ICT Engineering

Semester Process Report

Andreea Buturca 253691

Karolina Beliharova 253810

Marek Lowy 253652

Martin Janosik 253934

Date: 16. 12. 2016

VIA University College

Table of contents

1 Group policy

The members of the groups decided to create and respect all the policy rules:

1. Everyone agrees to come to group meetings.
2. The time and place of group meetings shall be agreed upon unanimously within our group.
3. We will have regularly meetings on Monday, Wednesday, and Friday mornings.
4. The meetings should have its own task with a specific deadline.
5. Make sure that when someone misses a group meeting, that person will contact the group as soon as possible.
6. Everybody will work on group's tasks collaboratively.
7. Everybody will work on what it is allotted to do, done, and respecting deadlines.
8. Regarding sick issues, the person which is not attending meetings, is responsible for notifying group members, being able afterwards to catch up the work performed during absence.
9. If somebody doesn't work on the task repeatedly, it will be prevented inside the group, support the penalty decided by group members.
10. Each member will be treated with respect and consideration.
11. Everybody will do everything in their capabilities to help group members understand each concept and problem.
12. If I do not understand a concept or solution, I will not hesitate to ask my group members for help.
13. I will communicate with my group members about any concerns I have regarding our group work.
14. I will be an active member of this group in all aspects. I will not take answers as fact, but instead confirm results.

2 SWOT analysis

Andreea Buturca

Strengths: Creative, good communication skills in team, enthusiastic, dynamic, committed to the success, experience with working in teams, project planning, group presentations, keeping portfolio.

Weakness: No experience with programming and databases, I have a strong, compulsive need to do things quickly and remove them from my "to do" list, and sometimes the quality of my work suffers as a result.

Opportunities: Improve English and communication skills, learn more about programming, improving skills by working in a group with motivated members from multicultural countries.

Threats: Bad weather and car problems, sickness.

Karolina Beliharova

Strengths: Positive mind, courage to overcome obstacles, challenging person, team player, open-minded, experience with working in teams.

Weakness: Sometimes indecisive, no experience with projects and databases.

Opportunities: Improve my programming skills and databases knowledge, learn more about team-work and different cultures background, improve my skills in English language and communication skills.

Threats: Bad weather, illness.

Marek Lowy

Strengths: Good communication skills, programming experiences, I can easily handle stress situation, quick-thinker, cooperative, highly-motivated.

Weakness: Forgetful, nervous, no experience with databases, sometimes focus too much on one think.

Opportunities: Improve my skills in English language and communication skills, learn more about team-work and other cultures, personal development.

Threats: Bad weather, illness.

Martin Janosik

Strengths: Knowledge in all programming courses, willing to help others, good logic thinking, decisiveness.

Weakness: Problems to be communicative, late arrivals, I focus too much on details to have my work perfect.

Opportunities: Learn more about the group work, teach other group‘s members java, improve my English and communication skills, and learn more about team-work.

Threats: Problem with car, bad weather, illness.

Team SWOT:

Strengths:

We are willing to help each other.

We have experience with working in group.

We are team players so we can motivate and inspire each other.

We are committed to hard working.

We divide the work, agree on schedule that fits to everybody in our group and we can work faster.

Weakness:

We have too much specialists.

Coming late.

We are not that good in splitting tasks.

Planning might not be our strongest ability.

Opportunities:

To improve English and communication skills.

To meet people with different cultures.

To improve our team skills.

To improve programming skills and knowledge about databases.

Threats:

Bad weather.

Some personal conflict can appear between the members of the same group.

Somebody could sabotage the project work.

3 Group roles

Andreea Buturca

As a coordinator, my job was to plan group meetings. Before 16th of November we met after courses in our class and after 16th of November we met in our class at 10 in the morning every Monday, Wednesday and Friday. Usually we stayed there until late afternoon. I made sure that the group stayed on track and focused work around the learning task. I made sure that every voice is hear at meetings.

As an implementer, I organized the work and put the team's ideas and plans into action. Working practical and efficient, I collaborate on each team task, so they could be delivered on time, respecting the deadlines.

Karolina Beliharova

As a recorder, I was responsible to write the group's ideas down and summary of the group work at the end. I put the ideas of my mates together and compared them. I made sure that each member of the team recorded work or data. I listened to others and gave them constructive feedback if they needed it. I served as group memory.

Marek Lowy

As a presenter, I was responsible to present my group's response. I wanted to be sure that communication in my group is easy-going and all tasks, solutions and strategies are discussed. I made summaries during every discussion for other members to help them made choice how they should do their work. I spoke for the group when the group is called on to answer a question or present to the class. I was responsible to talk with the teacher when there was some problem.

Martin Janosik

As a checker, I was looking for spelling mistakes when we had to send an assignment. I made sure all group members know what to do and helped the group members who were unsure. I made all work and discussion is of the highest standard and work met success criteria. I decided with my team how many hours we need to work for every hand-in and I made a schedule that fits everybody.

As a monitor, I was responsible for making sure that everyone in the group understood correctly to the group's task, solution and strategy. I collected supplies for the team. I ensured that everyone was helped when has a question. I was glad to help my group mates and in the end everyone understood the task.

4 Considerations

Andreea Buturca

I started working on this project, feeling that understanding someone else's code it could be the hardest thing for me, as I didn't have any prior experience in programming. I felt that it would be difficult to reach my colleagues' programming level, but I was very motivated to work hard, to support the team and throughout the project I felt very responsible for every task I needed to perform.

For me working in a team with multicultural members didn't seem hard in the-beginning, as I have big experience, from work and from my previous higher education, to work in diverse teams, to accomplish different goals with specific deadlines. So, I started, the project feeling confident in my ability to work in a multicultural environment. But things, started to change during the project. In my opinion there were some communication problems in the team. During our project meetings, some of the matters were discussed in Slovak, English being used only at my pressing request or when something regarding my tasks was discussed. This made me feel that I am losing essential parts of the project development. All things considered it was impossible for me to be 100% involved in this project as I intended at the beginning.

We started the project making the requirements, use case description, and activity diagrams, which were mandatory for our hands in. In the beginning, they weren't so well structured, because we needed a better overview of what should the system include. Also, being a person who processes information actively, I understood information better after every matter was discussed, ideas were written down, and after I started to apply them. As a global learner, I needed the big picture before I could master details. Each time before I began a new section of the project, I had to skim through the entire interview to get a better overview. I also preferred information being presented visual and as a visual learner, I concentrate more on graphical information. That's why, together with Martin we drew on paper the GUI interface and that helped a lot to understand better what the system must consists of. I accepted when Martin suggested building the GUI in JavaFX, instead of Java swing, because we could have certain guidance from him, and could make our job easier. I continued with some parts of functionality of GUI, coordinated by Martin, who is an experienced programmer, good to work with and to learn from. I consider that our group was leaded by him with a good structure and specific tasks and deadlines. He was a great help in debugging, as well, when we had troubles.

In addition, during the project, I perceived information relying too much on sensing, tending to prefer what is familiar, and concentrate on facts I knew instead of being all the time innovative. Being a sensory learner, if some parts of the project seemed difficult to accomplish, I tried to follow a safe (known) and fast path. One example could be the part of the project where I preferred to implement lists instead of tables, to display data on the screen view because it was much easier.

Moreover, during the project, I experienced some problems with GIT hub, our tool for merging and combining codes. Because of that, sometimes during weekends I've got stuck, having merging conflicts, and not being able to synchronise my work with the main project.

We didn't spend a lot of time together doing the project because some needed a shorter period of time to work on the same amount than others, on the other hand, some are B person, others A person – me for example. As seen on Git punch card, the hours that were spent on the project differed a lot from A to B person, working hours started from 10 am, and lasted until 3 am in the morning. And I had to respect that each person is productive in different moments of the day. Moreover, in the last week, I met at school only with Karolina, because Martin and Marek left home. It was difficult to stay in contact with them, we had a lot of dead time, waiting for some group decision to be made, especially when we discovered some bugs in the system, I didn't know if we should leave it like that, or another member could fix it. The communication and the interest reached a low level. I am aware that if we could spend more time together, doing the project, and if during the meetings, only English would be spoken, it would be a great advantage for me. And I think this thing can be improved on the next semester project.

Karolina Beliharova

During the semester I was working with Andreea, we have been programming partners since the beginning of the semester. We got used to each other and I can say I like working with her, of course we had some problems but I am open-minded that means I do not have a problem to accept criticism. We always talked about the problem and then figured out it together. Martin and Marek sat in front of us and because both of them had some programming experiences from high school, they were able to help us during the classes when we were completely lost. We worked together on some tasks so we found out how to cooperate together and decided to be group for SEP.

This project was kind of challenging for me. I learnt how to work in team at my high school, but it was easier and different because no one had the motivation to work as hard as us now here. I have to learn how to work with people with different cultures background. Working with Marek and Martin was not so big problem, because our cultures are similar but on the other hand we had to respect Andreea that sometimes was not observed.

Martin was something like a leader of group because his programming skills are on higher level than others. It was our big advantage because he exactly knew what to do and he could do it really quickly. He was able to correct every problem that we met with code. In the beginning we were thinking about that I and Andreea would work on documentation and guys would work on code. During the project, Andreea had interesting ideas how to make GUI so she worked on code with guys and I did documentation but in the end she was able to help me. I think this work everyone hates the most. It sounds easy but taking notes and documentation is actually harder than it looks like. I would like to say I am glad that in our group were two girls and two boys. I got experience how to work with both genders and I hope it would help me in the future.

We agreed on meetings every Monday, Wednesday and Friday at 10. It was rewarding for us because we had some days off when we could relax from the project but actually I think everyone was working almost every day because we wanted to finish as soon as possible and had the project on high level. Some days we had team buildings, we spent time together without codes and tried to know each other. We enjoyed entertainment so we became really good friends and working together was then easier and funnier.

In the last week of the project period there was complication with communication in our group because guys had some plans so they went home earlier and I and Andreea stayed there until the project was not done. Firstly I did not have problem with this because Martin did all his job well and he made the hardest part of the program. But suddenly we met many problems this week and we had to do a lot of changes. Still I think for the next project I would appreciate that everyone stays and leaves when the project will be uploaded.

During the project I received a lot of new knowledge and experiences and I think in future it will help me. I learnt more about team work and met new cultures. I was satisfied with my group and how we handled the project.

Marek Lowy

I have worked with Andreea, Karolina and Martin since the beginning of the semester. We have got used to working together, at least in my opinion. We have found out how to work together and communicate. That was one of the reasons why I chose them for semester project too. In next few paragraphs I will write my thoughts about the project, working in team, solving difficult tasks etc.

This was the first time I've been working on this kind of project. Even though I learned how to work in team at high school, I found out that it's still not that easy. We've been meeting every two days in a week and I quickly realized that it's quite hard to split work between four people and keep track of what they are working on at the same time. At the beginning we didn't have much to do so we split roles. Karolina was working on the documentation, Martin was designing GUI and me with Andreea were working on all classes in model. But after project moved to later phases, it got a bit chaotic. Sometimes one member didn't know what to do until next meeting or two members worked on the same thing. That was when I realized that communication in the team is not as easy as it seems.

During the whole project we encountered experience problems several times. I and Martin have some programming experiences, but girls don't. It was not that easy thinking together about the whole project. We had a plan right after being introduced to the task and we didn't think it's hard, at least it didn't seem to be. I admit our first plans and vision have been quite more than required and we quickly let of them when we saw the look at the girls faces or realized how much of a work it would be. We learned that we have to explain our thoughts and sometimes we had to find good arguments to prove us right. We discussed and decided to use JavaFX GUI, because it was easier to make and looked much nicer. At the beginning I was scared that girls would find it hard to understand, but I was wrong. They caught up very fast and started working with it without any major problems. In the end I didn't notice any problems and differences as I thought about first.

The coding wasn't hard for me. As I mentioned I have programmed before so I can think in a programmer way. I did what I was told to do and had no problems. If I encountered something I did not know I just opened google and voila. But after we made model and started on doing methods I quickly learned that having something around 15 classes in head and thinking about dataflow between the in the same time isn't that easy.

Next the language made some issues too. Before we decided to work together we made rule to talk only in English. But since we are two Slovaks and one Czech in the group, sometimes we slipped away. Never on important things though. But I found it much easier to brainstorm with Martin in Slovak and after that to conclude out mishmash in English to the rest of the group. I guess have to work on controlling that, because even though it's easier way I still find it mean to others.

The last thing I felt was difference in dedication to work. There were days when one of the group members wasn't having a "productive day", often me. Our schedule was meeting at school every Monday, Wednesday and Friday and I am that kind of person that wakes up feeling very productive at home and not at school someday or feeling tired at home but full of energy outside the other day. And that had sometimes conflicts with our schedule. I never saw that as a problem but this project showed me that I should really learn how to control that part of me.

In conclusion, this project showed me my weak sides in teamwork and communication on which I have to work. Also I learned doing this just from the interview. It is a great way to learn since I had free hand, and was forced to learn new stuff if I wanted to use it.

Martin Janosik

I decided to work with Marek, Andreea and Karolina because we have been working together since beginning of semester. I had no problems with cooperating with them during our project so I think it was good decision. The task we were given was simple and straight. I like idea of giving students interview since this is much closer to real life work than predefined model that has to be implemented. In interview we had specified some requirements, but we had free hand at the same time.

We planned our meetings for every two days since we came to conclusion that mostly we are more productive at home. I usually split work and told other members what they should do at home. We primarily used GitHub as our share point since it allowed us to code at the same time, share and merge code right away, and see all the changes other members did. On our meeting we discussed questions and worked on the code together. We have worked on the documentation for the project while coding, because we have found it more time efficient.

At the beginning I was designing GUI while other members worked on the model. This was the point where I encountered first complications. The interview didn't specify how the model should look like and every team member had different idea. We had to discuss, but it didn't take long to find reasonable solution. I found out that communication in the team is really important. It the beginning I was a bit worried. I have some professional programming experience and I was scared that other team members wouldn't understand my code, but I was wrong. The understood quickly and did their work very well.

Language wasn't problem at all. But it's true that sometimes I realized that I argued with mark about the code in Slovak. It wasn't big problem since he summarized our conclusion to the group afterwards. I even found it more efficient that way.

At the testing phase we found some bugs and things which didn't work. I have to admit some were pretty hard to find and solve. Most often the reason was that other member implemented some methods and I used them. It seems easy but understanding somebody's code is not easiest thing to do. The logic in the code is simple but the intention isn't. Fortunately, communication solved most of these problems for us. We understood each other and then quickly solved the problem. Also in the half of the project we have decided not to use MyDate class we have made during our SDJ lessons, but instead to use built in Java Date and LocalDate. The reason was simple; those were working better. But switching classes while in the middle of project was tricky. We had to redo a lot of code to work properly.

To sum it all up, I gained a lot of experiences while doing this project. It helped me to develop my communication skills, teamwork and cooperation. I liked the way task was presented and possibility to add things to the project. It made me feel more motivated, and I am always more productive while having fun and enjoying my project.

5 Target audience

Before we started the reports, we took into consideration the target audience. The project report and appendices have an expert audience and it was used a formal language, with specialized and technical terminology, except for the GUI User guide, where it was used an informal language, so each employee could understand the indications of the system that was implemented. The process report uses an informal language as we expressed our personal reflections and considerations.

6 List of tasks

Andreea Buturca

- Activity Diagrams
- Use Case Descriptions
- Requirements
- Use Case Diagram
- Introduction
- Design class diagram breakdown
- Testing
- User guide to the system
- Graphical user interface design – phase 1.
- GUI functionality: 1. Reservation controller: loadCustomerList, loadPassenggerList, addCustomer, addpassenger, removePassenger. 2. Chauffeur controller: addchauffeur, deleteChauffeur, loadlist
- Chauffeur Class, Customer, Passenger, toStrings in all classes.
- Group policy

Karolina Beliharova

- Activity Diagrams
- Use Case Descriptions
- Requirements
- Project Report
- Process Report
- Use Case Diagram

Marek Lowy

- Coding classes and methods in model
- BusController
- Consideration
- Use Case Diagram
- Activity Diagrams
- Use Case Descriptions
- Sequence Diagram
- JavaDoc

Martin Janosik

- Graphical user interface design and functionality
- Controllers, Trip and Reservation controllers, main controller
- Connection between controllers and GUI
- Supervising work on program, guiding
- Basic models design
- Work planning, dividing, giving tasks to members
- Datahander – connection between controllers and models, data reading and storing
- Validation methods
- Main class
- Alert windows
- Coding classes and methods in models
- Code cleaning and sorting

7 Supervisor meetings

The meetings with the supervisors were necessary at the beginning because we did not understand enough what exactly we need to prepare for the assignments. We met them weekly to clarify what are the expectations of product owner and to have explained some of the wishes from the list.

In the period of working for the process and project report, we needed to meet the supervisors again. We had a lot of questions because only one member had an experience with project planning and the rest had not the experience with the reports before.

At every meeting, the supervisors were willing to help us and give good advices for every task.

23.11.2016

The supervisor guide us on what should we change in the second. He helped us to develop a better overview of the system and gave us pieces of advice about requirements, the class diagram, GUI development, JavaFX.

30. 11. 2016

Consultation of requirements, use case diagram and activity diagram. We were asked to specify Trip's requirements. We received advice how to improve our activity diagram and use case description.

9. 12. 2016

Consultation of class diagram, functionality of GUI, to improve the prices for extra services according to passenger's age, consultation of use case diagram - to add a new use case with the list of tours sorted by date, questions about Javadoc, and bin file.

15.12. 2016

Consultation of group and Belbin roles and individual considerations. We were asked to write about writing styles. We received advice how to improve our project report specially introduction and abstract. We obtained information about format of our documentation.

8 Bloom's forms

Andreea Buturca

Fill in this form – include it in your portfolio – discuss it with the rest of the group Date.....	Bloom's level	Keeping a portfolio	Reflecting on learning	System development	SCRUM	Java Programming	Object oriented design and programming	UML	Web Programming	Database design	Written English	Spoken English	Team working	Sharing knowledge	Project planning	Presentation / exam skills
Excellent	6															
	5															
Good	4															
	3															
	2															
Basic	1															
No knowledge	0															

Karolina Beliharova

Fill in this form – include it in your portfolio – discuss it with the rest of the group Date.....	Bloom's level	Keeping a portfolio	Reflecting on learning	System development	SCRUM	Java Programming	Object-oriented design and programming	UML	Web Programming	Database design	Written English	Spoken English	Team working	Sharing knowledge	Project planning	Presentation / exam skills
Excellent	6															
	5															
Good	4															
	3															
	2															
Basic	1															
No knowledge	0															

Marek Lowy

Fill in this form – include it in your portfolio – discuss it with the rest of the group Date.....	Bloom´s level	Keeping a portfolio	Reflecting on learning	System development	SCRUM	Java Programming	Object oriented design and programming	UML	Web Programming	Database design	Written English	Spoken English	Team working	Sharing knowledge	Project planning	Presentation / exam skills
Excellent	6															
	5															
Good	4															
	3															
	2															
Basic	1															
No knowledge	0															

Martin Janosik

Fill in this form – include it in your portfolio – discuss it with the rest of the group Date.....	Bloom´s level	Keeping a portfolio	Reflecting on learning	System development	SCRUM	Java Programming	Object oriented design and programming	UML	Web Programming	Database design	Written English	Spoken English	Team working	Sharing knowledge	Project planning	Presentation / exam skills
Excellent	6															
	5															
Good	4															
	3															
	2															
Basic	1															
No knowledge	0															

ICT Engineering

SEP 1Y A16

Project Report

Group 1

Andreea Buturca 253691

Karolina Beliharova 253810

Marek Lowy 253652

Martin Janosik 253934

Supervisors

Steffen Vissing Andersen

Birgitte von Fyren Balsløv

Date: 16. 12. 2016

VIA University College

Table of contents

List of figures and tables

Abstract

VIA Bus is a company located in Horsens, Denmark with Trip Driver as the manager. At VIA Bus, you can either rent a bus with a chauffeur – driving to a destination of your choice, or travel by bus to one of the predefined locations, i.e. a few European countries and some Danish sites and events.

VIA Bus needs a system to keep track of the tours, chauffeurs, bus routes and customers, and therefore the manager Trip Driver decided to contact us to create it. The system is based on Java and meets the requirements of the customer. The system is user friendly, can be used by any employee of the company, it contains menus, tabs, lists and buttons exactly as Trip Driver required. There are several options in the main menu. User can choose between making new reservation, creating tour, finding reservation or tour in the system, adding busses or chauffeurs.

One of the main tasks is creating tour. Each tour can be created for a certain number of people. The number is assigned to that specific tour and depends on the bus type. The customer can be chosen for private tour, and also a date interval, destination, stops, bus type, and chauffeur.

1 Introduction

The project presented is based on an interview with VIA Bus Company that offers different types of tours for customers. They organize predefined trips and travels and make new trips by renting a bus with a chauffeur. The company is in need for a better system to monitor, record and organize all the services they offer.

The system must meet VIA Bus requirements, run on Java, which means it must be used on various platforms. Simple and usability tested, GUI must be implemented to make sure that the employees will have no problems using it. It must be a single user system, running on one single front desk.

The VIA Bus company needs a powerful tool that will handle every aspect which the employees might encounter. The system must be implemented simple, and must handle all reservations, trips and travels, to keep track of all the tours, chauffeurs must be able to manage prices for each reservation, prices for extra services, discounts. The system will automatically calculate prices for each reservation according to travel type, extra services, to create new tours and reservations, edit, cancel and remove them at any given time. Also it must be possible to register new chauffeurs as temps, when needed. The employees must find all available chauffeurs and busses, to assign them new tours according to their preferences and also must keep track of chauffeurs while they are on route. All the data has to be saved in computer using database which will be easy to back up in case of losing data.

The system is developed on requirements, use case descriptions and diagram, activity, class and sequence diagram, followed by implementation, test parts, giving clear and accurate results.

2 Analysis

2.1 Requirements

1. The employee should be able to search through the list of destinations for the two kinds of tours: standard trips and travels, and private trips and travels with a private bus and a chauffeur.
2. The employee should be able to see all available seats for each tour.
3. The employee should be able to make a reservation for available seats for given destination, date, and time.
4. The employee should be able to see details about reservation as price, extra services, discount, amount of passengers, customer's name, address, email, phone number, the number of points given to each customer, passenger's name, birthday, destination, departure time and date, bus type and chauffeur's dates.
5. For private customers, the employee should be able to register the name, address, the passenger's birthday, and when it is permitted the email address for the newsletter.
6. For companies, the employee should be able to register the name, address, phone number of the person that is making the reservation and the company's name.
7. The employee should be able to identify the profile customer: company or private person, including at least name, address, email, and phone number.
8. The employee should be able to find data about customer in the list of customers, to check the number of points and if it's a frequent customer.
9. The employee should be able to register chauffeurs by: name, address, email, phone, date of birthday, employee ID, preferences for tour distance: long, medium, short, for bus type: classic, mini, party luxury.
10. The employee should be able to identify the chauffeurs with full time contract and temps.
11. The employee should be able to see all the details about the chauffeurs in the list of chauffeur.
12. The employee should be able to register all the busses by selecting the type: classic, mini, party luxury, registering the number plates and number of seats per bus.

13. The employee should be able to see all the details about the busses in the list of busses
14. The employee must be able to register the hour, date and place for the tours' arrival, departure and stops.
15. The employee should be able, in creating tour, to select which bus is available for next tour, selecting in the list of busses one bus type.
16. The employee should be able to find an available chauffeur after the suitable bus was found and time and date for the tour were chosen.
17. The employee should be able to search through the list of chauffeurs: full time or temps.
18. The employee should be able to hire a bus-and-chauffeur, for a private tour, after searching through the list of tours.
19. The employee should be able to add extra services such as: food, accommodation, tickets, suitable for the private tours, according to customer requirements.
20. The employee must be able to find, edit or remove the tours.
21. The employee must be able to find, edit or remove the reservations.
22. The employee should be able to add additional notes for each reservation, in order to set the correct price for each customer.
23. The employee should be able to set the prices for extra services for each reservation according to passenger's age.
24. The employee should be able to set a lower price for extra services for persons under 18.
25. The employee should be able to reward and give discounts to customers before setting the total price.
26. The employee should be able to save reservation if there are some changes.
27. The system stores all the data in database.
28. The system must be implemented in Java

2.2 Use Case Diagram

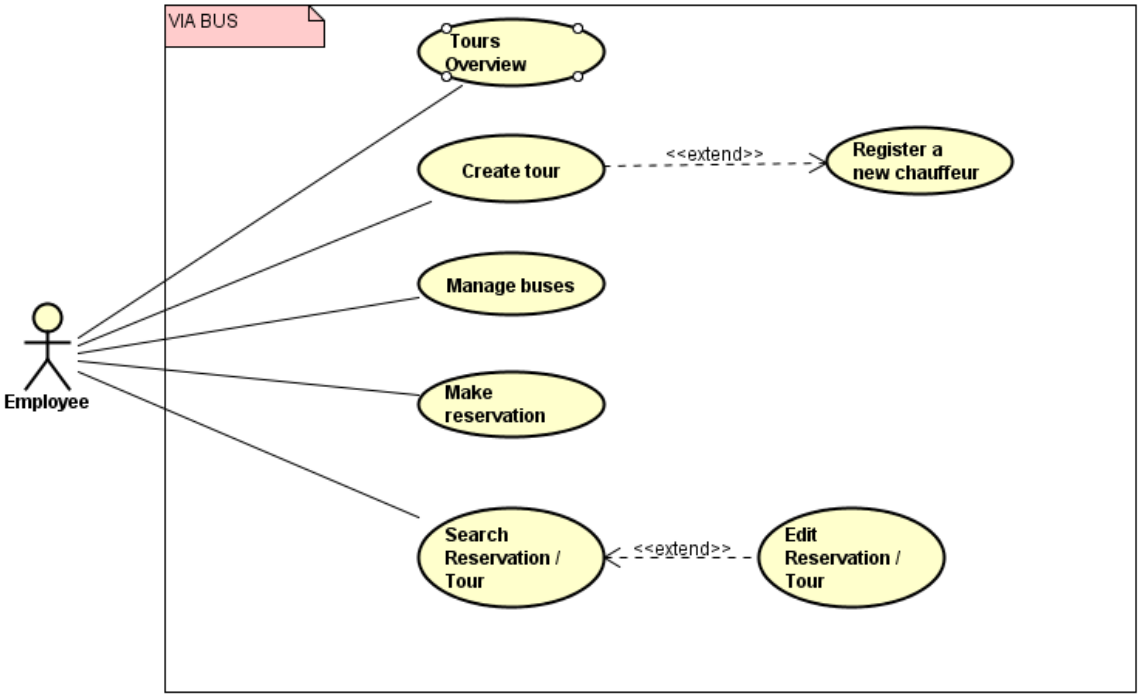


Figure 1: Priority Use Case Diagram

Tours Overview: All tours will be shown sorted by current date.

Create tour: The Employee is asked to enter information about the new tour: date, time, departure, destination, distance, he must choose a bus from the bus list and a chauffeur from the chauffeur list, fill in extra services, price, bus stop and time for break. If it is a private tour, the employee must fill data about customer, otherwise the tour is added to the list like a standard tour.

Register a new chauffeur: The Employee is asked to enter information about the new chauffeur using the following parameters: name, address, phone number, email address, employee ID (5-digit number), and type of contract: full-time or vicar. If it is full-time, the employee can specify wishes for bus, distance and the chauffeur is added to the list

Manage buses: The employee is asked to enter information about the new bus: type, registration plate and number of seats and the bus is added to the list. The employee is able to remove the bus from the list.

Make reservation: The employee can make a reservation, adding the customer and the passengers, setting the price for extra services, according to passenger's age, and discounts.

Find Reservation/Tour: The employee can search a reservation by customer data: name / company name, address, email and phone number. The employee is able to remove a reservation, can find a tour by searching with the following parameters: destination, departure, date, and is able to remove a tour from the list.

Edit reservation/tour: The employee can change any information about reservation/tour.

2.3 Use Case Description

"Create tour" is the most important part of the system because the purpose of this project is to develop a system that rents a bus with a chauffeur driving to a destination of customer's choice, or travel by bus to one of the predefined locations. The documentation for all other classes and Use Cases can be found in the appendices. The following pages of the documentation (sequence diagram, activity diagram, and implementation) will be orientated to the "Create tour" use case.

Check appendix 2 – Use Case Descriptions

ITEM	VALUE
UseCase	Create tour
Summary	System creates a new trips or travel
Actor	Employee
Precondition	none
Postcondition	New tour has been saved into database.
Base Sequence	1.User must choose the start date of the Tour. 2.User enters time of departure. 3.User choose the date of the arrival (end date). 4.User enters time of the arrival. 5.If User wants to enter name of the stop place, use case continues with branch sequence 1 . 6.User enters or selects the place of departure. 7.User enters or selects the destination. 8.User enters the number of kilometers. 9.User selects the Bus Type. 10.System displays the list of Busses according to criteria entered. 11.User selects a Bus from the list 12.System displays the list of available chauffeurs according to time and date, number of km and bus type. 13.User selects a chauffeur from the list of Chauffeur displayed on the screen. 14.User selects one or more of the extra services: food, accommodation, tickets. 15.User enters the price according to bus type, distance, duration and extra services. 16.If User wants a private Trip, use case continues with branch sequence 7. 17.User can save the Tour into database.
Branch Sequence	1.User enters the place of stops. 2.User must enter time of the stop (in min). 3.User can add the stop in the list. 4.User can repeat the step 1, if more stops needs to be added. 5.User can remove any stop. User can continue with base sequence 6. 6.User can add a customer by adding: name, company name (optional), address, email, phon e number. 7.User can choose one customer from the list of customers Use case continues with base sequence 17.
Exception Sequence	No available chauffeur after entering time and date. 1-4 base sequence Use case ends. No available chauffeur after entering number of kilometers. 8 base sequence Use case ends. No available chauffeur after selecting a bus type. 11 base sequence Use case ends. No available bus type after selecting the bus type from the list. 11 base sequence Use case ends. No selected Bus from the list of busses: 9 base sequence. Alert Window to select a Bus. No chauffeur selected from the list of chauffeur: 13 base sequence. Alert Window to select a Chauffeur. Wrong data input in the time fields. 2, 4 base sequence. Alert Window for time. Wrong data input in the price fields. 15 base sequence. Alert Window for price.
Sub UseCase	Register a new chauffeur
Note	Creating new tour can be canceled without saving at any time

Table 1: Create Tour

2.4 Activity Diagram

The Create tour activity diagram displays step-by-step the process of creating new tour. The special thing about this use case is that the user can create a standard tour, which will appear in the list of tours and can be used for later reservations and in the same time, the user, can create a private tour, by renting a bus and a chauffeur for only one use, assigning it to a customer. Everything said it is shown in the diagram bellow.

Check appendix 1 – Activity diagrams.

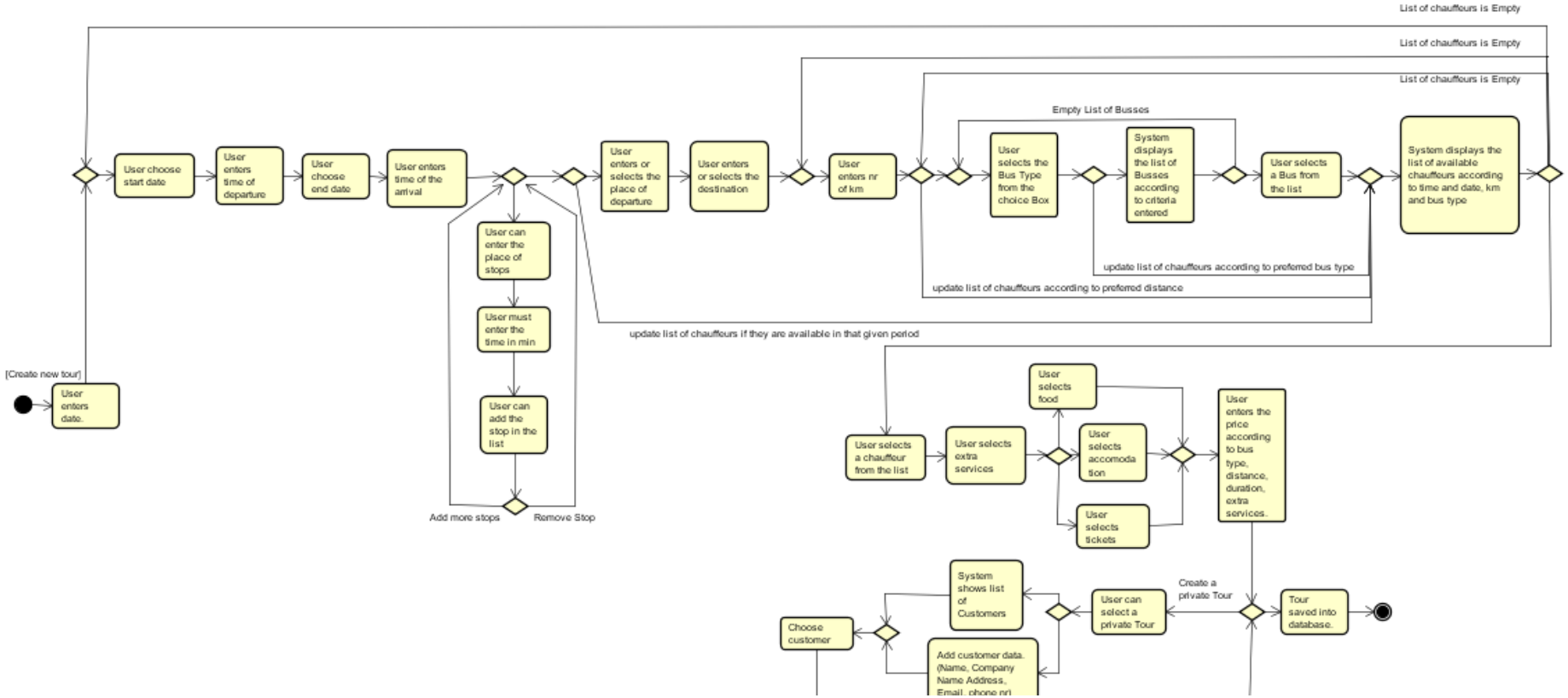


Figure 2: Create Tour

3 Design

3.1 Graphical User Interface (GUI)

In the beginning of the GUI implementation, the sketch was drawn on the paper, and that helped to understand better what the system must consists of. It was created a simple interface in JavaFX, in which it was used text fields, combo boxes, buttons, lists, radio buttons, check boxes, date picker, choice boxes.

One of the most important method from the program is "Make reservation". This method was designed on two pages, each representing a step. In first step, GUI was build taking into consideration that the user must search through the list of Tours, for the trip or travel, that customer needs to reserve. As seen in the picture bellow at this step it was used in implementation a text area for notes, 3 text fields for searching, and a list to display the tours.

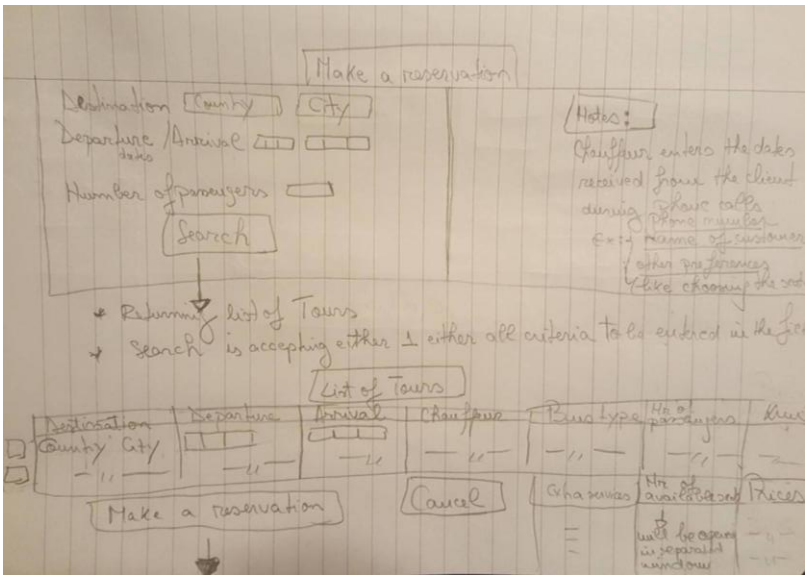


Figure 3: Making GUI 1

In the second step, GUI was developed so the user could enter the dates about the customer, passengers, price for extra services and discount. The user can make a reservation by selecting the customer, passengers from the lists. So, it was used 2 lists, date picker for birthday, and text fields to enter the dates.

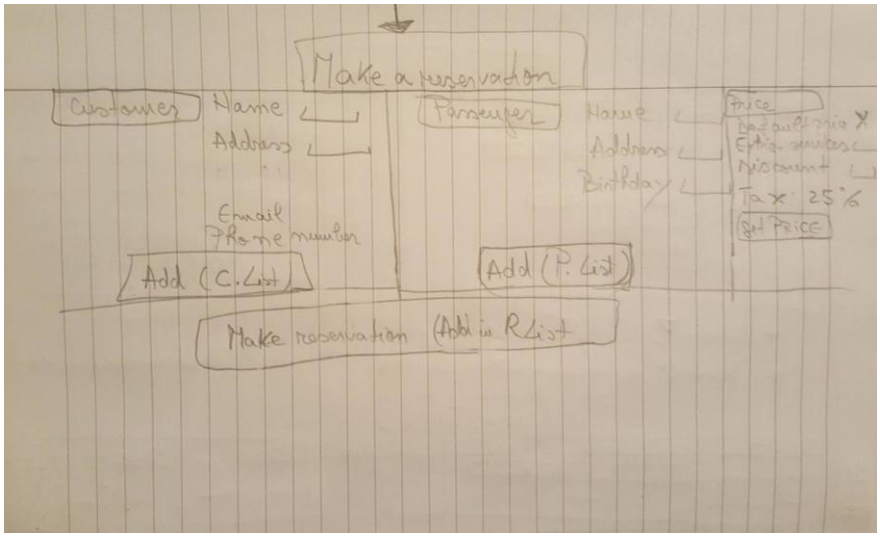


Figure 4: Making GUI 2

The result looks a little different, as they are better organized and implemented in JavaFX. It must be mentioned that, after clicking Choose Tour, the notes from first step will automatically appear in the second step of making reservation, with the same data, only if the user typed into the note section. Notes are a convenient way of storing information which the user needs in the workspace.

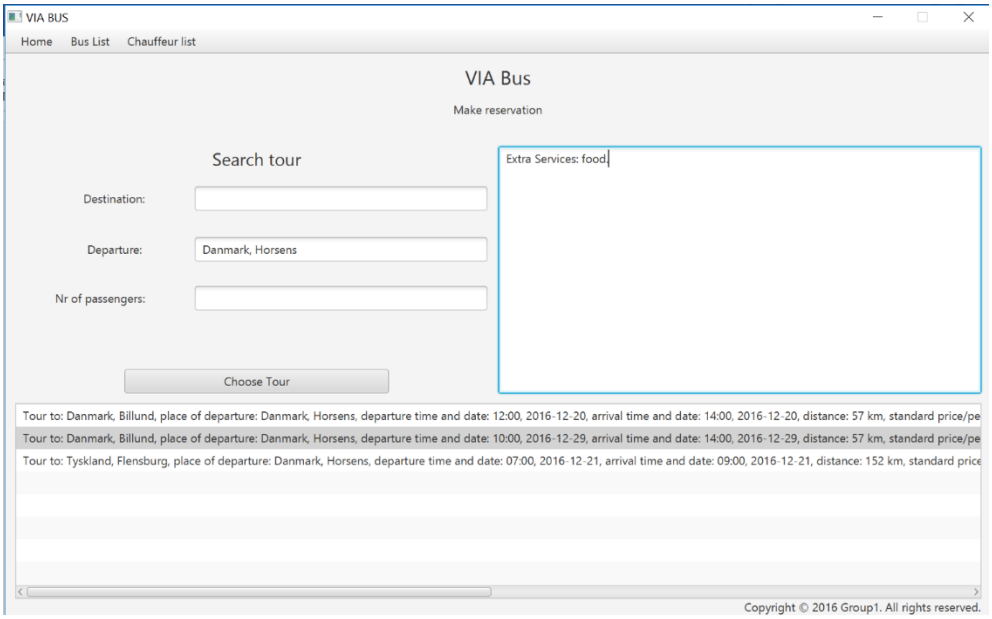


Figure 5: Making GUI 3

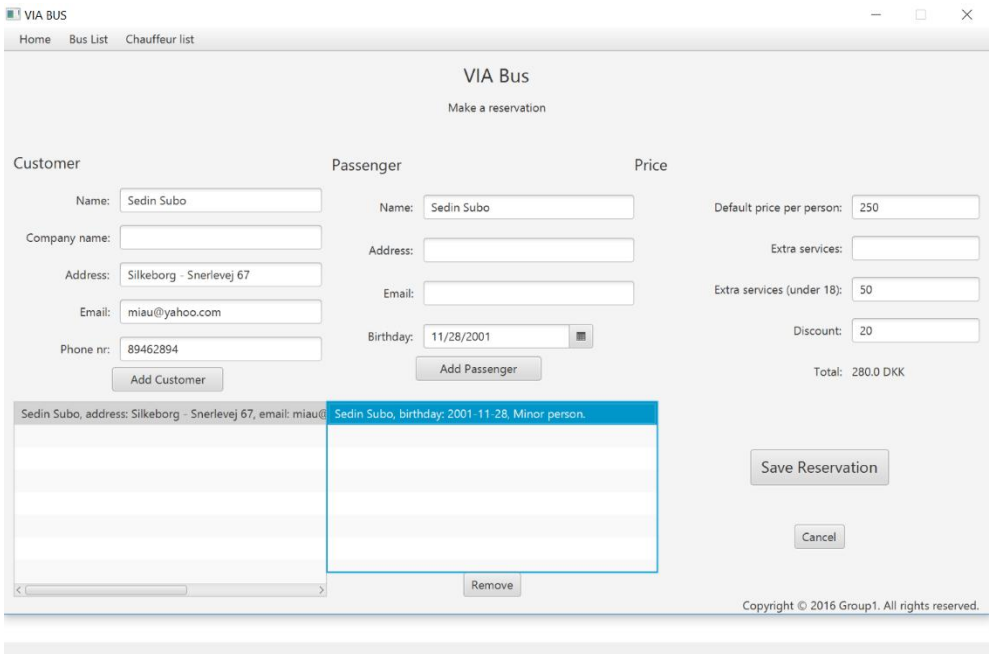


Figure 6: Making GUI 4

Check appendix 4 – User Guide

3.2 Model class diagram

This basic model shows us the structure of the system. The abstract class “Bus” is used to handle all information’s about the bus in the system such as registration plate and seat places, and through this class we get information about bus type from classes “LuxuryBus, PartyBus, ClassicBus, MiniBus”. The “Bus list” class stores all busses and information about them. The “Person” abstract class is extended by classes “Passenger, Customer, and Chauffeur” and is used to store data about them. “ChauffeurList” is used to hold a list of chauffeurs and it can sort chauffeurs according to their, availability, preferences for bus, distance. “CustomerList” is used to hold a list of customers and sort them according to find criteria and the same is for “PassengerList” with passengers. Every object of class “Reservation” stores information about each reservation. “ReservationList” is used to hold a list of reservation and it can filter them by customer’s dates. Information in class “Destination” is about place and stops and it is stored to “DestinationList”. In class “Trip” is created new tour and all data is saved to “TripList”, also the class TripController is responsible for creating tours. The Class Controller is responsible for verification of all the data. The class “Main” contains a main void function, helping to launch the system. The class “DataHandler” is used for connecting database with the “Main” class and helps us to have fixed connection with database when we do some changes like add bus/reservation/chauffeur.

3.3 Class diagram

Check appendix 3 – For extended Class Diagram

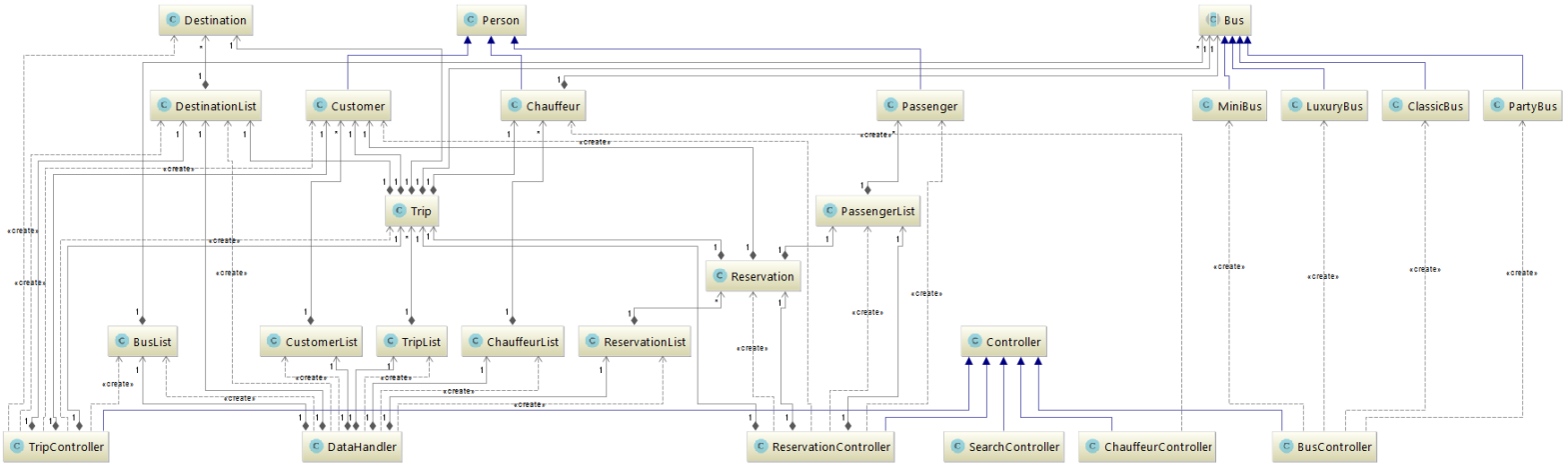


Figure 7: Class Diagram

3.3.1 Class Diagram Breakdown

One of the most important part of our program was to create new trip. In the Trip class, the user must specify all the dates needed to create a trip: the time – start and end (String variables), the date of the trip (taking variables of type LocalDate), the departure, the destination (from Destination class), the distance (integer), the bus type (Class Bus), the chauffeur (Class Chauffeur), available places and possibly extra services (food, accommodation, tickets - Booleans) and stops (Class DestinationList). At the end, the price is specified. The class has as methods getters for each instance variable, for extra services Booleans - getters and setters, for customer and the stops - setters. To return a proper format for the tours, the method toString returns the Booleans for extra services and for private tour, only if they are true, in String format. Also, it checks if there are stops during the tour and returns if they are tru. The method setCustomer is used only for private tours and assigns a customer to the trip, also it assigns the free places to 0, because a customer it's supposed to rent a full bus.

Trip	
freeSpaces	Integer
bus	Bus
chauffeur	Chauffeur
pickUpPoint	Destination
destination	Destination
stops	DestinationList
dateStart	LocalDate
dateEnd	LocalDate
timeStart	String
timeEnd	String
customer	Customer
isPrivate	boolean
privateString	String
distance	int
price	Integer
duration	String
dateObjStart	Date
dateObjEnd	Date
food	boolean
accommodation	boolean
tickets	boolean
Trip(Bus, Chauffeur, Destination, Destination, int, LocalDate, String, LocalDate, String,	
getDuration(Date, Date	String
isFood()	boolean
setFood(boolean)	void
isAccommodation()	boolean
setAccommodation(boolean)	void
isTickets()	boolean
setTickets(boolean)	void
getBus()	Bus
getTimeStart()	String
getTimeEnd()	String
getFreeSpaces()	Integer
getPickUpPoint()	Destination
getDestination()	Destination
getChauffeur()	Chauffeur
getStops()	DestinationList
setStops(DestinationList)	void
getDateStart()	LocalDate
getDateEnd()	LocalDate
getCustomer()	Customer
setCustomer(Customer)	void
isPrivate()	boolean
getDistance()	int
getPrice()	int
getDateObjStart()	Date
getDateObjEnd()	Date

Table 2: Class Diagram Breakdown

3.4 Sequence diagram

The sequence diagram shows process of creating new tour. Firstly, data about dates and places of departure and destination are entered. After entering distance and bus type, available chauffeurs are found and shown and then available buses are found the same way. Then extra services, bus, chauffeur are picked and price entered. At the end the stops can be entered one by one, by entering place of stop and time. After user decided to create tour the data is validated and if some mistakes occurred, alert is shown. Then trip is created with entered data and saved to list of trips. User is shown success dialog.

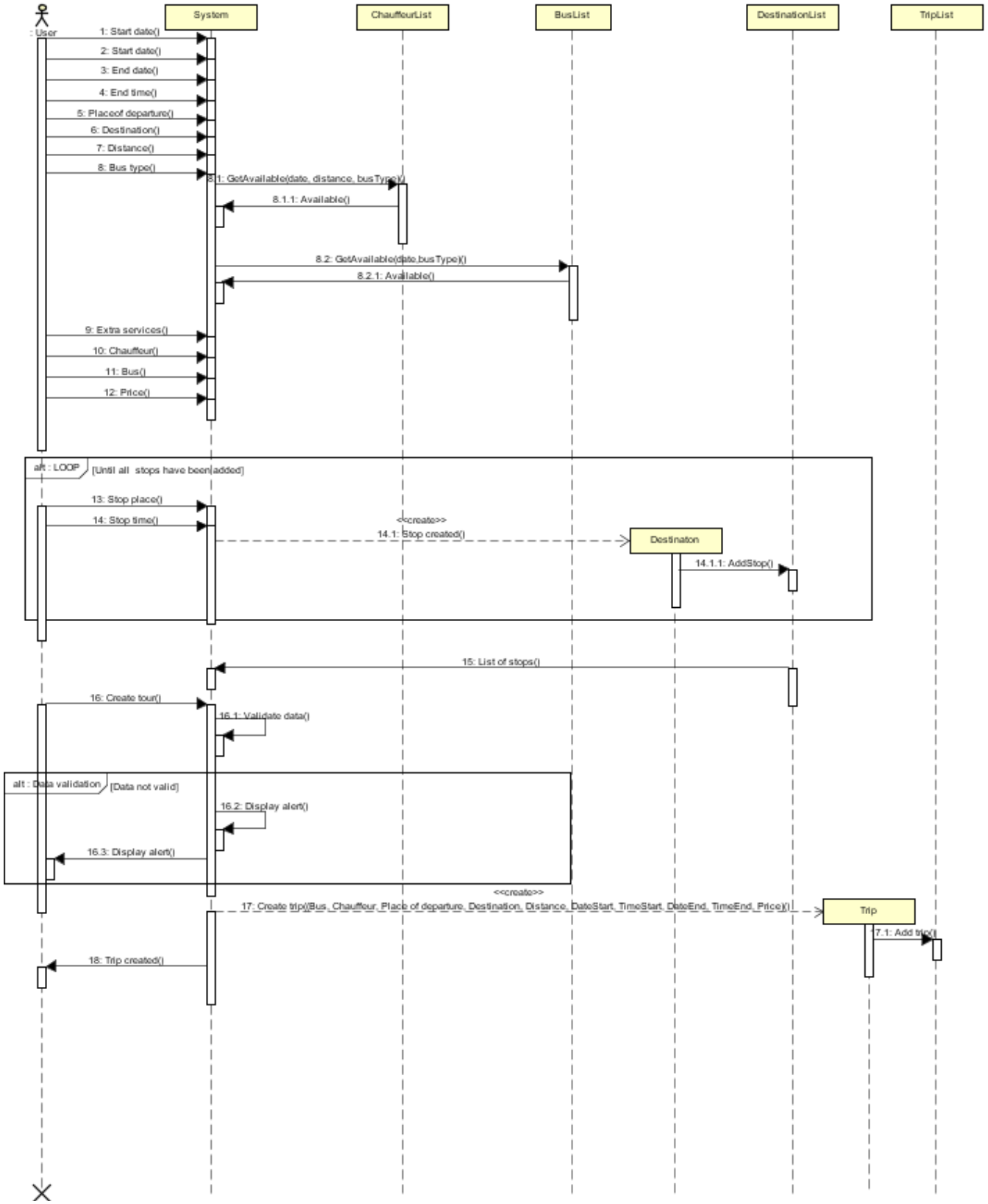


Figure 8: Sequence Diagram

4 Implementation

One of the most important part of our program was to create new trip/tour.
In order to create a new tour the user has to specify dates, time, departure, destination, distance, bus type and possibly extra services and stops. At the end the price is specified. The tour is created by clicking on button create tour.

```
package main.Controller;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyEvent;
import javafx.stage.Modality;
import javafx.stage.Stage;
import main.Main;
import main.Model.*;

import java.io.IOException;
import java.net.URL;
import java.util.Date;
import java.util.ResourceBundle;

/**
 * Class that manages trips.
 *
 * @author IT-1Y-A16 Group 1
 */

public class TripController extends Controller implements Initializable {


    public TextField fieldStartTime;
    public TextField fieldEndTime;
    public ComboBox fieldDestination;
    public ComboBox fieldDeparture;
    public TextField fieldDistance;
```


```
    public TextField fieldPrice;
    public DatePicker startDatePicker;
    public DatePicker endDatePicker;
    public CheckBox checkPrivateTrip;
    public Button CreateTourBtn;
    public ListView busListview;
    public ChoiceBox busType;
    public ComboBox stopName;
    public Button addStopBtn;
    public Button removeStopBtn;
    public ListView stopsList;
    public TextField stopTimeField;
    public ListView chauffeurList;
    public CheckBox foodCheckBox;
    public CheckBox accommodationCheckBox;
    public CheckBox ticketCheckBox;

    //Add customer
    public TextField fieldCustomerName;
    public TextField fieldCustomerCompany;
    public TextField fieldCustomerAddress;
    public TextField fieldCustomerEmail;
    public TextField fieldCustomerPhone;
    public ListView customerList;
    public Button saveCustomerBtn;
    public Label tourLabel;
    private Customer customer = null;
    private TripController mainController;

    //list for stops
    private DestinationList stops = new DestinationList();

    private Trip oldTrip;


    @Override
    public void initialize(URL location, ResourceBundle resources) {

        if (fieldStartTime != null) {
            loadBusList();
            loadChauffeurList();

            ObservableList<Destination> destinationItems =
FXCollections.observableArrayList();
```


```
destinationItems.addAll(DataHandler.getDestinationList().getArrayDestination());
            fieldDestination.setItems(destinationItems);
            fieldDeparture.setItems(destinationItems);
            stopName.setItems(destinationItems);
        }

        if (customerList != null) loadCustomerList();

    }

    public void getDataChoice(ActionEvent actionEvent) {
        loadBusList();
        loadChauffeurList();
    }

    public void getDataFromField(KeyEvent keyEvent) {
        loadBusList();
        loadChauffeurList();
    }

    public void callCustomerList(KeyEvent keyEvent) {
        loadCustomerList();
    }

    private void loadChauffeurList() {
        ChauffeurList chauffeurs;

        chauffeurs = DataHandler.getChauffeurList().copy();
        chauffeurs.removeAllVicars();

        if (validateEmptyField(fieldDistance) &&
validateNumberField(fieldDistance)) {
            chauffeurs =
chauffeurs.getAllByPrefferedDistance(Integer.parseInt(fieldDistance.getText()));
        }
        chauffeurs =
chauffeurs.getAllByPrefferedBus(busType.getValue().toString());

        for (Chauffeur chauffeur :
DataHandler.getChauffeurList().getAllVicars().getArrayChauffeur()) {
            chauffeurs.add(chauffeur);
        }
```


```
        if (startDatePicker.getValue() != null && endDatePicker.getValue() != null
&& validateTimeField(fieldStartTime) && validateTimeField(fieldEndTime)) {
            String[] lineToken = fieldStartTime.getText().split(":");
            int hours = Integer.parseInt(lineToken[0]);
            int minutes = Integer.parseInt(lineToken[1]);
            Date dateStart = new Date(startDatePicker.getValue().getYear() - 1900,
startDatePicker.getValue().getMonthValue(),
startDatePicker.getValue().getDayOfMonth(), hours, minutes);
            lineToken = fieldEndTime.getText().split(":");
            hours = Integer.parseInt(lineToken[0]);
            minutes = Integer.parseInt(lineToken[1]);
            Date dateEnd = new Date(endDatePicker.getValue().getYear() - 1900,
endDatePicker.getValue().getMonthValue(), endDatePicker.getValue().getDayOfMonth(),
hours, minutes);
            chauffeurs = chauffeurs.getAvailable(dateStart, dateEnd);
        }

        chauffeurList.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);
        ObservableList<Chauffeur> items = FXCollections.observableArrayList();
        if (chauffeurs.getSize() != 0) {
            items.addAll(chauffeurs.getArrayChauffeur());
        }
        chauffeurList.setItems(items);
    }

    private void loadBusList() {

        BusList buses;

        if (busType.getValue().equals("Mini Bus"))
            buses = new
BusList(DataHandler.getBusList().searchByType("main.Model.MiniBus"));
        else if (busType.getValue().equals("Party Bus"))
            buses = new
BusList(DataHandler.getBusList().searchByType("main.Model.PartyBus"));
        else if (busType.getValue().equals("Luxury Bus"))
            buses = new
BusList(DataHandler.getBusList().searchByType("main.Model.LuxuryBus"));
        else buses = new
BusList(DataHandler.getBusList().searchByType("main.Model.ClassicBus"));

        if (startDatePicker.getValue() != null && endDatePicker.getValue() != null
&& validateTimeField(fieldStartTime) && validateTimeField(fieldEndTime)) {
            String[] lineToken = fieldStartTime.getText().split(":");
```


```
            int hours = Integer.parseInt(lineToken[0]);
            int minutes = Integer.parseInt(lineToken[1]);
            Date dateStart = new Date(startDatePicker.getValue().getYear() - 1900,
startDatePicker.getValue().getMonthValue(),
startDatePicker.getValue().getDayOfMonth(), hours, minutes);
            lineToken = fieldEndTime.getText().split(":");
            hours = Integer.parseInt(lineToken[0]);
            minutes = Integer.parseInt(lineToken[1]);
            Date dateEnd = new Date(endDatePicker.getValue().getYear() - 1900,
endDatePicker.getValue().getMonthValue(), endDatePicker.getValue().getDayOfMonth(),
hours, minutes);
            buses = buses.getAvailable(dateStart, dateEnd);
        }

        busListview.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);
        ObservableList<Bus> items = FXCollections.observableArrayList();
        items.addAll(buses.getArrayBuses());
        busListview.setItems(items);


    }

    private void loadCustomerList() {
        CustomerList customers = DataHandler.getCustomerList();

        if (validateEmptyField(fieldCustomerName))
            if (customers.findAllByName(fieldCustomerName.getText()) != null)
                customers = customers.findAllByName(fieldCustomerName.getText());
        if (validateEmptyField(fieldCustomerPhone))
            if (customers.findAllByPhone(fieldCustomerPhone.getText()) != null)
                customers = customers.findAllByPhone(fieldCustomerPhone.getText());
        if (validateEmptyField(fieldCustomerCompany))
            if (customers.findAllByCompanyName(fieldCustomerCompany.getText()) !=
null)
                customers =
customers.findAllByCompanyName(fieldCustomerCompany.getText());

        ObservableList<Customer> customerItems =
FXCollections.observableArrayList();

        if (customers.getSize() != 0) {
            customerItems.addAll(customers.getArrayCustomer());
        } else {
            customerItems.addAll(DataHandler.getCustomerList().getArrayCustomer());
```


```
        }

        customerList.setItems(customerItems);
    }

    private void loadStops() {
        ObservableList<Destination> destinationItems =
FXCollections.observableArrayList();
        for (Destination destination : stops.getArrayDestination()) {
            destinationItems.add(destination);
        }
        stopsList.setItems(destinationItems);
    }

    public void handleStops(ActionEvent actionEvent) {
        if (actionEvent.getSource() == addStopBtn &&
validateNumberField(stopTimeField)) {
            stops.add(new Destination(stopName.getValue().toString(),
stopTimeField.getText()));
        }

        if (actionEvent.getSource() == removeStopBtn &&
stopsList.getSelectionModel().getSelectedItem() != null) {
            String[] lineToken =
stopsList.getSelectionModel().getSelectedItem().toString().split(", ");
            String stopNameTemp = lineToken[0].trim();
            stops.removeDestination(stops.findByName(stopNameTemp));
        }
        loadStops();
    }

    public void openCustomerView(ActionEvent actionEvent) {
        if (checkPrivateTrip != null) {
            Stage stage = Main.stage;
            if (checkPrivateTrip.isSelected()) {
                Stage window = new Stage();
                Parent root = null;
                try {
                    FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("../View/addCustomerData.fxml"));
                    root = fxmlLoader.load();
                    TripController CustomerViewController =
fxmlLoader.getController();
                    CustomerViewController.addMainController(this);
                } catch (IOException e) {
```


```
                    e.printStackTrace();
                }

                window.initModality(Modality.APPLICATION_MODAL);
                window.setTitle("Add Customer Data");
                window.setMinWidth(600);
                window.setMinHeight(400);
                window.setResizable(false);

                Scene scene = new Scene(root != null ? root : null);
                window.setScene(scene);
                window.show();
            }
        }

    }

    public void addMainController(TripController tripController) {
        mainController = tripController;
    }

    public void addCustomerData(ActionEvent actionEvent) {

        String alert = "There are some mistakes: ";
        int length = alert.length();

        if (!validateEmptyField(fieldCustomerName)) alert += "Name, ";
        if (!validateEmptyField(fieldCustomerAddress)) alert += "Address, ";
        if (!validateEmptyField(fieldCustomerEmail) ||
!validateEmail(fieldCustomerEmail))
            alert += "Email, ";
        if (!validateEmptyField(fieldCustomerPhone) ||
!validateLength(fieldCustomerPhone, 8)) alert += "Phone, ";

        if (length == alert.length()) {
            //save it DataHandler. .....
            boolean isCompany = validateEmptyField(fieldCustomerCompany);

            if (!isCompany) {
                DataHandler.getCustomerList().add(new
Customer(fieldCustomerName.getText(), fieldCustomerAddress.getText(),
fieldCustomerEmail.getText(), fieldCustomerPhone.getText()));
            } else if (isCompany) {
                DataHandler.getCustomerList().add(new
Customer(fieldCustomerName.getText(), fieldCustomerAddress.getText(),
```


```
fieldCustomerEmail.getText(), fieldCustomerPhone.getText(),
isCompany, fieldCustomerCompany.getText()));
            }
            DataHandler.save();
            successdisplay("Success", "Customer was created.");
            loadCustomerList();
        } else {
            //alert
            alertdisplay("Wrong Input", alert);
        }
    }

    public void chooseCustomer(ActionEvent actionEvent) {

        if (customerList.getSelectionModel().getSelectedItem() != null) {
            mainController.saveCustomerToMain((Customer)
customerList.getSelectionModel().getSelectedItem());

            Stage stage = (Stage) saveCustomerBtn.getScene().getWindow();
            stage.close();
        } else {
            alertdisplay("No customer", "Please choose one Customer");
        }
    }

    public void saveCustomerToMain(Customer customer) {
        this.customer = customer;
    }

    public void setEditData(Trip trip) {
        menu.setVisible(false);
        tourLabel.setText("Edit Trip");
        CreateTourBtn.setText("Edit");

        oldTrip = trip;

        fieldStartTime.setText(trip.getTimeStart());
        fieldEndTime.setText(trip.getTimeEnd());
        startDatePicker.setValue(trip.getDateStart());
        endDatePicker.setValue(trip.getDateEnd());
        fieldDestination.setValue(trip.getDestination().toString());
        fieldDeparture.setValue(trip.getPickUpPoint().toString());
        fieldDistance.setText(Integer.toString(trip.getDistance()));
        fieldPrice.setText(Integer.toString(trip.getPrice()));
        checkPrivateTrip.setSelected(trip.isPrivate());
        foodCheckBox.setSelected(trip.isFood());
```


```
        accommodationCheckBox.setSelected(trip.isAccommodation());
        ticketCheckBox.setSelected(trip.isTickets());

        busListview.getSelectionModel().select(trip.getBus());
        busType.setValue(trip.getBus().getBusType());

        if (trip.getStops() != null) {
            stops = trip.getStops();
            loadStops();
        }

        chauffeurList.getSelectionModel().select(trip.getChauffeur());

        if (trip.getChauffeur() != null) {
            chauffeurList.getSelectionModel().select(trip.getCustomer());
        }
    }

    public void createTour(ActionEvent actionEvent) throws IOException {


        String alert = "There are some mistakes: ";
        int length = alert.length();

        if (!validateEmptyDate(startDatePicker)) alert += "Start Date, ";
        if (!validateEmptyDate(endDatePicker) || validateAdultDate(endDatePicker))
alert += "End Date, ";
        if (!validateEmptyField(fieldStartTime) ||
!validateTimeField(fieldStartTime)) alert += "Start time, ";
        if (!validateEmptyField(fieldEndTime) || !validateTimeField(fieldEndTime))
alert += "End time, ";
        if (!validateEmptyField(fieldDistance) ||
!validateNumberField(fieldDistance)) alert += "Distance, ";
        if (!validateEmptyField(fieldPrice) || !validateNumberField(fieldPrice))
alert += "Price, ";
        if (!validateEmptyCombo(fieldDestination)) alert += "Destination, ";
        if (!validateEmptyCombo(fieldDeparture)) alert += "Departure, ";
        if (busListview.getSelectionModel().getSelectedItem() == null) alert +=
"Bus, ";
        if (chauffeurList.getSelectionModel().getSelectedItem() == null) alert +=
"Chauffeur, ";

        if (length == alert.length()) {
            //save it DataHandler. .....
            Bus bus = (Bus) busListview.getSelectionModel().getSelectedItem();
```


```
            Chauffeur chauffeur = (Chauffeur)
chauffeurList.getSelectionModel().getSelectedItem();

            Destination pickUp;
            Destination destination;

            if
(DataHandler.getDestinationList().findByName(fieldDeparture.getValue().toString())
!= null) {
                pickUp =
DataHandler.getDestinationList().findByName(fieldDeparture.getValue().toString());
            } else {
                pickUp = new Destination(fieldDeparture.getValue().toString());
                DataHandler.getDestinationList().add(pickUp);
            }

            if
(DataHandler.getDestinationList().findByName(fieldDestination.getValue().toString()
) != null) {
                destination =
DataHandler.getDestinationList().findByName(fieldDestination.getValue().toString())
;
            } else {
                destination = new
Destination(fieldDestination.getValue().toString());
                DataHandler.getDestinationList().add(destination);
            }

            int distance = Integer.parseInt(fieldDistance.getText());

            Trip trip = new Trip(bus, chauffeur, pickUp, destination, distance,
startDatePicker.getValue(), fieldStartTime.getText(), endDatePicker.getValue(),
fieldEndTime.getText(), Integer.parseInt(fieldPrice.getText()));

            if (stops != null) {
                trip.setStops(stops);
            }
            if (checkPrivateTrip.isSelected() && customer != null) {
                trip.setCustomer(customer);
            }


            if (foodCheckBox.isSelected()) {
                trip.setFood(true);
            }
```


```
            if (accommodationCheckBox.isSelected()) {
                trip.setAccommodation(true);
            }
            if (ticketCheckBox.isSelected()) {
                trip.setTickets(true);
            }

            if (oldTrip != null) {
                DataHandler.getTrips().remove(oldTrip);
            }

            DataHandler.getTrips().add(trip);
            if (oldTrip != null) {
                successdisplay("Edited", "Trip was edited.");

                Stage stage = (Stage) accommodationCheckBox.getScene().getWindow();
                stage.close();
            } else {
                successdisplay("Created", "Trip was created.");
            }
            DataHandler.save();
            Parent root =
FXMLLoader.load(getClass().getResource("../View/mainScreen.fxml"));
            Scene scene = new Scene(root);
            Main.stage.setScene(scene);
            Main.stage.show();

        } else {
            //alert
            alertdisplay("Wrong Input", alert);
        }
    }
}
```

Class TripController is responsible for creating tours. The button create tour calls method called createTour. In the method all the data from GUI is verified, if there are some mistakes the alert is shown. The class Controller is responsible for verification of data. If there were no mistakes all the data is stored in variables and then trip is created. If stops were included method calls setStops method from trip. The method createTour is also responsible for editing existing trips. If the check box private tour was ticked new window for customer will appear and after the trip becomes privat = bus and chauffeur. At the end of the method the data is saved.


```
package main.Controller;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.MouseEvent;
import main.Main;
import main.Model.DataHandler;
import main.Model.Trip;
import main.Model.TripList;

import java.io.IOException;
import java.net.URL;
import java.time.LocalDate;
import java.time.Period;
import java.util.ResourceBundle;

/**
 * Class that manages data and files.
 *
 * @author IT-1Y-A16 Group 1
 */

public class Controller implements Initializable {

    public MenuBar menu;
    public MenuItem homeHome;
    public MenuItem homeTour;
    public MenuItem homeReserve;
    public MenuItem homeSearch;
    public MenuItem homeBus;
    public MenuItem homeBusAdd;
    public MenuItem homeDriver;
    public MenuItem homeDriverAdd;

    //main screen
    public Button createTour;
    public Button mkReservation;
    public Button findTrip;
```


```
    //tripList
    public ListView tripList;

    //bus list
    public Button addBusView;

    //chauffeur list
    public Button addChauffeur;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        if (tripList != null) {
            showList();
        }
    }

    /**
     * Loads and displays list of trips on the listview.
     */

    private void showList() {
        TripList trips = DataHandler.getTrips();
        trips.sort();
        ObservableList<Trip> data = FXCollections.observableArrayList();
        for (int i = 0; i < trips.getSize(); i++) {
            if
(trips.getArrayTrip().get(i).getDateStart().isEqual(LocalDate.now()))
data.add(trips.get(i));
            if
(trips.getArrayTrip().get(i).getDateStart().isAfter(LocalDate.now()))
data.add(trips.get(i));
        }
        tripList.setItems(data);
    }

    /**
     * Changes view in GUI.
     */

    public void changeView(MouseEvent mouseEvent) throws IOException {

        Parent root = null;

        //main
        if ((mouseEvent.getSource() == createTour)) {
```


```
            root =
FXMLLoader.load(getClass().getResource("../View/createTour.fxml"));
        } else if ((mouseEvent.getSource() == mkReservation)) {
            root =
FXMLLoader.load(getClass().getResource("../View/makeReservation.fxml"));
        } else if ((mouseEvent.getSource() == findTrip)) {
            root = FXMLLoader.load(getClass().getResource("../View/search.fxml"));
        }

        //bus view
        else if ((mouseEvent.getSource() == addBusView)) {
            root = FXMLLoader.load(getClass().getResource("../View/addBus.fxml"));
        }

        //chauffeur view
        else if ((mouseEvent.getSource() == addChauffeur)) {
            root =
FXMLLoader.load(getClass().getResource("../View/AddChauffeur.fxml"));
        }


        if (root != null) {
            Scene scene = new Scene(root);

            Main.stage.setScene(scene);
            Main.stage.show();
        }
    }

    /**
     * Changes view in GUI through menu bar.
     */

    public void changeViewMenu(ActionEvent actionEvent) throws IOException {

        Parent root = null;

        if ((actionEvent.getSource() == homeHome)) {
            root =
FXMLLoader.load(getClass().getResource("../View/mainScreen.fxml"));
        } else if ((actionEvent.getSource() == homeTour)) {
            root =
FXMLLoader.load(getClass().getResource("../View/createTour.fxml"));
        } else if ((actionEvent.getSource() == homeBus)) {
            root = FXMLLoader.load(getClass().getResource("../View/busList.fxml"));
```


```
        } else if ((actionEvent.getSource() == homeBusAdd)) {
            root = FXMLLoader.load(getClass().getResource("../View/addBus.fxml"));
        } else if ((actionEvent.getSource() == homeReserve)) {
            root =
FXMLLoader.load(getClass().getResource("../View/makeReservation.fxml"));
        } else if ((actionEvent.getSource() == homeSearch)) {
            root = FXMLLoader.load(getClass().getResource("../View/search.fxml"));
        } else if ((actionEvent.getSource() == homeDriver)) {
            root =
FXMLLoader.load(getClass().getResource("../View/chauffeurList.fxml"));
        } else if ((actionEvent.getSource() == homeDriverAdd)) {
            root =
FXMLLoader.load(getClass().getResource("../View/addChauffeur.fxml"));
        }

        if (root != null) {
            Scene scene = new Scene(root);
            Main.stage.setScene(scene);
            Main.stage.show();
        }

    }

    /**
     * Validates if textfied is not empty.
     */

    protected boolean validateEmptyField(TextField textField) {
        return !textField.getText().isEmpty();

    }

    /**
     * Validates if textfied contains only numbers.
     */

    protected boolean validateNumberField(TextField textField) {
        return textField.getText().matches("[0-9]+");

    }

    /**
     * Validates if textfied contain double number.
     */
```


```
    protected boolean validateDoubleNumberField(TextField textField) {
        return textField.getText().matches("[0-9]+.[0-9]+");

    }

    /**
     * Validates if textfied contains registration plate, two letters and three
numbers.
     */

    protected boolean validateNumberPlate(TextField textField) {
        return textField.getText().matches("[A-Z]{2}[0-9]{5}");

    }

    /**
     * Validates if textfield text has given length.
     */

    protected boolean validateLength(TextField textField, int length) {
        if (length < 1) length = length * (-1);
        return textField.getText().length() == length;

    }

    /**
     * Validates if textfied contains time.
     */

    protected boolean validateTimeField(TextField textField) {
        return textField.getText().matches("([01]?[0-9]|2[0-3]):[0-5][0-9]");

    }

    /**
     * Validates if datepicker is empty.
     */

    protected boolean validateEmptyDate(DatePicker datePicker) {
        return datePicker.getValue() != null;

    }

    /**
```


```
     * Validates if person is adult.
     */

    protected boolean validateAdultDate(DatePicker datePicker) {
        LocalDate birthDate = datePicker.getValue();
        LocalDate now = LocalDate.now();
        if (birthDate != null) {
            int age = Period.between(birthDate, now).getYears();
            return (age >= 18);
        } else {
            return false;
        }

    }

    /**
     * Validates if combobox is empty.
     */

    protected boolean validateEmptyCombo(ComboBox comboBox) {
        return comboBox.getValue() != null;

    }

    /**
     * Validates if email is valid.
     */

    protected boolean validateEmail(TextField textField) {
        return textField.getText().matches("(?:[a-z0-9!#$%&'*+/=?^_`{|}~-
]+(?:\\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-
\\x7f])*\")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-
\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)\\])");

    }

    /**
     * Displays alert.
     *
     * @param title   title for alert
     * @param message alert's message
     */
```


```
    protected void alertdisplay(String title, String message) {

        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    /**
     * Displays success alert.
     *
     * @param title   tittle for mesage
     * @param message to show
     */

    protected void successdisplay(String title, String message) {

        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

Class Controller is responsible for validation of data all around the GUI, and more.

There is also a method in Chauffeur and Bus class that checks if its available.

```
package main.Model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;

/**
 * Class which represents a list of chauffeurs.
 *
 * @author IT-1Y-A16 Group 1
 */

public class ChauffeurList implements Serializable {

    private ArrayList<Chauffeur> chauffeurs;
```


```
/**
 * Constructs a list of chauffeurs.
 */

public ChauffeurList() {
    chauffeurs = new ArrayList<>();
}

/**
 * @return size of chauffeur list
 */

public int getSize() {
    return chauffeurs.size();
}

/**
 * Adds a given chauffeur to the list.
 *
 * @param chauffeur chauffeur to add
 */

public void add(Chauffeur chauffeur) {
    chauffeurs.add(chauffeur);
}

/**
 * Removes given chauffeur from the list.
 *
 * @param chauffeur chauffeur to remove
 */

public void removeChauffeur(Chauffeur chauffeur) {
    for (int i = 0; i < chauffeurs.size(); i++) {
        if (chauffeurs.get(i).equals(chauffeur))
            chauffeurs.remove(chauffeur);
    }
}

/**
 * Finds chauffeur at the given index.
 *
 * @param index index to look at
 * @return chauffeur at given index
```


```
 */

public Chauffeur getChauffeurByIndex(int index) {
    return chauffeurs.get(index);
}

/**
 * @return arraylist of all chauffeurs in the list
 */

public ArrayList<Chauffeur> getArrayChauffeur() {
    return chauffeurs;
}

/**
 * Finds chauffeur with the given name.
 *
 * @param name name to look by
 * @return chauuffeur with given name
 */

public Chauffeur getByName(String name) {
    for (Chauffeur chauffeur : chauffeurs) {
        if (chauffeur.getName().equals(name))
            return chauffeur;
    }
    return null;
}

/**
 * Checks if list contains given chauffeur.
 *
 * @param chauffeur chauffeur to check
 * @return true if list contains given chauffeur
 */

private boolean contains(Chauffeur chauffeur) {
    return chauffeurs.contains(chauffeur);
}

/**
 * Finds all chauffeurs with given preferred distance.
 *
 * @param prefferedDistance distance to look by
 * @return ChauffeurList of all chauffeurs with given preferred distance
```


```
     */

    public ChauffeurList getAllByPrefferedDistance(int prefferedDistance) {
        ChauffeurList result = new ChauffeurList();
        for (Chauffeur chauffeur : chauffeurs) {
            if (!chauffeur.isVikar()) {
                if ((chauffeur.getPreferredDistance() != null) &&
!(chauffeur.getPreferredDistance().isEmpty())) {
                    for (int j = 0; j < chauffeur.getPreferredDistance().size();
j++) {
                        if (chauffeur.getPreferredDistance().get(j) >
prefferedDistance) {
                            if (!result.contains(chauffeur))
                                result.add(chauffeur);
                        }
                    }
                } else if (chauffeur.getPreferredDistance().isEmpty()) {
                    result.add(chauffeur);
                }
            }
        }
        return result;
    }

    /**
     * Finds all chauffeurs with given preferred bustype.
     *
     * @param busType bustype to look by
     * @return ChauffeurList of all chauffeurs with given preferred bustype
     */

    public ChauffeurList getAllByPrefferedBus(String busType) {
        ChauffeurList result = new ChauffeurList();
        for (Chauffeur chauffeur : chauffeurs) {
            ArrayList<String> prefferedBuses = chauffeur.getArrayBusTypes();
            if (!chauffeur.isVikar()) {
                if (!prefferedBuses.isEmpty()) {
                    for (String prefferedBuse : prefferedBuses) {
                        if (busType.equals(prefferedBuse)) {
                            if (!result.contains(chauffeur))
                                result.add(chauffeur);
                        }
                    }
                } else
                    result.add(chauffeur);
```


```
        }
    }

    return result;
}

/**
 * @return ChauffeurList of all with vicar contract
 */

public ChauffeurList getAllVicars() {
    ChauffeurList result = new ChauffeurList();
    for (Chauffeur chauffeur : chauffeurs) {
        if (chauffeur.isVikar()) {
            result.add(chauffeur);
        }
    }
    return result;
}

/**
 * Removes all vicars in the list.
 */

public void removeAllVicars() {
    for (int i = 0; i < chauffeurs.size(); i++) {
        if (chauffeurs.get(i).isVikar()) {
            chauffeurs.remove(chauffeurs.get(i));
        }
    }
}

/**
 * @return List converted to String
 */

public String toString() {
    String s = "";
    for (int i = 0; i < chauffeurs.size(); i++) {
        s += chauffeurs.get(i);
        if (i < chauffeurs.size() - 1)
            s += "\n";
    }
    return s;
}
```


```
    /**
     * Finds all chauffeurs who are available in the given date interval.
     *
     * @param from start of date interval
     * @param to   end of date interval
     * @return ChauffeurList of all available in the given date interval
     */

    public ChauffeurList getAvailable(Date from, Date to) {
        ArrayList<Chauffeur> inTrips = new ArrayList<>();
        ChauffeurList result = new ChauffeurList();
        TripList trips = DataHandler.getTrips();
        for (int i = 0; i < trips.getArrayTrip().size(); i++) {
            for (Chauffeur chauffeur : chauffeurs) {
                if (chauffeur.equals(trips.getArrayTrip().get(i).getChauffeur())) {
                    if (!inTrips.contains(chauffeur))
                        inTrips.add(chauffeur);
                    if
(((from.before(trips.getArrayTrip().get(i).getDateObjStart())) &&
(to.before(trips.getArrayTrip().get(i).getDateObjStart())))
                            ||
((from.after(trips.getArrayTrip().get(i).getDateObjEnd())) &&
(to.after(trips.getArrayTrip().get(i).getDateObjEnd())))) {
                        result.add(chauffeur);
                    }
                }
            }
        }
        for (Chauffeur chauffeur : chauffeurs) {
            if (!inTrips.contains(chauffeur))
                result.add(chauffeur);
        }
        return result;
    }

    /**
     * Copies this list.
     *
     * @return Copy of this ChauffeurList
     */

    public ChauffeurList copy() {
        ChauffeurList chauffeurList = new ChauffeurList();
        for (Chauffeur chauffeur : chauffeurs) {
```


```
            chauffeurList.add(chauffeur);
        }
        return chauffeurList;
    }

}
```

After inserting the dates into the GUI, the method getAvailable is called, which will find all the chauffeurs that are free in the given date interval. After inserting the distance, the method getAllByPrefferedDistance returns all chauffeurs who prefer the given distance or have no preferences. After choosing the bus type, method getAllByPrefferedBus will find chauffeurs who prefer the specified bus or have no preferences. Also only the buses of the given type are shown. This is done by method loadBusList in TripController which also calls getAvailable method from BusList class. Adding stops to trip is handled by method handleStops.

5 Testing

In GUI testing, it was ensured the proper functionality of the graphical user interface for all applications, making sure it conforms to its written specifications, that all interactions, navigation, links work as required. During the system development, each method and functionality were checked for different bugs errors. The final version of the system was tested at the end of development and all the test actions were performed according to user guide.

Main page

1. User can select one of the buttons: Create tour, Make reservation, Find reservation/tour.	YES
2. The overview list view, displays the list of tours: standard and private sorted by date.	YES

1. Create tour

The test was performed by going through algorithm from user guide. Short description of actions performed:

1. User enters or selects a start date from the date picker (current or future dates)	YES
2. User enters the time of departure (format example 10:00)	YES
3. User enters or selects an end date from the date picker (future dates)	YES
4. User enters the time of arrival (format example 10:00)	YES
5. User enters departure/destination place in combo box or pressing the arrow should be able to select an item from the list. The list is scrollable and can be typed in any time.	YES
6. User enters the number of kilometers	YES
7. User selects a bus from the list pressing the arrow. By default, classic bus is selected.	YES
8. User can select any combination of extra services from checkboxes. Clicking the mouse on the box it should set or unset the checkbox.	YES
9. User enters the price for the tour.	YES

10. User can enter a stop name in combo box or pressing the arrow should be able to select an item from the list. The list is scrollable and can be typed in any time.	YES
11. User enters the time in minutes for stop.	YES
12. User selects the button Add stop to add the stop in the list.	YES
13. The previous stop automatically appears in the list of stops.	YES
14. User selects a single stop from the list.	YES
15. User can remove any stop from the list.	YES
16. User can select a chauffeur from the list of chauffeurs. The chauffeur list is sorted after user enters the date, time, and after the bus type is selected by calling the methods getAllByPrefferedBus, getAvailable from the class ChauffeurList.	YES
17. User can select a bus from the list of buses. The bus list is sorted after user selects the bus type by calling the method getAvailable from the class BusList.	YES
18. If user selects the check box private tour, by clicking the mouse on the box, the next step must be followed.	YES
19. User enters the customer's dates in the text fields: name, company name, address, email (only email format accepted, otherwise alert window will be displayed), phone number (only 8 digit number, otherwise alert window will be displayed)	YES
20. User selects a single customer from the list of customers	YES
21. User clicks the Add customer button	YES
22. User click the button Choose	YES
23. User click the button Create, the user is redirected to the main page, were he can see the previous created tour displayed in the list of tours.	YES

2. Make reservation

1. User can pick either one, or all the fields to find the best tour: destination, departure, number of passengers	YES
2. The list of tours is updated automatically according to the searches made by the user.	YES

3. User can take notes in the text area, by simply clicking on the text field and typing in.	YES
4. User can select a single tour from the list.	YES
5. User clicks the button choose tour and another window will appear on the screen for the second step in making reservation. Also the note window will be displayed in a separate window, with the same data.	YES
6. User enters the customer's dates in the text fields: name, company name, address, email (only email format accepted, otherwise alert window will be displayed at the end of the process), phone number (only 8-digit number, otherwise alert window will be displayed). All the fields are mandatory, only company name can be used when it is required.	YES
7. User can press the button Add customer and the customer will be saved and displayed in the list of customers.	YES
8. User can select a single customer from the list of customers.	YES
9. User enters the passenger's dates in the text fields: name, address, email (only email format accepted, otherwise alert window will be displayed at the end of the process), birthday. All the fields are mandatory, exception: address and email.	YES
10. User can press the button Add Passenger and the passenger will be saved and displayed in the list of passengers.	YES
11. User can remove any passengers from the list of passengers.	YES
12. User can select the passengers from the list of passengers.	YES
13. System displays the right price per person, token from the list of tours details, and displayed in the field Default price per person.	YES
14. User can enter the dates for extra services, discount.	YES
15. System calculates the total price and automatically displays on the total field.	YES
16. User can any time cancel the reservation by pressing the button Cancel, he will be redirected to the home page.	YES
17. User can select Save reservation and the system will store the reservation dates moreover will be displayed automatically in Search – list of reservation.	YES

3. Search
   3.1 Search Tour

1. The system displays the list of tours sorted by date.	YES
2. User can pick either one, or all the fields to find the tour using the following fields: destination, departure, date, or by clicking one of the check boxes: All, standard, private.	YES
3. The list of tours is updated automatically according to the searches made by the user.	YES
4. User can select any tour from the list	YES
5. User can click on remove tour button and the list will be updated without selected tour.	YES
6. If user selects the button edit tour, all the steps from Create tour must be repeated. The system will display the window with the fields filled in with all the dates from the selected tour.	YES

3.2 Search reservation

1. The system displays the list of reservations.	YES
2. User can pick either one, or all the fields to find the reservation using the following fields: name, company name, address, email, phone number.	YES
3. The list of reservations is updated automatically according to the searches made by the user.	YES
4. User can select any reservation from the list.	YES
5. User can click on remove reservation button and the list will be updated without selected one.	YES
6. If user selects the button edit reservation, all the steps from Make reservation must be repeated. The system will display the window with the fields filled in with all the dates from the selected reservation.	YES

4. Bus list

1. System displays all the busses.	YES
2. User can click remove Bus and the list will be updated.	YES

3. User can press the button Add Bus and will follow the steps from Add bus.	YES

4.1 Add Bus

1. User selects a bus from the list pressing the arrow. By default, classic bus is selected.	YES
2. User enters the registration plate, which must be 2 capital letters followed by 5-digit number (otherwise an alert window will be displayed at the end of the process). All the fields are mandatory.	YES
3. User enters the number of seats	YES
4. User clicks the button Add bus and the dates will be stored in the list of buses.	YES

5. Chauffeur list

1. System displays all chauffeurs.	YES
2. User can click delete chauffeur and the list will be updated.	YES
3. User can press the button Add chauffeur and will follow the steps from Add chauffeur.	YES

5.1 Add chauffeur

1. User enters the name, address, email (only email format accepted), phone number (only 8-digit number), date of birthday (only chauffeur with the age > 18 accepted), employee ID (only 5-digit number). All the fields are mandatory (otherwise an alert window will be displayed at the end of the process).	YES
2. User can select any combination of check boxes: vicar, preferences for distance, and bus type.	YES
3. User clicks the button Add chauffeur and the dates will be stored in the list of chauffeurs.	YES

6 Results

The system was created in order to lighten the processes involved in a creating tour, making reservation and hiring chauffeurs. When the employee creates a new tour, bus and chauffeur are reserved for a specific time period. When the employee wants to make a reservation for a customer he can look into the database to see if the customer is already in database.

The main purpose of the system is to a create tour, list all available chauffeurs and buses, offer private tour for the customer, to store information like customer, chauffeur and bus data in a database and keep track of reservations. The system is created in easy way for comfortable using.

7 Conclusion

The main goal of the project was to create a system for VIA Bus company that complies all customers and manager requirements. With VIA Bus company, it can be rented a bus and chauffeur for tours to destination chosen by customer, or travel by bus to one of the predefined locations. The system was developed for easy and clear use by different actors for multiple actions. Every Use Case description and Activity Diagram were checked and compared to the system. This system is comprehensive, fast and simple to use by all with no specific computer knowledge.

During the project development, some problems were encountered. We had mainly problems with Java code and different types of errors in the system. We always had to decide how it should work and agree on it together.

As a conclusion, the project is successfully completed and the system complies all the requirements. It was created a powerful software that is easy to work with. The final version of the system is designed and created in unique way for VIA Bus company.

8 References

8.1 Books

Gaddis, T. and Deep Dhiman, V., 2015. *Starting out with Java: early objects.* 5. ed., gl ed. Harlow, Essex: Pearson.

8.2 Additional resources

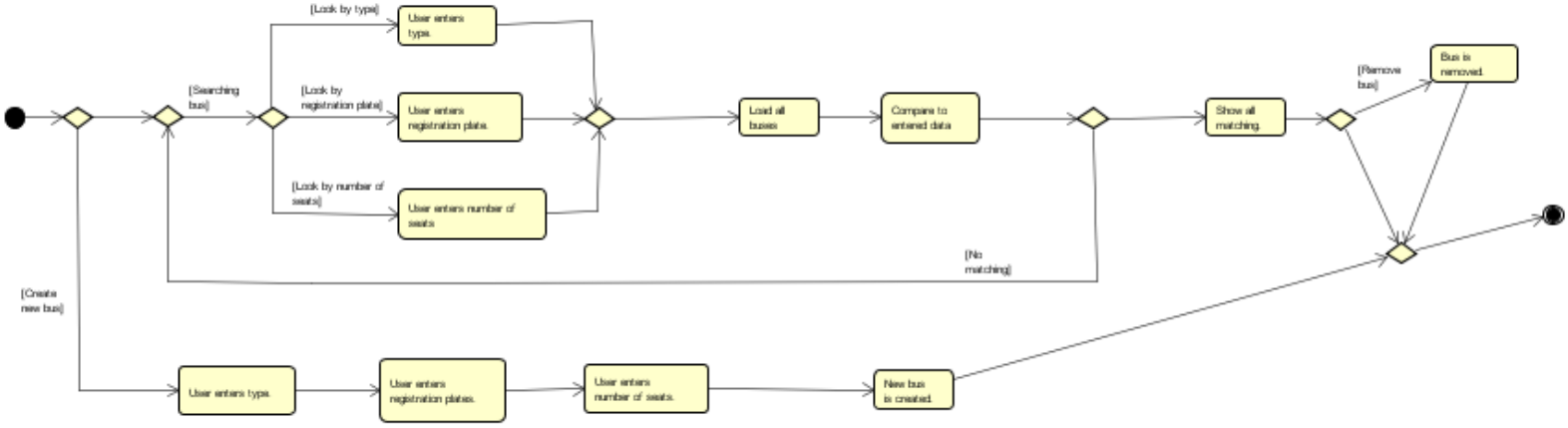
Presentations from classes

JAVA 1 and UML (SDJ1)

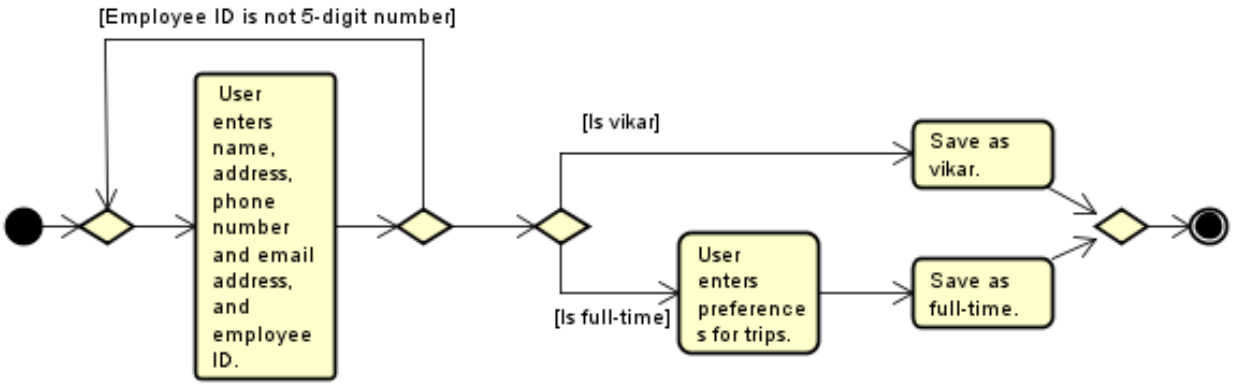
9 Appendices

9.1 Appendix 1 – Activity diagram

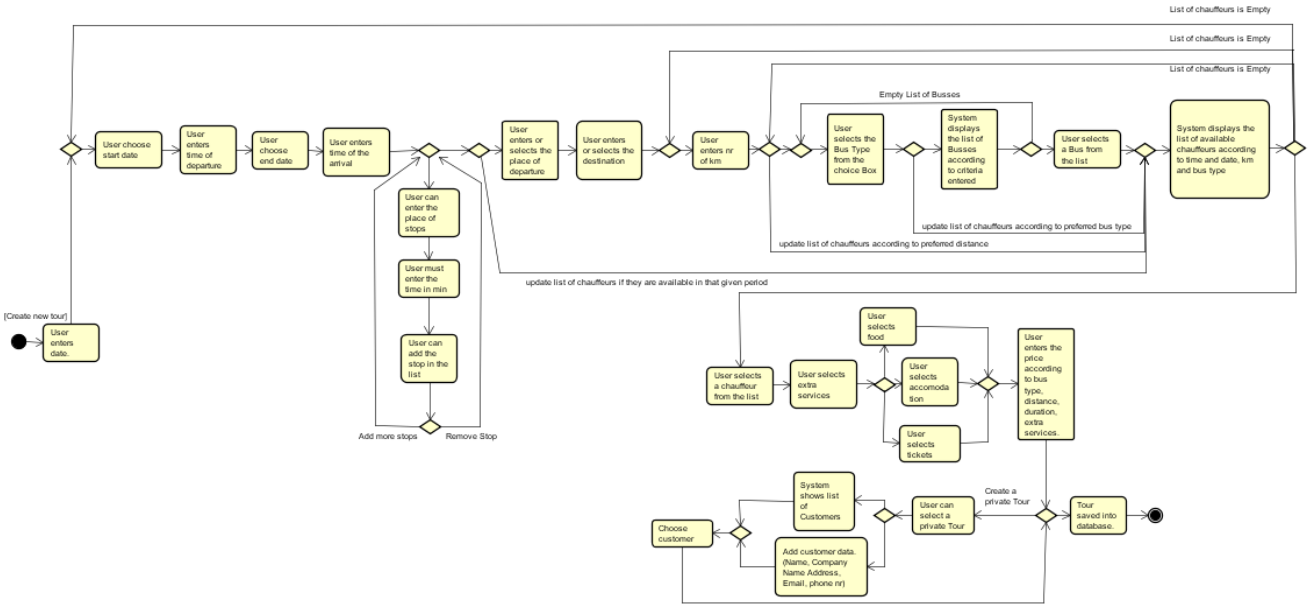
Manage buses:



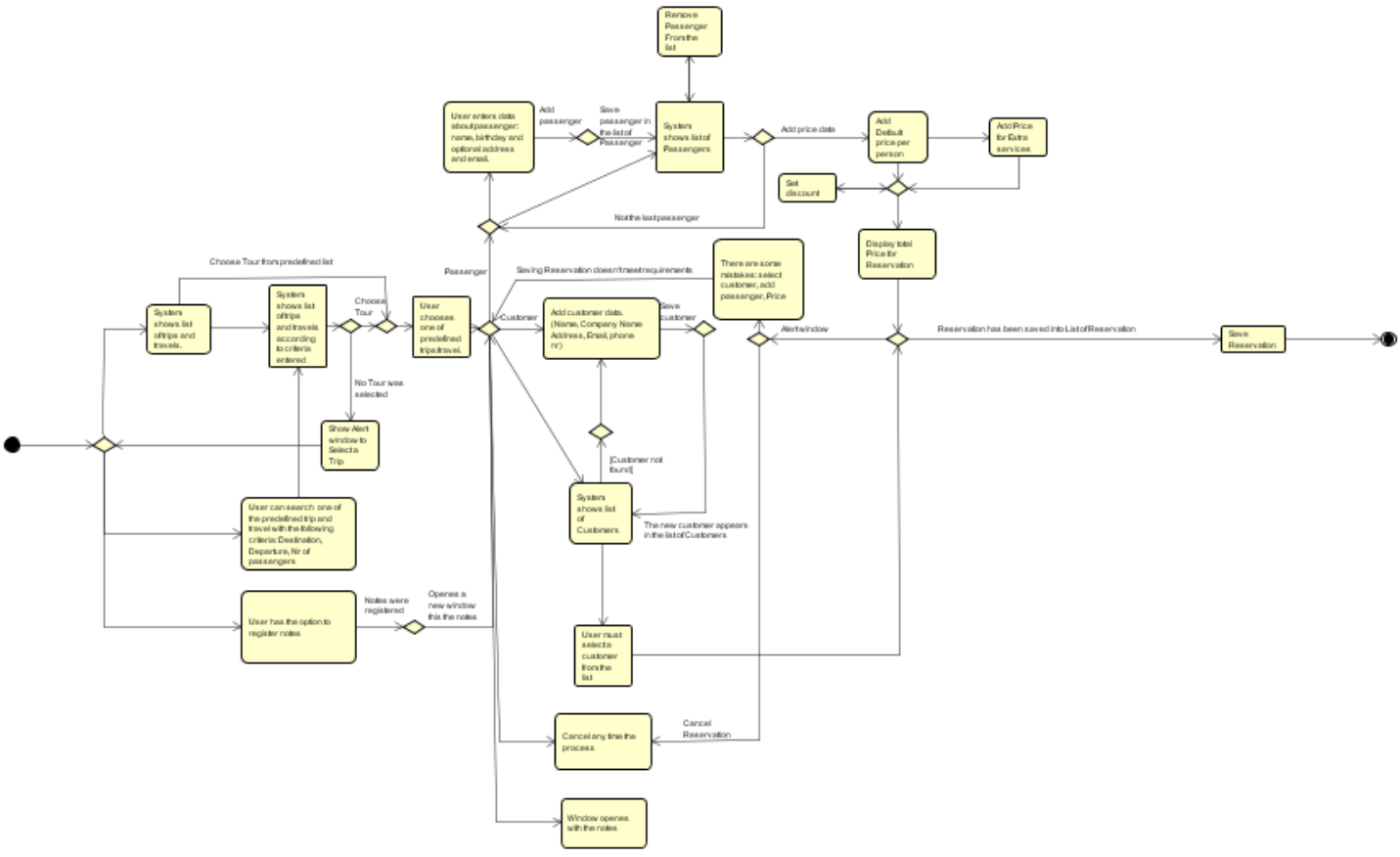
Register a new chauffeur:



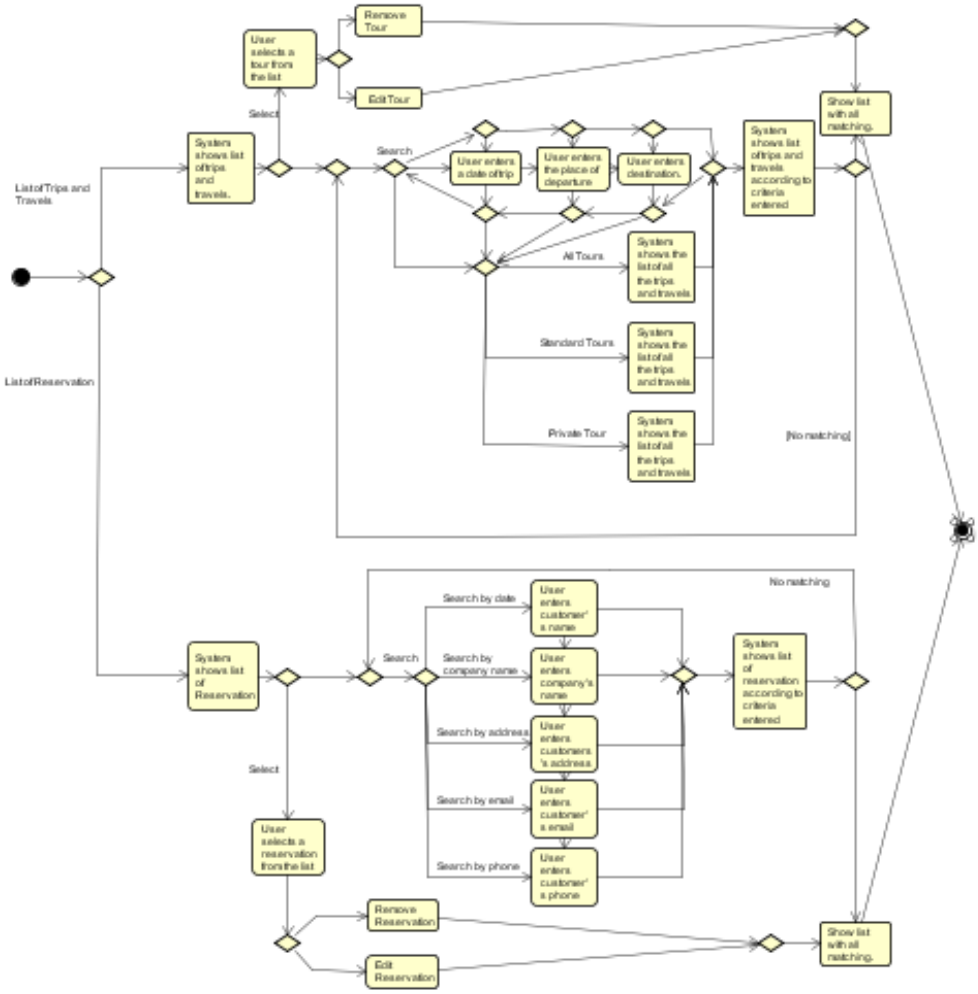
Create a tour:



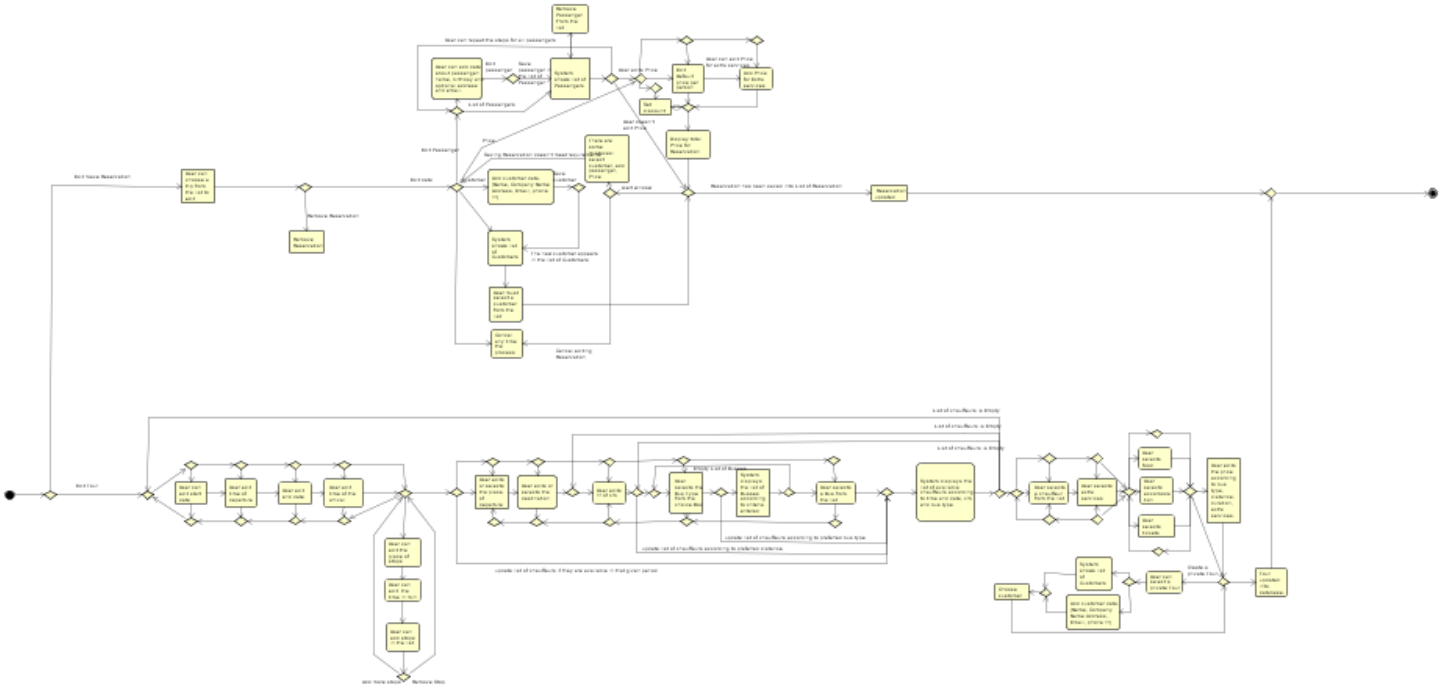
Make reservation:



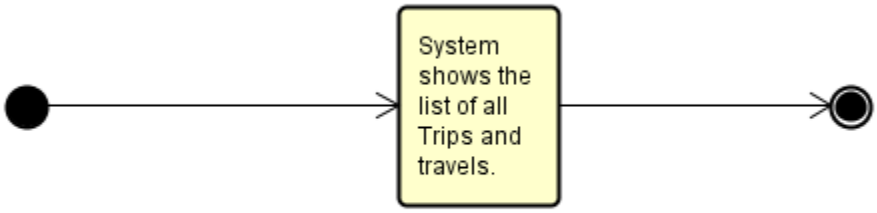
Search Reservation/Tour:



Edit Reservation/Tour:



Tours Overview:



9.2 Appendix 2 – Use Case Description

Manage buses:

ITEM	VALUE
UseCase	Manage buses
Summary	Information about new bus has been stored in the system.
Actor	Employee
Precondition	none
Postcondition	New bus has been managed.
Base Sequence	1. System shows list of all buses. 2. User will select one of more buses. 3. User can choose wether to add or remove bus. If add was chosen: 4. User will enter type of bus, registration plate and number of seats 5. New bus was created and added to list of busses.
Branch Sequence	If remove was chosen: 4. Selected bus or buses was removed from list of buses and list was saved.
Exception Sequence	Wrong data input: Base sequence 1. Invalid data type, use case ends.
Sub UseCase	
Note	

Register a new chauffeur:

ITEM	VALUE
UseCase	Register a new chauffeur
Summary	Information about new employee has been stored in the system.
Actor	
Precondition	Employee ID must be 5-digit number.
Postcondition	New chauffeur has been employed.
Base Sequence	1. User enters name, address, phone number and email address, and employee ID. Condition: Employee ID must be 5-digit number. If ID is incorrect, go to step 1. 2. User will choose employees type of contract. Full-time or vikar. If it is full time employee, user will specify preferences for trips. 3. System saves all data and add employee to coresponding list of employees.
Branch Sequence	
Exception Sequence	Wrong data input: Base sequence 1. Invalid data type, use case ends.
Sub UseCase	
Note	Vikar contract can be canceled at any time.

Create a tour:

ITEM	VALUE
UseCase	Create tour
Summary	System creates a new trips or travel
Actor	Employee
Precondition	none
Postcondition	New tour has been saved into database.
Base Sequence	1.User must choose the start date of the Tour. 2.User enters time of departure. 3.User choose the date of the arrival (end date). 4.User enters time of the arrival. 5.If User wants to enter name of the stop place, use case continues with branch sequence 1 . 6.User enters or selects the place of departure. 7.User enters or selects the destination. 8.User enters the number of kilometers. 9.User selects the Bus Type. 10.System displays the list of Busses according to criteria entered. 11.User selects a Bus from the list 12.System displays the list of available chauffeurs according to time and date, number of km and bus type. 13.User selects a chauffeur from the list of Chauffeur displayed on the screen. 14.User selects one or more of the extra services: food, accommodation, tickets. 15.User enters the price according to bus type, distance, duration and extra services. 16.If User wants a private Trip, use case continues with branch sequence 7. 17.User can save the Tour into database.
Branch Sequence	1.User enters the place of stops. 2.User must enter time of the stop (in min). 3.User can add the stop in the list. 4.User can repeat the step 1, if more stops needs to be added. 5.User can remove any stop. User can continue with base sequence 6. 6.User can add a customer by adding: name, company name (optional), address, email, phone number. 7.User can choose one customer from the list of customers Use case continues with base sequence 17.
Exception Sequence	No available chauffeur after entering time and date. 1-4 base sequence Use case ends. No available chauffeur after entering number of kilometers. 8 base sequence Use case ends. No available chauffeur after selecting a bus type. 11 base sequence Use case ends. No available bus type after selecting the bus type from the list. 11 base sequence Use case ends. No selected Bus from the list of busses: 9 base sequence. Alert Window to select a Bus. No chauffeur selected from the list of chauffeur: 13 base sequence. Alert Window to select a Chauffeur. Wrong data input in the time fields. 2, 4 base sequence. Alert Window for time. Wrong data input in the price fields. 15 base sequence. Alert Window for price.
Sub UseCase	Register a new chauffeur
Note	Creating new tour can be canceled without saving at any time.

Make reservation:

ITEM	VALUE
UseCase	Make reservation
Summary	User enters all data needed and makes reservation
Actor	Employee
Precondition	List of Tours should not be empty
Postcondition	Reservation has been made and all data have been saved into list of Reservation.
Base Sequence	1.System displays the list of all trips and travels. 2.User can enter one of the following criteria to search in the List of Tours: destination, departure, number of passengers. 3.System selects and displays the list of trips and travels, according to the criteria entered. 4.User can enter in the Note block, information received from the customer (ex: nr. of passengers, preferences for extra services). 5.User must select the Tour from trips and travels displayed in the list. 6.User can select the customer from the list of customers. If customer is not in the list, branch sequence 1. 7.User must enter name, address, email and birthday of the passenger. 8.User must add passenger in the list of passengers. 9.User must repeat step 6 for each passenger. 10.User can remove any passenger from the list any time. 11.User must enter the default price per person. 12.User must enter price for extra services, if they appear in the note window. 13.User can enter a discount for frequent customers. 14.System will display the total price of reservation. 15.Reservation can be saved. 16.System will save reservation into list of reservation.
Branch Sequence	Base sequence 4: 1.User must enter name, company name(if the customer is a company), address, email, and phone of the customer(8 digit number). 2.User must add the customer in the list of customers. 3.System will continue from base sequence 6.
Exception Sequence	No available tour after searching into the list of tours: 2 base sequence User can create a private tour with bus and chauffeur. No selected tour from the list: 5 branch sequence. Alert Window will appear on the screen to select a Trip. Wrong data input in the phone field (required 8 digit number): Branch sequence 1. Invalid data type, alert window. No name, address, email or phone entered for the customer. Branch sequence 1. Alert window. No selected customer from the list: Base sequence 6. Alert window. No passenger added. Base sequence 8. Alert window No birthday or email entered for the passenger. Base sequence 7. Alert window. No default price entered. Base sequence 11.
Sub UseCase	Register new employee Create tour
Note	Reservation can be canceled any time.

Search Reservation/Tour:

ITEM	VALUE
UseCase	Search Reservation / Tour
Summary	System will find Reservation or a Trip according to data entered by user.
Actor	Employee
Precondition	List of Tours / Reservation should not be empty
Postcondition	Reservation / Tour has been found and can be edited.
Base Sequence	Reservation 1.System loads the reservations lists. 2.User can search a reservation whether by customer's name, company's name, customer's address, customer's email, and phone. 3.System will show list of all matching reservations on the screen. 4.User can select one of the reservations. 5.User can modify or cancel the reservation. 6.If any changes have been made, user will be asked to save the changes.
Branch Sequence	Tour 1.System loads the lists with trips and travels. 2.User can search a Tour whether by destination, date of the trip or place of departure. 3. User can select one of the following options: all, standard or private tours. 3.System will show list of all matching trips and travels on the screen. 4.User can select one of the tours. 5.User can modify or cancel the tour. 6.If any changes have been made, user will be asked to save the changes.
Exception Sequence	No matching reservation has been found: Base sequence 3. List is shown empty. Use case continues from base sequence 2. No matching trips has been found: Branch sequence 3. List is shown empty. Use case continues from base sequence 2. Wrong input for name, company name, address, email or phone entered for the customer. Base sequence 2. List shown is not matching the criterias. Use case continues from base sequence 2. Wrong input for destination, date or place of departure. Branch sequence 2. List shown is not matching the criterias. Use case continues from base sequence 2. No selection has been made. Base sequence 4. Branch sequence 4. Use case ends.
Sub UseCase	Edit Reservation / Tour
Note	Find Reservation/Find Tour can be canceled any time.

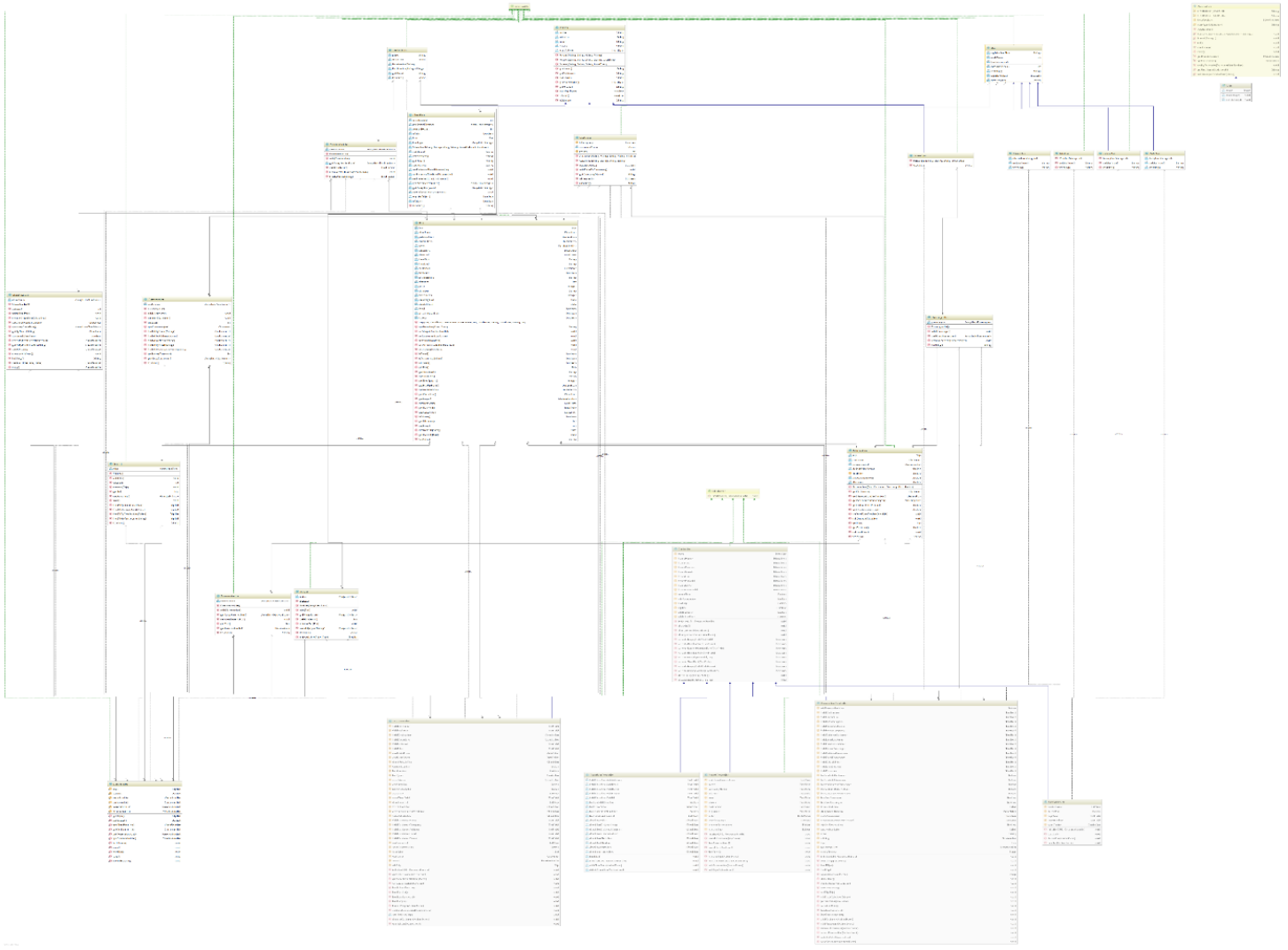
Edit Reservation/Tour:

ITEM	VALUE
UseCase	Edit Reservation / Tour
Summary	Reservation / Tour is edited.
Actor	
Precondition	Edit Reservation must have been opened through Search reservation option. Edit Tour must have been opened through Search Tour option.
Postcondition	Reservation has been modified or canceled. File has been updated.
Base Sequence	1. Reservation is loaded from the file. 2. User can edit the reservation by selecting a different customer or passenger from the lists or adding a new customer or passenger, or edi If user decide to cancel reservation, reservation is canceled. 3. User will be asked to save the changes. 4. All changes have been saved into the system.
Branch Sequence	Base sequence 4: If user does not want to save the changes. 1. User can choose to exit without change or to modify again. If modify again has been chosen, continue from base sequence 3. Tour 2. Tour is loaded from the file. 3. User can edit the tour by selecting a different chauffeur or bus from the lists or can edit a ny field from the screen. If user decide to cancel reservation, reservation is canceled. 4. User will be asked to save the changes. 5. All changes have been saved into the system.
Exception Sequence	Wrong input for phone entered for the customer(8 digit number). Base sequence 2. Alert window. Wrong input for Tour date or time . Branch sequence 3. Alert window. No selection has been made. Base sequence 2, Branch sequence 3. Use case ends.
Sub UseCase	
Note	Modifying can be canceled at any time. It is possible not to save the changes.

Tours Overview:

ITEM	VALUE
UseCase	Tours Overview
Summary	Displays the list of all trips and travels.
Actor	Employee
Precondition	none
Postcondition	none
Base Sequence	1.Show all tours.
Branch Sequence	
Exception Sequence	
Sub UseCase	
Note	

9.3 Appendix 3 – Class Diagram

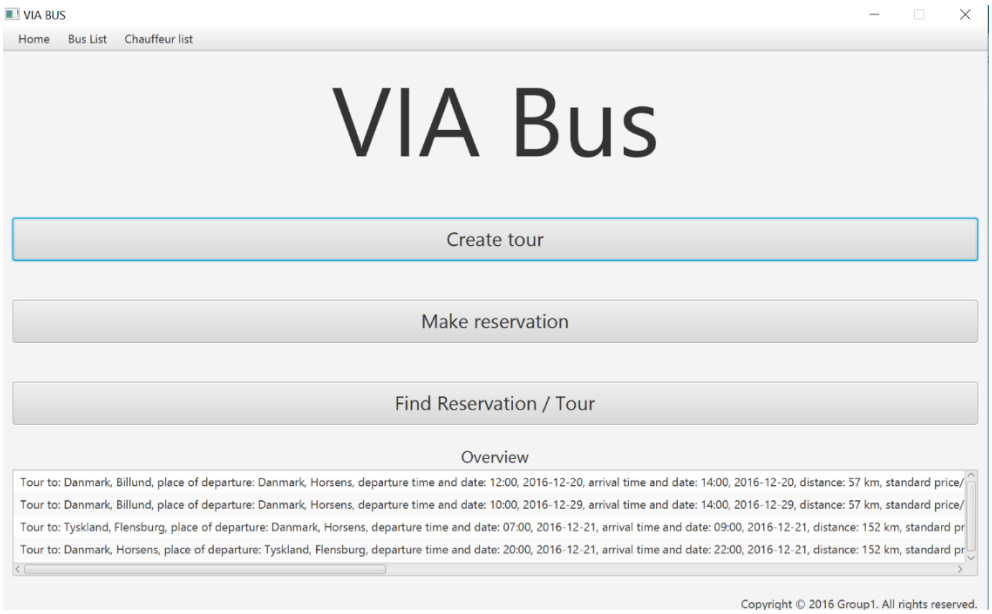


9.4 Appendix 4 – User Guide

Home

The home page is displayed when the program is opened. The Home page allows user to access Create tour, Make reservation, Find reservation / tour and displays the list of tours.

On the following pages it will be elaborated all the functions of the program.

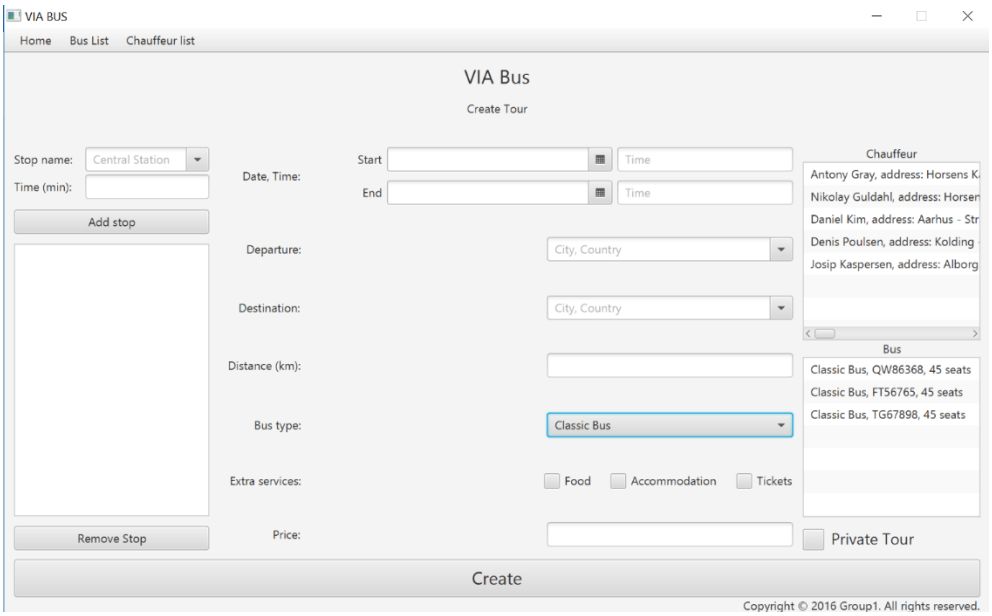


Create tour

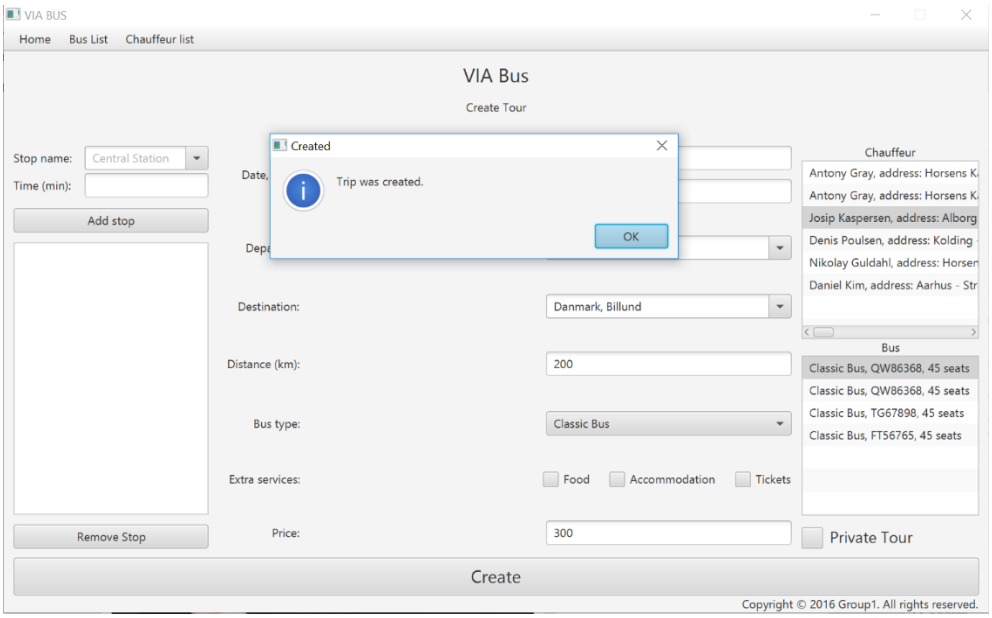
Create tour – standard

The user must fill in all the dates about start time(format example 10:00), date of the departure (can be picked up or filled in with the following format : month/day/year: 12/01/2016), end time and date of the arrival, the departure place, destination place, distance in km, the buss type can be choosen from the list, price. The chauffeur list is sorting after user enters the date, time, and after the bus type is selected. The bus list is sorting after user selects the bus type. In order to create a new Tour, the user must select a chauffeur and a bus from the lists. In case one of the fields is not filled in, an alert window will be displayed on the screen with the name of the field that should be filled in properly

The check boxes for extra services and adding stops fields are not mandatory.

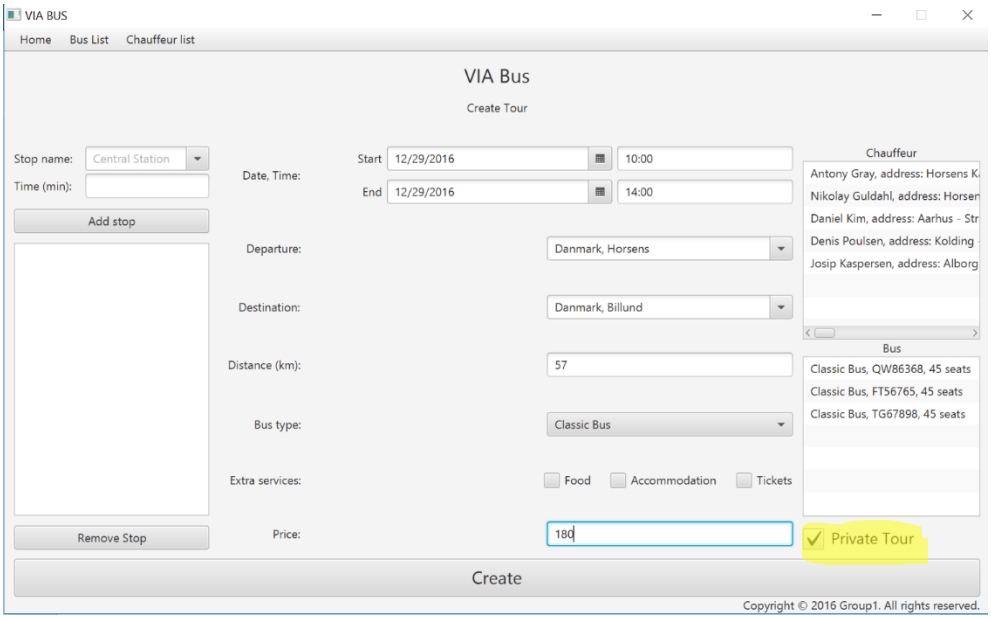


In order to finish creating a new tour, the user must press the Create button and an alert window will appear on the screen, like the one displayed bellow.



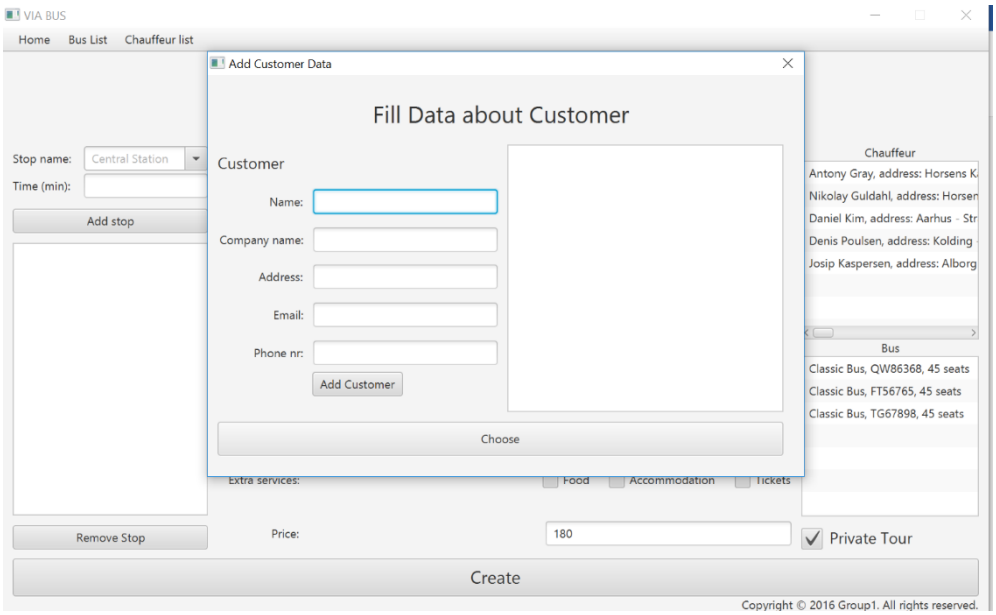
Create private tour

If the user wants to create a private tour, he must follow the steps from Create tour, and press at the end the button Private Tour.

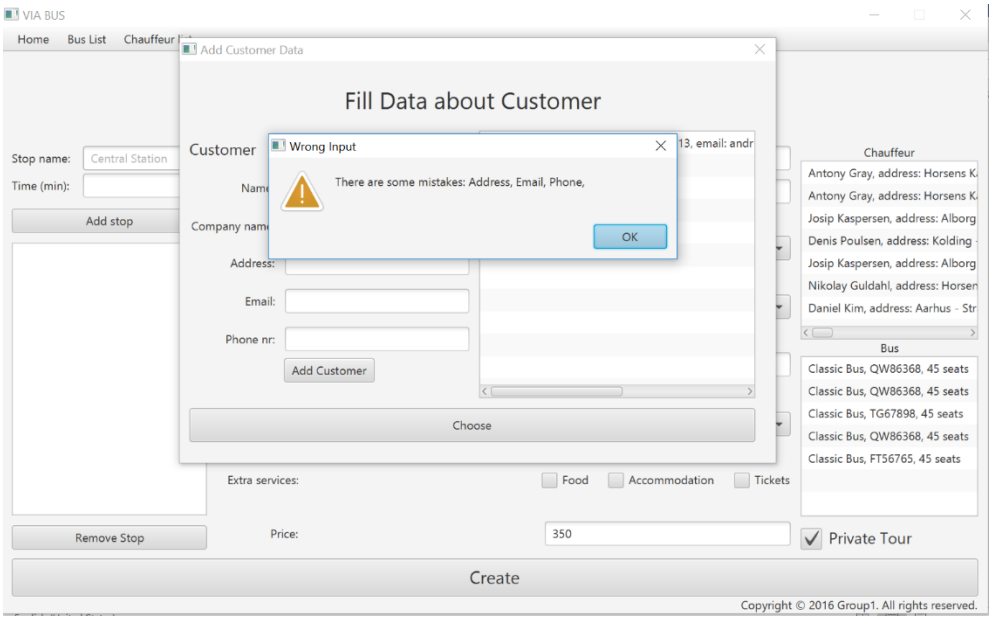


After this, another window will appear on the screen with: Fill data about customer.

This window will enable the user either to choose from the list of customers displayed on the right side of the window, or to fill in all the dates about customer: name, company name (only if the order is made by a company), address, email, and phone number



If the user doesn't fill all the mandatory fields about the customer, an alert window will appear, like the one below, indicating which field must be filled in properly.



Make reservation

Make reservation appears in the Home tab, on the third place of the list.

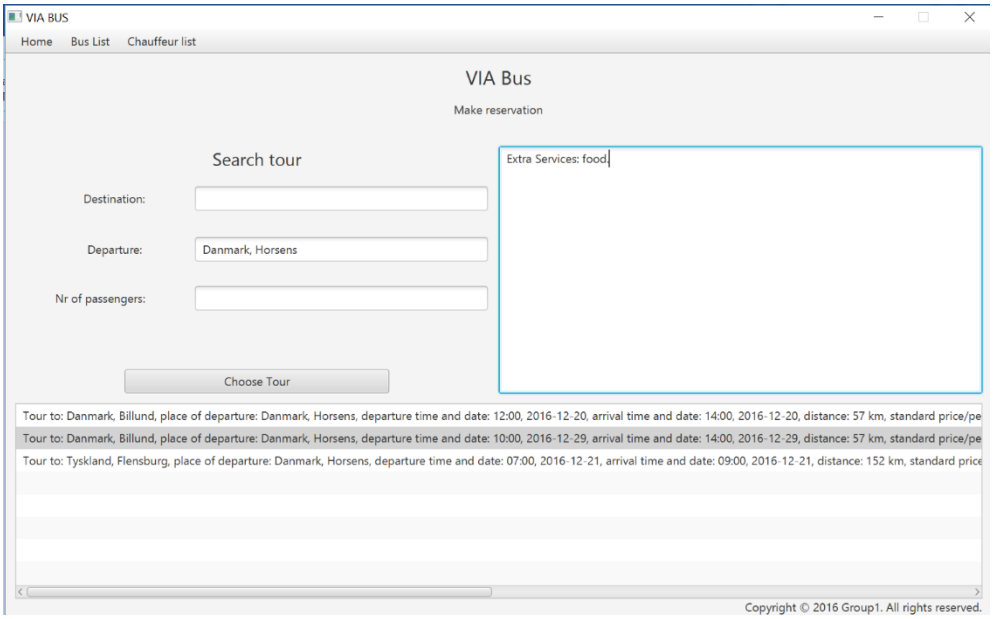
Make reservation - Search Tour

The screen Make reservation its split in 3 sections. First section has three fields used to search in the list of tours with the following parameters: destination, departure, number of passengers. The user can pick either one, or all the fields in order to find the best tour, according to customer requirements.

Second section, it enables the user to make notes in the right side of the screen, by simply clicking on the text field and typing in. The notes will automatically appear in the next step of making reservation, with the same data, only if the user typed into the note section. Notes are a convenient way of storing information which the user needs in the workspace.

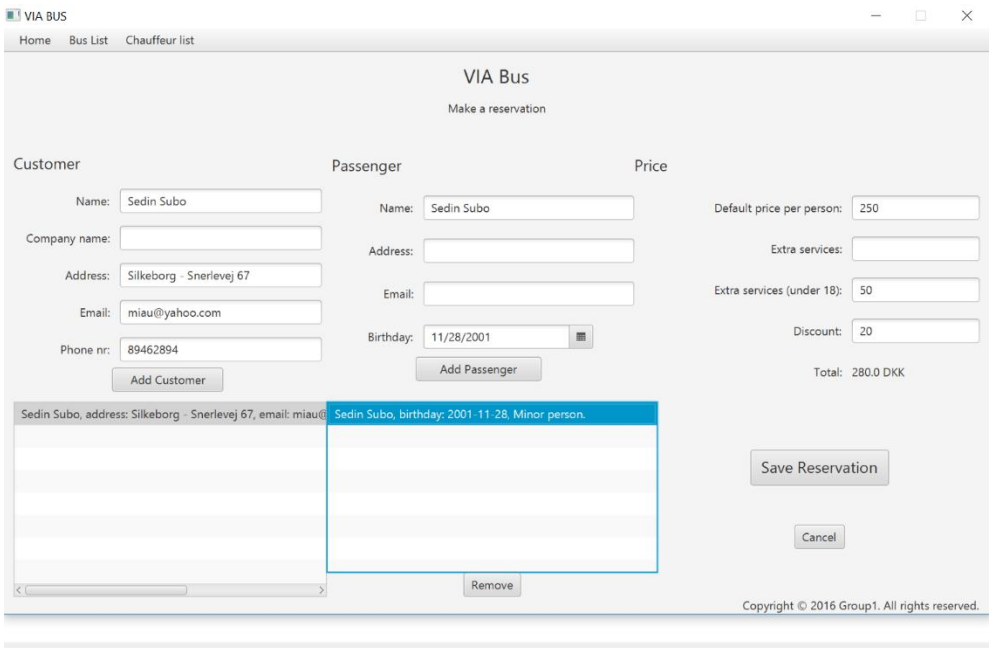
The third section of this screen is the list of the Tours displayed in the bottom of the screen. The list is updated automatically according to the searches made by the user.

Before the user presses the button Choose tour, he must select a tour from the list.



Make reservation

The user must select the customer, from the list of customers, wich is dispalyed under the Customer section, otherwise he must fill in all the dates about the customer, if the customer is at the first order, with the following dates: name, company name, address, email, phone number. All the fields about customer are mandatory, the field company name must be filled in, only if the order is made by a company.

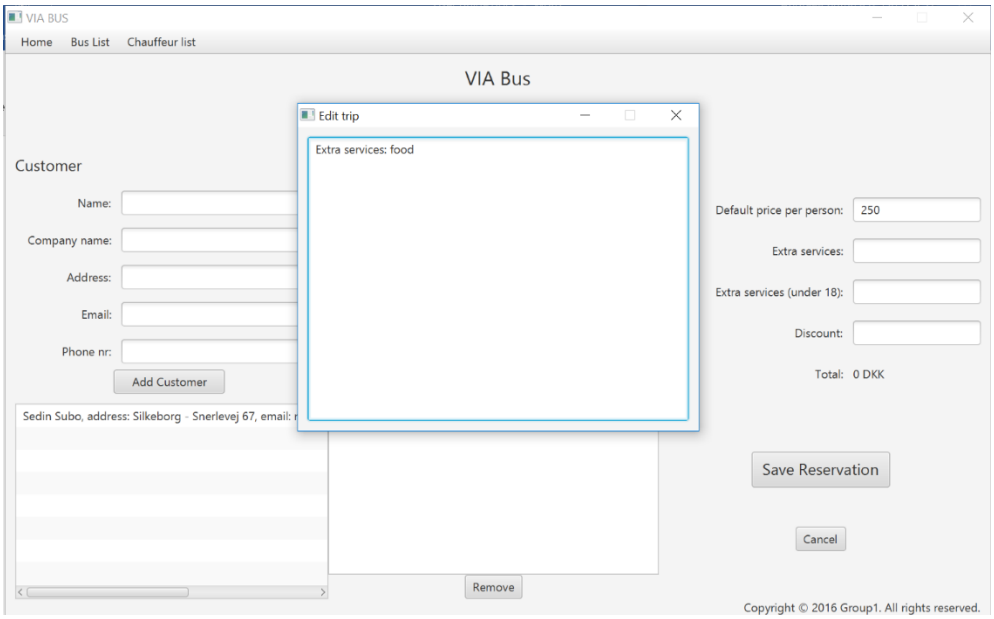


The next step in “Making a reservation” is to add the passengers, one by one, by pressing Add passenger. Each passenger will appear in the list of passengers displayed under the passenger section. The fields name and birthday are mandatory. The fields address and email are used only if the customer wants to give more details to receive the newsletter.

The next step is to fill in the price per person for extra services. Remark that the field default price per person is already filled in with the price took from the selected tour field.

The user will adjust the price for extra services according to the notes he took in step 1 of “Make a reservation”. This notes appear like in the picture displayed bellow. The user can also give a discount to the customer if necessary in the field called Discount.

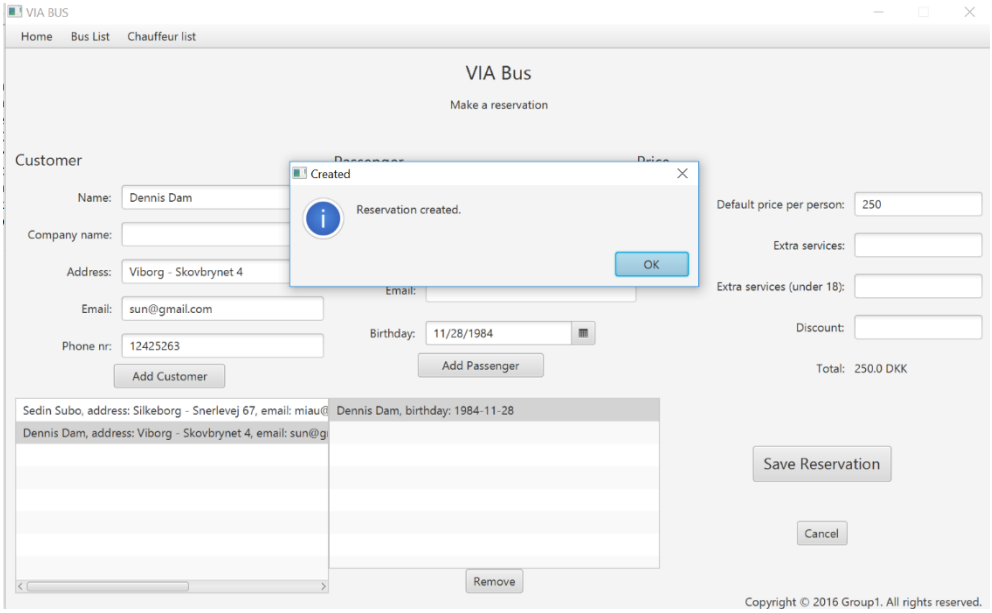
All the prices must be typed in per person and the total price of the reservation will appear in the Total section.



The user can any time press cancel reservation, and he will be directed to the home screen.

After all the steps enumerated above, the user can save the reservation, and an alert window will appear on the screen, which will confirm that the reservation was created.

The alert window will be exactly like the one displayed bellow.

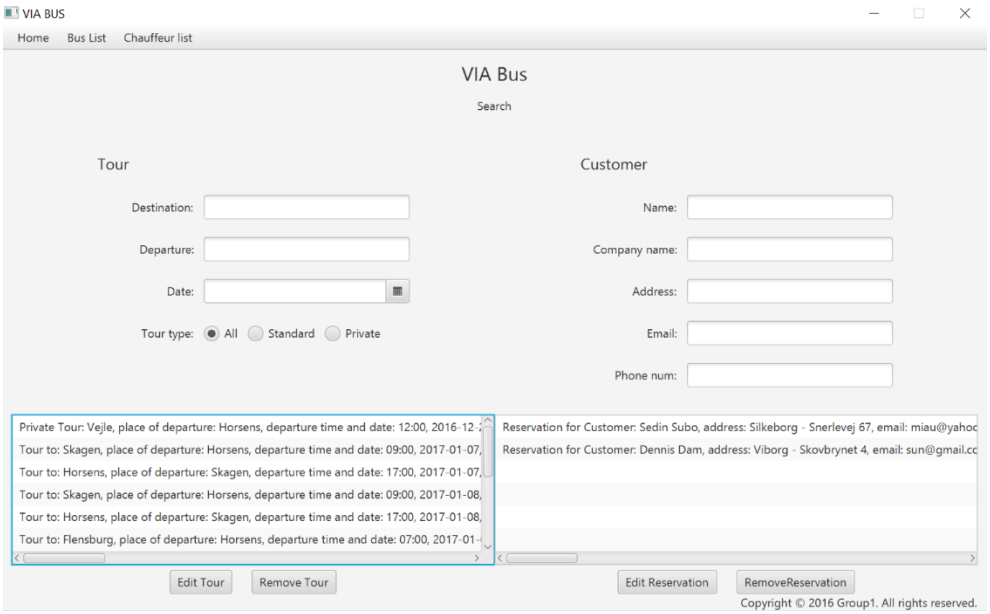


Find Reservation/Tour

Find Reservation/Tour appears in the Home tab, on the last place of the list. The screen it's split in 2 sections, the left half of the page, is for Tours and the right half of the page is for Reservations.

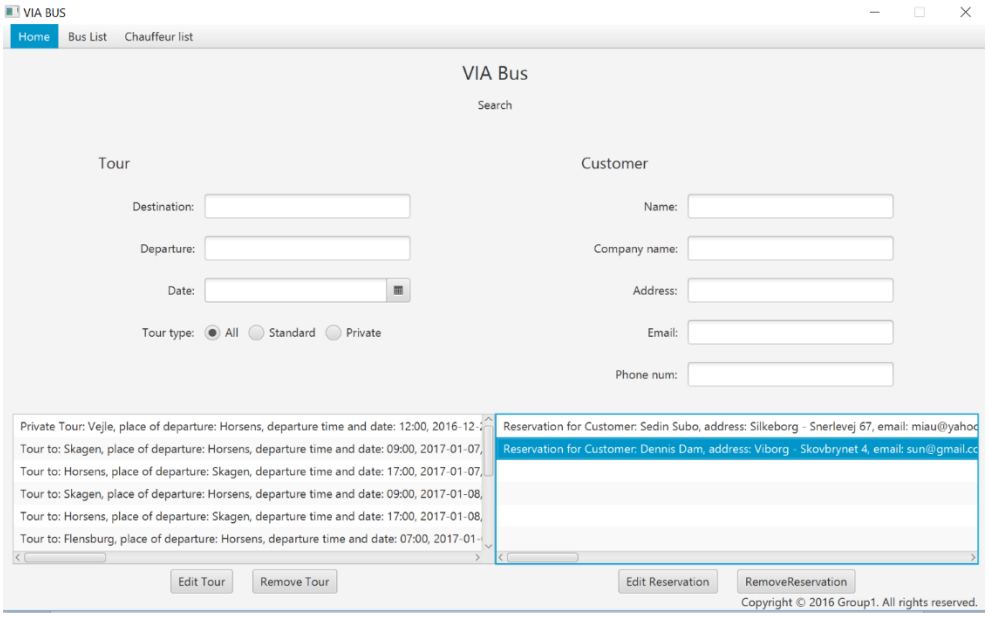
The tour section, has three fields used to search in the list of tours with the following parameters: destination, departure, and date. The user can use either one, or all the fields in order to find the tour, according to customer requirements. And, three radio buttons for all tours, standard or private tours. The list is updated automatically according to the fields used by the user.

In order to go on edit tour or remove tour, the user must select one from the list.



The Reservation section, has five fields used to search in the list of reservations with the following parameters about the customer that made the reservation: name, company name, address email, phone number. The user can use either one, or all the fields in order to find the right reservation. The list is updated automatically according to the fields used by the user.

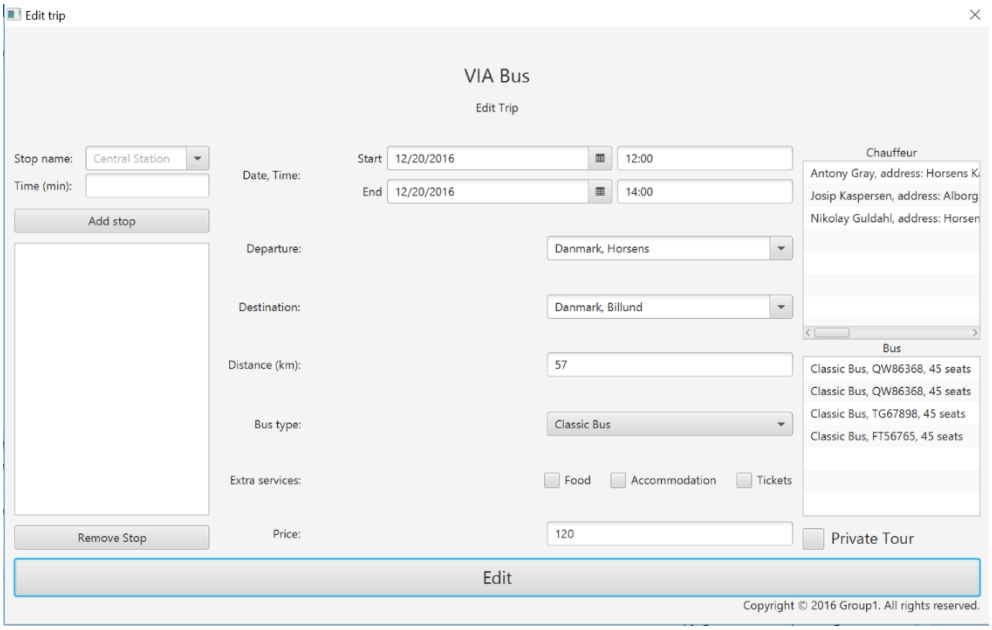
In order to go on edit reservation or remove reservation, the user must select one from the list.



Edit tour

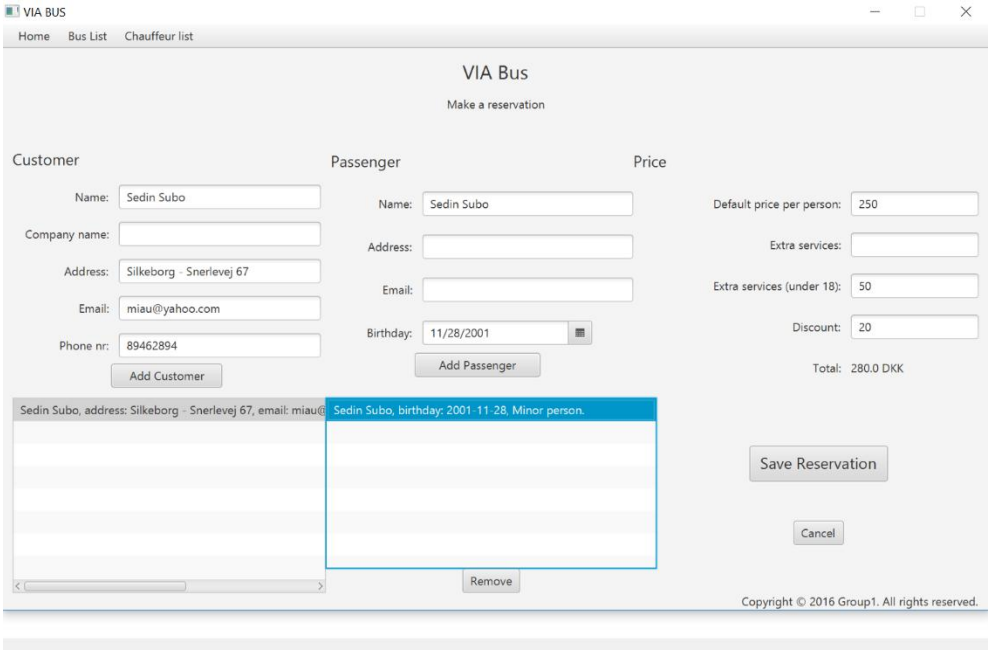
When the user selects edit tour, a new window will appear on the screen with the same interface like in create tour, but in this case, all the fields will be filled in with the dates from reservation selected in the previous step.

The user can edit any of the fields he wants to change, but in order to succeed, it is mandatory to select again the chauffeurs and the buss from the list.



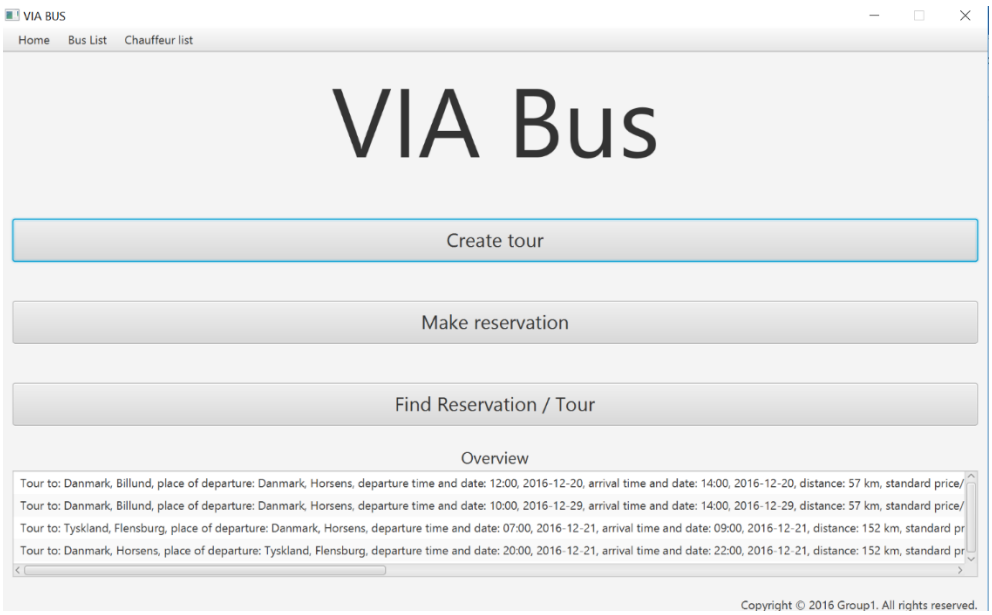
Edit reservation

When the user selects edit reservation, a new window will appear on the screen with the same interface like the one in Make Reservation. The user can add new customer, passengers, remove passengers or set new price or discount. To save the changes, it is mandatory to select again the customer from the list.



Overview

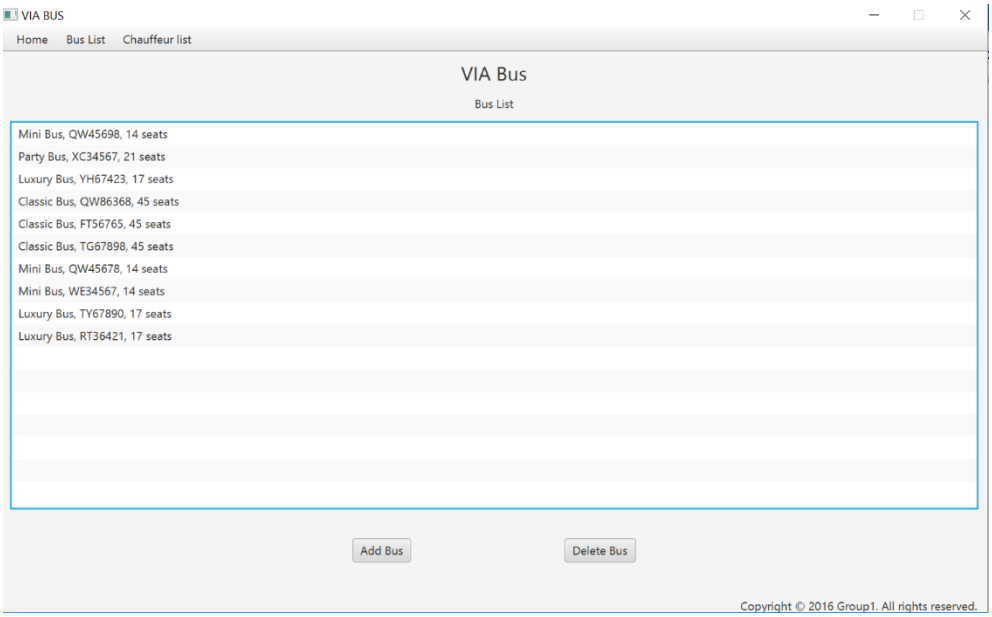
The main page shows in the bottom of the page the list of tours made by the VIA Bus company, sorted by current date.



Bus List

Bus List

The user must go on the second tab of the menu and select Bus List in order to see all busses, displayed in a list, like the one bellow.



Add buss

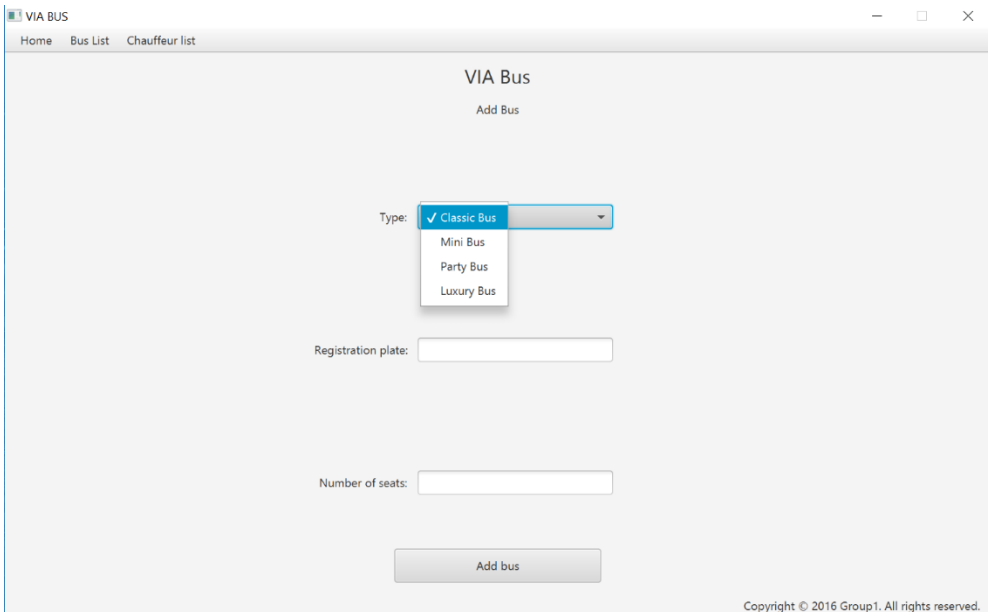
First step, the user must select one of the busses: classic bus, mini bus, party bus or luxury bus.

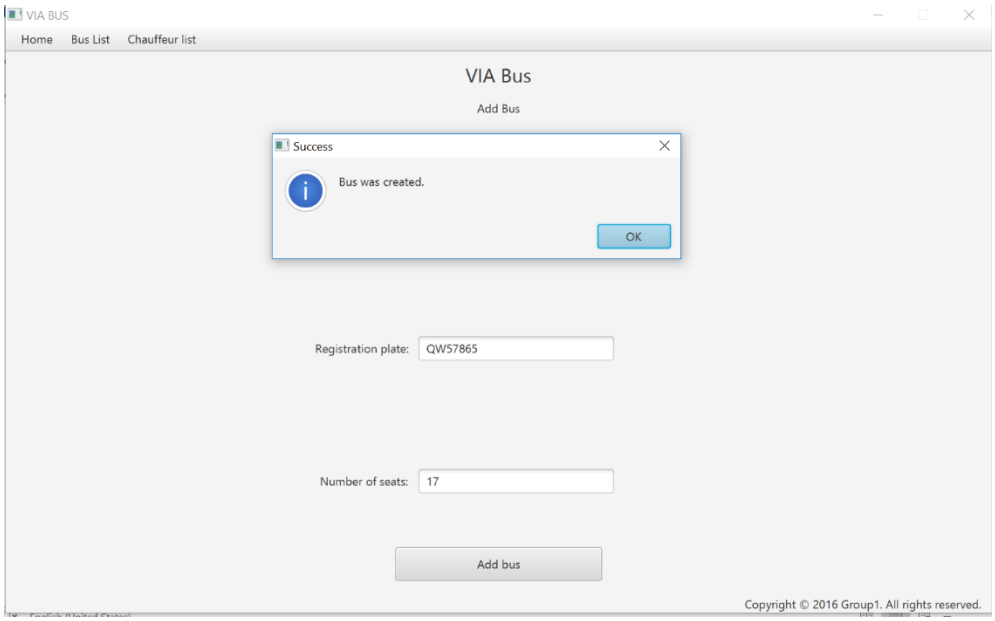
The second step is to add a registration plate number, which must respect to following format: 2 uppercase letters and 5 digit number (example: AS12345).

Third step: enter the number of seats, per buss.

Last step: press add button.

If one of the above steps it’s not respected, an alert window will appear on the screen, saying which field must be filled properly, otherwise a Succes window will be displayed, showing that the bus was created.

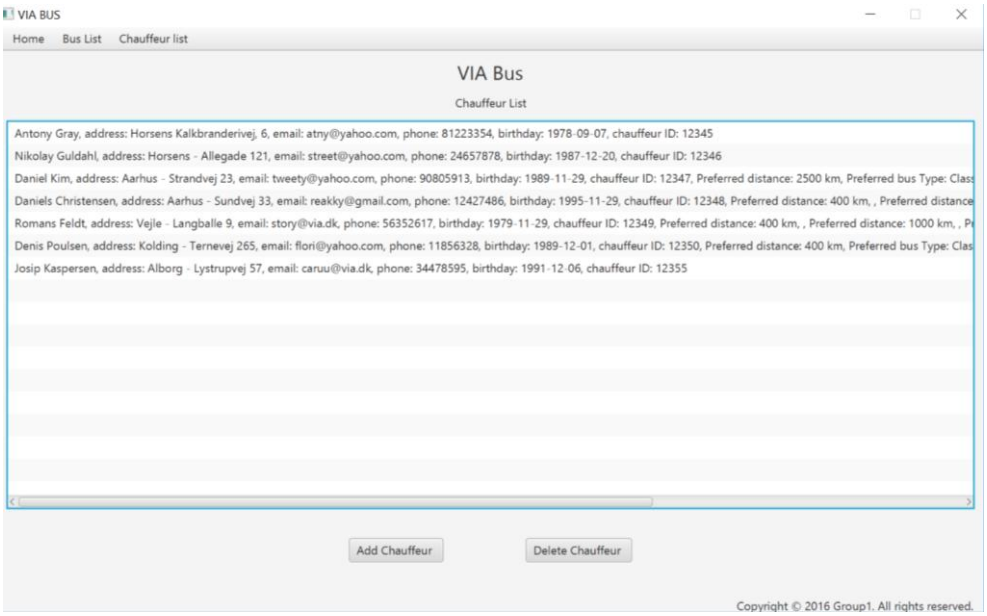




Chauffeur List

Chauffeur List

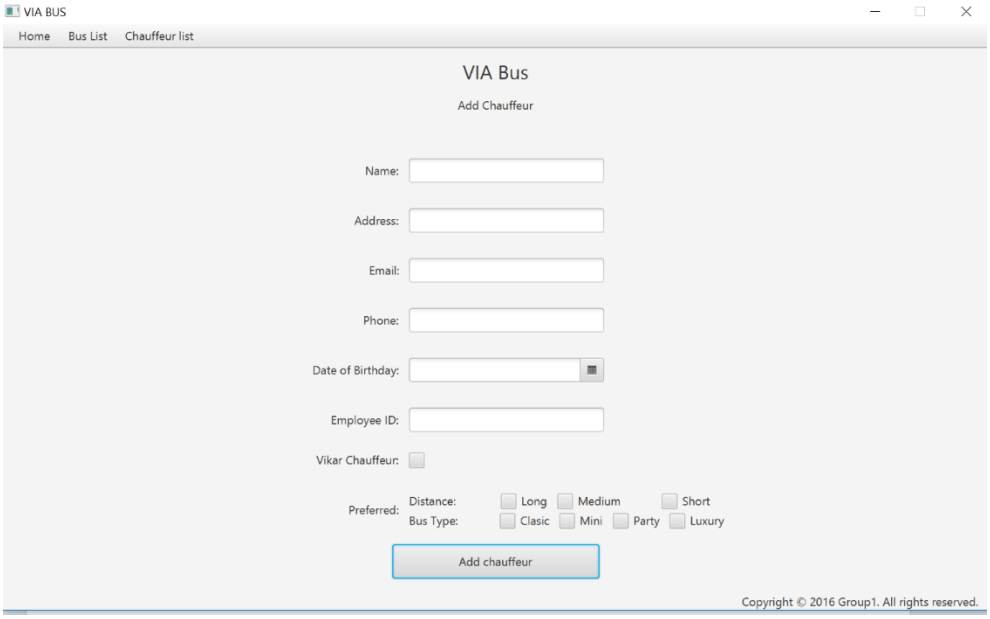
The user must go on the last tab of the menu and select Chauffeur List in order to see all chauffeurs displayed in a list

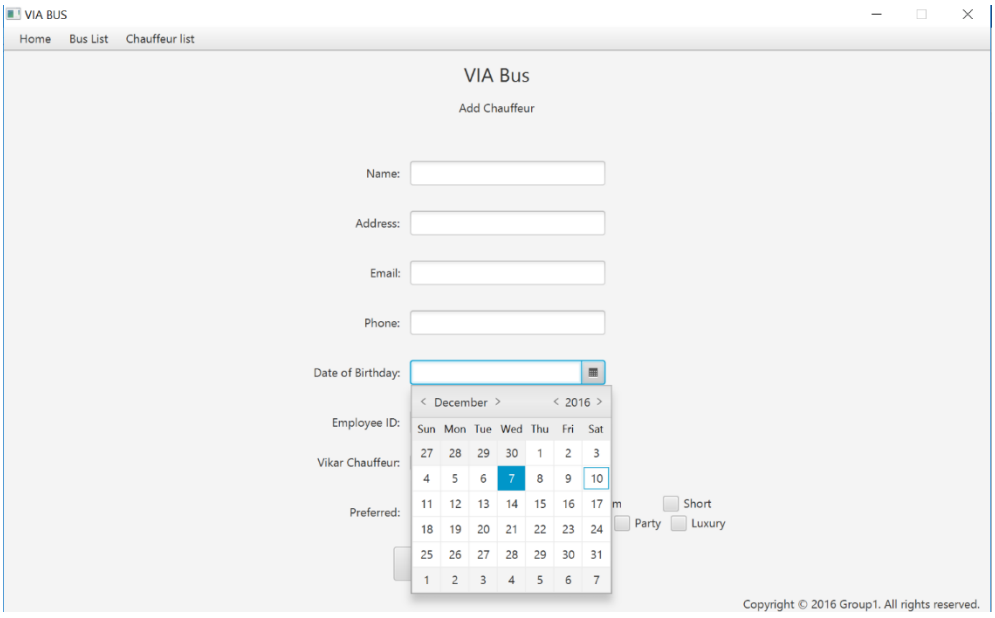


Add chauffeur

In the Add chauffeur, the user must fill in all the fields with the following mandatory data: name, address, email, phone number (8 digit), date of birthday(can be picked up or filled in with the following format month/day/year), employee ID (5 digit). The user can also select either one, or all the check boxes about chauffeur's preferrences: distance and bus type. For long distance, the chauffeur prefers only tours with a distance bigger or equal with 2500 km, for medium between 400 and 2500 km and for short distance less, or equal with 400 km. If the user adds a vikar chuffeur, no preferrences are needed to be selected.


VIA
BUS


VIA BUS
Home
Bus List
Chauffeur list
VIA Bus
Add Chauffeur
Name:
Address:
Email:
Phone:
Date of Birthday:
Employee ID:
Vikar Chauffeur:
Preferred:
Distance:
Long
Medium
Short
Bus Type:
Clasic
Mini
Party
Luxury
Add chauffeur
Copyright © 2016 Group1. All rights reserved.


VIA BUS
Home
Bus List
Chauffeur list
VIA Bus
Add Chauffeur
Name:
Address:
Email:
Phone:
Date of Birthday:
Employee ID:
Vikar Chauffeur:
Preferred:
December
2016
Sun Mon Tue Wed Thu Fri Sat
27 28 29 30 1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
1 2 3 4 5 6 7
Short
Party
Luxury
Copyright © 2016 Group1. All rights reserved.