the left side a list with all the chauffeurs, displayed with minimal information (name, phone number and employee id). In the middle part there is a form which can be used to type down information about a new chauffeur or when a chauffeurs is selected from the list, the form will display the chauffeurs information that can be edited. The tab also features three buttons: Add (for adding the new chauffeur to the system), Save (for saving the modified information about a chauffeur) and Remove (for removing a chauffeur from the system).

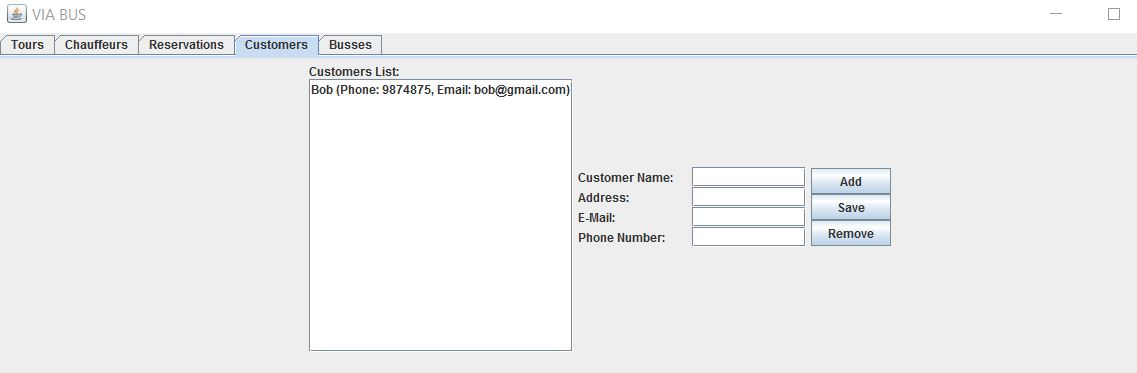


The “Reservations” tab is the most complex tab because it features another two sub-tabs.

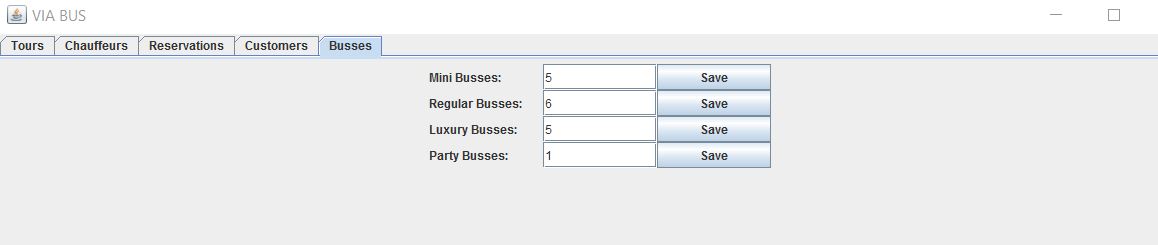
1. The “Trips & Journeys” sub-tab is the place where the reservations for trips and journeys can be managed. It contains a list with all reservations that were made displayed with minimal information about who mae the reservation, their phone number and email. Also there is a list displaying all the trips and journeys with essential information. This sub-tab also features a form that has two functionalities: a part containing information about the customer, number of seats they want to reserve and the price, and another part which is another form used for adding or removing passengers from the passengers list (on the right side). On the far right side there are three buttons: Create (for creating a new reservation for the selected tour), Save (for saving the modifications did to a reservation) and Remove (for removing a reservation from the system).



2. The “Bus and Chauffeur” sub-tab is where reservation for the Bus-and-Chauffeur service can be made. This sub-tab features a list of all Bus-and-Chauffeur reservations displaying minimal information about each reservation. Next to this list there is another list called “Available Chauffeurs” that will be updated with the chauffeurs available after inserting the departure date and arrival date or when a reservation is selected. Under this list there is a button (“Add Chauffeur”) for assigning the selected chauffeur form the Available Chauffeurs to the selected reservation. This sub-tab also features a form that contains information about the customer and the tour. Also there are 2 labels containing the assigned chauffeurs to the selected reservation with “Remove” buttons for removing the assigned chauffeurs.



The “Customers” tab is similar to the “Chauffeurs” tab and contains on the left side a list with all the customers, displayed with minimal information (name, phone number and email). In the middle part there is a form which can be used to type down information about a new customer or when a customer is selected from the list, the form will display the chauffeurs information that can be edited. The tab also features three buttons: Add (for adding the new customer to the system), Save (for saving the modified information about a customer) and Remove (for removing a customer from the system).



The “Busses” tab contains all the information about the busses used by the company. Beside each category of busses there are text fields containing the current available busses. The number of available busses can be edited than saved using the “Save” buttons located after each category.

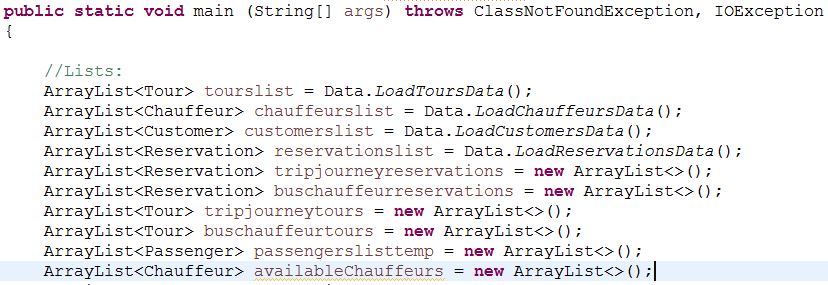
**Sequence Diagram**

**OCTAVIAN YOUR JOB**

**Implementation**

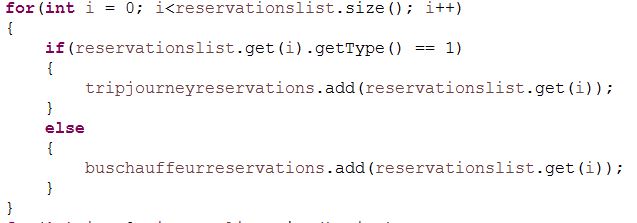
Our group thought of the best ways to display all necessary information from the system. There are lists containing tours, reservations, chauffeurs, passengers and customers. In order to edit or to add an item of a list we made a form-like design to be filled out really easily. The GUI is easy to understand and most important is user friendly.

**Code Snippet 1:**



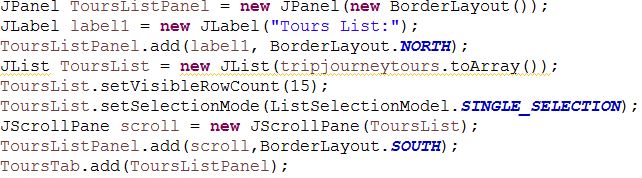
The GUI class starts out with these lines in which all the Arraylists that are necessary to the program are created and where all the saved data is loaded.

**Code Snippet 2:**



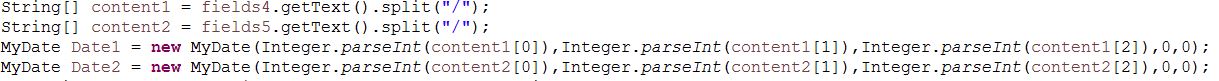
This snippet is responsible for separating the reservations by their type. Reservations for trips & journeys are placed in the “tripjourneyreservations” ArrayList and the reservations for Bus-and-Chauffeur are stored in the “buschauffeurreservations” ArrayList.

**Code Snippet 3:**



This code snippet is responsible for creating the Tours List JList that will contain the “tripjourneytours” ArrayList. This list will have a scroll bar if there are more than 15 elements inside it. All the other JLists created in this program are made in a similar way in order to maintain consistency.

**Code Snippet 4:**



This part of the code is responsible for reading the Departure Date and Arrival Date from the text fields of the GUI. The text fields contents are split at each “/” and creating two MyDate object (taking arguments the fields contents changed into integers).

**Code Snippet 5:**



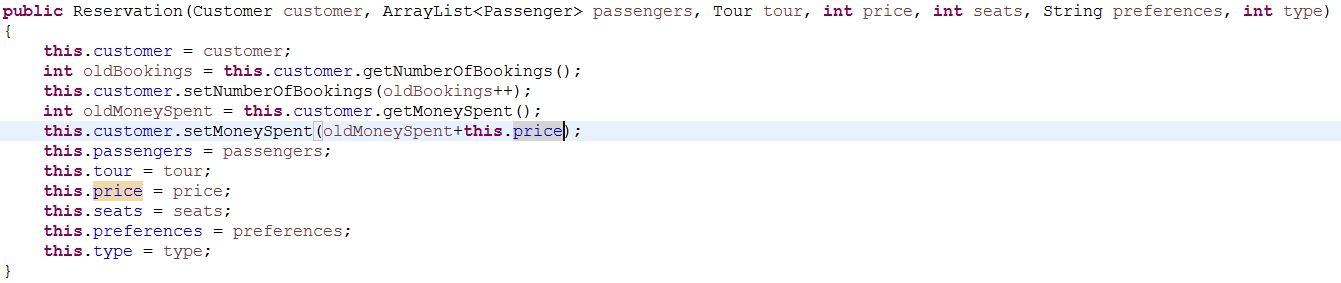
This code snippet is responsible for loading all the Tours from the “Tours.bin” file. The program will read an ArrayList of objects (of type Tour) and then return them (the returned object is used in the GUI class). All the other objects (Customer,Passenger,Chauffeur,Reservation) have a similar loading method.

**Code Snippet 6:**



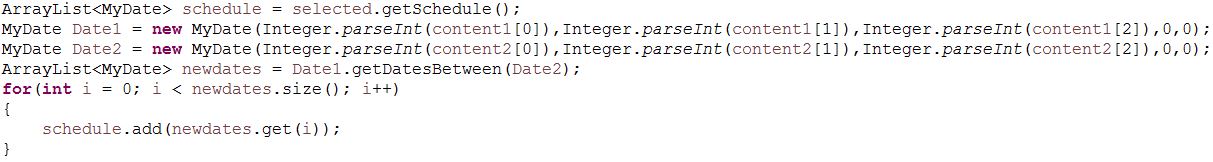
This code snippet is responsible for getting all the available chauffeurs and adding them to the “availableChauffeurs” Array List. It will compare each of the Chauffeur’s current schedule (an ArrayList of MyDate objects in which the chauffeur is busy) with the Tour’s dates (stored in “dates” ArrayList). If there are two equal MyDate objects in these two Array Lists the chauffeur will be unavailable.

**Code Snippet 7:**



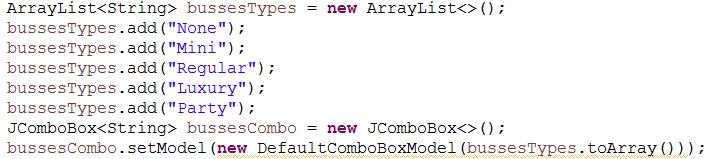
This is the constructor for the Reservation object. When creating a Reservation object, besides setting all instance variables, the customer that made the reservation will have the number of bookings increase by 1 and the money spent increase by the price of the Reservation made.

**Code Snippet 8:**



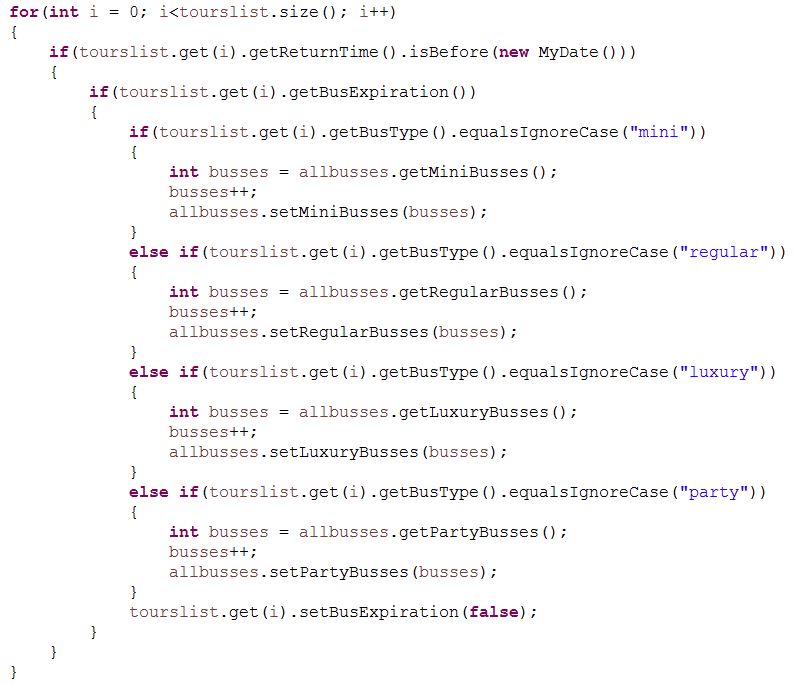
This snippet is part of setting a chauffeur to a tour. This will create two MyDate objects based on the date interval of the tour and then add all the dates in this interval to the chauffeur’s schedule.

**Code Snippet 9:**



This part of the code is responsible for creating a combo box filled with objects of type String. This combo box is used when creating a tour or making a bus-and-chauffeur reservation and lets the employee to select the bus.

**Code Snippet 10:**



This part of the code is responsible for going thourgh all the tours and checking if there are expired tours in order to set the bus used on that tour to be available. Besides checking if the tour is expired is also checking the Boolean variable inside each tour that is returned by the “getBusExpiration” method in order to avoid the duplication of the same bus.

**Tests**

**Results**

|  |  |
| --- | --- |
| 1.The system must allow to add, edit or remove a tour, chauffeur, reservation | Working |
| 2. An employee can make a reservation for a customer (trips, journeys and Bus-and-Chauffeur) | Working |
| 3. An employee can add extras (e.g food preferences) | Working |
| 4.Employee can select up to 2 chauffeurs for any tour. | Working |
| 5. An employee can register information about customers | Working |
| 6. An employee can register information about the passengers only for trips and journeys. | Working |
| 7. An employee can include the customer’s or company’s name if it is a bus-chauffeur service | Working |
| 8. Customer’s information must contain: name, address, phone number | Working |
| 9. Each passenger must contain: name, address and age only for trips and journeys. | Working |
| 10. An employee can specify the number of busses of a specific type. | Working |
| 11. An employee can edit a reservation, customer’s info, chauffeurs’ info. | Working |
| 12. An employee can to add, edit or remove a Chauffeur | Working |
| 13. A customer can reserve a party guide for the bus-and-chauffeur services that have a party bus | Working |
| 14. An employee can set a discount and a final price for reservation. | Working |
| 15. An employee can see while making a reservation if customer has been on any tours before, and how much money he has spent on them. | Working |
| 16. An employee can see lists with Tours, Reservations, Chauffeurs and the number of busses at any time. | Working |

**Conclusion**

The making of our first single user system had many challenges. First, we knew that in terms of interface, the program should be as simple as possible, so that it is easy for an employee to navigate without much trouble. The system needed to be capable of handling very important for the company tasks. To simplify it further for the owners, we included a self-teaching User Guide. However, we tried to make the design and overall look of the system very intuitive. We took the requirements of VIA Bus into consideration and created a way that they can save a large amount of data without hurdle. The number of actions that can be performed by the employee are limited within what the company asked for. We made sure everything was as straightforward as possible. It was important to us that the system we provided the company with would prove to be a very efficient way of dealing with the large number of customers they have.

As can be seen from the final product, we achieved our goal of successfully implementing the requirements that the company gave us. More importantly, we made a software that makes the employees’ jobs easier and doesn’t require a strong background with computers.

To sum up – it gives the employees the ability to easily make tours(trip/journey/Bus-and-Chauffeur) and reservations for those tours, as well as keep track of the customers, passengers and the chauffeurs.

**References**

|  |  |
| --- | --- |
| Sequence Diagram | <http://www.ibm.com/developerworks/rational/library/3101.html> |
| GUI Tutorial | <https://www.youtube.com/watch?v=jJjg4JweJZU&index=50&list=PLFE2CE09D83EE3E28> |
| GUI Tutorial | <https://www.youtube.com/watch?v=3XB3in9Xqy8&list=PLE7E8B7F4856C9B19&index=20> |
| Github Guide | <https://guides.github.com/activities/hello-world/> |
| MVC Pattern | <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm> <https://www.codecademy.com/articles/mvc> |