SOEN 343 Project (Hexagon)

Phase 4

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

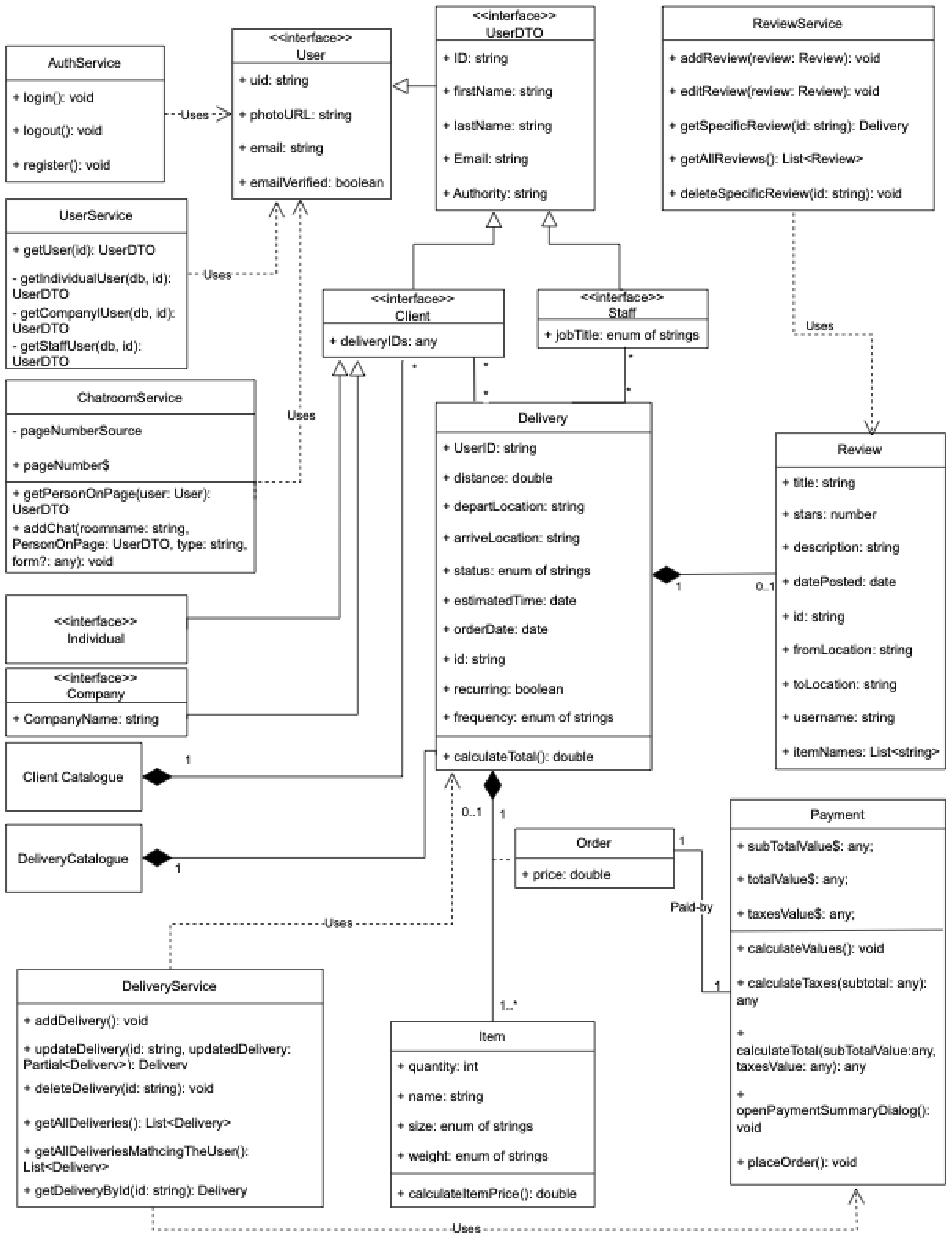
Oliver Vilney
ID: 40214948 Role: Developer

SOEN 343: Software Architecture and Design
Professor: Joumana Dargham
December 4th, 2023

Table of Contents

2. 1. Diagram Refinement/ addition

3. 1.1. Class Diagram



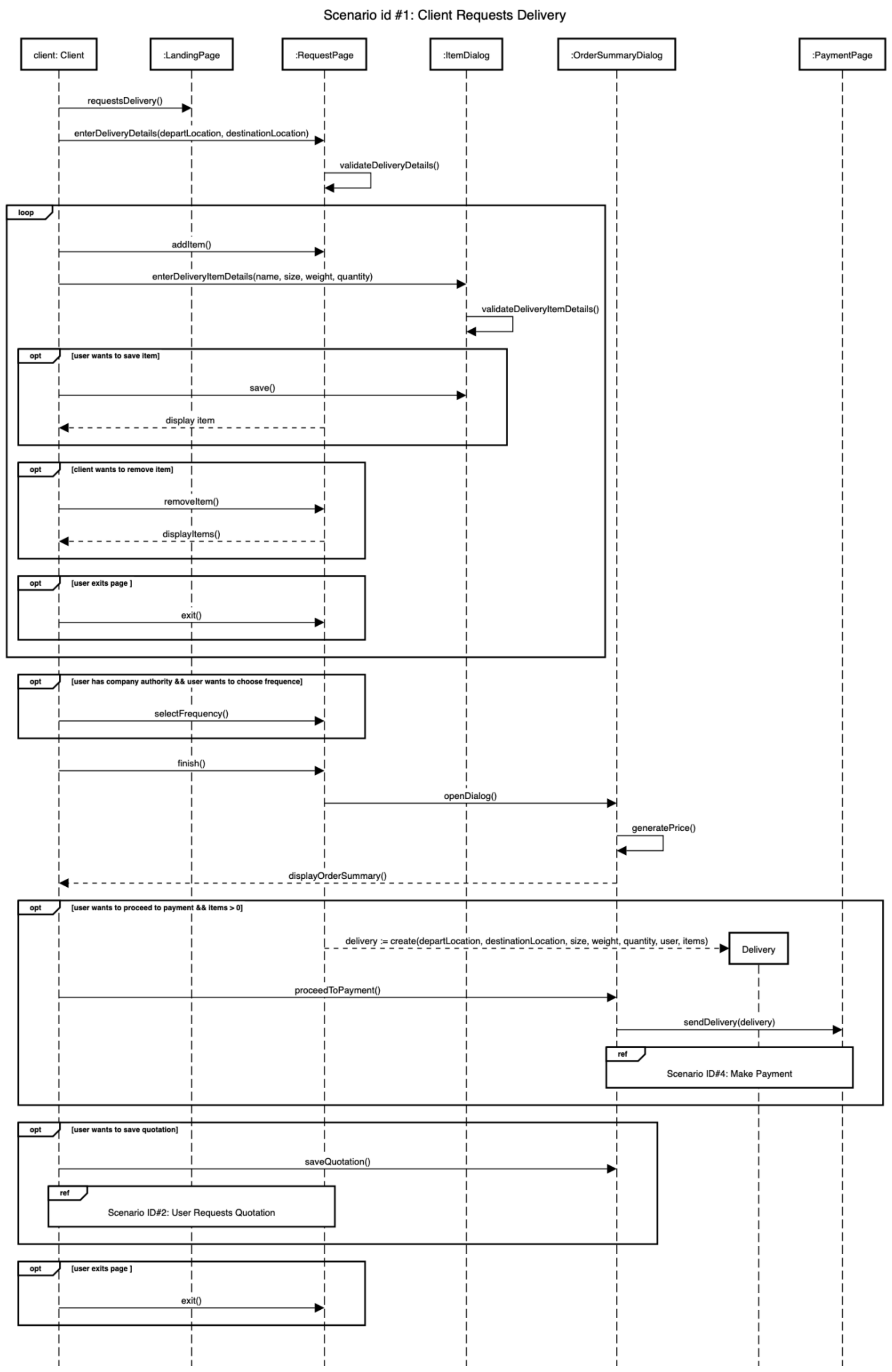
We decided to make the review a composite of delivery since every delivery has at most one review that can only exist if the delivery exists. We also added some services. We introduced a delivery service, review service, authentication service, user service and chatroom service. This was done to reduce coupling so that database access is done in a dedicated class.
We also added some missing attributes to the classes as the project evolved and we had a better understanding of what we needed.

Changes:
-Added more attributes to the delivery class to address missing data.
-Added more attributes to the review class.
-Changed association from delivery to review to be a composite.
-Added services to reduce coupling.
-Updated the methods in the deliveryService class to align with the final methods.
-Updated signature of methods in the reviewService class to add proper return type.
-Removed Admin user and reorganized User class hierarchy.
-Added missing attributes to Item class.
-Updated attributes and methods in Payment class.
-Added the methods used in the Payment class.

Importance:
These changes were made to better represent the system we were designing. They addressed some changing requirements, provided better clarity on the associations between classes, and allowed for a better design by reducing coupling and increasing cohesion.

4. 1.2. Sequence Diagrams

System Sequence Diagram #1: Client Requests Delivery

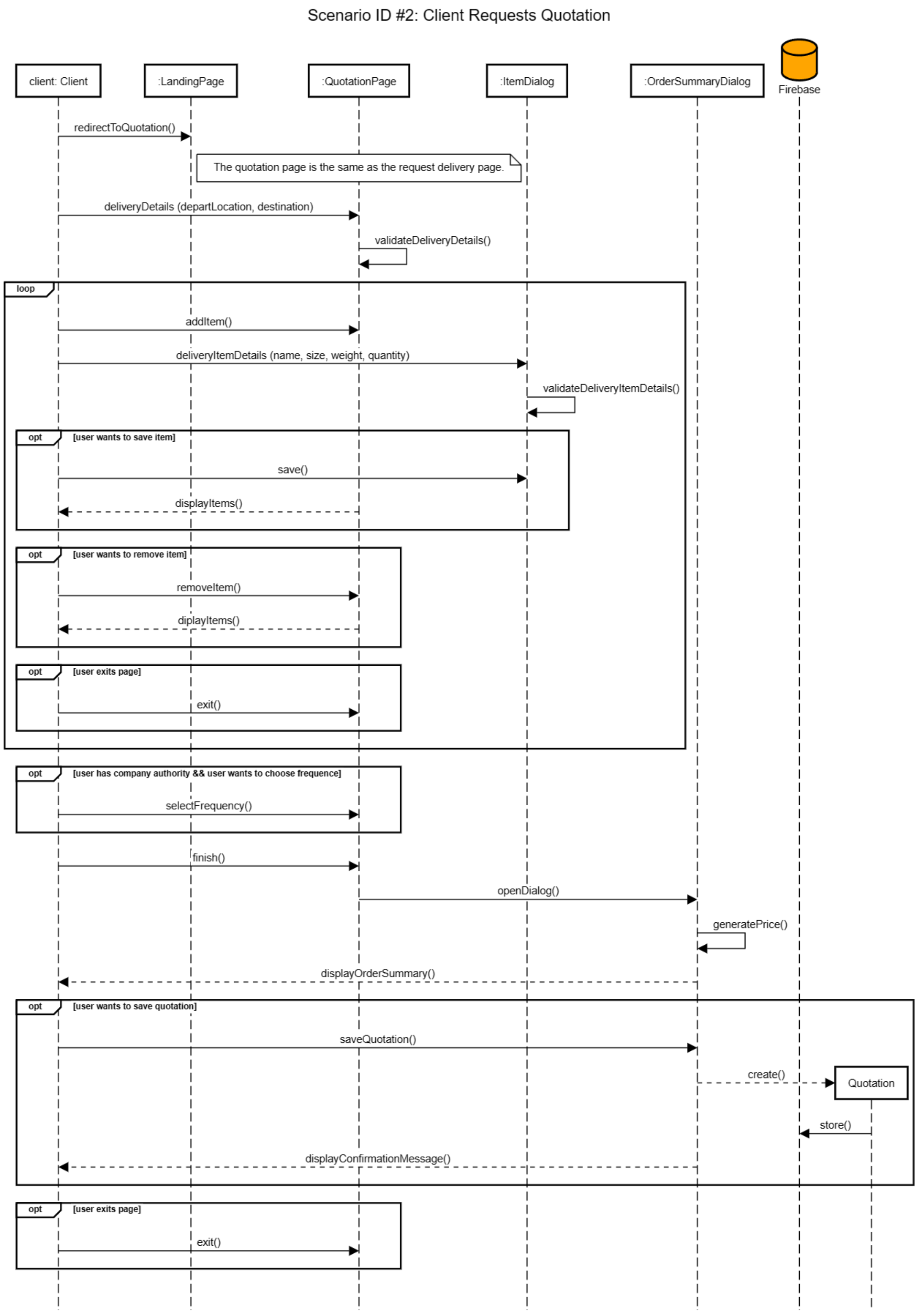

Scenario id #1: Client Requests Delivery
client: Client
:LandingPage
:RequestPage
:ItemDialog
:OrderSummaryDialog
:PaymentPage
requestsDelivery()
enterDeliveryDetails(departLocation, destinationLocation)
validateDeliveryDetails()
loop
addItem()
enterDeliveryItemDetails(name, size, weight, quantity)
validateDeliveryItemDetails()
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()

The revised diagram presents several modifications compared to its initial iteration drafted during Sprint 2. Notably, significant improvements involve the introduction of additional functionalities through the inclusion of 'opt' options. These options enable users to either exit from the page, proceed to payment, save the quotation, remove delivery item or choose frequency if the authority is company, enriching the overall user experience. Many of these newly added options refer back to specific sequence diagrams.

Moreover, the updated SSD incorporates the itemDialog component, showcasing its role in gathering user input. Additionally, the integration of the OrderSummaryDialog serves to offer users final options related to delivery, contributing to a more comprehensive and informative user interface.

As part of refining the diagram's details, alterations were made to method names, with 'submit()' being updated to 'finish()', aligning it more closely with the user interface. Furthermore, parameters were introduced within the 'create' method, directed at the Delivery component.

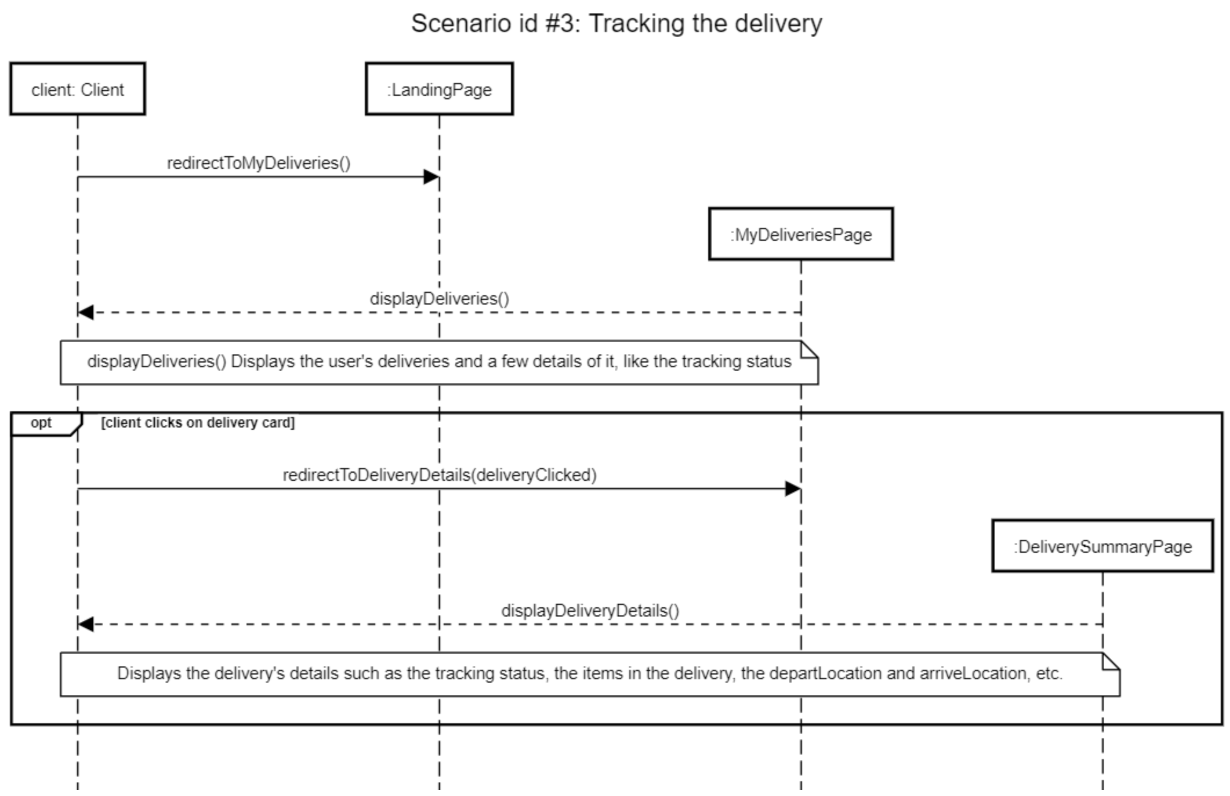
System Sequence Diagram #2: Client Requests Quotation


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()

The refined system sequence diagram seen above for when a user requests a quotation has several modifications to the one made in Sprint 2. Namely, “:ItemDialog” and “OrderSummaryDialog” were added as participants to better represent the current functionality of our delivery platform. These dialogs that pop-up upon a button click event were included to have a greater separation of tasks among modules (i.e: to prevent the quotation page from doing all the work) and, thus, increase cohesion. In sprint 2, it was specified that the quotation was stored in “DeliveryCatalogue”. However, in sprint 4 we changed it to Firebase in order to be more specific and representative of our website’s backend functionality.

Additionally, the option to remove an item was included in this SSD and in our website to allow users to make modifications to their delivery for better user experience. The option to exit the page was also made available to enhance user experience. The option to request a recurring delivery for company users was also added to differentiate features available for different types of users.

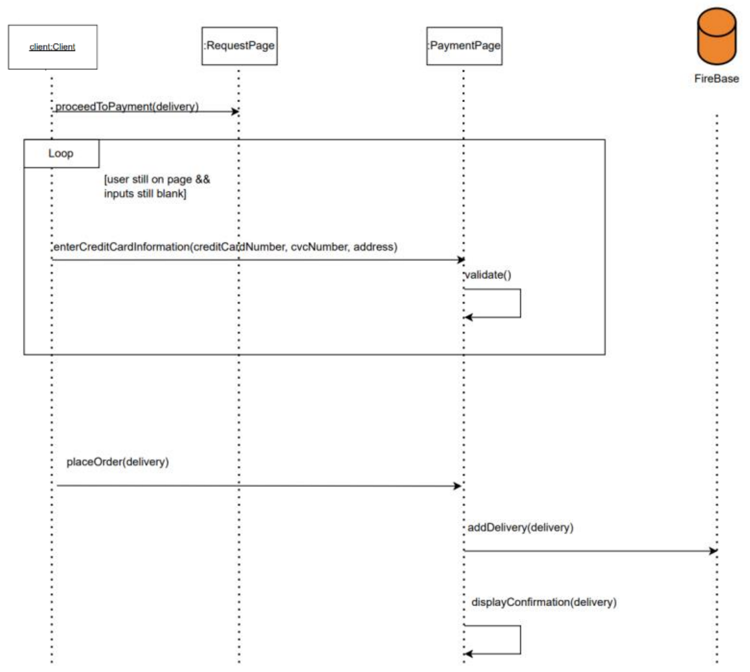
System Sequence Diagram #3: Tracking the delivery


Scenario id #3: Tracking the delivery
client: Client
:LandingPage
:MyDeliveriesPage
:DeliverySummaryPage
redirectToMyDeliveries()
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)
displayDeliveryDetails()
Displays the delivery's details such as the tracking status, the items in the delivery, the departLocation and arriveLocation, etc.

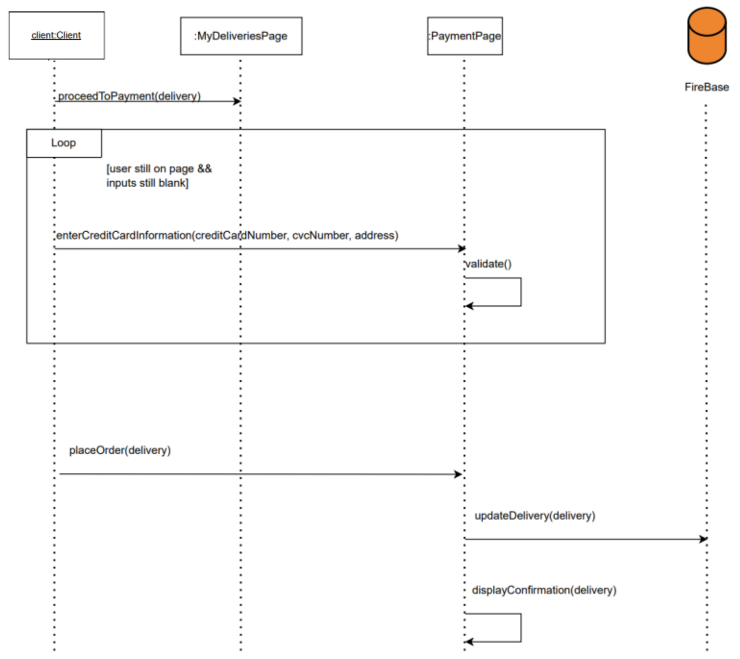
The refined system sequence diagram seen above for when a user wants to see the tracking of their delivery only has a few modifications to the one made in Sprint 2. The change that was made is that we added the parameter “deliveryClicked” to the function redirectToDeliveryDetails(deliveryClicked) to show that when the user clicks on a delivery card, the object is sent to the DeliverySummaryPage so that this page is able to display the information related to that specific delivery.

System Sequence Diagram #4: Make Delivery

SSD: Make Payment from Request Page


client:Client
: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

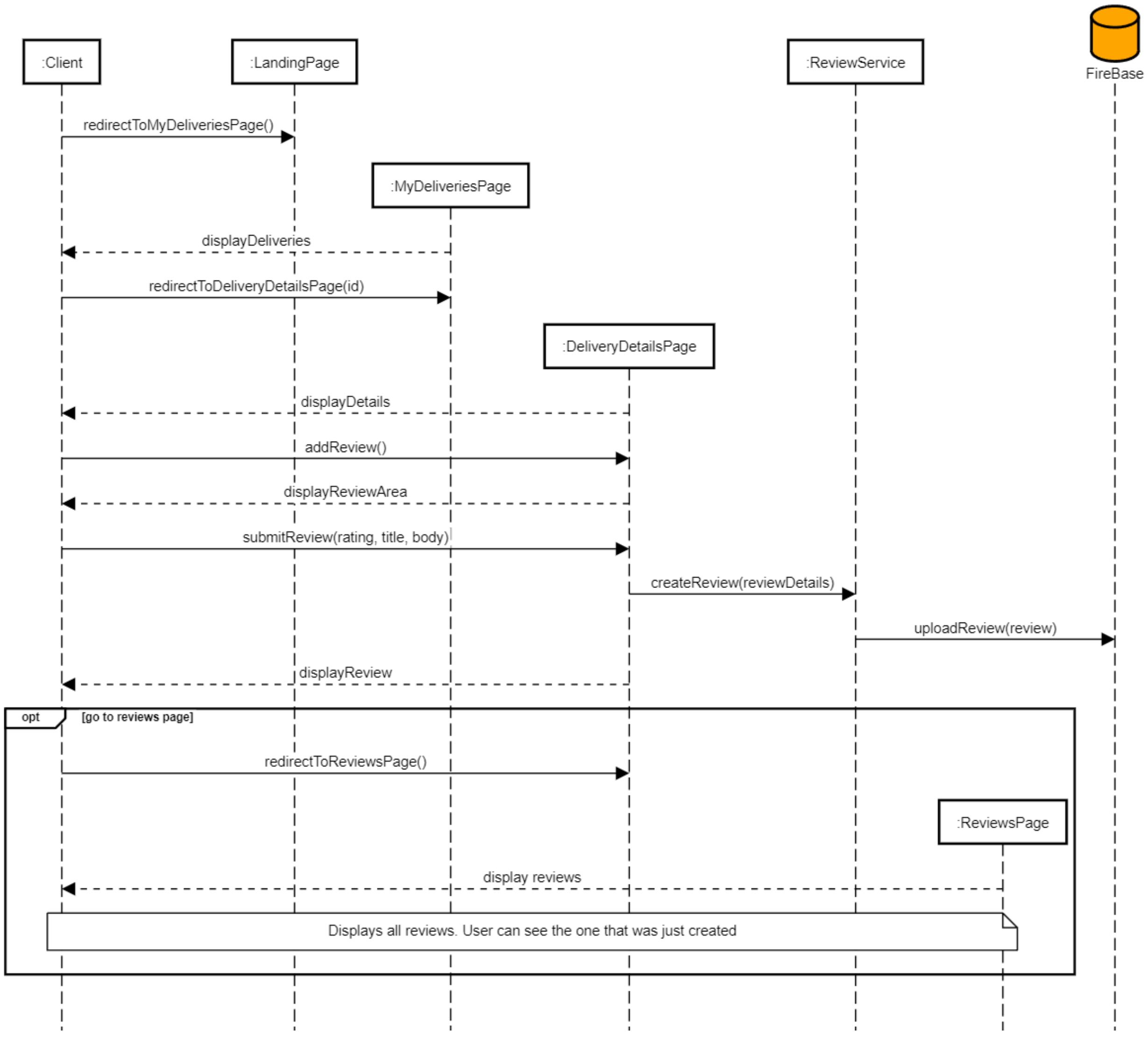

client:Client
:MyDeliveriesPage
:PaymentPage
FireBase
proceedToPayment(delivery)
Loop
[user still on page && inputs still blank]
enterCreditCardInformation(creditCardNumber, cvcNumber, address)
validate()
placeOrder(delivery)
updateDelivery(delivery)
displayConfirmation(delivery)

Several modifications were made to this system diagram. Firstly, the add to delivery page and assign to driver function were removed. Therefore, the class Driver and DeliveryPage were also removed. Secondly, a guard was added to the loop. Thirdly, since the customer can proceed to payment from two different pages, namely from MyDeliveries and from RequestDeliveryPage, I created two different sequence diagrams for these two scenarios. Fourthly, I changed the naming of certain functions so it is more clear and matches the coding name convention. Lastly, for the second SSD that makes payment from MyDeliveries, another function is accessing the Firebase. Instead of adding, we modify an already present object.

These changes were crucial for better programming practices. To start, the removal of the functions were necessary, since this promotes high cohesion and low coupling. These functions do not belong to the responsibility of the payment page. Therefore, the Payment page focuses just on its responsibilities and all unnecessary interconnections are deleted. A guard was added to the loop, so that there is a point of exit. One major change was that two system sequence diagrams were made. This is due to the fact that the user can navigate to payment from two different pages, and then the action will vary slightly from there. In one case, the object delivery is local and therefore when the user pays, it needs to be added to Firebase. In the other context, the delivery item already exists in the database but as a quotation status, therefore, when the user pays the status of the object is modified.

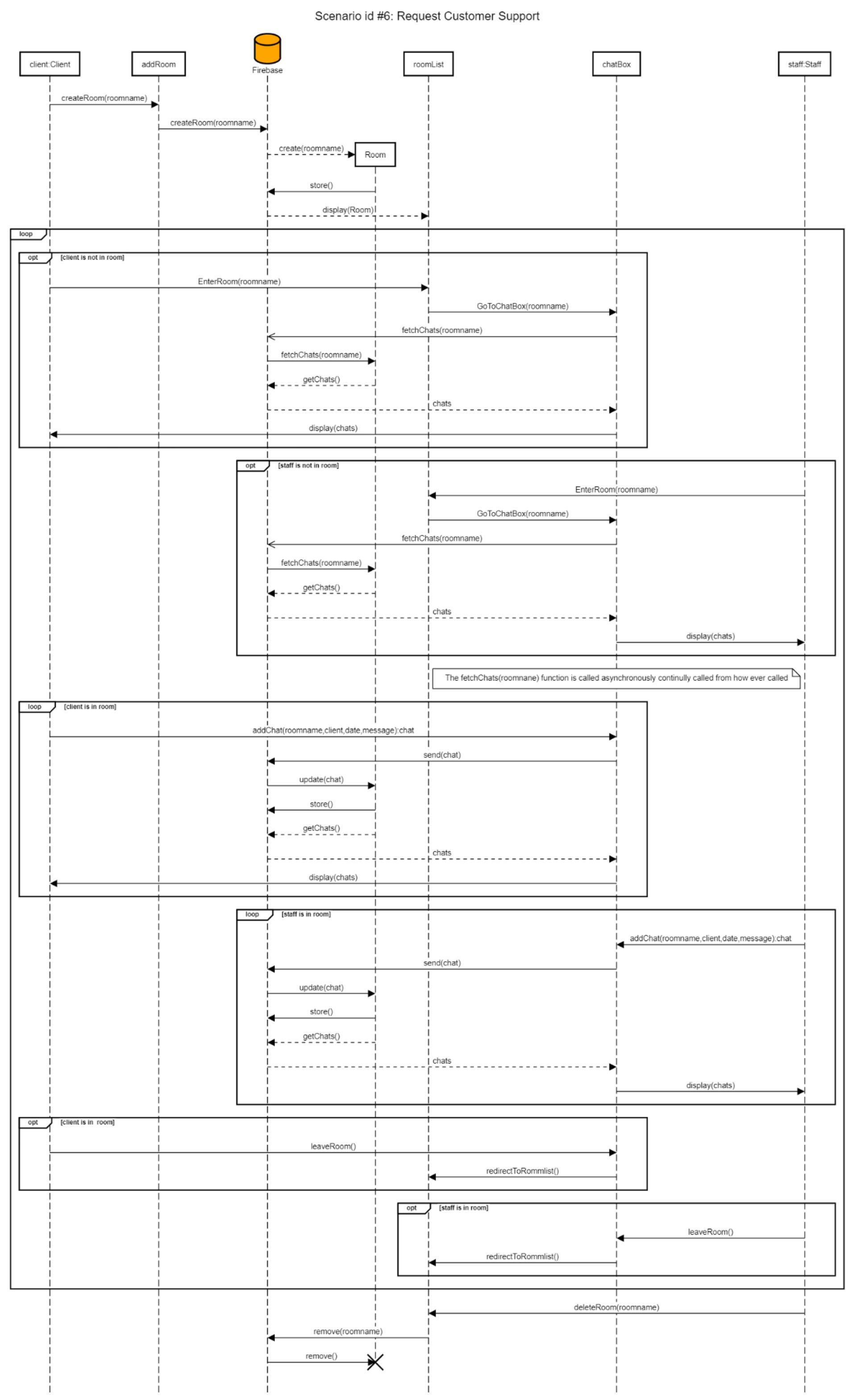
System Sequence Diagram #5: Client Creates a Review

Scenario id #5: Client Creates a Review



No changes were made on this sequence diagram since the final code represented closely the steps followed in the sequence diagram.

System Sequence Diagram #6: Customer service



The changes done in this sequence diagram were the addition of elements and more precise function execution calls. The elements added were the firebase database, the room object, the roomlist and addromlist classes.

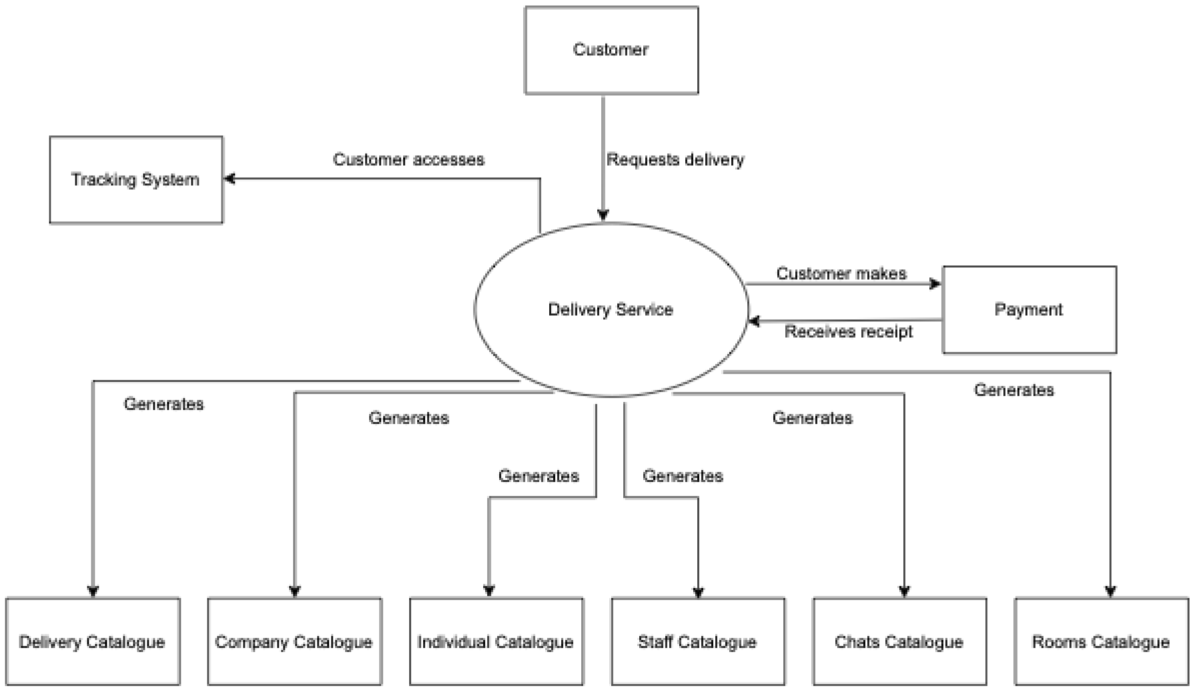
These changes promote a better visualization of the steps necessary to interact with the system especially when it comes to adding and removing rooms. It also helps to clarify how each component contributes to the overall function demonstrated by the diagram. The added function clarifies how each element interacts with each other.

The added rigor brought by adding functions paints a more faithful picture of what needs to be implemented for the system to work. In the diagrams it is shown how the client and the staff intercat in directly with the database in order to make the chats appear and be able to interact with.

Comprehensively this diagram shows that in the case of customer service a client creates a room and a staff said room. Each can enter a room and the room’s chat history will be displayed. At each message sent it is added to the room object and refetches the chat’s history, and the chat history is updated as long as you are on the chatBox page (the room’s page). client and staff can enter and leave the chat.

All of the above are made all the more clearer with the new sequence diagram.

5. 1.3. Context Diagram



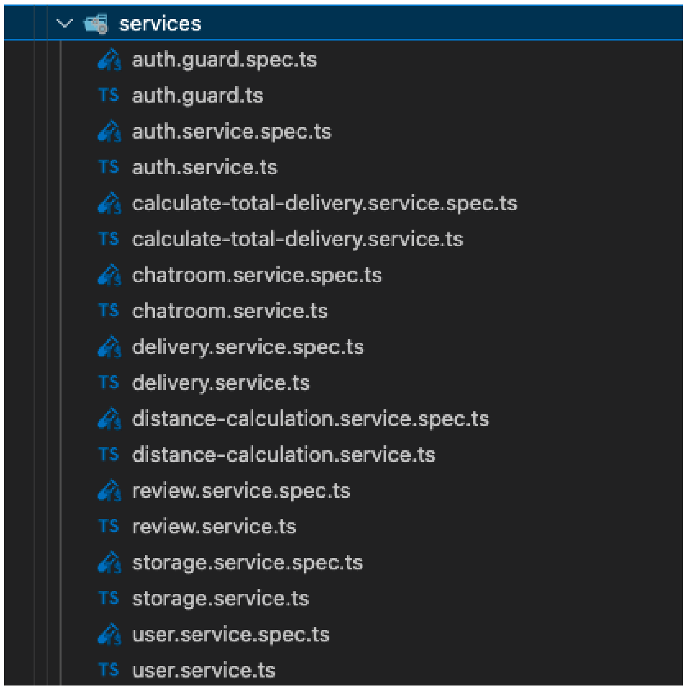
This revised diagram was significantly modified from the first original draft of Sprint 1. Unfortunately, our first draft was developed without fully understanding the concept of the domain model yet. Therefore, it was important for the team to go back and modify it. The changes included removing entities 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.

2. Code refactoring

Refactoring Actions:

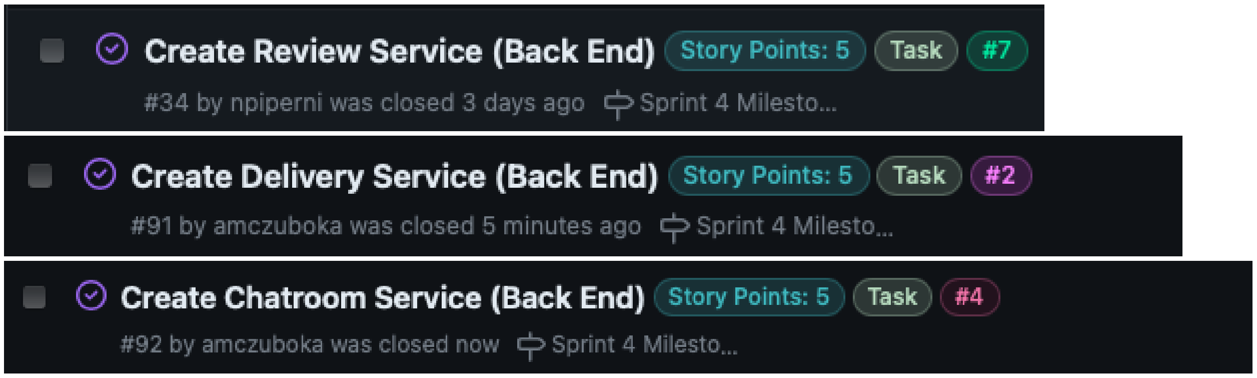
Significant code refactoring activities were executed during this sprint. Mainly, these activities involved the creation of service classes. Some of these classes act as a controller class to bridge between the GUI and domain while delegating the main system logic to the Firebase API. The other classes 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, the result of creating these service classes was better code organization and separation of concerns. As a result, the codebase experienced an uplift in its quality, fostering improved readability, scalability, and ease of maintenance.

Below are all of the service classes that were created to keep our code readable, maintainable, and organized, ensuring streamlined functionality across our application's various components:



```
services
  auth.guard.spec.ts
  auth.guard.ts
  auth.service.spec.ts
  auth.service.ts
  calculate-total-delivery.service.spec.ts
  calculate-total-delivery.service.ts
  chatroom.service.spec.ts
  chatroom.service.ts
  delivery.service.spec.ts
  delivery.service.ts
  distance-calculation.service.spec.ts
  distance-calculation.service.ts
  review.service.spec.ts
  review.service.ts
  storage.service.spec.ts
  storage.service.ts
  user.service.spec.ts
  user.service.ts
```

These were the main Service classes that englobe the majority of the domain logic related to the 6 main functionalities of the application:



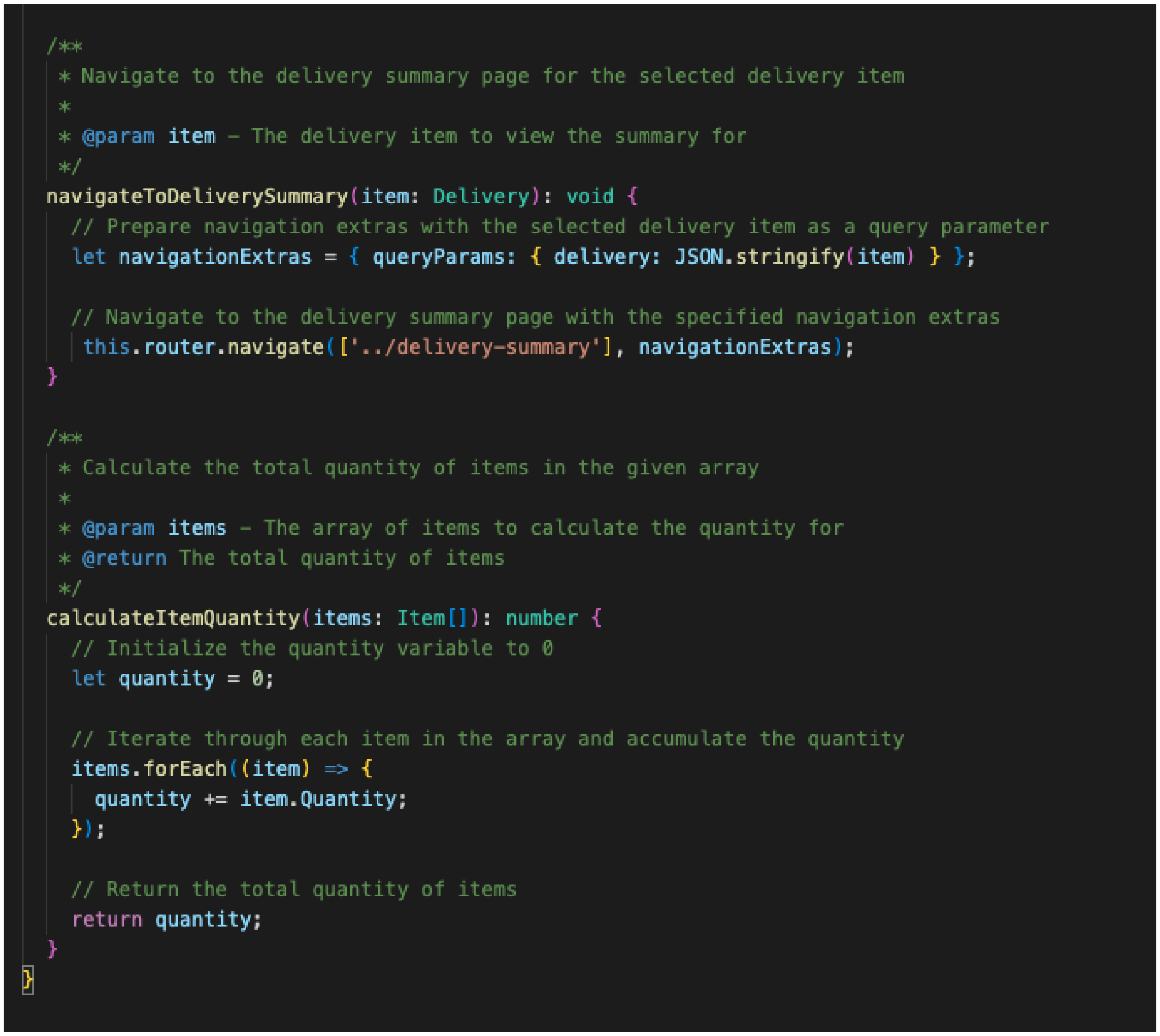
Create Review Service (Back End) Story Points: 5 Task #7
#34 by npiperni was closed 3 days ago Sprint 4 Milesto...

Create Delivery Service (Back End) Story Points: 5 Task #2
#91 by amczuboka was closed 5 minutes ago Sprint 4 Milesto...

Create Chatroom Service (Back End) Story Points: 5 Task #4
#92 by amczuboka was closed now Sprint 4 Milesto...

Lasty, one important refactoring activity the team did, was comment the code adequately. Although this does not change the structure of code, it helps with the readability and maintainability of code.

Here is how the code looks when it is commented:



```
  /**
   * Navigate to the delivery summary page for the selected delivery item
   *
   * @param item - The delivery item to view the summary for
   */
  navigateToDeliverySummary(item: Delivery): void {
    // Prepare navigation extras with the selected delivery item as a query parameter
    let navigationExtras = { queryParams: { delivery: JSON.stringify(item) } };

    // Navigate to the delivery summary page with the specified navigation extras
    this.router.navigate(['../delivery-summary'], navigationExtras);
  }

  /**
   * Calculate the total quantity of items in the given array
   *
   * @param items - The array of items to calculate the quantity for
   * @return The total quantity of items
   */
  calculateItemQuantity(items: Item[]): number {
    // Initialize the quantity variable to 0
    let quantity = 0;

    // Iterate through each item in the array and accumulate the quantity
    items.forEach((item) => {
      quantity += item.Quantity;
    });

    // Return the total quantity of items
    return quantity;
  }
}
```

3. Features implementation

Feature Implementation:

This web application offers a delivery service catering to its clientele. Clients can seamlessly request deliveries, receive detailed quotations, stay informed throughout the service process via effective communication channels, track their orders, complete secure payment transactions, and provide feedback through reviews.

Request for delivery

The request for delivery is done by providing a source and destination address as well as a description of different objects such as their weight size and quantities.

Proposal of a quotation for the service

Once the user fills out the request for delivery information as mentioned above, they can either save it as a quotation or proceed to payment. When they choose to save it as a quotation, it will be saved as a delivery in the MyDeliveries page under the status quotation.

Communication about the service

A chat box is present at the bottom of the landing page where the user can create rooms that they can name as they wish for different issues or questions they are facing. Here, they can message staff and get replies in that room in real time.

Tracking the order

Inside the page MyDeliveries where the user can see all the deliveries they have made including just the quotations, a status is shown. So each delivery is shown as a card and the bottom of the card shows the tracking bar with words like pending, en route and delivered.

Payment

Once you proceed to payment from either My Delivery or RequestDelivery page, it will bring you to a form where you fill out important information like name, credit card information and billing address. Once this is complete, you can place the order. Here, it will add a new delivery to the backend or change the status of a delivery that was a quotation to a delivery that is pending.

Review of service

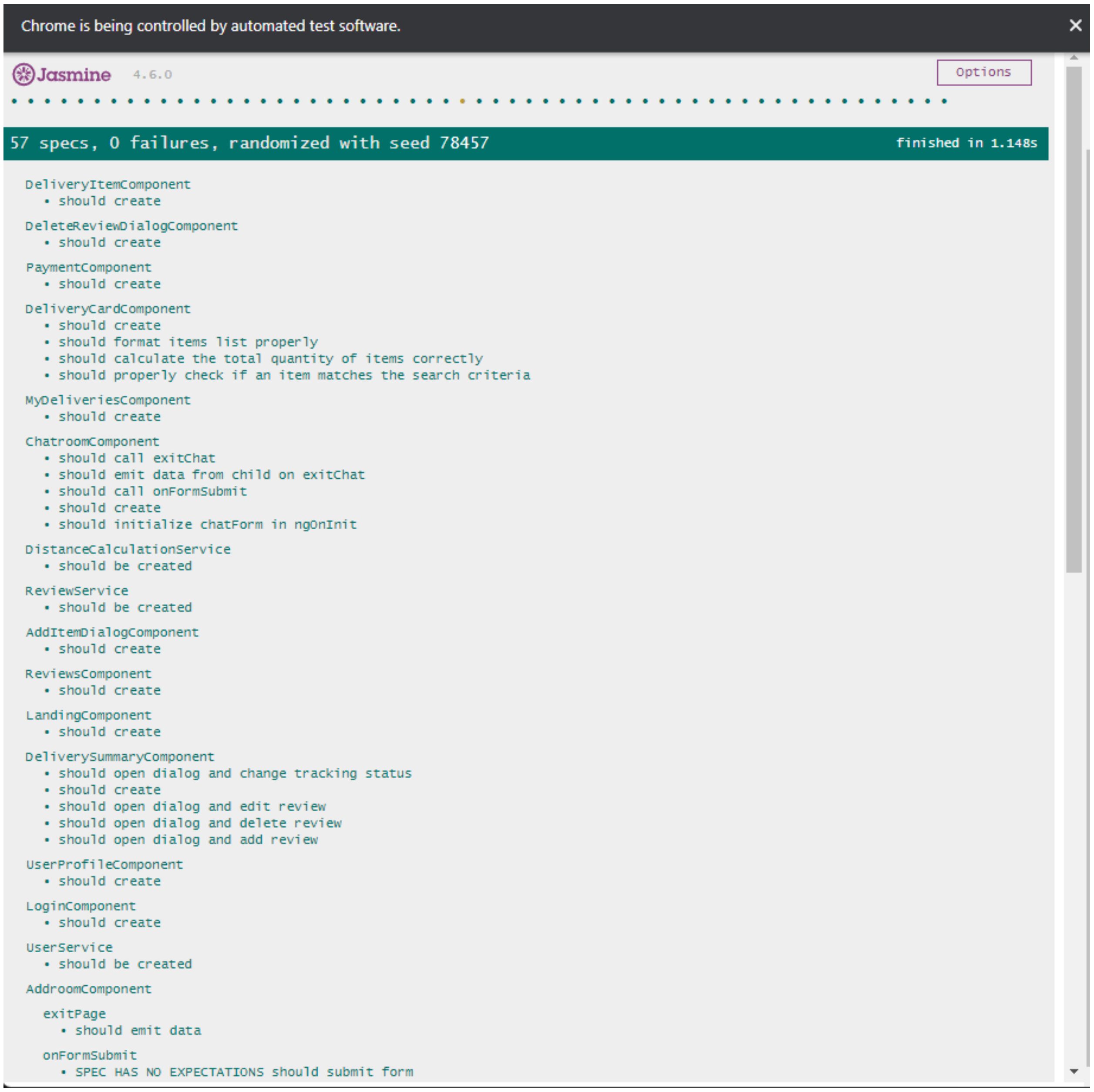
Once a delivery has been placed, inside the MyDeliveries page, under the delivery, you can add a review. Here, you can add a text and rate it with a start system. Furthermore, there is a section called reviews, where all clients can see everyone's reviews.

Testing:

Throughout the implementation phase, each component added to the repository is accompanied by a dedicated spec file housing unit tests. Every component undergoes a foundational test, ensuring its successful creation. Complementing these essential tests, our team crafted unit tests for every method developed, excluding those interacting with the firebase API. These comprehensive tests encompass a range of checks, such as validating dialog functionality (opening and closing), ensuring accurate price computations, and confirming seamless entity creation. In Angular, the term "spec" typically refers to Jasmine specifications, residing in *.spec.ts files. These specifications assess the behavior and performance of individual code units.

Executing these Jasmine specs is straightforward; we simply trigger the 'ng test' command within Visual Studio Code's terminal.

Below is an image of what the tests look like after successfully running them with the command.



```
Chrome is being controlled by automated test software.

Jasmine 4.6.0                                                          Options

57 specs, 0 failures, randomized with seed 78457               finished in 1.148s

DeliveryItemComponent
  • should create
DeleteReviewDialogComponent
  • should create
PaymentComponent
  • should create
DeliveryCardComponent
  • should create
  • should format items list properly
  • should calculate the total quantity of items correctly
  • should properly check if an item matches the search criteria
MyDeliveriesComponent
  • should create
ChatroomComponent
  • should call exitChat
  • should emit data from child on exitChat
  • should call onFormSubmit
  • should create
  • should initialize chatForm in ngOnInit
DistanceCalculationService
  • should be created
ReviewService
  • should be created
AddItemDialogComponent
  • should create
ReviewsComponent
  • should create
LandingComponent
  • should create
DeliverySummaryComponent
  • should open dialog and change tracking status
  • should create
  • should open dialog and edit review
  • should open dialog and delete review
  • should open dialog and add review
UserProfileComponent
  • should create
LoginComponent
  • should create
UserService
  • should be created
AddroomComponent

  exitPage
    • should emit data

  onFormSubmit
    • SPEC HAS NO EXPECTATIONS should submit form
```

4. Conclusion

Achievements:

We had many achievements throughout this project. The first achievement we had was communication. Our team has already done projects in the past together. Therefore, the team is very comfortable discussing several matters and staying on top of each other's work. For example, while working in GiHub, it was easy to progress with our work since everyone was actively participating in the review of our code through pull requests. Since we had a lot of work to do in Sprint 4 in a limited time, this quality helped our team stay up to date.

Another achievement that helped our team be more efficient was transparency and being helpful. There were a lot of times that a member of the team needed to code something that they were unsure how to do. In these moments it was easy to ask for help from another team member for an explanation on how to do it since it was likely they had done something like it in the past.

Challenges:

A challenge we faced in sprint 4 was documenting changes we made to the diagrams. Since these diagrams undergo smaller modifications during the process, it was sometimes hard to remember why we made changes to them without leaving comments. So when it came down to explaining the modifications, it was hard to remember all the details.

Another challenge we had was testing. We had left testing to the last sprint since it was not a requirement for the previous sprints. Testing can be a very lengthy process when left at the end, and if many tests fail, it can be very overwhelming to fix at the last minute.

Lessons Learned:

This project was the first coding project that was done with engineering practices. And with that in mind, we learned several things. While coding with analysis and design in mind can be very jarring at first, it makes the coding process much easier in the long run. So while at the beginning, more effort was made, this process helps eliminate issues in the coding process near the end of the coding process in Sprint 4.

Because of the challenge we faced in explaining modification to all the diagrams, we learned to keep track of all the differences through comments for next time. This way it is easy to view our progression in terms of analysis and design.