ILP CW2

B194750

November 2023

Reflection on the Practical Implementation

Aspects of my implementation differed from the initial plan, as usually happens with most projects. While many requirements stayed the same and implementation went as expected, a few were altered. As intended, the spec is vague, giving little detail into the specifics. Because of this, Piazza became my major source of keeping up with current requirements. With almost daily updates and clarifications being posted, Piazza was vital in keeping my implementation in line with current requirements.

One thing the spec clarified was the release of a new source Jar to replace the one for CW1. Combined with Maven, this proved to be one of the biggest challenges in my implementation. While the addition of the Jar was not a challenge, getting the .xml file to compile the project and create an executable jar was. The source .xml file found in the GitHub repository helped but there were still aspects that I figured out myself. Maven plugins such as shade, install, and main were added to ensure that the executable jar could read the source Jar, and find and execute the Main method correctly. Additionally, adding Maven dependencies to the .xml proved a challenge at first, but I quickly adapted to the task. All this combined with never having used Maven made this a significant difficulty in my project.

The first thing implemented was deserialization, which also happened to be one of the parts that differed the most from my initial plan. Before starting, I had researched what method to use for deserialization. I concluded that Jackson would be used for deserialization, as it is a widely documented method with lots of online support. However, when it came to implementing the deserialization I discovered some difficulties with Jackson. While Jackson offers support for more complex situations, Gson strives in its simplicity[1], and it was this simplicity that I desired in my implementation. Using Gson would also make it easier to integrate the LocalDateDeserializer, a crucial part of deserialization.

The requirement that took the most time, and was by far the biggest challenge was the pathing system. Initially, I had planned to utilize the A* algorithm, but a naive version of A* for simplicity. After a first attempt, this proved to be a wrong approach. Not only was my result non-optimal, but the requirement that the drone can move in 16 directions forced the algorithm to be as efficient as possible as the search space was increasing exponentially to the power of 16. The majority of my time spent on CW2 was fine-tuning the A* algorithm.

It was during this that I realized that my knowledge of Java's data structures was not up to scratch. I first tried using a hash map to store the frontier but quickly realized that searching the entire frontier at each step was highly inefficient. Using instead a priority queue with a custom heuristic and weighting on the distance to the goal was key in making the algorithm run in under 60 seconds, another requirement[2]. However, the thing that reduced run time the most was the dynamic searching I added. If none of the possible points from the current position were in a no-fly zone, the best move from that list is the next point. But if a no-fly zone is present, all possible points are stored in the frontier to facilitate backtracking in true A* fashion.

Something I could have done differently would be to calculate the path to the restaurant and then reverse it. I then wouldn't need to calculate the path back. While this would cut the calculations in half, I felt that without it my algorithm was good enough and the extra work needed to implement it would not be worth the possible efficiency increase.

I think overall, my implementation went quite well. While keeping with original requirements but also adapting to new ones, an optimal, working project was created. There were challenges to overcome, but with research and an improvement in my skills, I overcame these. In the final steps of my project, I learned the benefit of splitting up the Main method into separate classes, static where possible[3]. Splitting up these classes into packages further added to the clarity of my implementation. The addition of in-line and Javadoc comments was the final touch to my project[4]. More improvements would be further optimizing the efficiency of my A* algorithm and removing unnecessary methods and processes, streamlining my code.

The PizzaDronz Mobile App

So far, all of the work on this project has been technical. From gathering requirements, drawing up UML diagrams, and implementing the system itself. There has been no design or work done on the user side of the project. The users currently have no way to interact with the system and order pizzas. In having a working system, it is time to focus on the user and link them to the system. Designing how someone will use and interact with the app is as paramount as having a working system. There are many aspects to the usability of an app, as the app needs to be as accessible and user-friendly as possible.

Before any visual design can take place, the user's flow through the app must be considered. How will the user interact with the system? When the user opens the app, they will be greeted with a home screen containing two options, begin order and login/create account. The benefit of the user creating an account is the ability to save their payment details and name for future payments. The 'name' field will also be added to all orders to identify a user's order. If the user does not log in at the beginning, the opportunity to log in at any future point will still be present. Upon deciding to order, the app will display all the restaurants the user can order from. Upon selecting a restaurant, the user can see the menu and add pizzas to their order. The user will be limited to adding a maximum of 4 pizzas to their order and ordering from 1 restaurant as these are both requirements.

Every pizza on a menu will have a picture displaying it, the ingredients, the price, the allergy information, and the dietary information, e.g. vegan, gluten-free. It is important to be transparent about what the pizzas contain as it could be potentially life-threatening for some users if not informed of ingredients and any necessary information. We must also ensure that, with all of the restaurants, there are suitable vegetarian, vegan, and gluten-free options, as catering to everyone will only draw in more users.

Once a user is happy with their order, they can continue with the payment. Before any payment, the user will have the ability to confirm that their order is correct. Once confirmed, the user will be taken to a page where they can enter their payment details. Logged-in users will simply have to confirm existing payment information. Included in the payment details will be an email address, a new addition to the system. As per requirements, every order will be delivered to Appleton Tower, removing the need for the user to enter any delivery address. If the order is valid, an email will be sent to the user confirming that their order is valid and that it will be delivered.

Next, the user interface must be designed. Being the part of the system the user interacts with, it is one of the most important sections to get right. Making the app simple to use for the user is something I will focus on. Not only will simplicity lead to better usability, drawing in more users, but it will also be easier to maintain and quicker to release to the market[5]. A simple UI is achieved using multiple methods. Not having too much information on the screen at one time, overwhelming the user; not having too many decisions, paralyzing the user; having simple, easy-to-understand menus and signposted buttons, ensuring the app is intuitive to use.

Another thing to consider when designing a UI is user accessibility. Over 1 billion people worldwide have a disability that would prevent them from using a website that does not consider accessibility. In not considering it, I am cutting off a significant user base[6]. There are multiple things I can do to ensure accessibility:

- The app will have sufficient text-to-speech support.
- Ensure that text can be resized without jeopardising the users experience.
- Have text descriptions for the images of the pizzas.
- Have colourblind modes available that change the displayed colours.
- Have a high-contrast mode.
- Use consistent layout, font, and style.

Combining these ideas, I can create a preliminary design for the UI. The three areas I will highlight are the home page and the payment page.

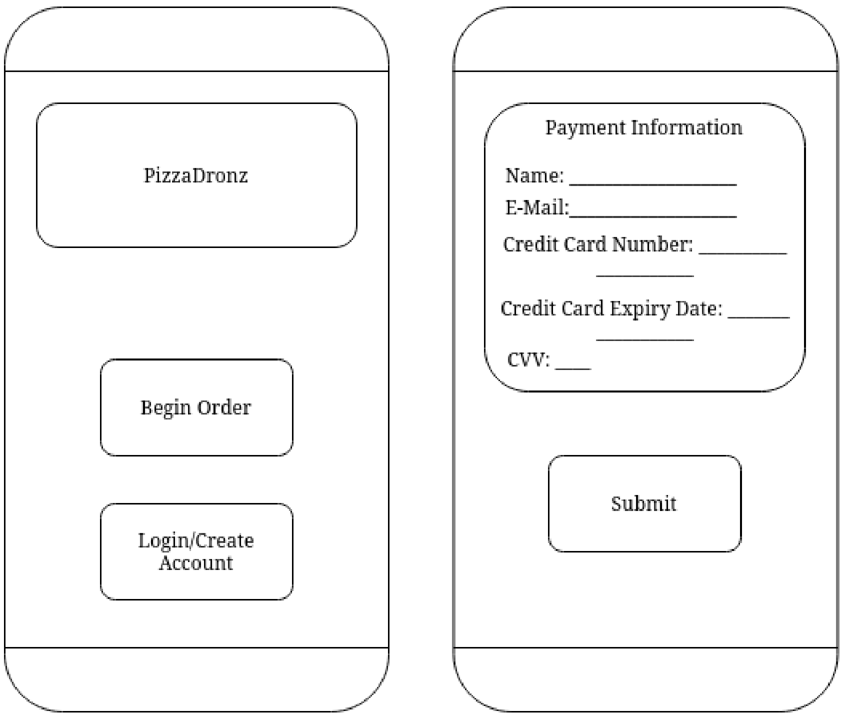

Figure 1: Left: The home page. Right: The payment page.

Even after the design stage and implementation are completed, and the app is released, work is still to be done. Listening and responding to user feedback is crucial in delivering a good user experience. Making sure the customers feel heard and their concerns are being addressed will not only bring them back but attract new ones[7]. Issuing forms to customers, perhaps through email with the user's consent, can be a way to gather user experience as well as letting users leave a review after their payment. I should also provide consistent updates to the app after release to maintain the system, improve aspects of the implementation, and apply user feedback.

Focusing on the user-centric design as well as the technical side is imperative for the success of the app. Prioritizing user experience through considered UI design, accessibility, and user feedback ensures sustained customer satisfaction. Integrating features like text-to-speech support and colorblind modes ensures inclusively, catering to a broader user base. Combined with a simple user flow, this will create a usable app that adheres to existing requirements and links the user to the implementation.

References

[1] Baeldung, *Jackson vs Gson*, Baeldung, 6 November 2023

[2] Mohit Manoj, *Hash map and Priority Queue: Data Structures Explained*, Medium, 22 January 2022

[3] Edeh Israel Chidera, *Static Variables in Java – Why and How to Use Static Methods*, freeCodeCamp, 7 March 2023

[4] RYMS, *10 reasons why you should place comments in your code*, Medium, 15 February 2021

[5] Jonathon Hensly, *UI Design: The Intuitive vs. Easy Paradox in Digital Products*, Emerge, 30 May 2023

[6] Lesa Seibert, *Understanding The Importance Of Web Accessibility*, Forbes, 20 March 2023

[7] Kirill Tšernov, *Why You Should Listen to Your Customers (And Why You Shouldn't)*, Qminder