Final Report

Hexagon

Delivery System Project:

Team Members:

Guillaume Lachapelle

ID: 40203325 Role: Team leader / Developer

Ann-Marie Czuboka
ID: 40209452 Role: Developer

Isabelle Czuboka
ID: 40209525 Role: Developer

Nicholas Piperni
ID: 40207764 Role: Developer

Karina Sanchez-Duran
ID: 40189860 Role: Developer

Oliver Vilney
ID: 40214948 Role: Developer

December 4th 2023

Table of Contents

- 1. Introduction

The purpose of the delivery system platform is to provide a user-friendly and effective alternative to current delivery services for individuals and companies. With the rise of e-commerce sales and the subsequent increased demand for delivery services, our website provides companies with a viable solution to their delivery problem needs by taking care of all the logistics of the delivery. Otherwise, these companies would have to dedicate time and additional resources to handling delivery requests for their customers. Our website also provides individual clients with a viable solution to their delivery service needs.

Our delivery system platform provides national deliveries in Canada. With our website, a client can request a delivery, request (and save) a quotation, track their delivery, write a review and see the reviews of other clients in the "Reviews" section of the website, proceed to payment and request customer service by speaking to a customer service representative in a chat room in real time. The following report explains, in detail, the analysis and design of the system, the functionality of our delivery service platform, as well as the evolution of the entire project.

- 2. Combined Sprint Summaries

Sprint 1 - Problem Domain Study and Initial Diagrams

In Sprint 1, we provided the problem statement and produced a context diagram and domain model. In our problem statement, we explain how our delivery service platform provides an effective solution to companies and individuals who wish to make a delivery and find other means to be more expensive and/or complex in terms of logistics. The context diagram provides a visual depiction of the interactions between our delivery service platform and external systems and actors. Additionally, the context-diagram defines the scope of the project. The domain model illustrates the domain concepts of the system and how they interact with each other. The domain model also acts as the base layer to the creation of the class diagram later on. Both the context-diagram and the domain model are key to convey a simplified explanation of the system to all stakeholders.

Sprint 2 - Use Case Diagrams and Sequence diagrams

The goal of Sprint 2 was to showcase the different use cases of the system, based on its main functionalities. We first started by looking at the six main features and assigned one feature per person. From there, we built the use case diagram that represents all use cases for the six main features, and then every person designed the sequence diagrams for the specific use case to which they were assigned. We then gathered the whole team to review all the diagrams and made some small tweaks so that they fit the system better. From there, we were ready to continue to Sprint 3. The main goal of this sprint was to think like a user and think of how we could provide a good user experience while designing a good, efficient, and easy-to-use system. When combining those ideas, it was a lot easier to design the diagrams and understand the scope and goal of the system.

Sprint 3 - Implementation and Design Class Diagram

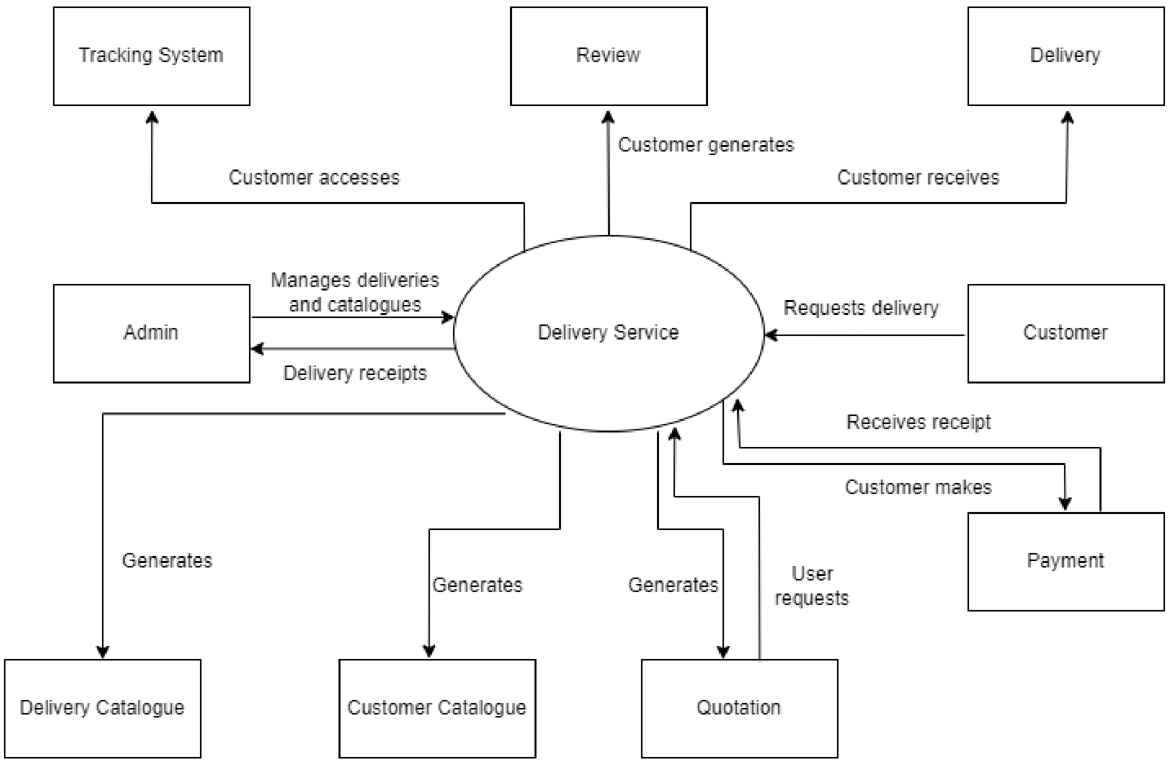
The goal of Sprint 3 was to finish the diagram designs and decide on the design patterns and architecture to be used. This sprint was crucial for good development. The class diagram was built from the context diagram, domain model, use case diagrams and sequence diagrams. While discussing the architecture and design patterns to be used, we made sure to refine the class diagram to include patterns such as controller classes called “services” that would ensure communication between the UI and the Firebase backend. The class diagram had to be refactored quite a few times to make sure that it accurately and precisely shows how the different classes will interact and what patterns should be used during the development. This sprint also focused on implementing the UI for at least 4 main features. We decided to implement the UI for all 6 main features, as well as some extra features such as a search bar, and a Reviews page where a client can see everyone’s review on their delivery, as well as a bit of information about that delivery. From there, we made sure that the way we implemented the UI would make it easy to add backend functionality during Sprint 4.

Sprint 4 - Refinement and Finalization

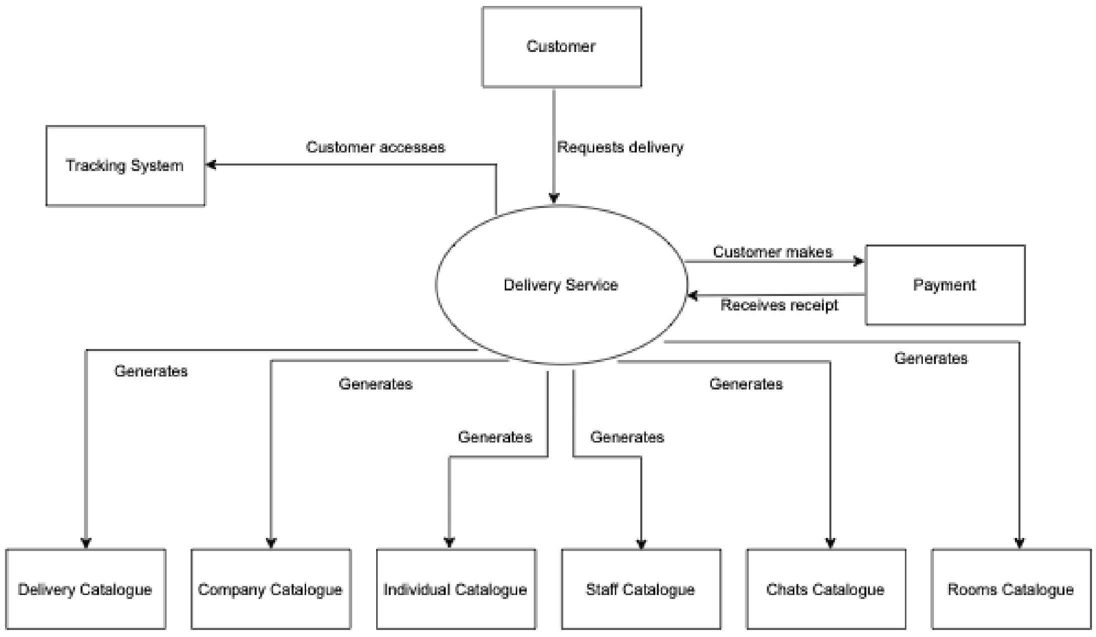
The goal of Sprint 4 was to refine the diagrams, implement the backend functionalities of the system, code refactoring, and conclusion of the project, including testing. We assigned backend functionality to everyone’s use case (that was assigned during Sprint 2). We made sure to refine the diagrams as we implemented the code, especially the class diagram, to make sure to update the team on possible interface changes, or communication changes, such as adding services for code reusability, as well as higher cohesion in the classes. After updating those diagrams, we made sure to refactor the code as much as possible, such as putting repeated code inside a service, for fewer code smells, repetition, and higher cohesion. Once all that was over, we made sure to modify our code to make sure the tests passed. We spent quite some time on that, making sure that every test passes and that every component is tested. In the future, we would like to do that as we develop the code so that the testing isn’t left for the end of the project. This would not only simplify the testing part at the end of the project but could also help us avoid bugs during the development process. Finally, we were able to add another extra feature to the system, which is a profile page that displays the user’s information, and implement the backend for a feature giving the client a way to save a quotation and order the delivery later.

- 3. Diagrams Evolution

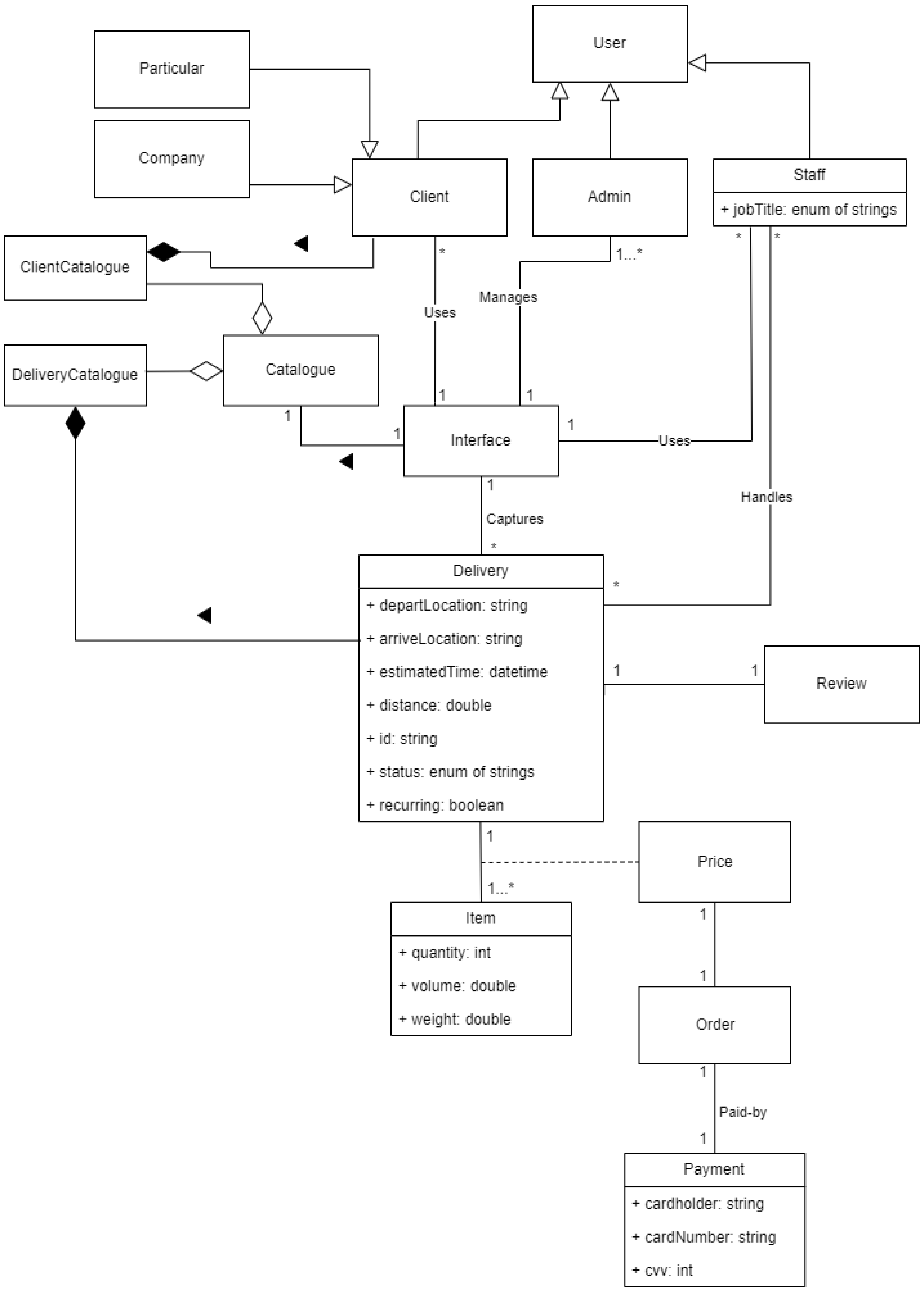
Old Context Diagram:



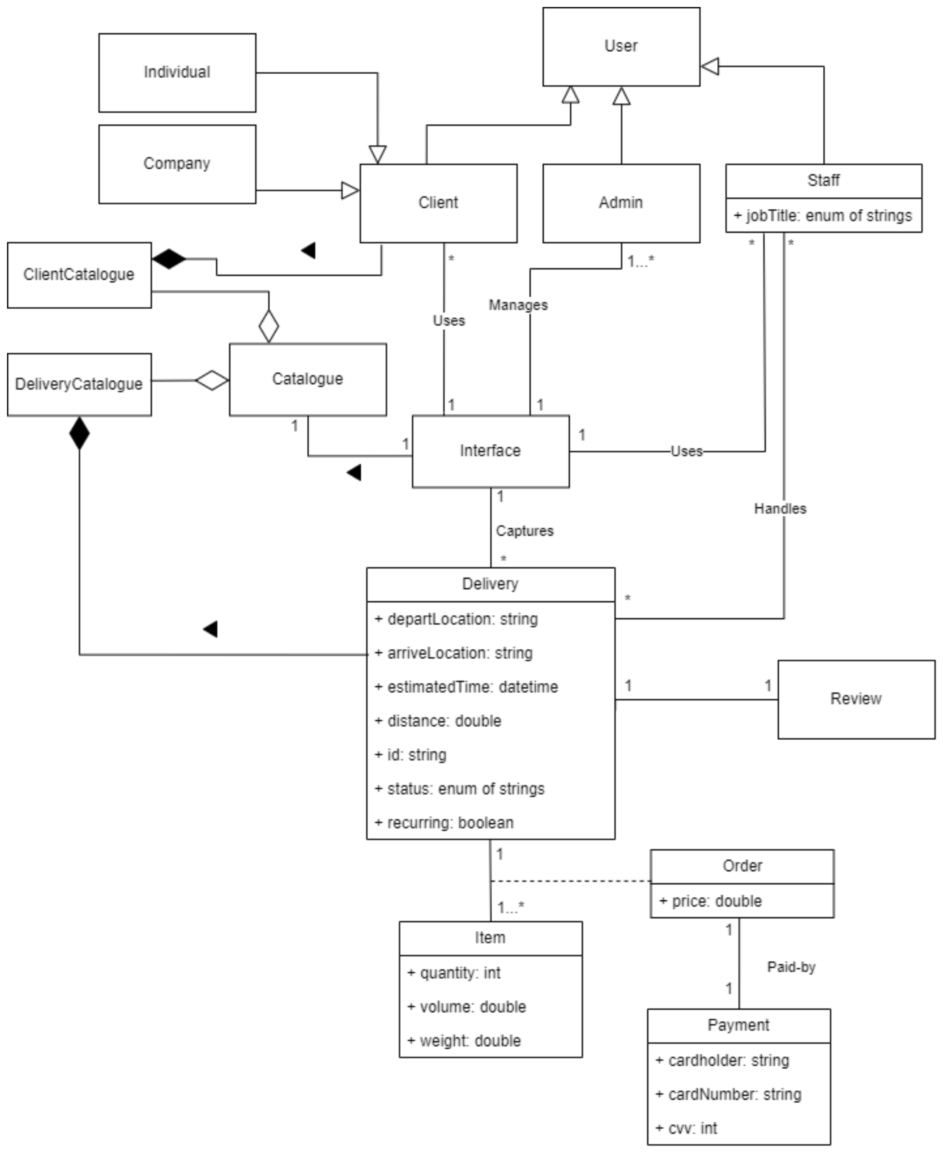
New Context Diagram:



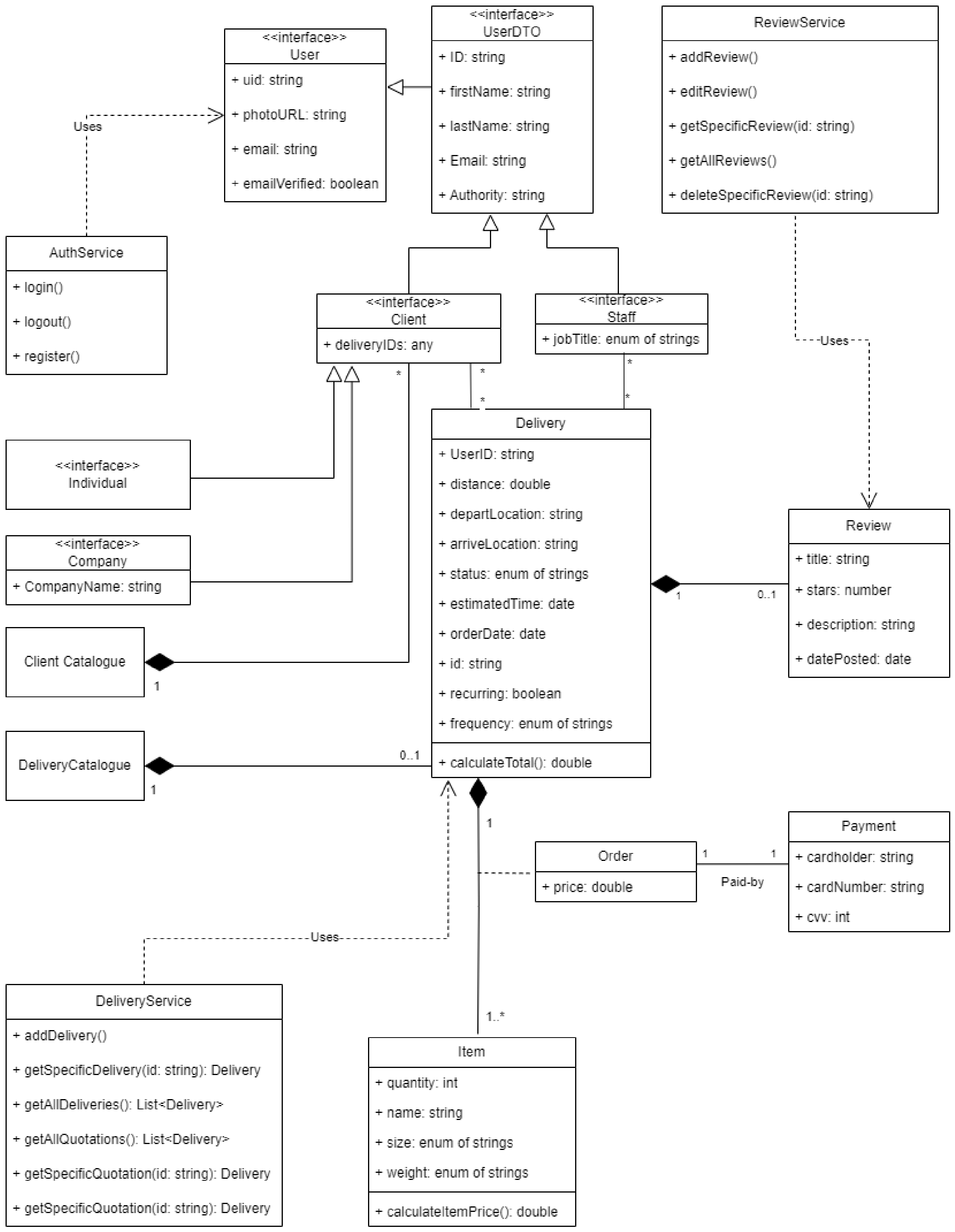
Old Domain Diagram:



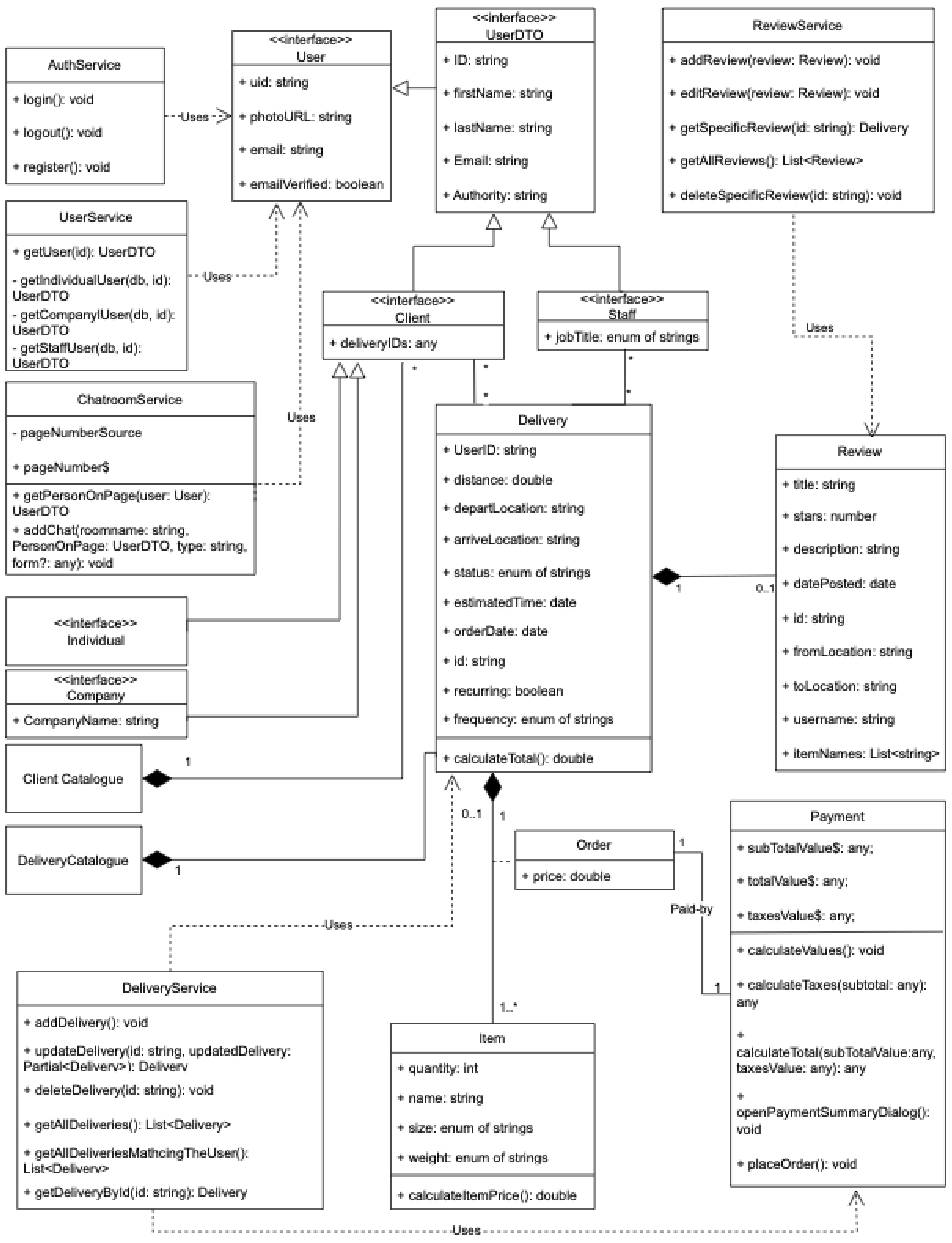
New Domain Diagram:



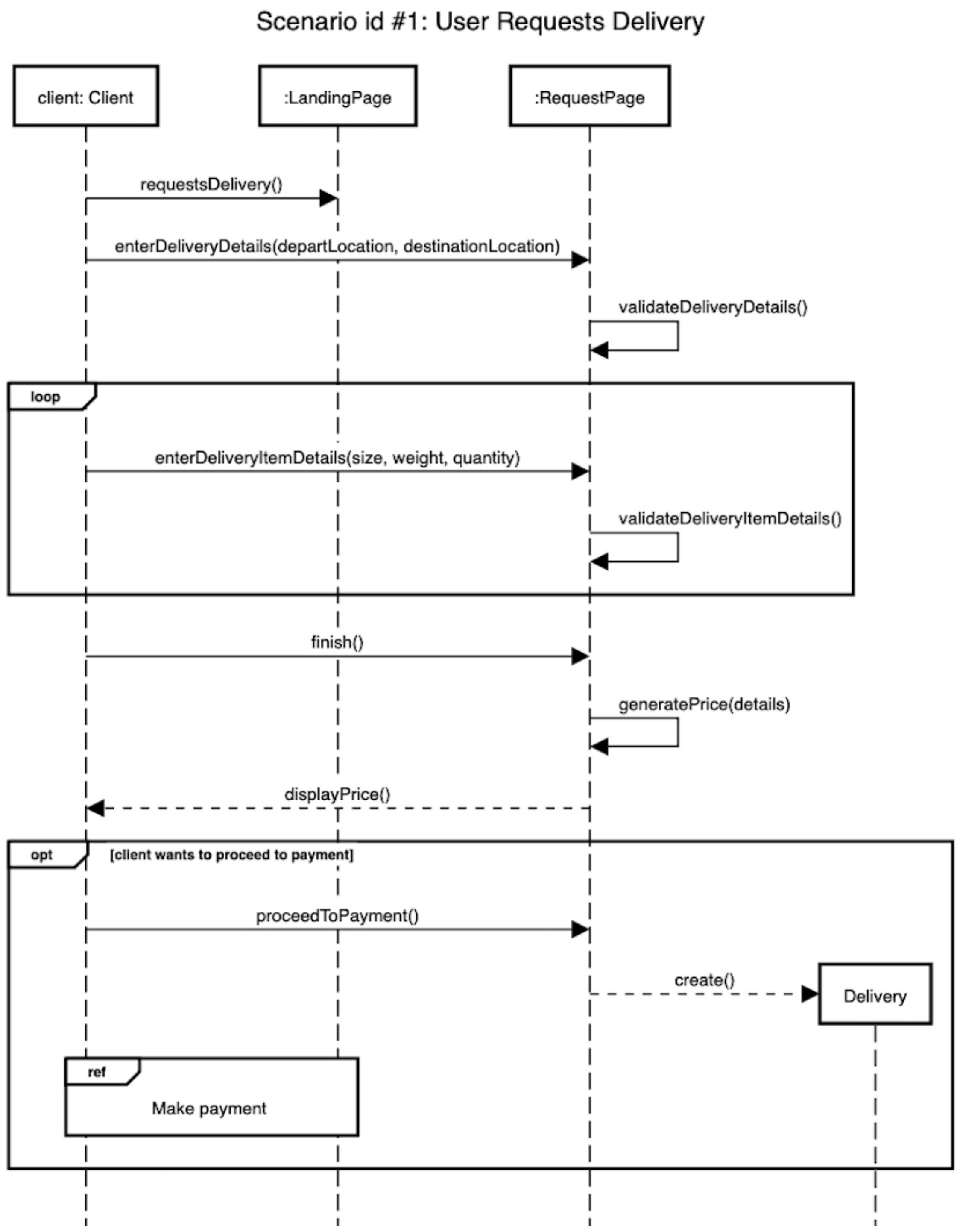
Old Class Diagram:



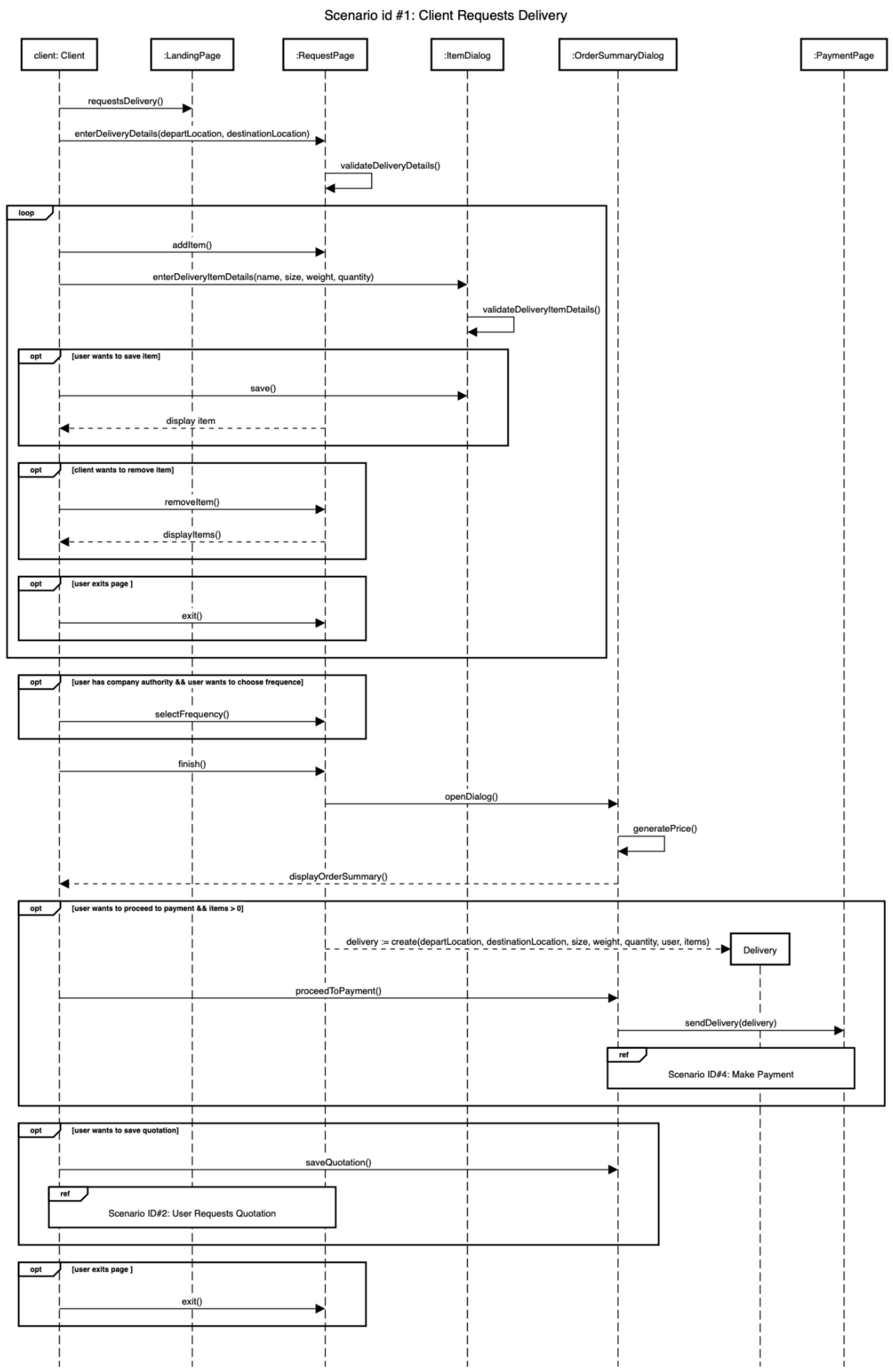
New Class Diagram:



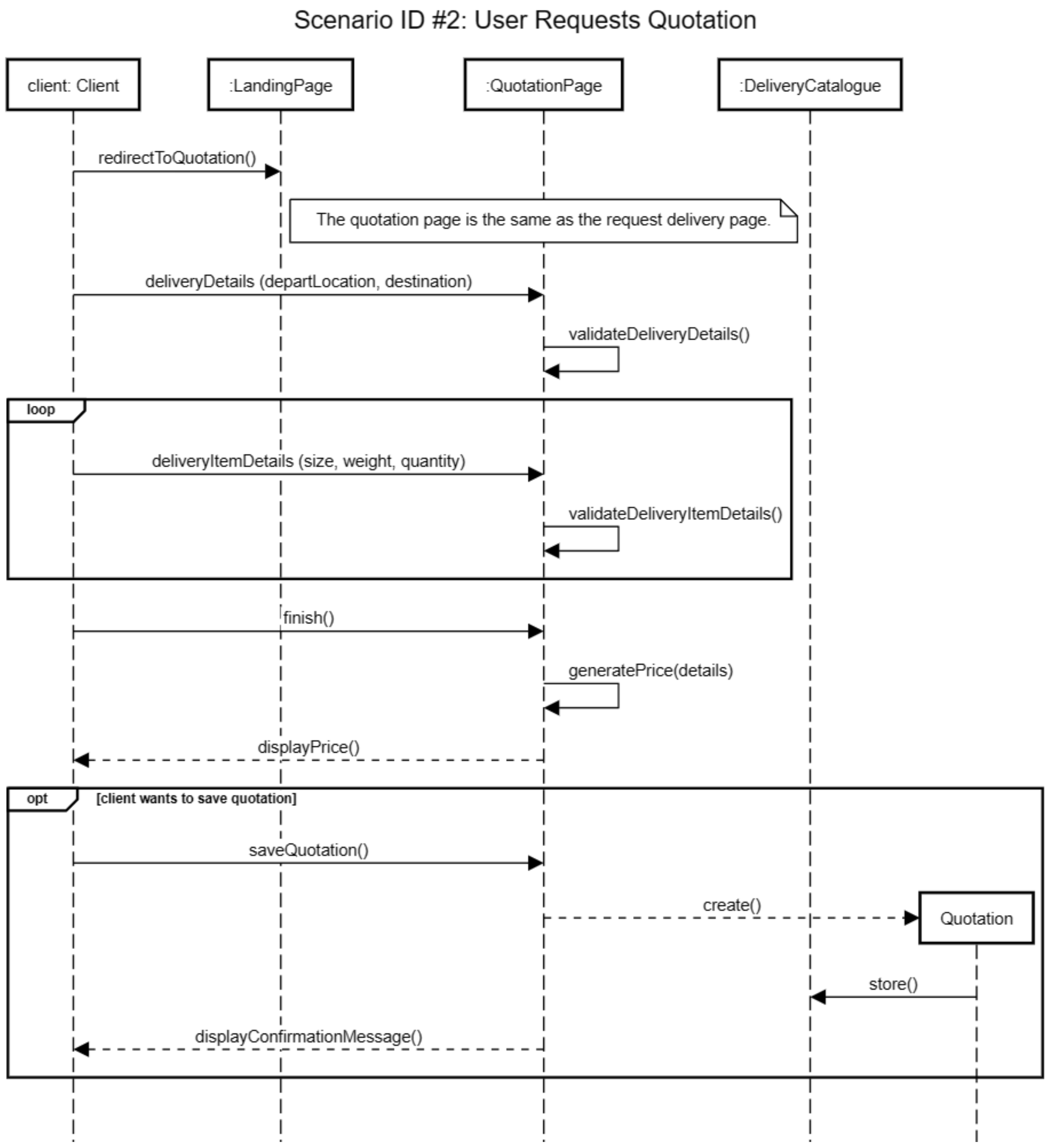
Old System Sequence Diagram #1: Client Requests Delivery


Scenario id #1: User Requests Delivery
client: Client
:LandingPage
:RequestPage
requestsDelivery()
enterDeliveryDetails(departLocation, destinationLocation)
validateDeliveryDetails()
loop
enterDeliveryItemDetails(size, weight, quantity)
validateDeliveryItemDetails()
finish()
generatePrice(details)
displayPrice()
opt
[client wants to proceed to payment]
proceedToPayment()
create()
Delivery
ref
Make payment

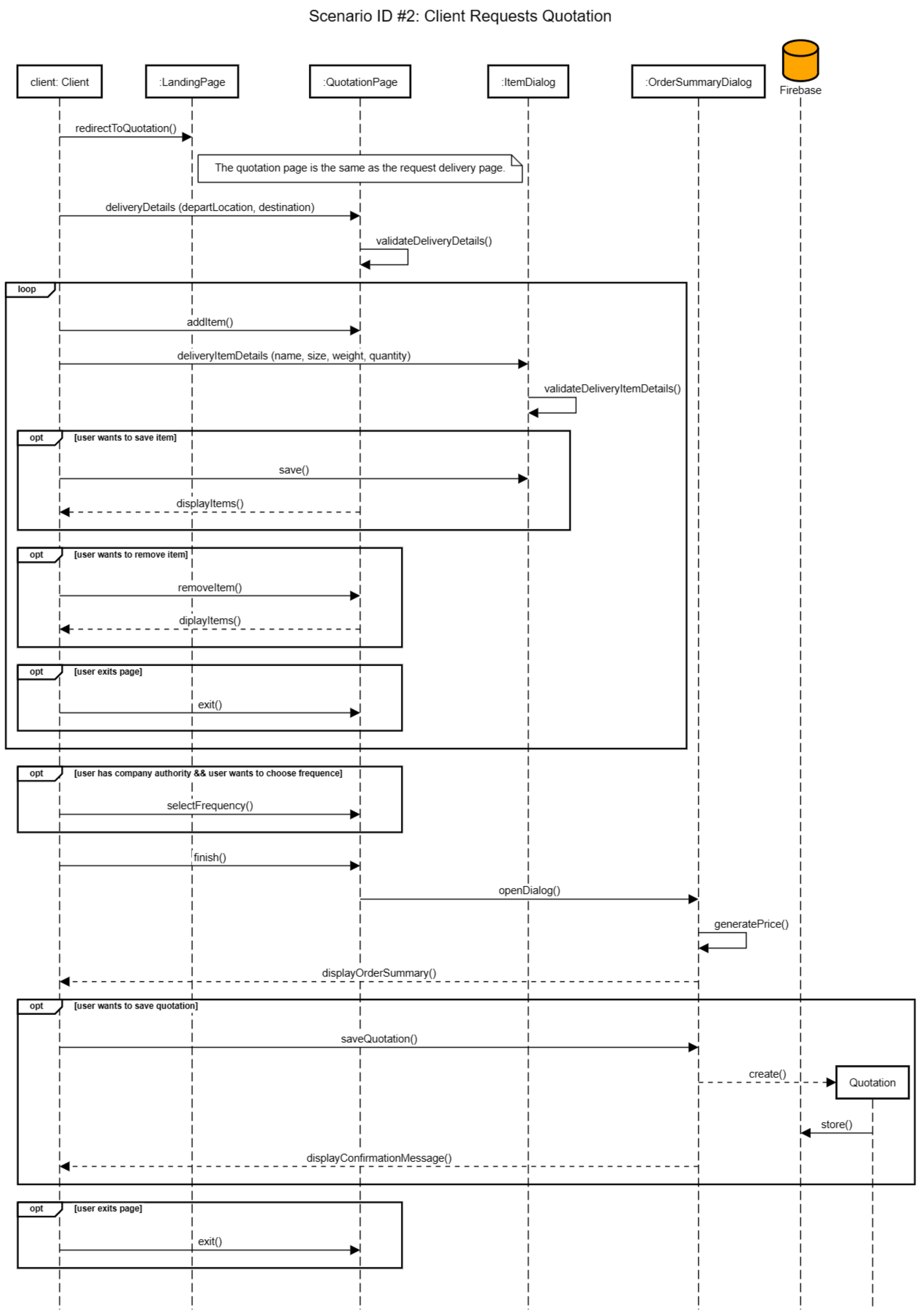
New System Sequence Diagram #1: Client Requests Delivery


Scenario id #1: Client Requests Delivery
client: Client
:LandingPage
:RequestPage
:ItemDialog
:OrderSummaryDialog
:PaymentPage
requestsDelivery()
enterDeliveryDetails(departLocation, destinationLocation)
validateDeliveryDetails()
loop
addItem()
enterDeliveryItemDetails(name, size, weight, quantity)
validateDeliveryItemDetails()
opt
[user wants to save item]
save()
display item
opt
[client wants to remove item]
removeItem()
displayItems()
opt
[user exits page]
exit()
opt
[user has company authority && user wants to choose frequence]
selectFrequency()
finish()
openDialog()
generatePrice()
displayOrderSummary()
opt
[user wants to proceed to payment && items > 0]
delivery := create(departLocation, destinationLocation, size, weight, quantity, user, items)
Delivery
proceedToPayment()
sendDelivery(delivery)
ref
Scenario ID#4: Make Payment
opt
[user wants to save quotation]
saveQuotation()
ref
Scenario ID#2: User Requests Quotation
opt
[user exits page]
exit()

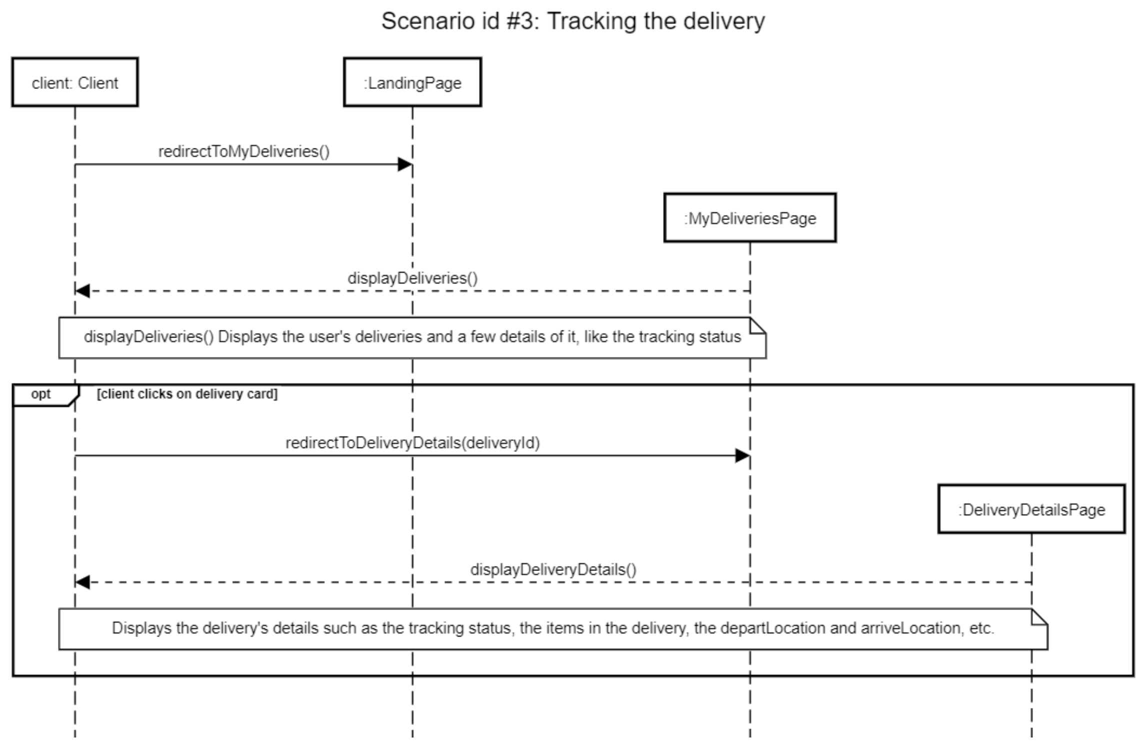
Old System Sequence Diagram #2: Client Requests Quotation


Scenario ID #2: User Requests Quotation
client: Client
:LandingPage
:QuotationPage
:DeliveryCatalogue
redirectToQuotation()
The quotation page is the same as the request delivery page.
deliveryDetails (departLocation, destination)
validateDeliveryDetails()
loop
deliveryItemDetails (size, weight, quantity)
validateDeliveryItemDetails()
finish()
generatePrice(details)
displayPrice()
opt
[client wants to save quotation]
saveQuotation()
create()
Quotation
store()
displayConfirmationMessage()

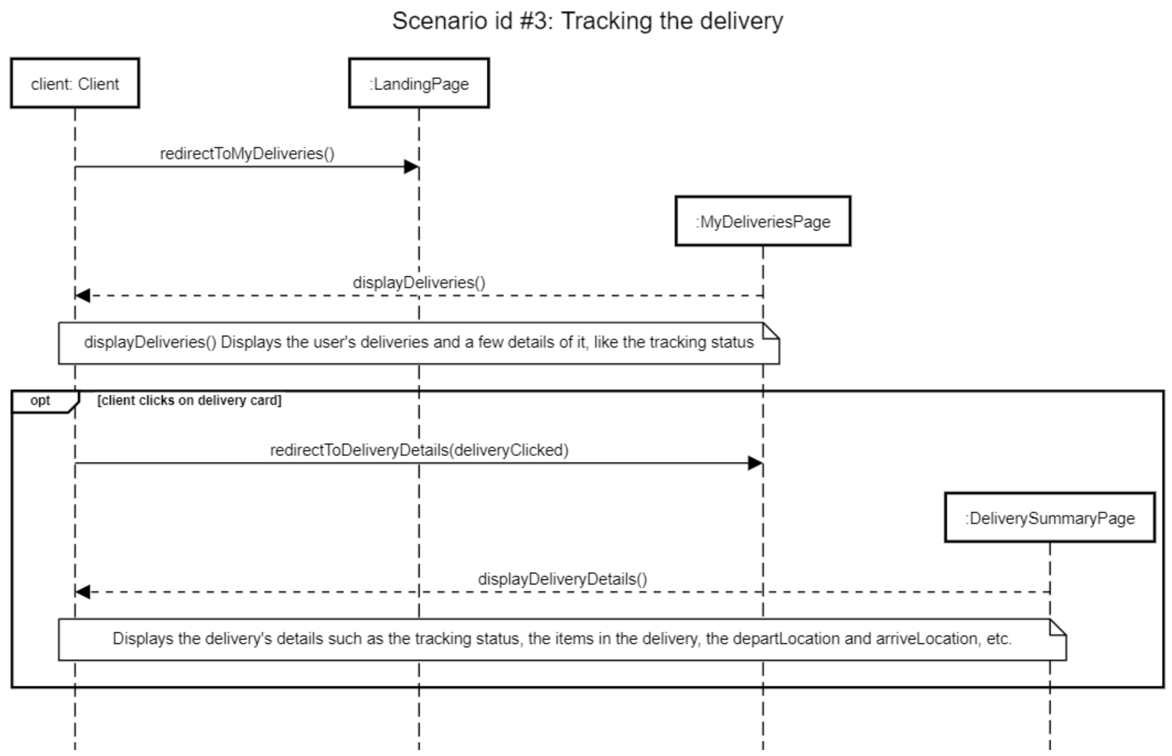
New System Sequence Diagram #2: Client Requests Quotation


Scenario ID #2: Client Requests Quotation
client: Client
:LandingPage
:QuotationPage
:ItemDialog
:OrderSummaryDialog
Firebase
redirectToQuotation()
The quotation page is the same as the request delivery page.
deliveryDetails (departLocation, destination)
validateDeliveryDetails()
loop
addItem()
deliveryItemDetails (name, size, weight, quantity)
validateDeliveryItemDetails()
opt [user wants to save item]
save()
displayItems()
opt [user wants to remove item]
removeItem()
diplayItems()
opt [user exits page]
exit()
opt [user has company authority && user wants to choose frequence]
selectFrequency()
finish()
openDialog()
generatePrice()
displayOrderSummary()
opt [user wants to save quotation]
saveQuotation()
create()
Quotation
store()
displayConfirmationMessage()
opt [user exits page]
exit()

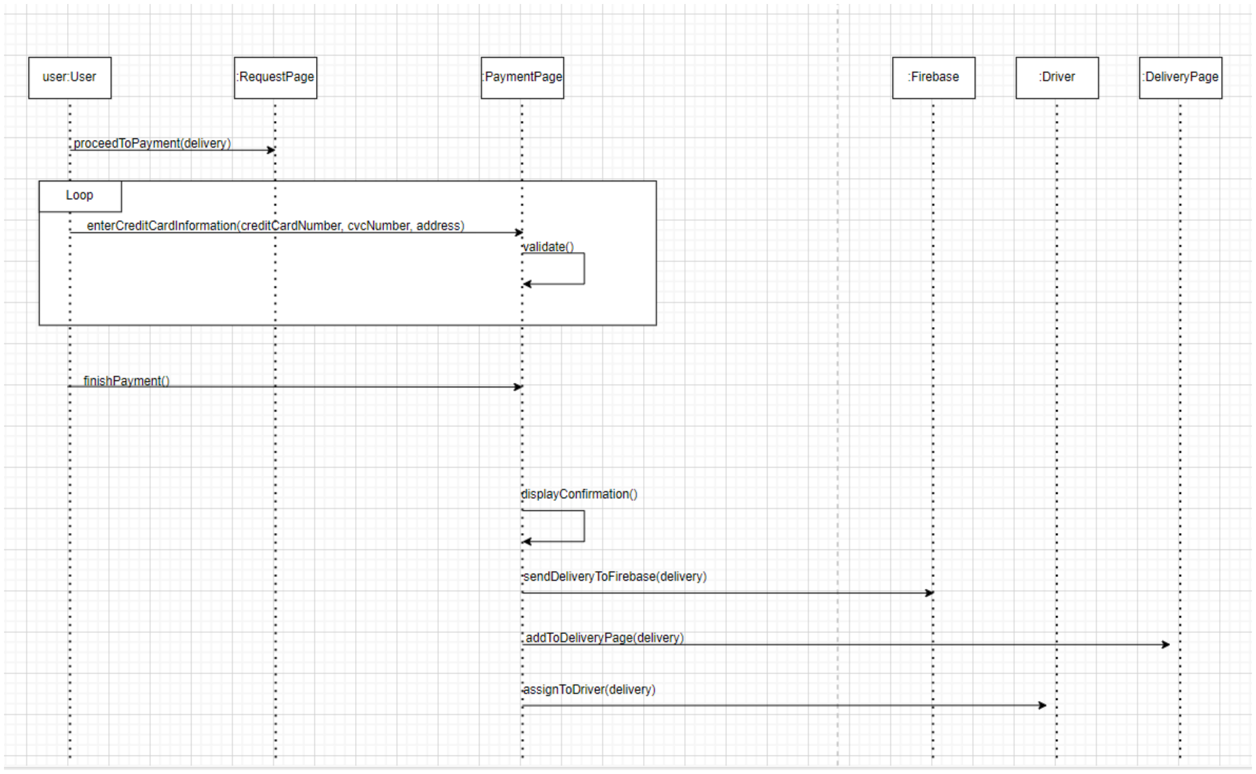
Old System Sequence Diagram #3: Tracking the Delivery


Scenario id #3: Tracking the delivery
client: Client
:LandingPage
redirectToMyDeliveries()
:MyDeliveriesPage
displayDeliveries()
displayDeliveries() Displays the user's deliveries and a few details of it, like the tracking status
opt
[client clicks on delivery card]
redirectToDeliveryDetails(deliveryId)
:DeliveryDetailsPage
displayDeliveryDetails()
Displays the delivery's details such as the tracking status, the items in the delivery, the departLocation and arriveLocation, etc.

New System Sequence Diagram #3: Tracking the Delivery

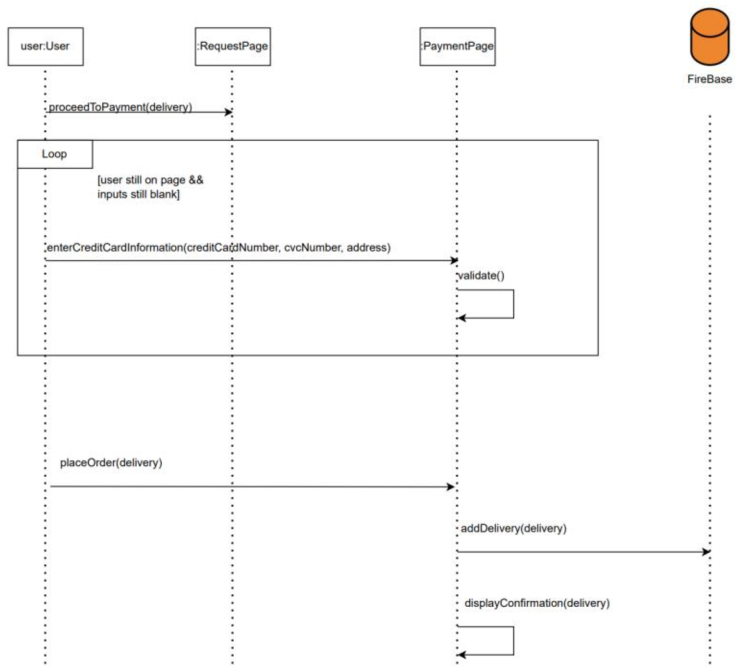

Scenario id #3: Tracking the delivery
client: Client
:LandingPage
redirectToMyDeliveries()
:MyDeliveriesPage
displayDeliveries()
displayDeliveries() Displays the user's deliveries and a few details of it, like the tracking status
opt
[client clicks on delivery card]
redirectToDeliveryDetails(deliveryClicked)
:DeliverySummaryPage
displayDeliveryDetails()
Displays the delivery's details such as the tracking status, the items in the delivery, the departLocation and arriveLocation, etc.

Old System Sequence Diagram #4: Make Payment

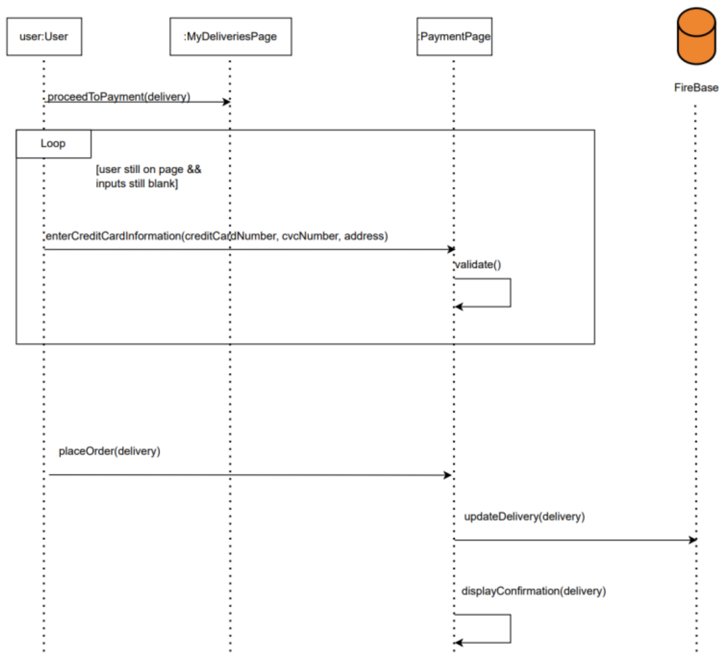

user:User
:RequestPage
:PaymentPage
:Firebase
:Driver
:DeliveryPage
proceedToPayment(delivery)
Loop
enterCreditCardInformation(creditCardNumber, cvcNumber, address)
validate()
finishPayment()
displayConfirmation()
sendDeliveryToFirebase(delivery)
addToDeliveryPage(delivery)
assignToDriver(delivery)

New System Sequence Diagram #4: Make Payment

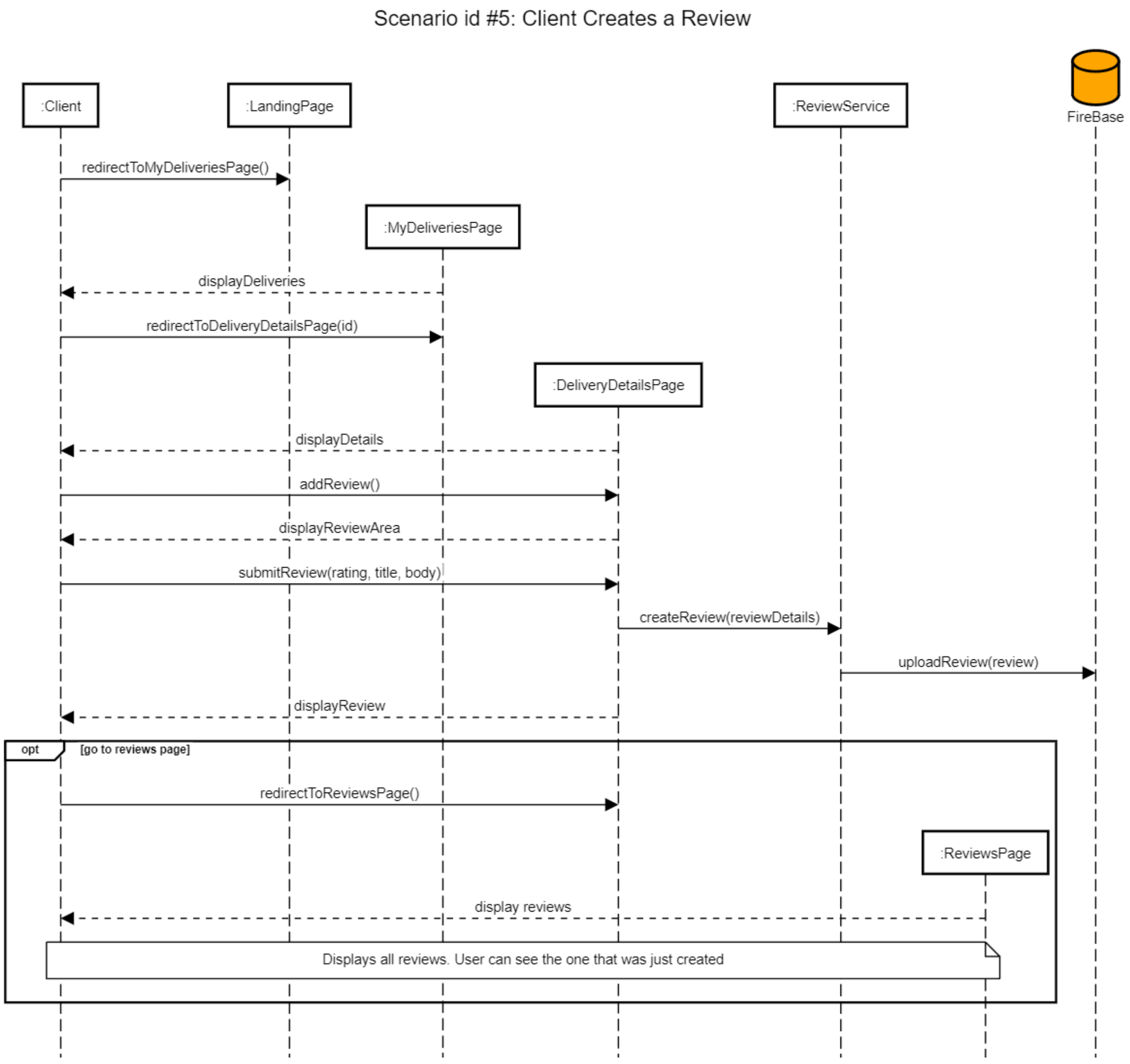
SSD: Make Payment from Request Page


user:User
:RequestPage
:PaymentPage
FireBase
proceedToPayment(delivery)
Loop
[user still on page && inputs still blank]
enterCreditCardInformation(creditCardNumber, cvcNumber, address)
validate()
placeOrder(delivery)
addDelivery(delivery)
displayConfirmation(delivery)

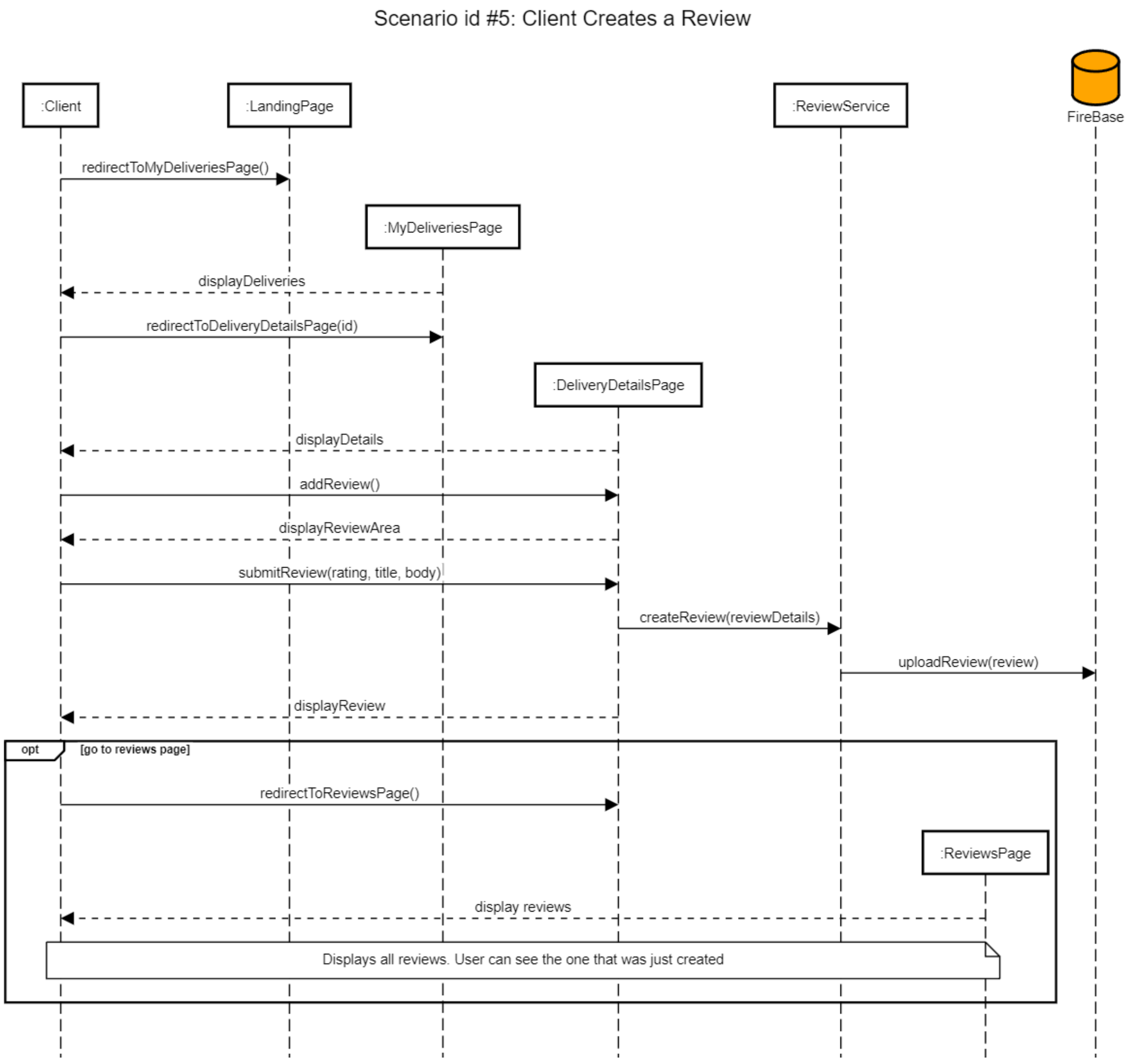
SSD: Make Payment from MyDeliveries


user:User
:MyDeliveriesPage
:PaymentPage
FireBase
proceedToPayment(delivery)
Loop
[user still on page && inputs still blank]
enterCreditCardInformation(creditCardNumber, cvcNumber, address)
validate()
placeOrder(delivery)
updateDelivery(delivery)
displayConfirmation(delivery)

Old System Sequence Diagram #5: Client Creates a Review

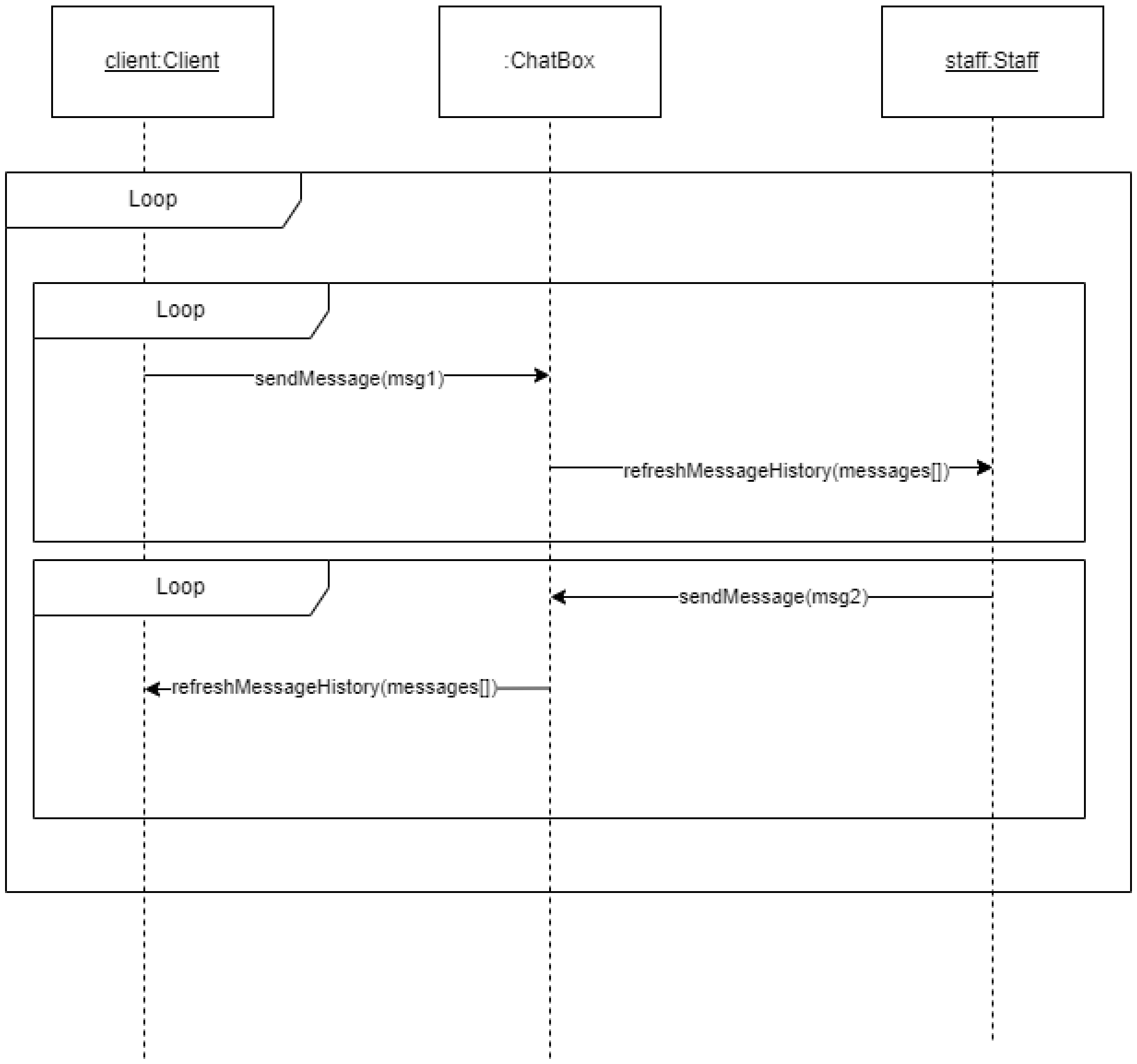


New System Sequence Diagram #5: Client Creates a Review (no change)

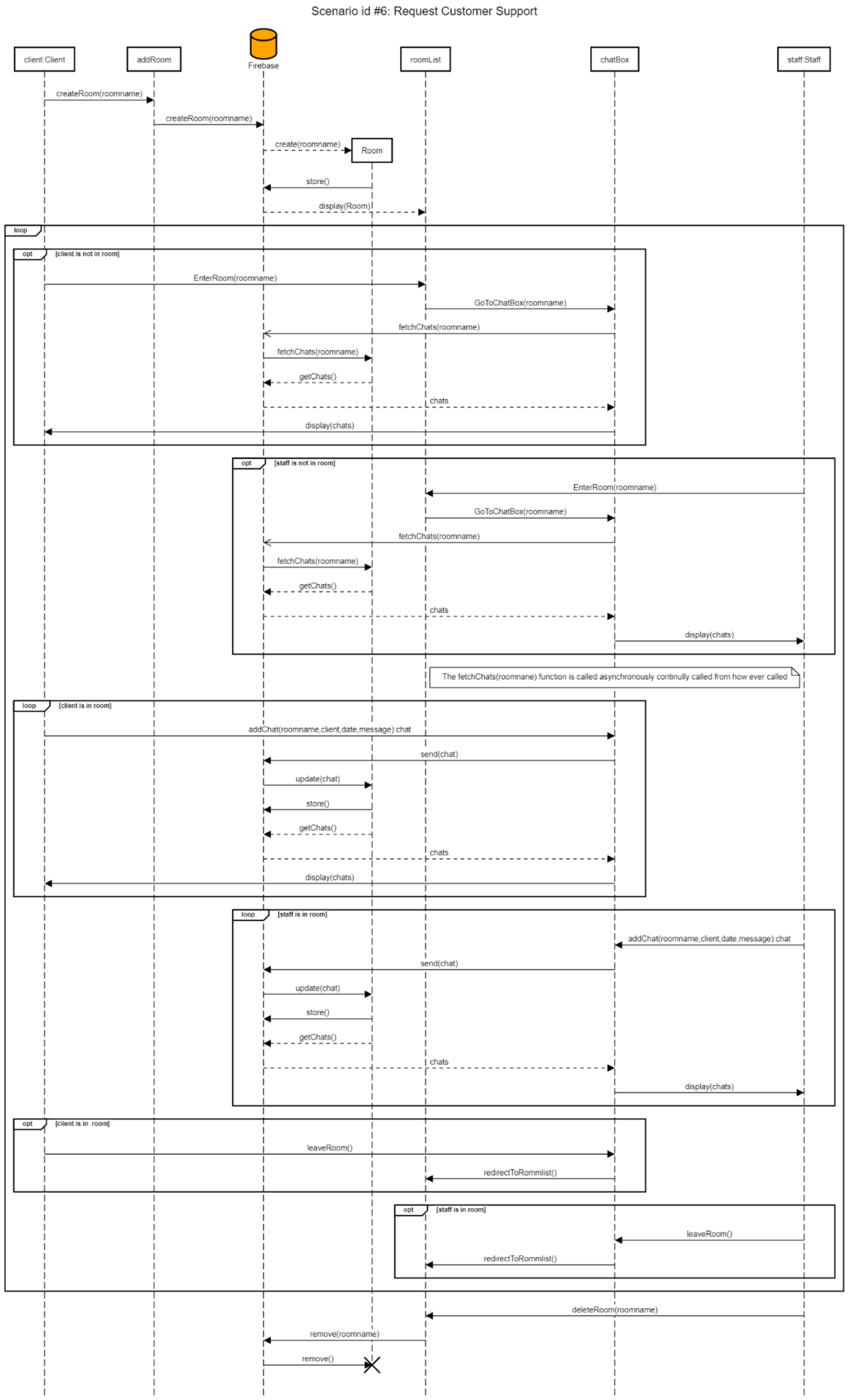


Old System Sequence Diagram #6: Client Requests Customer Support

Scenario id #6: Request Customer Support



New System Sequence Diagram #6: Client Requests Customer Support



- 4. Code Refactoring and Implementation

Significant code refactoring activities were executed during sprint 3 and sprint 4. Mainly, these activities involved the creation of service classes. Some of these classes act as a controller class to bridge between the GUI and domain while delegating the main system logic to the Firebase API. The other classes are more focused on reusability and maintainability by encapsulating functionality within services making the same logic reusable across different components. The services were effectively abstracted from intricate details, promoting a more modular and maintainable structure.

Furthermore, throughout sprints 3 and 4, modifications to some interfaces were made. This included adding attributes that were necessary to have that only arose during implementation. For example, the *Review* interface had new attributes during Sprint 4, since the attributes fromLocation, toLocation, username, itemNames and id made the fetching of information much simpler and faster than having to fetch this information from the delivery object.

- 5. Discussion on Design Patterns

***Delivery* Responsibilities:**

- Acts as the Information Expert for Item, computing the total delivery cost by summing item prices.
- Functions as a Creator for *Item* and *Review*, reflecting their whole-part relationship by managing their instantiation within the *Delivery* scope.

Polymorphism Implementation:

- *UserDTO* and *Client* classes embody polymorphism principles. Both *Individual* and *Company* extend from *Client*, differing in attributes like CompanyName. Additionally, the inheritance hierarchy includes *Client* and *Staff*, both extending from *UserDTO* and differing in attributes such as deliveryIDs and jobTitle. The interfaces were primarily employed to customize content for distinct user types.

Protected Variance:

- All the interfaces that we have related to a user reflect protected variance. *UserDTO* is the wrapper interface, while *Individual*, *Company* and *Staff* are all protected variations of said wrapper.

Controller Classes (DeliveryService and ReviewService):

- Serve as intermediaries between the GUI and domain, orchestrating all database-modifying interactions.

- Ensure focused operations on specific entities, such as the DeliveryService for the delivery entity. This approach fosters high cohesion within classes and can be seen as an instance of pure fabrication due to its purposeful design and segregation of concerns.

- 6. Refinement and Discussion (Sprint 4)

Regarding the context diagram, it was significantly modified from the first original draft of Sprint 1. The changes included removing entities found inside the system represented as external entities such as *Review, Delivery, Admin, Quotation.* The last step was to specify the different catalogs created by the Delivery Service depicted in the image above. These changes clarified the external entities that the system interacts with and delimited the scope of the system.

Regarding the domain model, some real-world entities were removed, such as the *Price* since we discovered that this will instead be an attribute depicted in the class *Order* inside the class diagram instead. Other than that change, we modified the name of the *Particular* entity with *Individual* since it better represented the domain entity and portrayed a clearer meaning to stakeholders.

Regarding the system sequence diagrams, significant improvements involve the introduction of additional functionalities through the inclusion of 'opt' options. These options enable users to accomplish extra functionalities. Many of these newly added options refer back to specific sequence diagrams. The options were added to the SSDs to align with the functionalities that were implemented throughout the final sprint. To add on, new entities have been added to the system diagram namely dialog components. This was done to showcase their role in gathering user input. The new component additions contribute to a more comprehensive and informative user interface. Lastly, we added the interactions with Firebase to depict more clearly the external interactions of the system.

Regarding the class diagram, we added missing attributes in the Delivery class, the Review, the Item class and the Payment class. We also updated methods from DeliveryService and added missing methods in PaymentService. We also updated the return type of methods in ReviewService. We changed the association between Delivery and Review to be a composite relationship where Delivery is the “whole” part of the relationship. To reduce coupling, we added services to the class diagram as well. Finally, we removed the Admin user from the class diagram as it was not needed.

- 7. Project Goal and Objectives

The original goals and objectives of the delivery system project are rooted in addressing the escalating demand and significance of e-commerce in today's market. With the increase in online shopping, directly affecting increased delivery requirements, the fundamental objective of the delivery application revolves around meeting the evolving needs of this expanding market. The primary goal is to develop a user-friendly and efficient system that aligns with the rising demand for better delivery services in the e-commerce sector.

The key objectives revolve around providing a seamless and user-friendly interface that enhances customer experience, caters to diverse delivery needs, and ensures a consistent provision of timely and exceptional service feedback. This encompasses creating an intuitive website capable of catering to individual and company users, enabling them to initiate delivery orders effortlessly and receive quotations. Specifically for company users, the application provides them the option to have recurring deliveries according to their company needs.

Furthermore, the project integrates an email service for ideal authentication of a user. Additionally, a robust tracking system for active orders enhances transparency, allowing users to monitor order progress.

The project also addresses the operational aspect by providing accounts for staff members, enabling them to access and manage orders by changing the status respectively. This feature facilitates seamless coordination and real-time updates on delivery statuses (received, en route, delivered and quotation). Ultimately, the project strives to build a delivery system that meets the needs of the expanding market of online shopping, enhancing customer satisfaction and operational efficiency.

- 8. Learning Outcomes and Importance of Architecture and Design

In this project, our team encountered typical teamwork aspects such as collaboration, communication, and issue tracking using GitHub. However, a pivotal shift occurred in our approach, notably in understanding the significance of analysis and design of a software system prior to plunging into implementation, a deviation from our past practices.

The creation of detailed diagrams (context diagram, domain model, system sequence diagrams and class diagram) exemplified a more disciplined approach to software engineering. The context diagram provides the system's scope and external interactions. It paves the way for the domain model, which encapsulates core entities and relationships, being a great tool to share system information among stakeholders. The SSDs further refine this understanding, visualizing user-system interactions, helping in requirement identification, and guiding the design of necessary functionalities. Every functionality is modeled by an SSD. Finally, the class diagram solidifies the system's structure, providing a detailed blueprint for implementation, ensuring alignment with previously defined contexts, domains, and SSDs.

Together, these diagrams sequentially build upon one another, fostering a structured and informed development approach, crucial in achieving project objectives by guiding the team from conceptualization to implementation. These visual representations proved instrumental in developing a structured, efficient, and easily maintainable codebase. They served not merely as visuals but as the cornerstone of a robust and well-organized system. This experience taught us the value of a methodical approach grounded in analysis and design, helping us to navigate a project from inception to completion efficiently.

- 9. Alternative Approaches

For this project we used the Firebase service for our application's backend, notably the authentication service and realtime database. We could have alternatively created our own backend or used a different service but we decided to use Firebase since we were already familiar with it and given the limited time to complete this project and the fact that this class is focused more on design and planification rather than implementation, we felt it was not worth it to create our own backend. We followed the agile software development methodology to organize the planning and development. We opened issues on GitHub and assigned them story points. We broke down the development into separate sprints focusing on specific features and use cases. Team members participated in pair programming to make debugging code easier, but most of the features were developed in parallel and independently. These were the methodologies that we found got the most work done for us and in the most efficient way. Therefore, we adapted the methodology we used based on the situation since Agile was easier for the overall development, but pair programming was easier for debugging. We thought pair programming for the whole project would make the development less efficient since we would essentially "lose" developers so that they could work on a common feature, which is why we decided not to use pair programming for the whole project, but only for specific situations such as debugging and roadblocks.

- 10. Conclusion

We have accomplished numerous milestones during the course of the project. Firstly, one of the main features we really wanted to implement was a chat box in the website that the user could use to communicate. This was a heavier feature since it was never done before by any of the team members. We wanted this feature to be something that stuck out from other groups. And so we are proud that we had enough time and will to complete this aspect of the website. Another achievement of our delivery service is the UI. We believe the look of a website really affects the customer experience. Since the main goal of the service was the ease-of-use for the customer's enjoyment, the look of the project was important to us. In the end, we believe we have conveyed a professional look in the UI. An important goal of the project was high cohesion and low coupling. Since the point of the project was to build the website with engineering practices in mind, these attributes are an essential part in bringing efficient and reliable code into the world. With the presence of multiple service classes, we believe that we have achieved these goals. One of the hardest things we believe that a lot of companies unfortunately face is the lack of communication.

During the course of our project, we ensured that we were constantly staying in touch. When it came time to approve everyone's PR, this was done in good time since everyone kept available. Therefore, there was little wasting of time waiting on other people. Also, since the team has done past projects together, we have a good sense of community. Therefore, we remain transparent and helpful to each other. This really helped when one of us was in charge of a task they were unfamiliar with. In many cases, another team member who was more knowledgeable would explain the process.

A challenge that we faced during the project was designing the project in the early stages only. We wanted to design a well built system in the simplest way. One of the areas that we discussed at length was how to save objects in Firebase. For example, instead of saving quotations as a separate object, we decided to save them as a delivery object but with a quotation status, since they are no different than a regular delivery object. In addition, in sprint 4, it was difficult to remember the reasoning behind all the diagram modifications since we did not extensively document these changes. On top of that, sprint 4 brought us another challenge. This was the implementation of testing. Because this aspect was only mentioned in the later sprint, it was overwhelming to assure the passing of these tests in such little time.

There are many lessons we can take with us to future projects. The first is the importance of analysis and design documentation. Even if this is a lengthier and meticulous part of the development process, it is well worth the extra commitment. Since it reduces issues in the long wrong when the programmers begin coding. Moreover, a lesson we will take with us is to document the changes made to the various diagrams and requirements needed for a project. It is best to leave comments or have a separate documentation for the reasons for changes so if it is asked later on you can easily prove to them why modifications were made. Another lesson that we believe is very important to bring with us to future projects is the use of tasks, story points and milestones. These GitHub features really allow the team to fairly separate tasks, but also help us stay on track with our progression. The use of milestones help envision the work that needs to be done by a certain date. These tools lower the chances of delays since it is clear what needs to be done.

In all, we believe this was a successful project. We ended with a final result that had all the features wanted at the start of the semester and more. We did not settle for less in terms of front-end either. There are aspects of the project that could be more realistic if ever it were to be used in the real world. For example, we would need to integrate an actual payment service that adds to a company account. As well, we would have to double check that our pricing calculator is similar to other delivery companies. Moreover, we would probably need to assign deliveries to specific drivers when the website becomes more large-scale, because right now any staff can modify the state of any delivery. Overall, we believe we have delivered all the necessary requirements in a robust and efficient way.