BeegShip

Shipping Subsystem

Brice Higa, Christopher Koepke, Lorenzo Cabrera, Lance Martinez, Josue Cruz

CS3650-01

Cal Poly Pomona

Pomona, CA

[bmhiga@cpp.edu](mailto:bmhiga@cpp.edu), [ckoepke@cpp.edu](mailto:ckoepke@cpp.edu), [josuec@cpp.edu](mailto:josuec@cpp.edu), [lancem@cpp.edu](mailto:lancem@cpp.edu), [lvcabrera@cpp.edu](mailto:lvcabrera@cpp.edu)

***Abstract*—We are creating a shipping subsystem to allow companies to adapt to the post-Covid-19 world and offer online shipping to their customers. The system will be able to receive shipping requests, create orders, update orders, and contact customers. The system will be able to store the information of the customers and recipients. This system will be implemented using Java and Microsoft Access Database.**

***Keywords—Java; SQL; Shipping***

# Introduction

Our goal is to create a shipping subsystem. Thus, we decided to create BeegShip, the name of our shipping company, which will provide shipping across and outside of the US and provide the best shipping rates.

## *Problem Description*

The issue we are trying to address has to do with the drop of in-store traffic due to the current pandemic and the resulting shift to online shopping. With retailers across the nation seeing a decrease in in-store purchases, online activity and purchases has increased. A company, that we will call Company X, seeks to meet this increased demand of online shopping by acquiring the services of a third party, which has a system and the necessary infrastructure capable of shipping products globally. Thus, we need to build the system for BeegShip to be able to handle the increased shipping requests by Company X and any other businesses that want to use our services. The system must also maintain a database for record keeping purposes and to help streamline our shipping process.

*B. Proposed Solution*

The shipping subsystem will be able to receive packages, transport packages globally, deliver packages to the recipient, and maintain an up-to-date status on any given order. Users can place new orders as well as view existing ones, add new representatives for their companies, and see when the status of a package is updated. The system will also store the banking information that the client will use to pay for services to make future payments more efficient.

# Analysis

## *A. System Requirements*

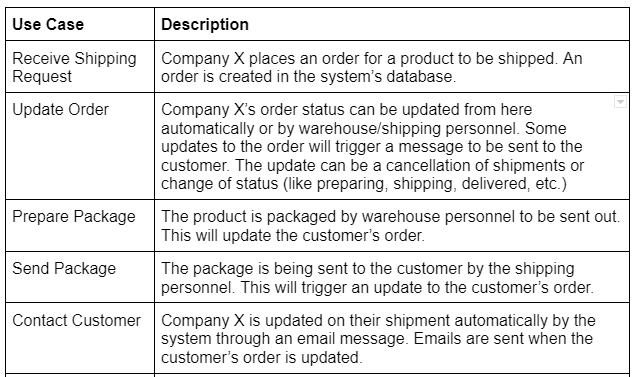
When analyzing the system requirements, the majority of our functionality revolves around users placing new orders under Company X, who will ship their orders through BeegShip. The most important part of addressing this was to first decide what types of customers that we would allow, whether we would allow only businesses to use our service or to allow walk-in customers as well. We decided that only businesses and not walk-in customers would be allowed to place orders through our service. For our customers, it is necessary that they can add and remove representatives for their company from the database. Allowing them to view and delete these representatives allows the companies to see who is in charge of their packages in case they get lost. Some other functionality that we needed to consider was the ability to update the status of packages. This is used by our employees to notify the customers of their package's status. User Interface wise, we need to to allow for packages of the customer and their statuses be listed out for the

## *Actors and Stakeholders*

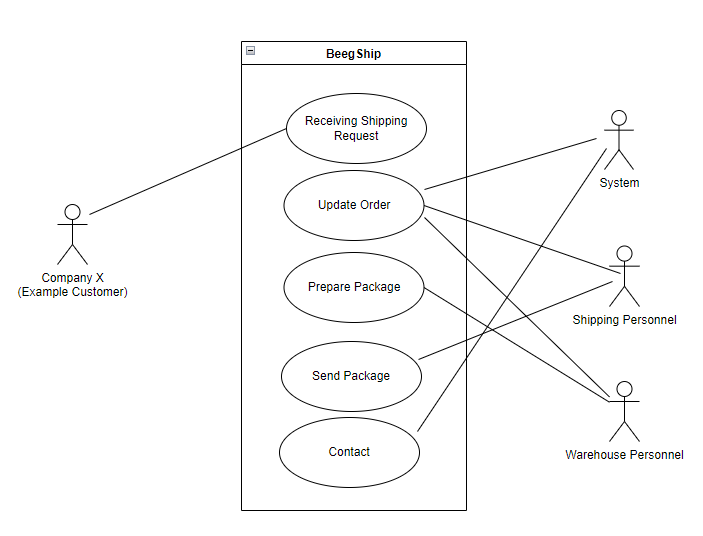
The actors for BeegShip will be the warehouse employees, the shipping personnel, and the customers.

The stakeholders for BeegShip will be the BeegShip executives, warehouse personnel, shipping personnel, Company X, and other business partners, such as investors and banking institutions.

*C. Use Cases*

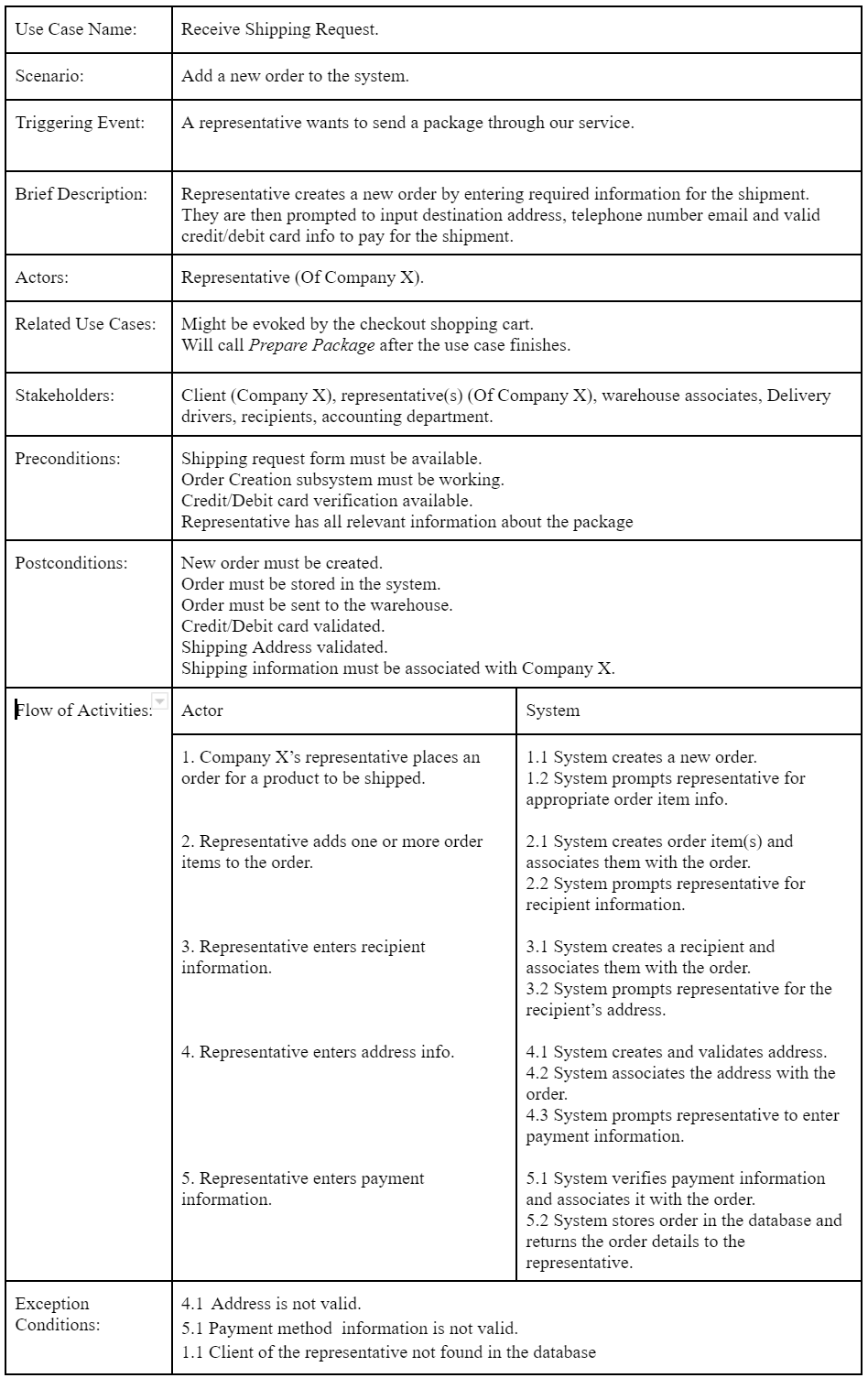


Elaboration on the use cases shown in the previous page. The “receive shipping request” is a use case in which a client wants their packages to be shipped through our system. “Update Order,” “Prepare Package,” and “Send Package” all relate to the entire shipment process of an order and will be used to update an order’s status that is viewable by the client. “Contact Customer” use case is a notification system for the client and recipient to update them on the status of their order.



Use case diagram that shows how our classes can be interacted with and by what actors. Shipping and Warehouse Personnel update the status of an order through some type of handheld device that is connected to our system. The System itself will send status information to the client and recipient when the order’s status is updated. The client (Company X) will request a shipment of their package and provide relevant information to complete the process.

*D. Fully Developed Use Case*



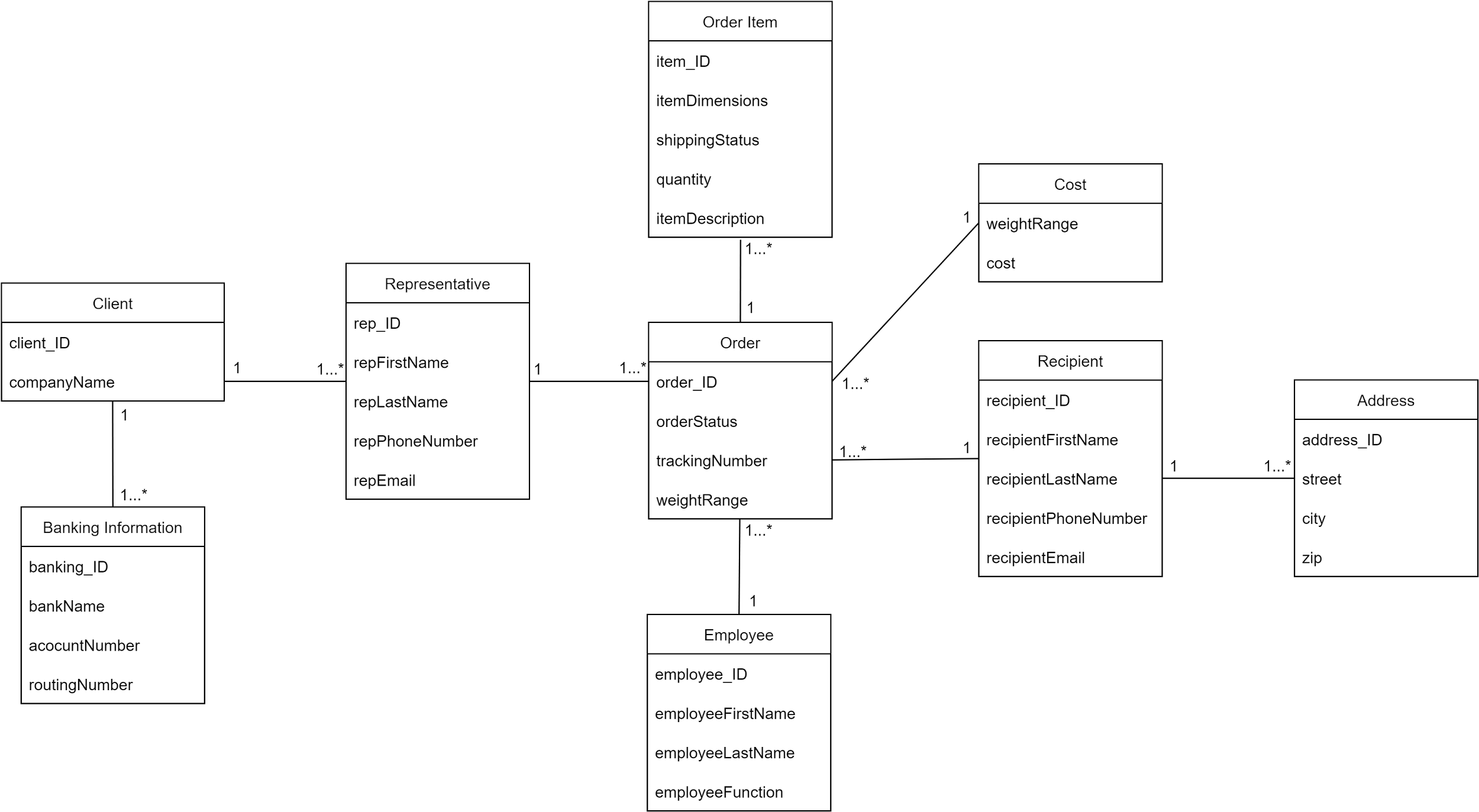
This fully developed use case demonstrates the use case “Receive Shipping Request”. The scenario that will call for this use case is when a new order is to be added to the system. This is triggered by the client's (Company X) desire to send a package through BeegShip. Upon expressing a desire to create a new order, the client will have to enter the necessary information for the shipment (information concerning the order items). The system will then prompt the user to enter the recipient’s address, phone number, and email address. Payment information that is entered will then be validated by interacting with the account’s banking system.

Actors associated with this use case are the representative(s) placing the shipping request. This use case will be invoked when interacting with something similar to a “checkout cart” and will be called “Prepare Package,” after the use case is complete. Stakeholders of the use case include: client (Company X), representative(s) (Of Company X), warehouse associates, delivery drivers, recipients, accounting department. In order for this use case to occur, the shipping request form must be available, the subsystem responsible for creating orders must be functional, payment method verification must be available, and the representative must have all required information about the order. After the use case has been invoked, a new order must be created, stored in the system, sent to the warehouse subsystem, payment method verified, shipping address verified, and shipping info must be associated with Company X’s representative.

The flow of activities begins with the representative of Company X placing an order. The system creates a new order and prompts the representative for the appropriate order item info. The system will create one or more order items after the representative has entered the required info for the order item, the representative may continue to add more order items to the order. Each order item is associated with the order.

After the representative is finished adding their order items to the order, the system will prompt the representative to enter the recipient’s information. The system will create a recipient and associate them with the order. The system will then prompt the user to enter the recipient's address information and create the address. The system will then associate the address with the recipient and order and then prompt the representative to enter payment method information. Upon validation of the payment information, the system will store the order in the database and return the order details to the representative.

*E. Domain Model Diagram*



This class domain diagram demonstrates the various relationships involved in the Beeg Ship shipping sub-system. As demonstrated, a client (Company X) is capable of having one or more bank accounts and one or more representatives. The representative(s) are responsible for placing one or more orders on behalf of the client they are representing.

Each Order will have one or more order items included in it. An Employee will be responsible for handling one or more orders. This means that an employee, either warehouse associate or delivery driver, will be responsible for sorting, transporting, and delivering one or many orders. The cost of an order is dependent on its weight and in what weight range it falls under. Each of these weight ranges are associated with a cost in the Cost class. This allows our company to easily change our rates without needing to update the cost of each order. One or more orders is associated with a single recipient. The recipient will have one or more addresses associated with them.

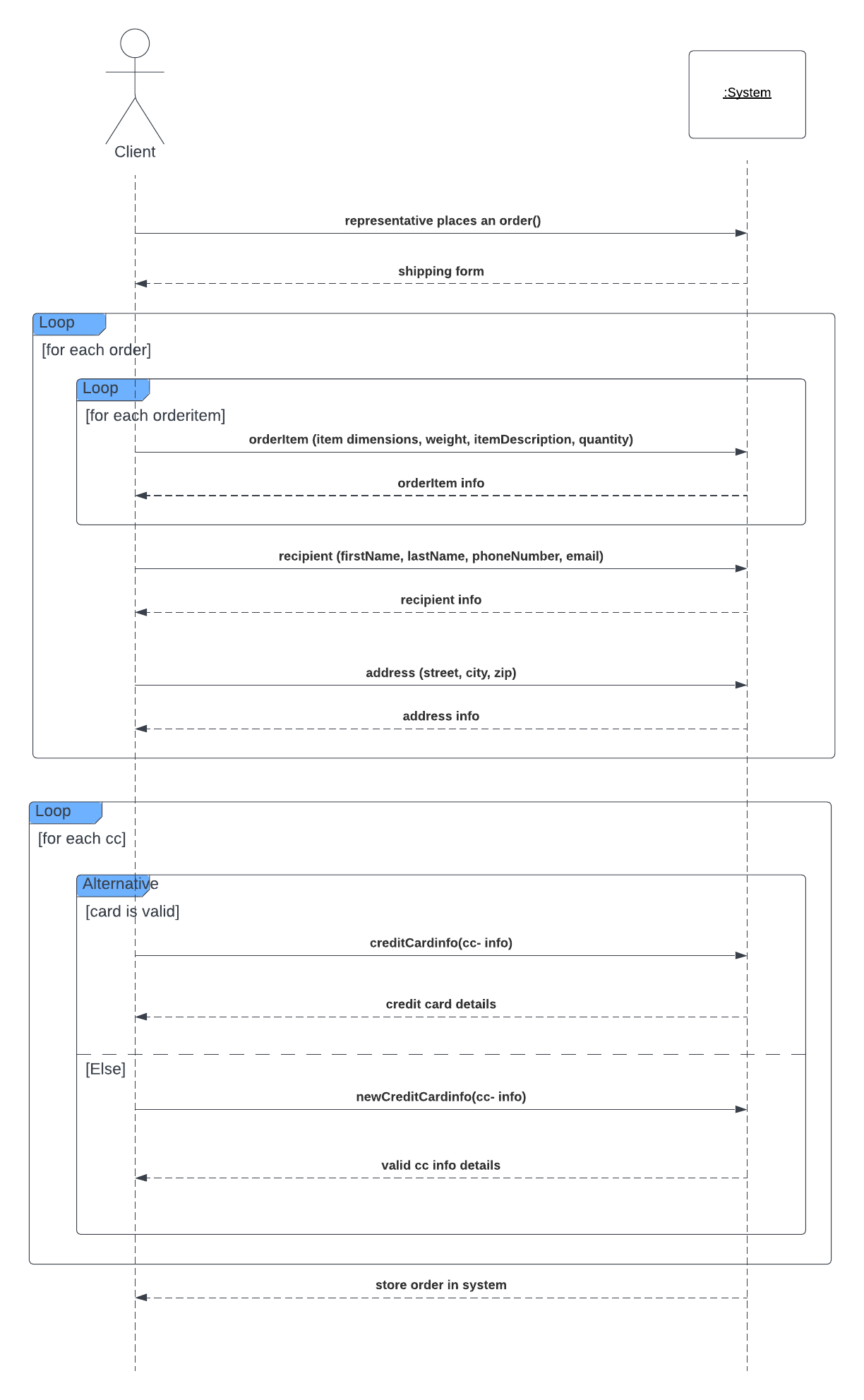
Our domain class model diagram has seen many iterations. Many of the changes were due to improper normalization to begin with. With each iteration, our classes approached and eventually conformed to the third normal form. Other changes, the addition of the Cost class in particular, were made out of utility.

*F. Activity Diagram*

[**](https://app.diagrams.net/?page-id=pMkfZoMgGv83hZ-eCbkn&scale=auto#G1X0oAWFn6gPecUDBcydWB2wdjB84o_7Te)

This is our activity diagram for the “receive shipping request” use case. The process starts when a representative wants to place an order. In response to this, the system will create a new order in the database. For each order item that the representative adds, the system will create a new order item for the order. The system will also prompt the representative to enter the item’s dimensions, quantity, and weight. The representative will add a recipient for the order and the system will create a new recipient object. The system will prompt the representative to enter the information for the recipient and add the information into the new recipient object. This process can be repeated for several orders. Lastly, the system will prompt the representative for payment info and will store the order into the system after validating that the payment info is valid. If the payment information is not valid, then the representative will need to input their payment info again.

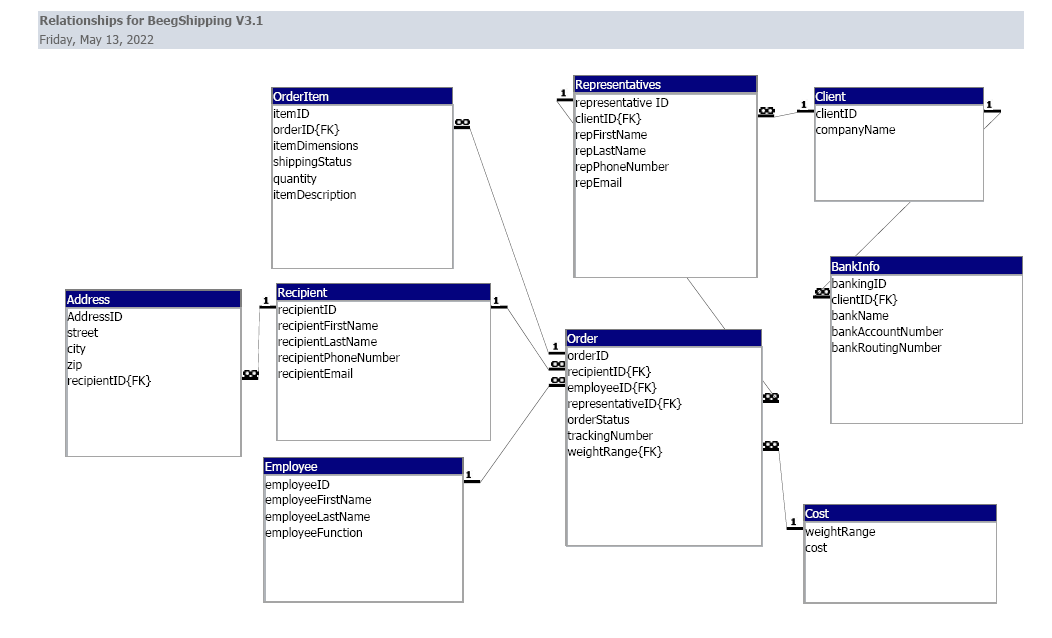
*G. System Sequence Diagram*

[**](https://lucid.app/documents/edit/efaacf7a-3bb5-4a0b-9dcf-e4af943001b0/0?callback=close&name=docs&callback_type=back&v=670&s=612)

This is our system sequence diagram for the receive shipping request use case. This diagram shows a client requesting a shipment of their package(s) and the system returning a shipping form. The client will input all relevant package information inside the loops, which will enter information into the correct class. The recipient will be created and its information entered. Afterwards, the address will be verified. Once complete, the client will enter new credit card information or use an existing credit card in the system (the diagram does not show this part). Once verification of new credit card information is valid, the system will store the information and return an order complete form.

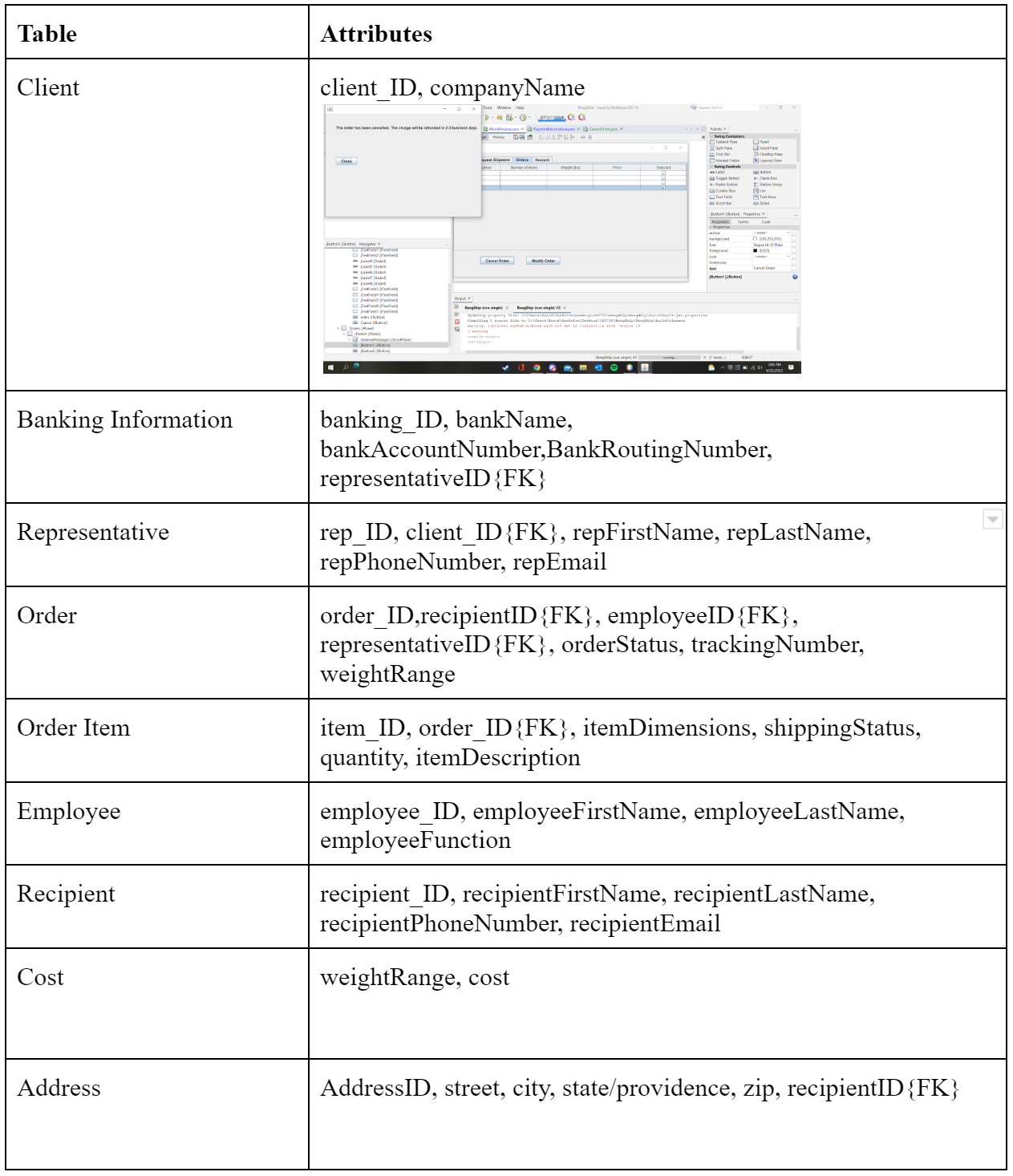
# Design

*A. Class Design*



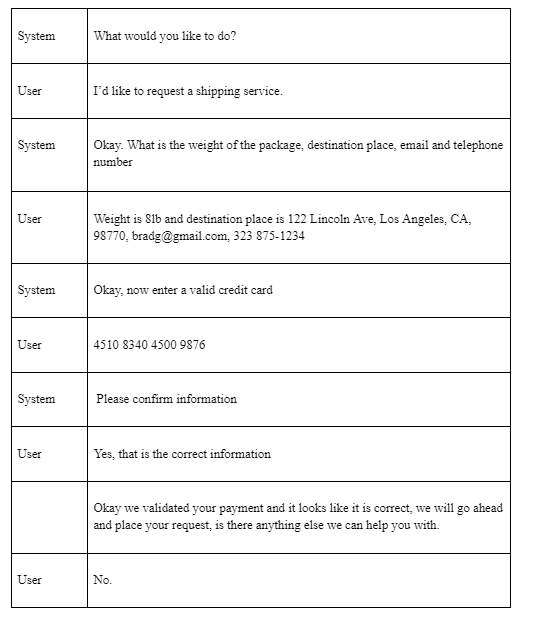
In order to create a database schema, tables for every class were created and then normalized to the third normal form. The classes from the domain model changed because of the normalization. BeegShip system ended up having nine 3N form tables: client, banking information, preparing packages for delivery, Representative order, order item, employee, recipient, cost, and address. The use of foreign and primary keys showed which classes were parent(s) and child(ren), and how they interacted within the system. Also, the relationship between the tables changed from numbers to the crossfit notation.

The Beegship database was created using Microsoft Access. Collaborating with Microsoft Access was easier than MySQL because it was simpler to set up and easier to add records to the tables. In order to connect the database with the system, we used UCanAccess, an open-source Java JDBC driver. This was used because it allows the driver to work on Windows and non-Windows operating systems, allowing us to port our work between operating systems. The only issue with this driver is that it can only be used to connect to MS Access databases. Since we do not need an online portal for our system as intended, MS Access is appropriate and the driver allows easier access than other drivers that are available for other database systems.

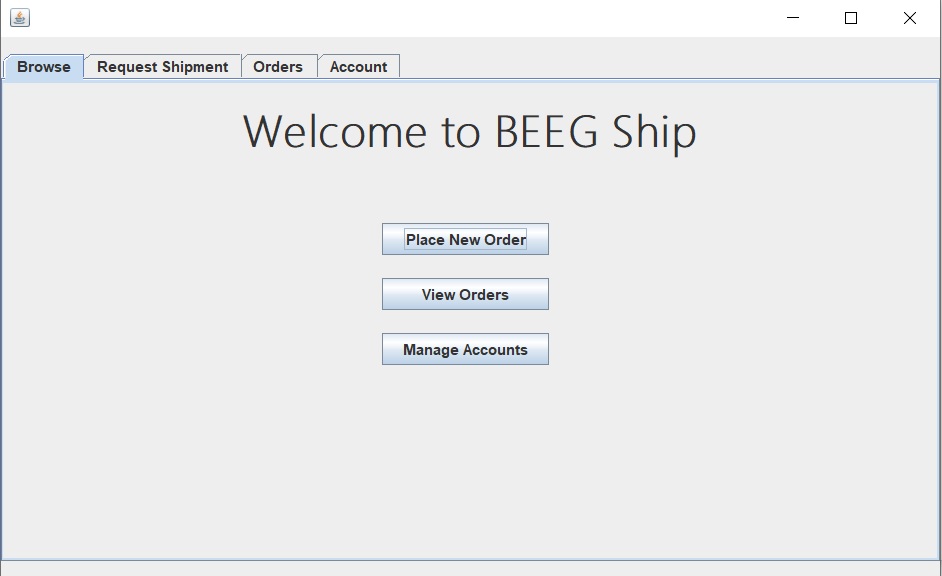


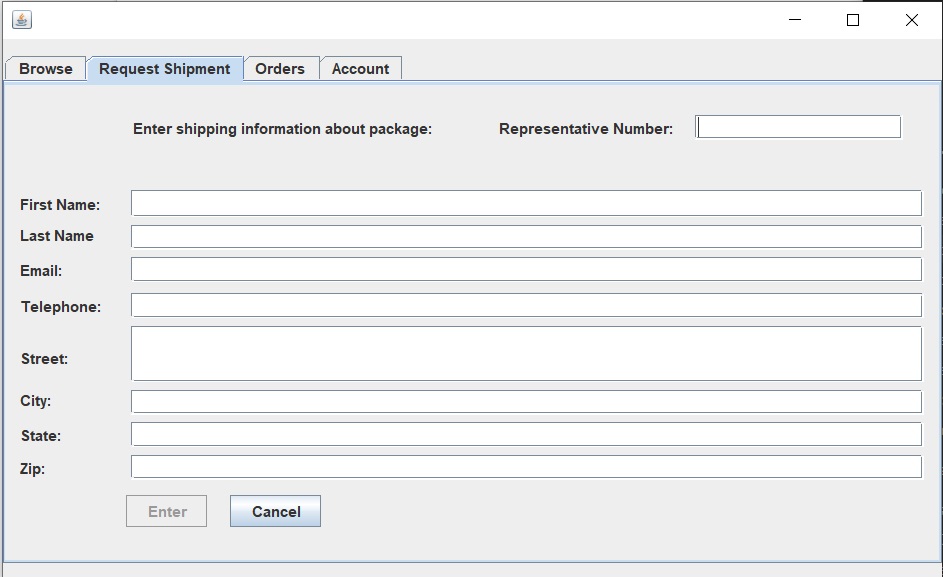
*B. Relationship Tables*

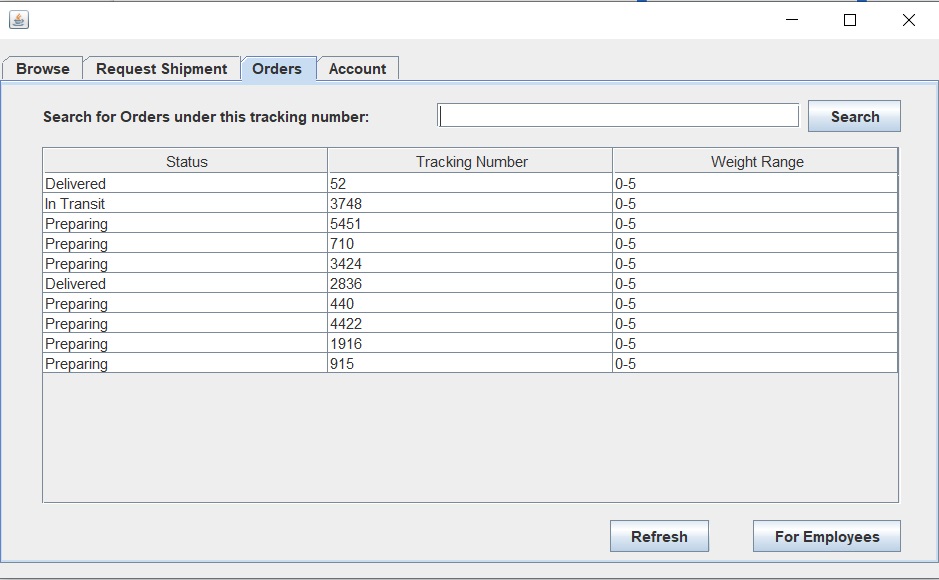
After creating and normalizing the database we brainstormed ideas as to how messages would be exchanged between a potential client and BeegShip system. We developed different scenarios and made dialogs and storyboards. The following chart shows the dialog between a customer requesting a shipping request:



The story boards helped understand how to build and implement our Graphical User Interface (GUI). The conversation between the user and the system identifies which scenes will collaborate together. In the C section, we will display visuals of the graphical interfaces. Using the same example of requesting a package, interfaces will collaborate in the following manner: The first window allows a client to enter information for a new shipping request. Then, the second window will take the user to specify the package information and dimensions. Lastly, the user will get redirected to input a valid card to process the payment. This will eventually store the order in the database with client’s information.

*C. Graphical User Interface (GUI)*





The graphical user interface for BeegShip consisted of several scenes. Some of the tools that were used to create the GUI were JavaFX and Java swing. We decided to use these tools because we were using Java as our main language to create Beegship. The interfaces were easier to connect with the system and the database. The logic around scenes was to help clients easily navigate around the system. The initial scene contains a menu for placing new orders, viewing orders or managing an account. In addition, there were tabs that helped browse and interact with other options BeegShip provides.

The GUI allows clients to manage their orders and check their status. Whenever a client placed an order, he or she would be prompted to fill a form with the appropriate information about the package. All orders were saved in our database, so in order to display the data, we used queries such as “Select, Insert, Update” followed by the tables names. The data displayed in the GUI helped check if data was being sent over correctly and to the appropriate tables.

# Findings and Suggestions

## *A. Findings*

We found that the implementation of our project was the most difficult compared to other steps. This is because there were many instances where our code was not working and we needed to bugfix. Another reason for this was because we made errors during our analysis phase that needed to be fixed during development.

## *B. Suggestions for the Future*

Due to time constraints we didn’t have enough time to implement a lot of the features that we wanted into our project. In the future, we think that it would be better to implement our database in a way that it could be interacted with online. As it currently stands, our program is meant to run with an offline database. This could be done using cloud services like Amazon Web Services. However, our program would need to be updated to work with the online database system.

There are also other subsystems that we could create which were outside the scope of this project. Such as, an employee subsystem to allow logins to our application. We could also improve functionality and readability of our graphical user interface.

# Conclusion

## *A. Challenges*

There were several challenges that we encountered during the creation of BeegShip. Possibly the most significant issue that we faced was that most of us had little to no experience with the software and/or technology that we used during the creation of our project. Thus, we needed to quickly learn how to use new software and languages during development.

Another challenge for us was during the implementation phase of our project. We needed to make many changes to our project since there were normalization issues and other adjustments that we thought were necessary. The difficulty occurred when we needed to update the skeleton code for our classes, the database tables, graphical user interface, and implementation code all at the same time; while keeping everything in working order.

We also had difficulties deciding on what database management system to use. We had initially decided to use MySQL for our database. However, we had trouble sharing the database over Github because of the way that MySQL stores the database. We decided that it would be better if we use Microsoft Access for our database management system. There were some issues we had with using Microsoft Access for the creation of our database schema, but they were solved by just creating it manually. Since Microsoft Access stores the database in a single file, we could simply push updates to Github so that we could all work with the same database without having to recreate anything or use other software to make an online database.

## *B. What We Learned*

We have also learned to make a variety of diagrams to help with the analysis of a system. We used the brainstorming technique to create use cases, use case description diagrams, and fully-developed use case description diagrams. We also created activity diagrams and system sequence diagrams to detail how the user would interact with the system.

We learned how to develop relational databases and implement them in a way that meets our business’ needs. We learned how to use SQL queries for input and output in order to manage our database tables. We learned of the strategies used that normalize our database tables in order to create cleaner implementation and prevent unneeded redundancy in the database.

With the information we have learned, we could apply the same process to other problems throughout the world, like creating a bookstore database, a car dealership subsystem, keeping an inventory count of store products, and many other relevant areas of interest.

## *C. Recap*

Overall, we were able to create a working implementation of the shipping subsystem called BeegShip. Due to time constraints and lack of extensive knowledge, we decided to limit the scope of our project to a size that we could manage. However, given more time, we would be able to make a more extensive system with more features.

##### References

1. “The Java® Language Specification,” docs.oracle.com. https://docs.oracle.com/javase/specs/jls/se17/html/index.html (accessed May 13, 2022).
2. “Database Software and Applications | Microsoft Access,” www.microsoft.com. <https://www.microsoft.com/en-us/microsoft-365/access>
3. “Trail: Creating a GUI With Swing (The JavaTM Tutorials),” docs.oracle.com. <https://docs.oracle.com/javase/tutorial/uiswing/index.html>
4. “MySQL :: Why MySQL?,” www.mysql.com. <https://www.mysql.com/why-mysql/>
5. “SQL Tutorial,” W3schools.com, 2019. <https://www.w3schools.com/sql/>
6. “Git Tutorial,” www.w3schools.com. https://www.w3schools.com/git/default.asp (accessed May 13, 2022).