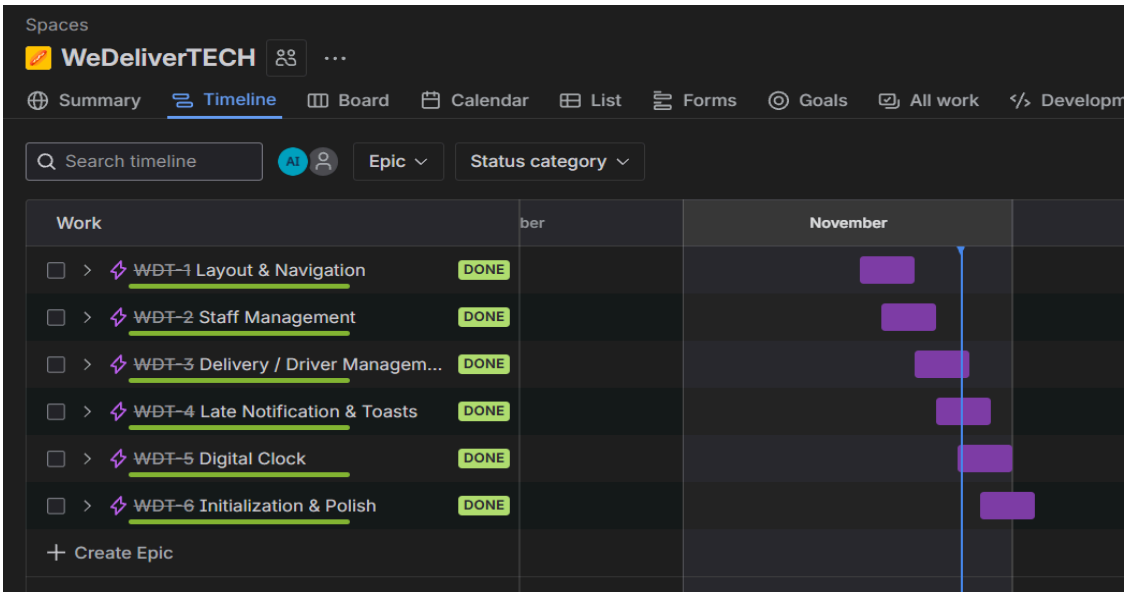
WeDeliverTECH – SEMESTER PROJECT 1
JIRA PROJECT MANAGEMENT REPORT

Student: Alp ISIK
Class: ALP_ISIK_SP1_CA_AUG25FT
Project Duration: 4 Weeks

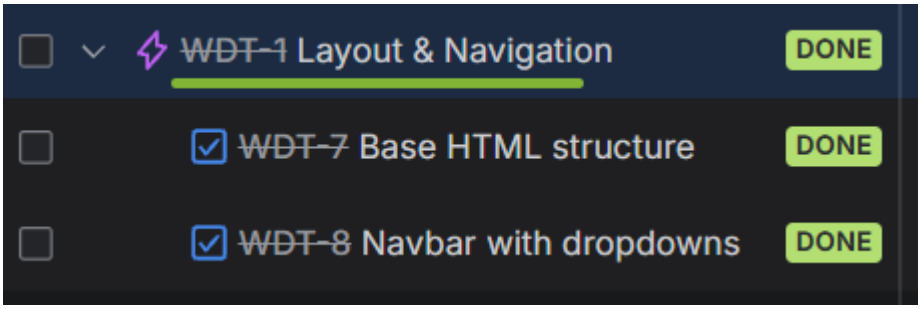
1. Introduction
This document shows how I planned and managed the WeDeliverTECH Reception Management Dashboard using Jira. I included screenshots of my Timeline, Sprints, and Epics to give a clear picture of how the project was organised. Breaking the work into smaller tasks helped me focus on the highest-priority requirements and keep the project under control. The reflection section goes deeper into my thought process during development, which parts were challenging, what went well, and how I approached the project overall.

2. Jira Sprints / Timeline

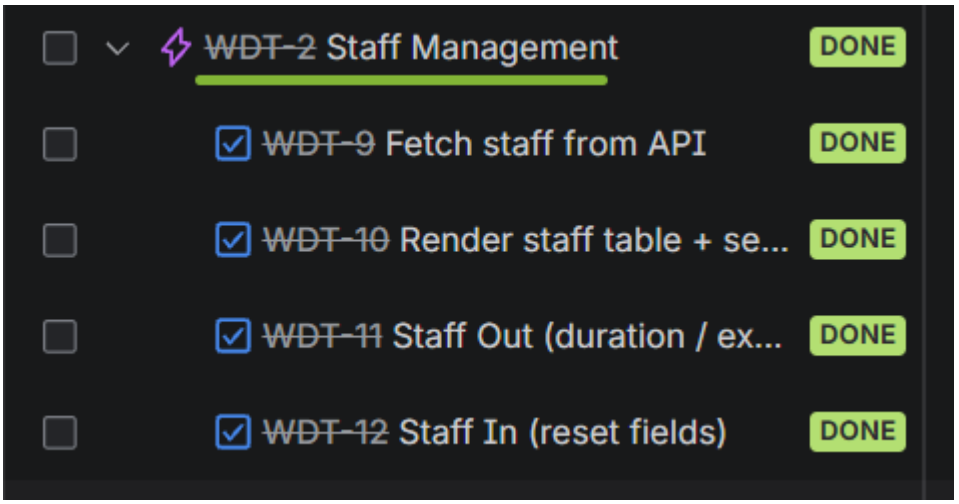


Explanation:
I started the first sprint by getting the basic layout in place, because without a proper structure the rest of the project would just feel messy. After that, I moved on to the staff table and the API fetch, and this is where I set up the OOP classes. Since the assignment required inheritance, I made the Employee → Staff setup early so the API data could flow into actual objects instead of loose values. Once I saw the real staff data showing up in the table, everything else became easier to plan. While doing the layout, I also got a good sense of what parts would be harder later, so I decided to finish the digital clock early since it was quick and didn't depend on anything else. Doing things in this order made the next sprints smoother and kept me from getting stuck on random details.

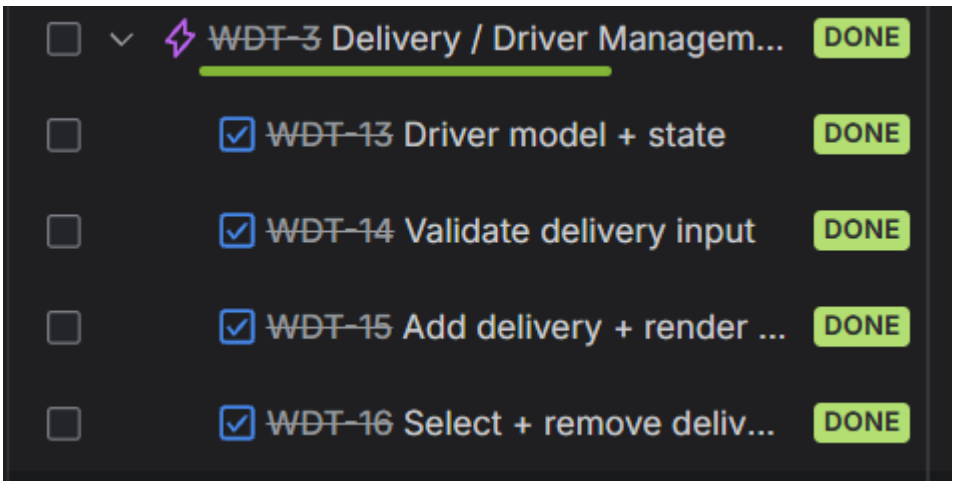
3. Jira Epics
-Layout & Navigation



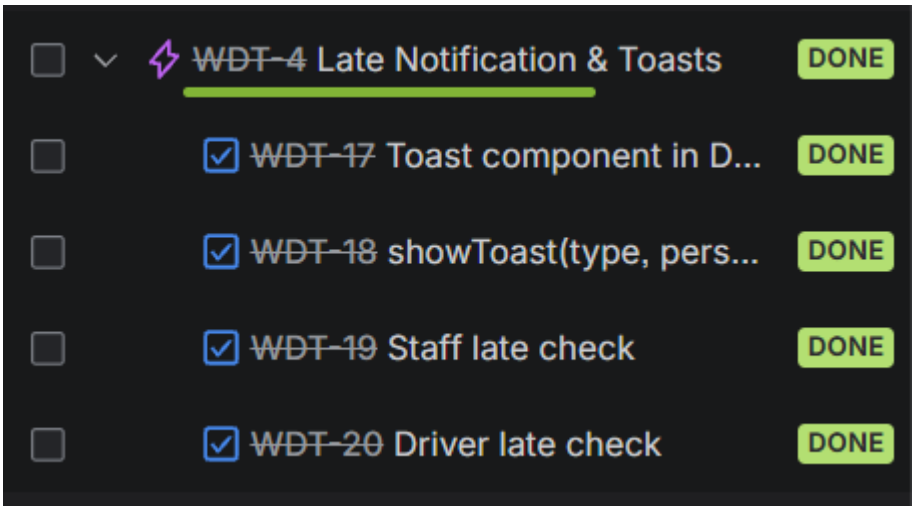
-Staff Management



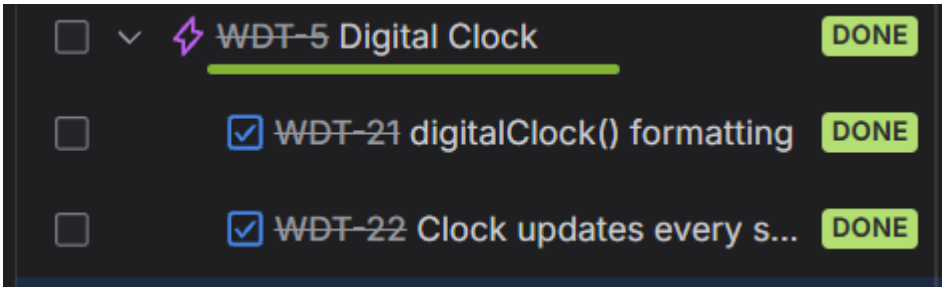
-Delivery / Driver Management



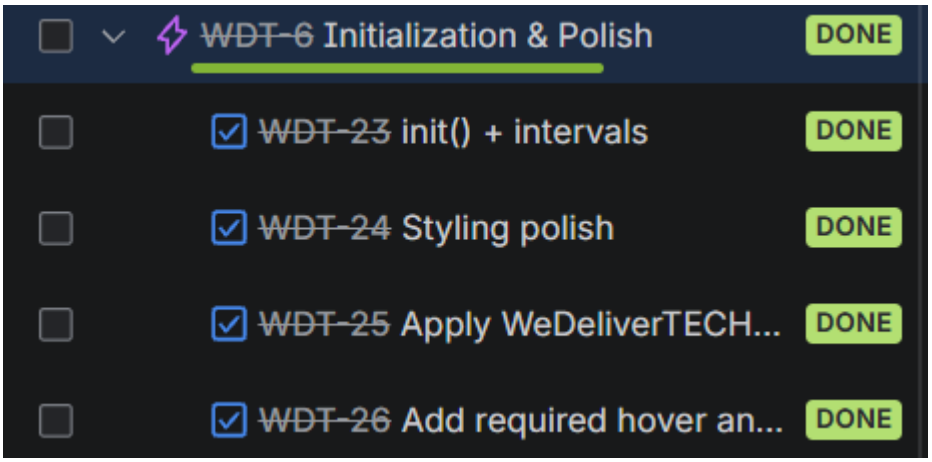
-Late Notification & Toasts



-Digital Clock



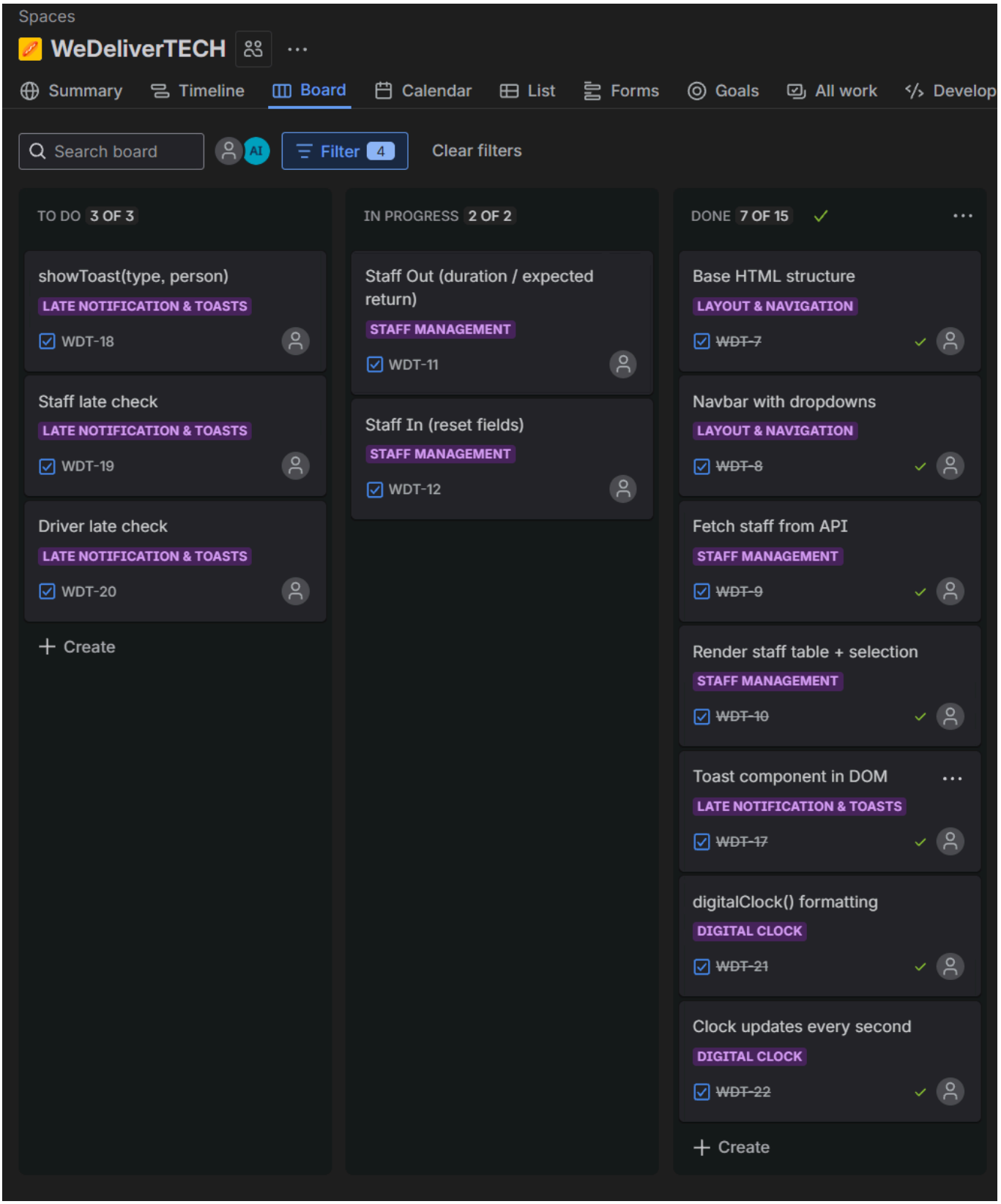
-Initialization & Polish



Explanation:
I grouped the tasks into these epics based on the real execution order of the project. Each epic represents a section I needed to finish before moving on. The layout came first because the whole dashboard sits on top of that structure. Staff Management and Delivery Management were split since they each have their own data and flow. The late notification system got its own epic because it connects to both tables and needed to be handled separately. The Digital Clock and Initialization parts were smaller but still important to keep organised. Setting things up this way made it easier to see what needed to be done next and kept the project clear and manageable.

4. Jira To-Do List / Issues / Backlog
The To-Do list was built by breaking every requirement into small, clear tasks that I could work through one by one. The instructions also specified several functions that had to exist, like staffUserGet, staffOut, staffIn, addDelivery, and the late check functions, so I created tasks around those as well. This made it easier to shape the system the way the brief expected and kept everything consistent with the required structure. I wrote the tasks in the actual execution order of the project, which helped me follow the flow without jumping around. Each item represents a single step, from fetching API data to handling late notifications. Keeping the tasks simple and tied to the required functions made tracking progress straightforward and made it easy to move things across the board as soon as they were done.

5. Jira Board (Kanban)



Explanation:
The board shows how tasks moved from To Do → In Progress → Done during development.
This made it easy to see active work and keep the project on track.

7. Reflection (500–1000 words)

This project was a good test of how I plan, structure, and build a full feature set from scratch. Using Jira actually helped more than I expected, mainly because the requirements were very specific about what

functions I needed to implement. Splitting those into tasks and grouping them into epics gave me a clear roadmap, so at no point did I feel stuck about what to do next. I ended up rebuilding my Jira setup a couple of times because the first versions felt messy, and cleaning it up made the whole flow much easier to follow. Once the structure was fixed, the system basically built itself step by step.

Starting with the layout was the right call, because getting the page structure, navigation bar, and sections in place made it easier to visualise how all the logic would fit in. After that, I moved to the staff table and the API fetch. This is where I set up the classes and inheritance, because the brief clearly required an OOP approach with Employee → Staff → Driver. Having that foundation early made the rest of the data flow way more organised. Once real API data showed up in the table, the rest of the project became easier to think about.

I tried to build everything in the same order the website actually runs. Before worrying about late checks or toast notifications, I made sure selecting a staff member worked, status updates worked, and the table re-rendered properly. I used the same approach for deliveries: first the inputs, then validation, then object creation, then rendering. This kept the features from stepping on each other. I also finished the digital clock early because it was simple and didn't block anything else, so clearing small tasks like that helped mentally.

The biggest challenges came from dynamic tables, row selection, and making sure data updated correctly after re-rendering. When you rebuild a table each time, you can lose the selected row or break event listeners, so I had to rethink how I stored and reapplied the selected index. Late notifications were another challenge — the logic needed to activate only once, show the correct info, and not keep spamming toasts every second. Adding a lateNotified flag solved that cleanly.

What went well was the structure and readability of the code. Keeping functions simple and updating only when needed made the project predictable. Bootstrap covered most of the styling, so I focused on logic and data flow. Hover animations and branding also helped the UI feel more polished without overcomplicating anything.

Overall, this project taught me a lot about planning ahead and breaking requirements into manageable tasks instead of rushing into the code. Using OOP properly made handling staff and driver data much smoother. Jira helped track progress and keep things organised, especially after restarting the board to clean up the structure. In the end, the dashboard feels stable and matches the requirements exactly. It's something I can confidently put in my portfolio later, since it shows both the technical side and the planning side of how I build projects. gg wp