Högskolan Kristianstad

Faculty of Natural Science
Department of Computer Science
Kristianstad University
Course: DA122B
Program: Project Course 2
Authors: Ivan Marosan and Niclas Everlönn

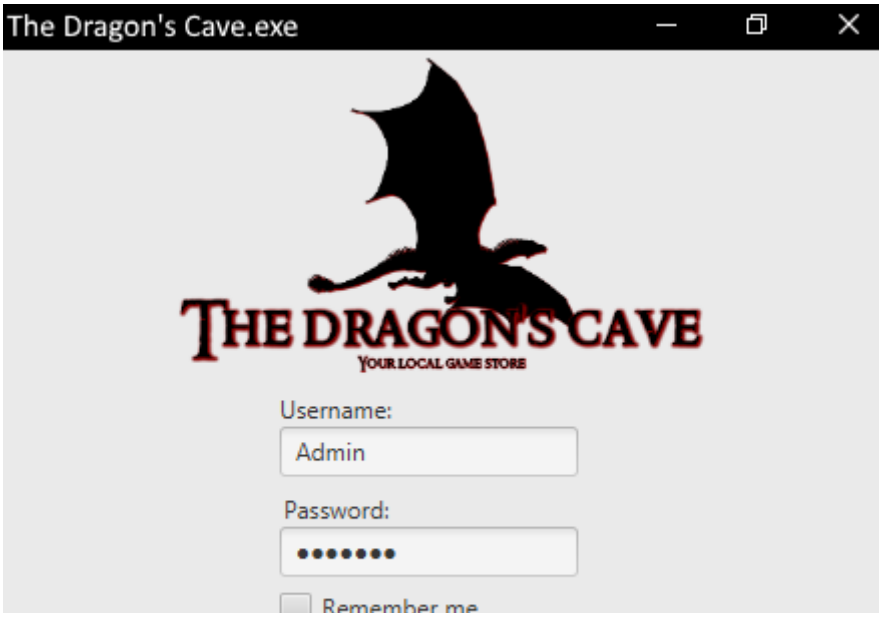
The dragon's cave – An online hobby shop

Table of Contents

Innehållsförteckning

1. Introduction

This application that is being created is a shopping application utilizing JavaFX. The main purpose of this application is to be able to handle items that are in stock that a user can find if they want to purchase or just view the items that the store has in stock. The application itself is a hobby shop of sorts, dealing in board games, figurines card games and other such merchandise you would find in a similar physical store. This is how we want the application to work, a software where a person will be able buy for goods online. The reason why this application is needed is because we are moving to a more global economy, and with a global economy also comes a global market. More and more people these days are shopping for items online rather going to physical stores to buy items and such stores needs to adapt this. One can just look online, and one will see that there are plenty of stores who happens to have an online division where someone can buy items from; Amazon and Mediamarkt to name a few stores who operates online. Thus, stores who don't have an online division or have an online outlet will be severely crippled in today's market. Therefore, a shopping software is important, because without it some companies will just be terminated today's competitive market. Henceforth the software we're creating will function like an online retailer where a user can login and find items to their liking and then buy said items if they choose to.



2. Requirements

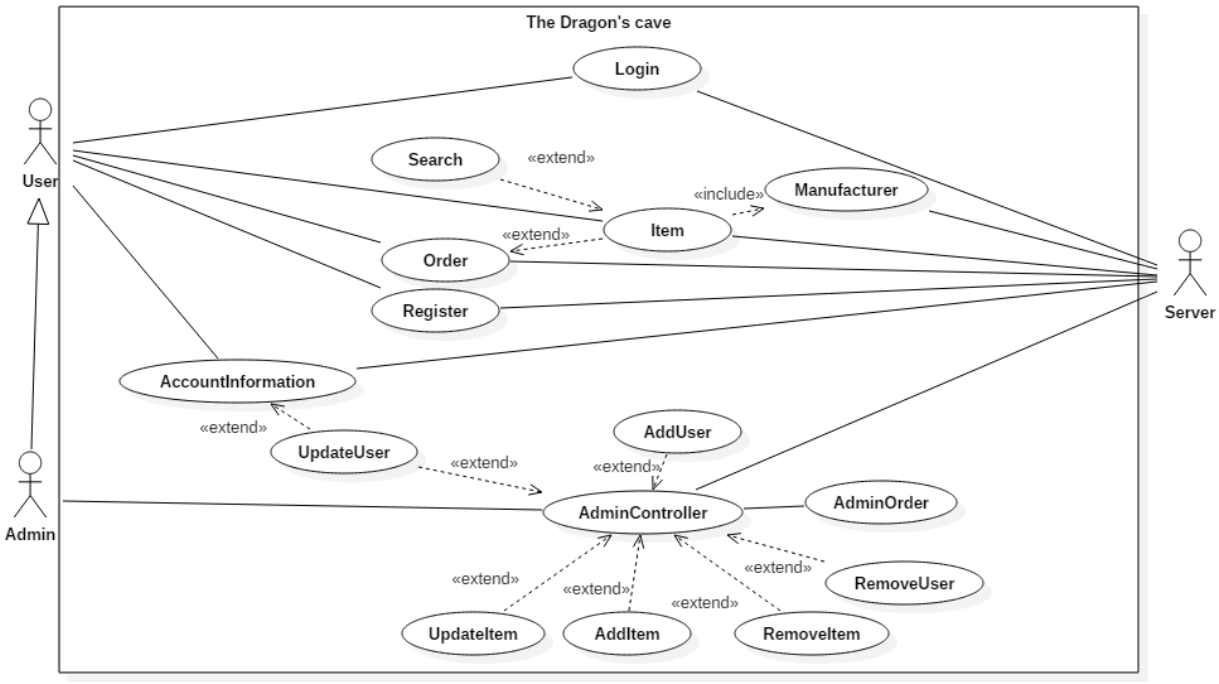
These are the requirements for our software:

Table 1 - Requirements

Req. No	Req. Name: Description
1	**User** This is the main actor for our software. The user shall be able to view, search and buy items that are in the stock. The user will buy items via cart where they can add and remove items. If there are no users in the system one will be tasked to create a user.
2	**Admin** This is a sub-actor that does nearly the same thing as a normal user, with the exception is that the admin is a worker at the store, so they should be able to add items to the stock and add a price tag to them.
3	**Database** This is required to store various information that might be important. Like user information, order history and items that are in stock, etc
4	**Order** A user should be able to put an order of items that they want to buy
5	**Login** To be able to shop in the store you need a user and to have a user you need login information to be able to log in the store.
6	**Search** Browsing items can be time-consuming especially when one wants to find one or two specific items. Thus, a search function will be needed to find specific items if they exists.
7	**Order history** A user should be able to view their order history of what they have bought.
8	**Administrative tools** This is tools for various tasks that the Admin role needs, for instance like changing stock and adding prices to them, remove users, and to order from manufacturers and such.
9	**Registration** A user should be able to create an account and fill it with information such as billing address.
10	**Display** A shop needs to have items on display, while an online store doesn't have counters to place items on it does have a store page with items to show and which categories an items belong to.

2.1 Use Case Overview

This is our use case diagram. It shows the most important functions of our software and how it will pan out. You have a user who is the actor who mainly interacts with the software, they will be able to login to the server, view items, register an account and make an order. The user shall also be able to update their account information if the choose to. Afterwards we have an admin that inherits from the user. The admin is a user in the system but unlike a regular user the admin has special properties to it and it's in the form of the AdminController case. This case is meant specially for the admin and it's meant to give the admin the ability to change objects in the server. Such as adding and removing users and items or updating them with new information. Lastly there server itself has relationship with every major use case since it's there where data is sent to and received from. Below will describe the major use cases in the system and their function inside the system.



2.1.1 Search

This use case's purpose is to give the ability to search for a specific item of the users choice. It extends to items since it's primarily there where the search function is used. The reason why we is mainly so the user doesn't have to scroll through an entire list of items in the system if they just want specific items.

2.1.2 Item

This use case is like a store counter or display. It's meant to show the user what items are currently available in the shop and how much they cost. The user will be able to make an order of said items when they buy from the store.

2.1.3 Order

The Order use case's function is for the user to place an order in the shop. Akin to a cart it's meant to store items that a user wants to buy and if they chose to buy said items they will be able to checkout with their order. The user shall also have an order history where they can see what previous orders they had. The reason why it extends rather than include is because a customer can login without having to make an order of items if they so choose to.

2.1.4 Manufacturer
This use case is to mainly highlight any potential manufacturers an item might have. The reason why this is a separate use case is because manufacturers is it's own table and class in this project. The reasoning why is because some manufacturers have more board games/creations and thus it would be better to make it a separate function rather than bulk it in with items

2.1.5 Login
The actors who interacts with this use case are user and server. The reason why these two interacts with Login is because a user has to login in to the store if they want to use any functions that the store has, and to do so the server must be able to check if said user exists in the system before a user can do anything inside the store.

2.1.6 Register
Similar to the Login use case; the actors who utilize this use case is the user and the server. This use case is meant to represent the ability to register an account in the server. To do so the user creates an account which the server receives to check if such a user already exists. If not the a new user will be made; however if a user with same name exists in the system it will produce an error stating why.

2.1.7 AccountInformation
This use case represents the user's ability to view their own personal account information. Such things might be one of the following: their first and last name, phone number, age, address and email they entered in during the registration process. Another use case extends to this and it's the UpdateUser use case. The reasoning for this is because people never stay the same. People move, changes their name or contact information and thus the user should be given the opportunity to update and correct the information so the items they bought don't get sent to a random person or address. The reason why it extends and not include is because if the user doesn't want to update their information they don't have to. The information to this use case is passed through the server since it stores all the information that's necessary.

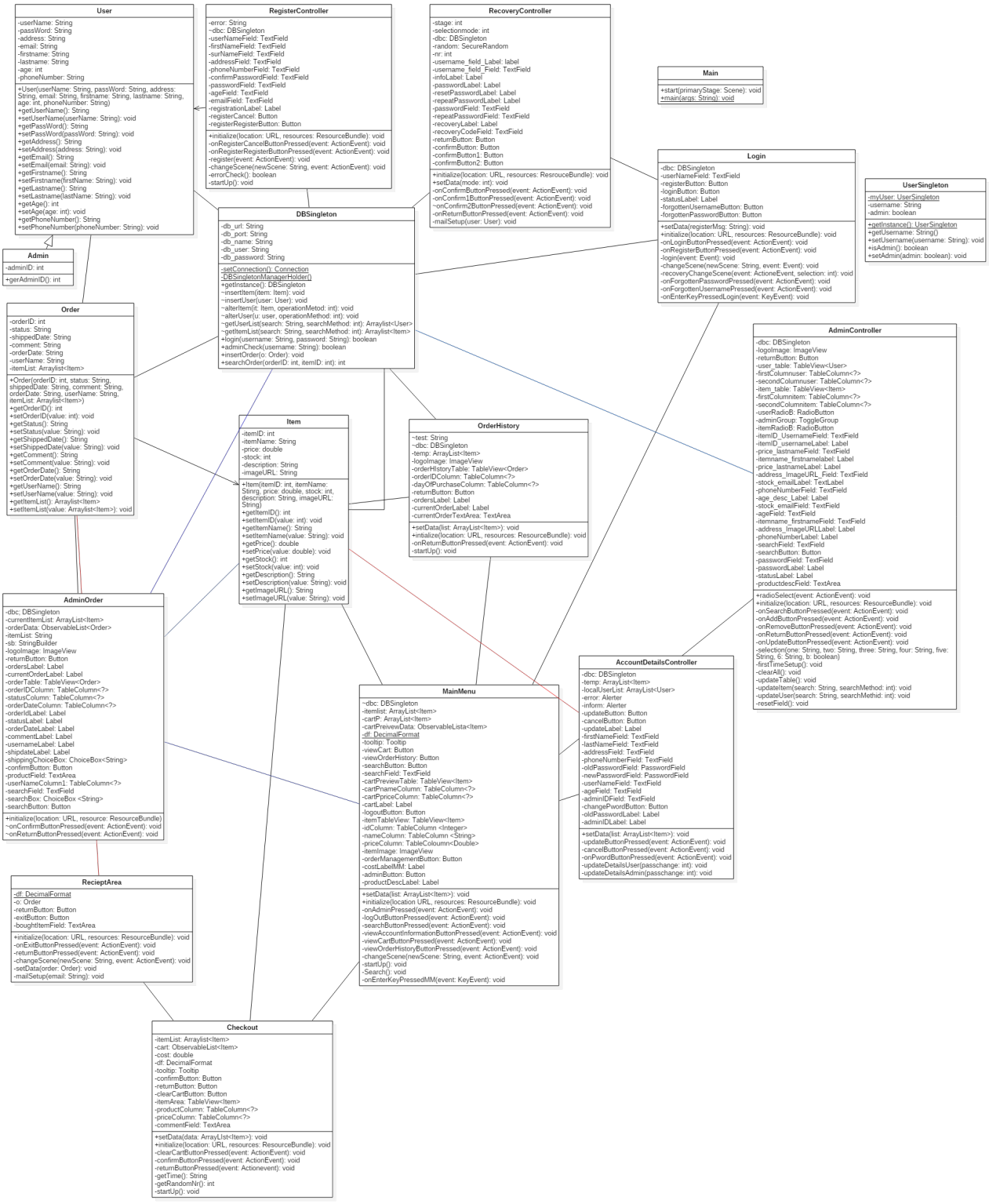
2.1.8 AdminController
This use case is only meant for the admin user and the server. This use case holds all the important functions that an admin should have when they work. As such there are six use cases that all extends to this because while they are necessary for the admin to his or her job. They are not required to do should the admin choose not to. While it's six use cases that extends it's only three really unique ones because their functionality are quite similar. The key difference is one affects the users in the system while the other affects items in the system. Henceforth they will be address as add, remove and update use cases when covered in this segment. The add use cases is meant for AddUser and AddItem. Their function is to give the admin the ability to insert a user or item in the server. While it's quite rare for an admin to just insert a user on a day to day basis it is important if they choose to employ a new recruit. The crème de la crème is the AddItem use case. It's function is for the admin to add new items in the system which are going to be displayed at the store. The remove use cases are meant for the admin to remove users or items in the system, signified by RemoveUser and RemoveItem use case. While removing a user is a rare thing to happen there might come a time when such a function might be needed, such an employee getting the pink slip or a user requesting that their account gets deleted. Items being removed from the system however is more common thing that's going to happen. A manufacturer might one day choose to cease all production of an item and when the last item from the stock is sold there serves little to no point to display such an item. Lastly are the update use case which are represented by UpdateUser and UpdateItem. Their functionality is to give the admin the ability to manually update a user if said user requests, and lastly to update the price/id/quantity of a product in the store.

2.1.9 AdminOrder

This use case represents the ability to change the status of an order. For instance if the order is shipped, in the process of being shipped or if the order has been canceled. This use case has some minor association with the Use Case AdminController due to the fact that it's something that an admin is supposed to do and it's something that is modifiable. However in the software it's not something tied to the actual scene: "AdminController" itself.

3. Design and Implementation

Here one will find our class diagrams and EER-diagram to see how the overall look on the software will be.



3.1 Classes

Here's the class diagram for our software. While there's nothing particular to point out in this class diagram there is however one thing to point out and that's that almost every single class in one way or another has a relation with the DBSingleton class. This is due to the fact that this is the class that manages everything between the database and the software.

3.1.1 MainMenuController
This class is the main menu in our software. This is here where a user has the ability to make a purchase of items that they want. In this scene there's also other functions such as logging out if they desire to log out. There's also the ability to view older orders that one has made. One shall also have the ability to go to their account information from this scene. One thing to add that this scene contains two more options if you're an admin. These two are an admin button that will take you to an admin page where one can modify users or items and lastly an order management button where an admin can manage orders in the system.

3.1.2 AccountInformationController
This class represents the scene where a user can see all their account information that they made during registration, and if the needs arrive, they can update their account information by the various text fields. This is important because in the events if one would change their name, place of living, phone number or password this information needs to be update since a shop that ships item to the wrong place isn't a good shop to begin with. The two methods are for the buttons in the scene and they either take you back to the main menu if they want to return, they do so via the cancelButtonPressed method. The other method is to confirm that the information one entered will be updated to the latest.

3.1.3 RegisterController
This class is meant for a user to register an account into the system. The user shall fill in all the fields in this scene if they want to create an account. This includes the following: Username, first- and lastname, age, address, email, phone number and lastly password. When all these information has been filled an attempt to make a user will be made in the database. If said username already exists in the system the creation process will fail and the user will be prompted to give themselves a new username.

3.1.4 LoginScreenController
The purpose of this class is the able to log into the shop system. To do so one needs an account and there's a button with the method registerButtonPressed. Pressing this button will bring forth the RegisterController scene and one can make an account there. If they do have an account, they just fill in the fields and login in. However if said user would have forgotten their login information there will be an option to recover said information

3.1.5 RecoveryController
This class represents the methods for one to recover their username or password if they've forgotten them. The user shall be asked to give their email from where the database will check if a user with said email exists. When done the user will receive a recovery code which they can use to recover their password and/or username.

3.1.6 OrderHistory
This class function is to represent a list of orders that one has made into the system. The user shall see what date the order was made as well as the status of the order such as if it's shipped, pending or canceled.

3.1.7 DBSingleton
The function of this class is to connect the database with the software and pass through information between them.

3.1.8 User
This class is meant to represent a user object. The point of this class is for the software to pull users from the database into the software. Thus the purpose of this class is to make sure that we can display the information from the database in the software.

3.1.9 Admin
This class inherits from the user class and it's similar to the user class for what purpose it exists for. It's meant to represent an admin object from the database that we can display in the software.

3.1.10 Item
Similar to the User and Admin class. It's meant to make objects out of the information in the database for us to display in the software.

3.1.11 Order
Just like the three previous mentioned classes. It's meant to make order objects out of the information the Order table from the database.

3.1.12 Checkout
This scene represents a bridge between the main menu and the receipt. One shall have the ability to remove unwanted items from the cart without having to go back to the main menu. The user shall also have the ability to added any comments to their order if they chose to do so.

3.1.13 RecieptArea
This scene represents the a receipt for the user that has made a purchase. The user shall be able to see what they bought from the store and as well have a receipt sent to their email.

3.1.14 AdminController
This class represents a scene where the admin has their administrative rights. Only an admin can access this page and for that a user needs an admin ID. In this scene an admin shall have the ability to modify information in the database. These things are items; such as adding, updating and removing them from the system. They shall also have the same ability with users. To add, update or remove users from the system. This page is also the only option for someone to update usernames and age of a user.

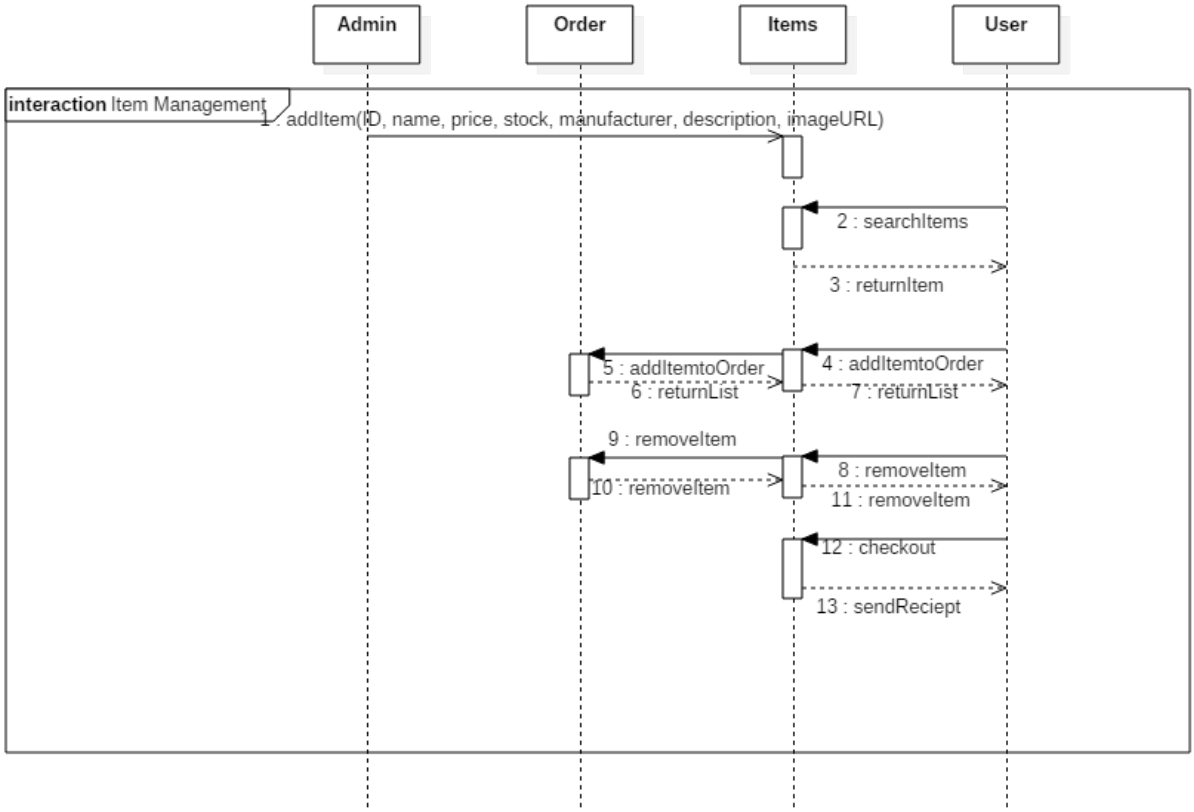
3.1.15 AdminOrder
This class is for the admin to update orders in the system. The admin shall have a list of various orders in the system and will have the ability to update them accordingly. The admin shall also have the ability to search for specific orders from the following things: Order ID, Username, Date of Purchase and Status.

3.1.16 UserSingleton
This class makes a single object of a user when they log in to the system. This is meant to reduce the time it takes between the database and the software to parse information between each other. Since without this class the software would have to collect a list of all users in the database before finding a specific user, and if said list of users is long it will take a long time before the software would display the information.

3.2 Class Interactions and Use Case mappings

Here are a couple of sequence diagram for how our software will operate.

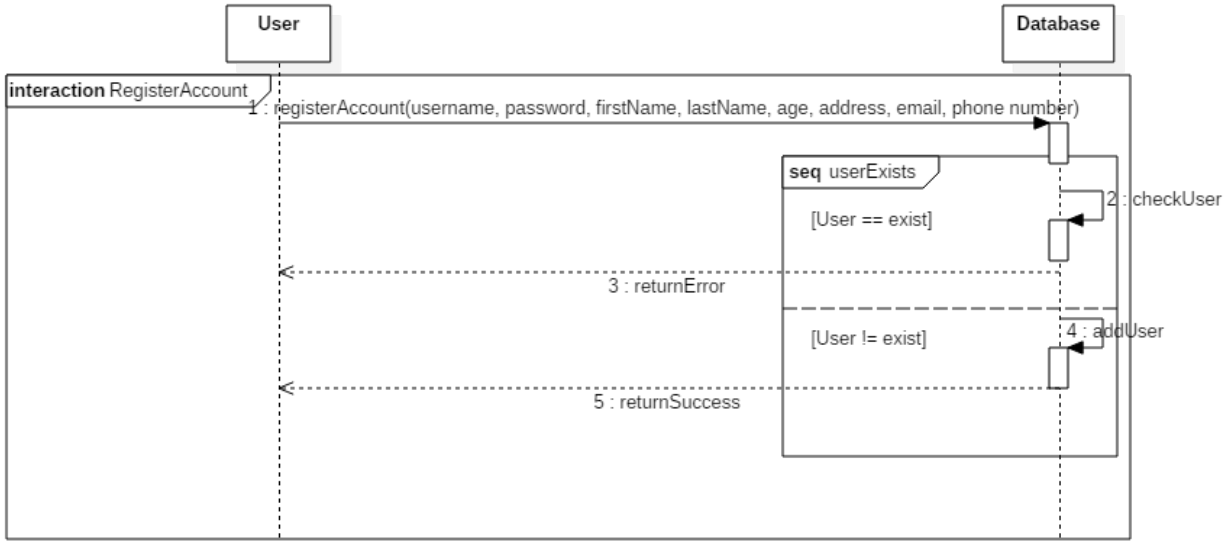
3.2.1 Item management



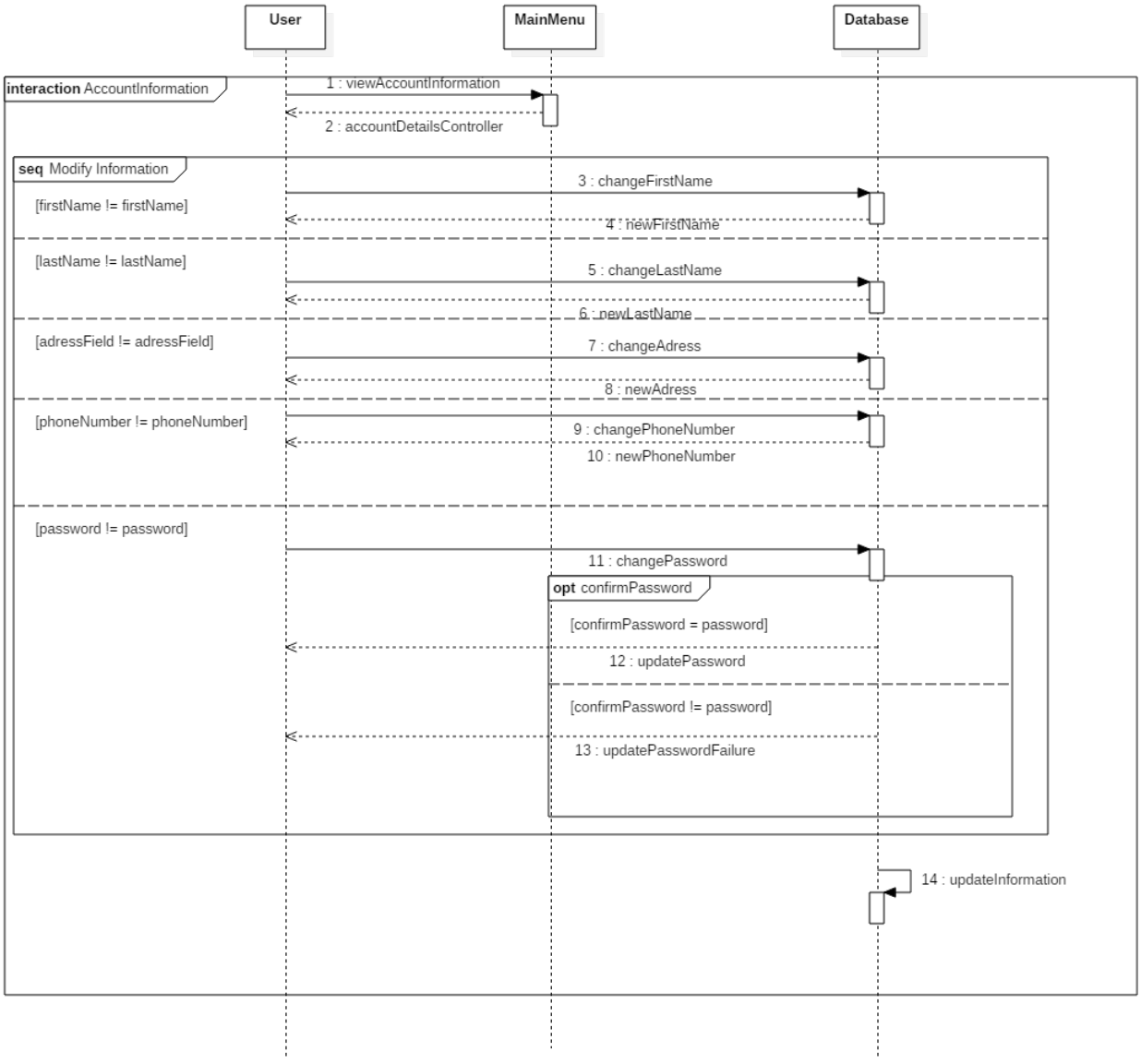
This sequence diagram shows how items in the systems are handled or rather how it's supposed to handle. It first starts off with that the admin adding items to Items. The Admin will give the item an ID, name, price and finally the quantity named "stock". Then the User will be able to view and find items. The user shall also be able to add items and the reason why this message is asynchronous rather than synchronous is because there's nothing to return back to the user. They will see that an item is in their cart. The reason why removeItem is synchronous is to update the cart information so there's no of said item in the cart. Finally the customer will be able to checkout and by doing so the admin will be notified that there has been a change in the items. The User will also receive a receipt that confirms their purchase.

3.2.2 Registration

Here's a sequence diagram on how account registration will happen in the software. The user will send a create account message to the database/server and then there will be an opt operation that will happen. In this operation there will be two guards that checks if it's possible to create an account. The first guard checks if the account exists. If it does not exists the database will create an Account and return a message saying that registration is complete. If said account exists the database will send a registrationFailure message and the account won't be created due to conflicting information. What this guard will check is just the username. Since a user can have many accounts but each account can only have one username.



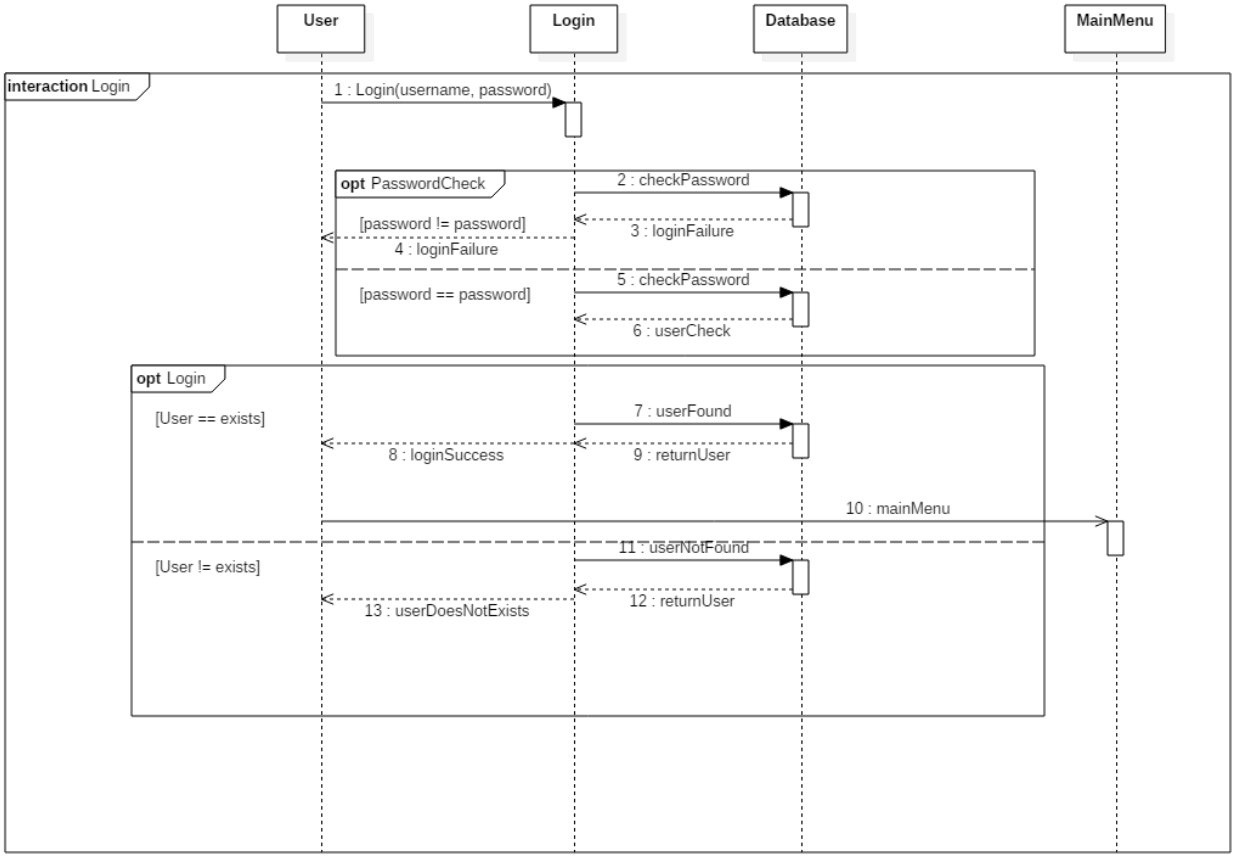
3.2.3 Information update



Above is the sequence diagram that shows how updating account information will work out. The software will go through each editable field and check if they match with the information in the

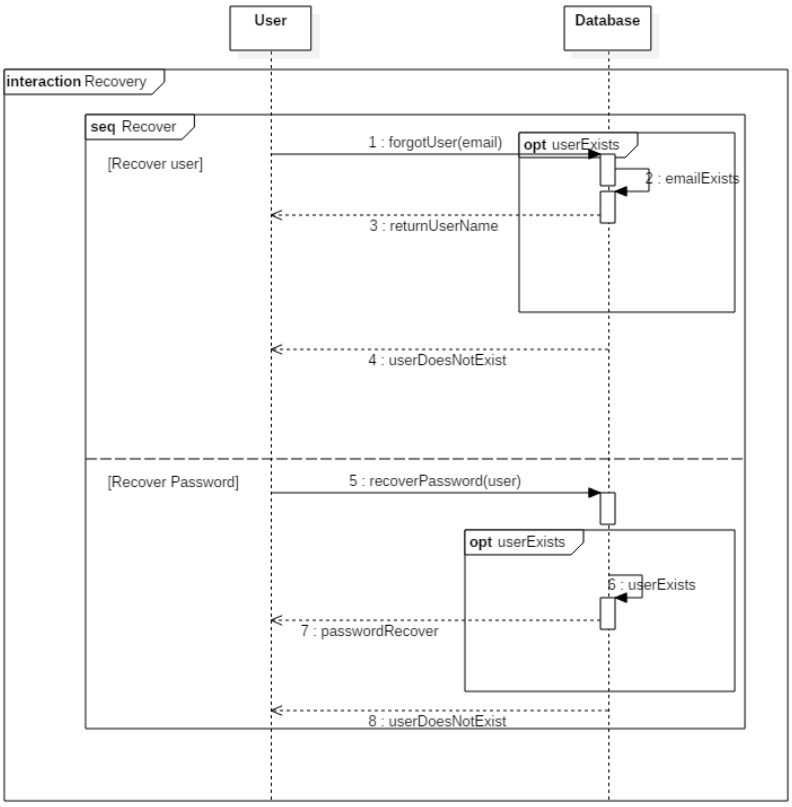
database. If said information does not match it will update the information accordingly and the user will be notified by this. The fields that we want to be able to edit are both the first and last name. This is due that people can legally change their name at any given point. We also want the user to be able to change address; this is due to the fact that people have a tendency to move away and thus need to update their shipping information. We also want the user to update their phone numbers just so the store has the potential to call the user in case something happened to their order. Lastly we want the user to be able to update their password. This is mostly for security reasons. Finally when all the checks have been made the database will update accordingly. The reason why this self message is last is because it would be quite the time constraint if the user made a lot of changes and then it had to update every single time.

3.2.4 Login



Here's a sequence diagram over how login in to the sore would look like. First the user attempts to login. The user will attempt to login and the software will check if the user exists and will try to login and if the user exists in the system the login will be successful and the user will be taken to the main menu. If not the software will return that the user does not exists.

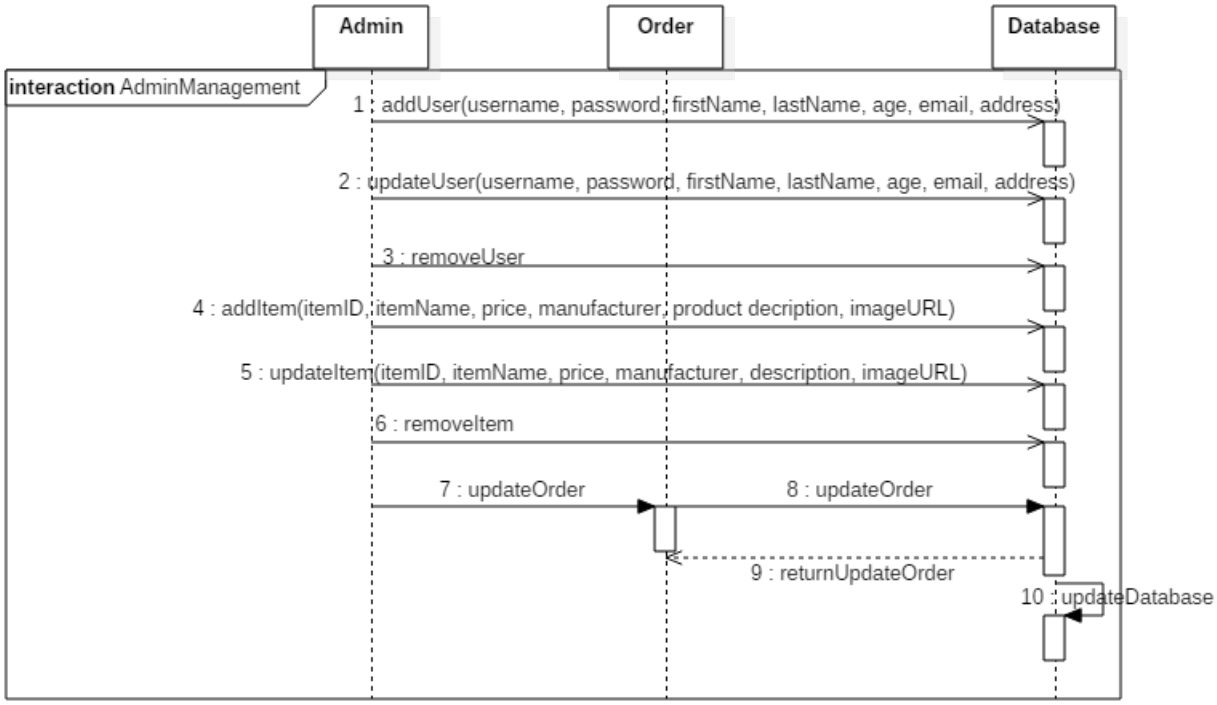
3.2.5 Recover information



This is sequence diagram for when a user wants to recover their login information. The user shall be given the option if they want to recover user or password. After their choice has been made they will be asked to fill in their email or username for password. If said information matches with the information the database a recovery email will be sent to the email that's connected to the user.

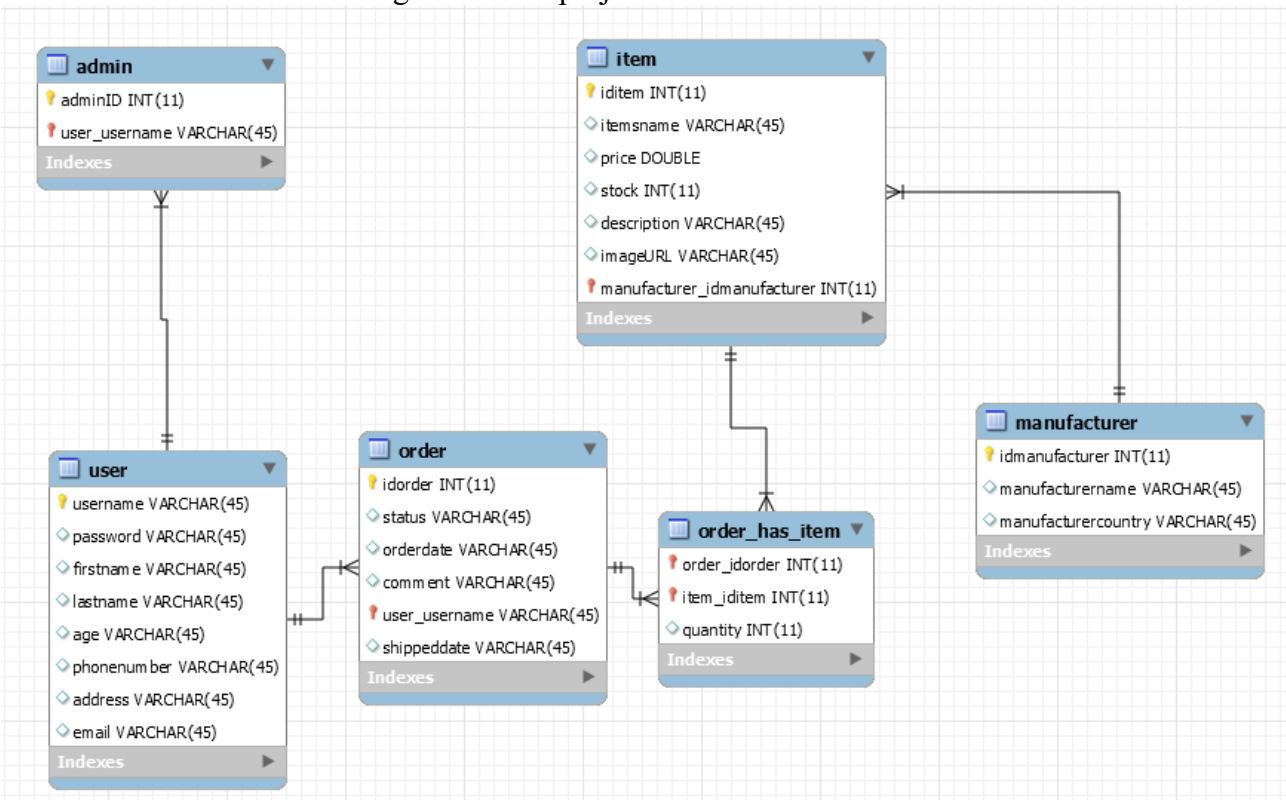
3.2.6 AdminManagement

The sequence below shows how the AdminController class interacts with the admin. The admin shall have the options to update information about users and items. They shall also have the ability to remove or add users and items in the system. Lastly the admin shall have the ability to update the status of an order



3.3 Database

Here one will see our EER-diagram for the project.



4. Test Results

Here's the current list of requirements are are either passed, failed, partially implemented or not yet implemented in our software.

Table 2 - Test Results

Req. No	Req. Name	Test Result
1	**User**	**Passed** We've established a user class and the ability to register a user into the database
2	**Admin**	**Passed** There's an admin and the function to add an admin to the system
3	**Database**	**Passed** We have a database on a server and one can use it to fill in information.
4	**Order**	**Passed** A user is able to make an order that will send receipt confirming their purchase
5	**Login**	**Passed** We have a login in screen where one can use an account and log in to the system.
6	**Search**	**Passed** We have a search function where one is able to search for items, or users if you're an admin
7	**Order history**	**Passed** One is able to place and see their older orders with order management
8	**Administrative tools**	**Passed** We currently have an admin page where one can see users and items in the system. One is also able to remove or add user or items here.
9	**Registration**	**Passed** One is able to register an account into the database.
10	**Display**	**Passed** We have a functioning GUI where one can see what items are on display.

5. Summary and Conclusion

Here is the weekly report for what the members of the group have been doing through the project.

5.1 Weekly Progress

5.1.1 Week 1

During this week we as a group started to discuss about the design on our software. There were plenty of back and forth on what kind of software we wanted to create, everything from a web page to an FXML application. We decide that a shopping software that because it would be something fun to do.
We started to design the use case and the overall design of the software and the use case in the report. There were no overall difficulties this week; everyone worked with what they were tasked and there were no complaints about it. What was done this week as the gannt chart, use case and everyone committing something to gitHub. Overall the first week was an overall success with everyone worked with their assigned task and everyone learn how to commit something up to gitHub and how to make use of a gannt chart.

5.1.2 Week 2

During this week we as a group started to work out the general design for our software. We finished up the class diagram and made one sequence diagram that shows how items will be handled in the system. We all gave each other assignments to do individually first. Both of the group members decided to handle the database and it's connection this week. While there was some frustrating moments of it not being properly connected it was solvable. During this week the group did split up the work load for the software. One worked on fixing up the GUI elements like various scenes while the other made some basic classes like Item. Overall this week was a productive week where lots of the smaller things was done to make the software at least testable so people can see how the software is supposed to look.

5.1.3 Week 3

During the third week we decided to split our work into two categories One group member focused on making a prototype for the software where you can at least navigate through the menus. The only noteworthy code to add here is the code for switching scenes and a testing class where one could hard code elements. Otherwise there's not much to it. The testing class exists solely to test out various variables that one would use in the software. The other group member focused on fixing up the report and the opposition report that was due to the first seminar. There was a little bit of help with proofreading in the reports due to the overwhelming about of writing that had to be done. The rest of this week was spent preparing for the first seminar that the group had which was an opposition. Due to this there was a bit of conflict in time management due to the fact that the preparing for the opposition took a bit longer than estimated. Despite this poor management of time, this week was overall successful. The software was filled up with various testing variables that could be used and show how the software is supposed to work.

5.1.4 Week 4

During this week the group decided to finish up the DBSingleton class by finally adding in SQL queries to it. During this week almost all of the rudimentary SQL queries were added to see if the connection works, and to see if we could populate and collect the information from the database. Other than the group decided to finish up at least 95 % of the software so it reached the deadline for this week. This was done by making the basics works for instance such as adding a user to the database and logging in with a user that exists in the database. During this week we also took to heart what our opponents told us during the opposition. These things involves fixing up some

redundancy in the code and making a method for changing scenes rather than writing every time the code for changing scenes.

5.1.5 Week 5
During this week the group decided to start working on the last details of the project. These things involving numerous stuff such as music in the main menu. The ability for the admin to handle orders, and lastly the entire database connection and all of its SQL queries. Alongside with this the group decided to spruce up the project a bit. This involves improving the database connection making it more efficiency. Cleaning up old/dummied out code During this week everyone was focused on finishing up the report due to extra time was given to finish it up and such nearly meet every single day so the project would finish up.

5.1.6 Week 6
During the last week the group decided to add many quality of life improvements to the software. These functions are the following: Information on how to add items, setting up an email to send to a user, the ability to recover account information and many more options. During this week there was also a lot of clean up in many aspects of both the code and the design part of the project. These things involves: Fixing older design that did not match our current projects status, removing dummied out code yet again. During this week there was also a good communication between the group members and everyone helped each other out to make sure that the software and it's design are up to snuff.

5.2 Difficulties and challenges
Here's a list of some troubles that we managed to stumble upon our project

5.2.1 The dynamic duo
One of the biggest difficulty that this group faced was the fact that we were just two people working on it. While the software in of itself lived up to what we envisioned it to be; a shopping application for board- and card games. There was naturally some difficulties due to the fact that there was only two of us. This stems from the fact that two people working on a project of this scope is not an easy feat due to the amount of work one was expected to do. This naturally caused some stress issues which did not help. However the way that this group mitigated that was to make sure that there were daily meetings so the group could work on the project. This involved meeting up and discussing on what tasks/requirements the group should tackle down today and make sure that said task/requirement was meet for the day.

The lesson that could have been learned from this is to ask the professor to see if there are available people for us to add in the group. While this is a rather rudimentary lesson and not a stellar lesson. Since it pins more work for the professor to find people; it is still a lesson. The other way one can look at this lesson is a much harsher way but would ultimately be a bit fair on the professor's part to find suitable group would be to split up the group and put the members in different groups. Naturally this also has it's own problems. Mainly that a group may or may not be accepting of a new member just hopping in. Overall for the next project the solution would simple be: find more people that wants to co-operate. As basic as it sounds it is the lesson that we will bring with us for the next project.

Other than that there weren't any other big issues with this group. There might have been some smaller problems but they were too trivial or not really difficulties. Just a small hassle

5.3 Time estimates

The time estimations for this project were on the mark. When the gannt chart was designed we decided to prioritize things that might take up a lot of time would get more allotted time than something smaller. Example of this would be the administrative tools. This was given almost a full work week to be done. This as we saw it; an admin has to have all the necessary tools that they need to control various elements in the software, and such thing might actually take time. While something such as making various classes like Item or User was given less than a day due to how simple it is to make a class that only holds variables in it. Naturally there were some things that were off the mark. Not by much and such would be making a database. From how we saw it: making a database would take up a lot of our time and thus we allotted a lot of time to it. Unfortunately that proved not to be the case at all. The database we had envisioned us did not take up a lot of time. In fact it took way less time design and create than we anticipated.

The lesson that we will bring us for the next project is not a big one but it's still a lesson. That is to talk over the group on what features are really important and what features are less important. Because when one has grasped on what is important for a software one can usually make a relatively accurate time estimation with a small margin error. Another thing that we will bring with us and that it's better to allocate more time to something. This is because if becomes complete earlier than intended you get more time to do other things.

5.4 Priority estimates

The priorities for this project were on correct for most of the part. Looking back there's not much that we as a group want to change around with the priorities. Even if we wanted to change things around it wouldn't be much other than moving some things up or down by one point or two on the priority sheet. So overall we as a group did priorities the correct requirements that we assigned however due to certain circumstances we're not quite sure if we managed to deliver the highest priorities to their fullest potential but we did however managed to deliver the highest priorities tasks without much of hassle.

The lesson that we learned from prioritizing is to spend time before making priorities. We first threw out some priorities/tasks that we would think that a project would need and then we discussed in the group what would seems important and what would seem unnecessary. Anything that was deemed useless or extra fluff would be the first thing that got tossed out due to they aren't the core part of a software. Afterwards we assigned points to the priorities/tasks based on what we deemed would be important. An example in this application was that we priorities a user heavily due to the fact a user is the person that interacts with the software; while we didn't priorities the GUI heavily and kept it rather simple. This is because a GUI needs to be functional first before looking pretty. This is a lesson that the group members will bring with them during the next project: spend time when working on priorities. Without a good priority list it becomes difficult to work with a project.

5.5 Conclusion

The overall experience for this project were stressful. While the communication between the two members were good and the work that was made was of decent quality. There were some things that unfortunately took a bit of a hit due to these circumstances. This was mostly the visual aspect of the software since our GUI is rather basic and sleek. This stemmed from the fact that we could not put enough time into it. Despite these struggles the overall project has been a big learning moment for the group. That is to utilize our strengths to help each other out where we can. Because we are humans and as such we're not perfect beings. There are some things that others are better at hence

why it’s important to make fill in the gaps where ones knowledge may falter. Another thing that was learned from this is to keep the spirit/morale of the group up. The importance of this cannot be stressed enough. Since when the morale is low the overall productivity gets lowered. However by keeping the morale high we could deliver the work that we did.

If there’s one thing looking back with this project and that was that Gannt chart wasn’t really properly utilized, and with this it’s with the overall usage of it. First off it weren’t followed by the group members which is a fault on our end. Secondly it wasn’t properly brought in during the project meetings which felt like it wasn’t important in the grand scheme of things. Despite that the Gannt chart had to be brought in during every project meeting with the report. With it being pushed aside made it feel tacked on rather than a set of instructions for the group to follow. This will probably be brought in during the next project course. To utilize the Gannt chart more efficiently.