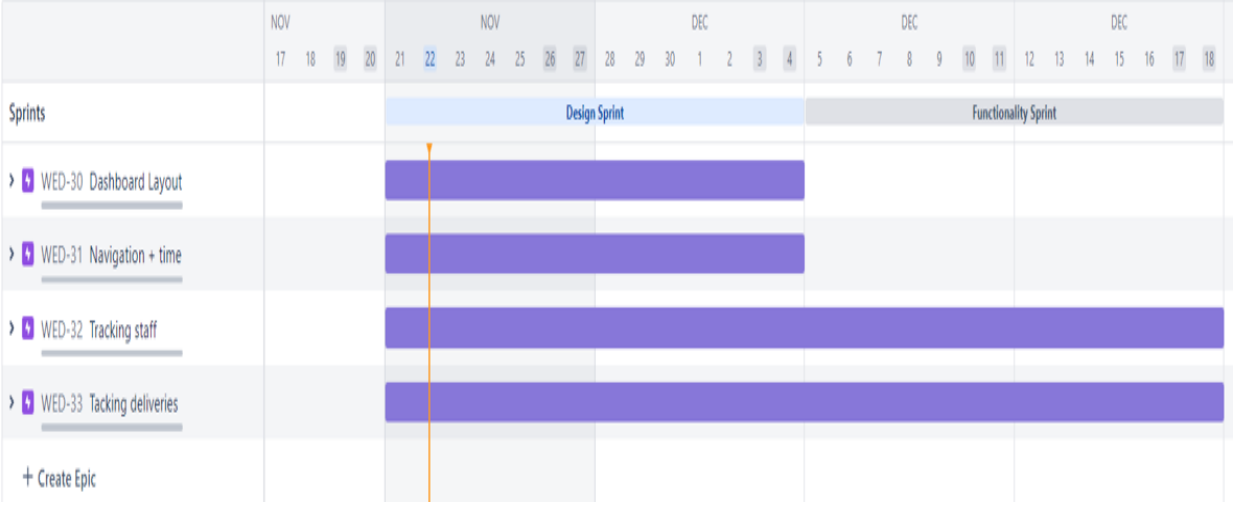
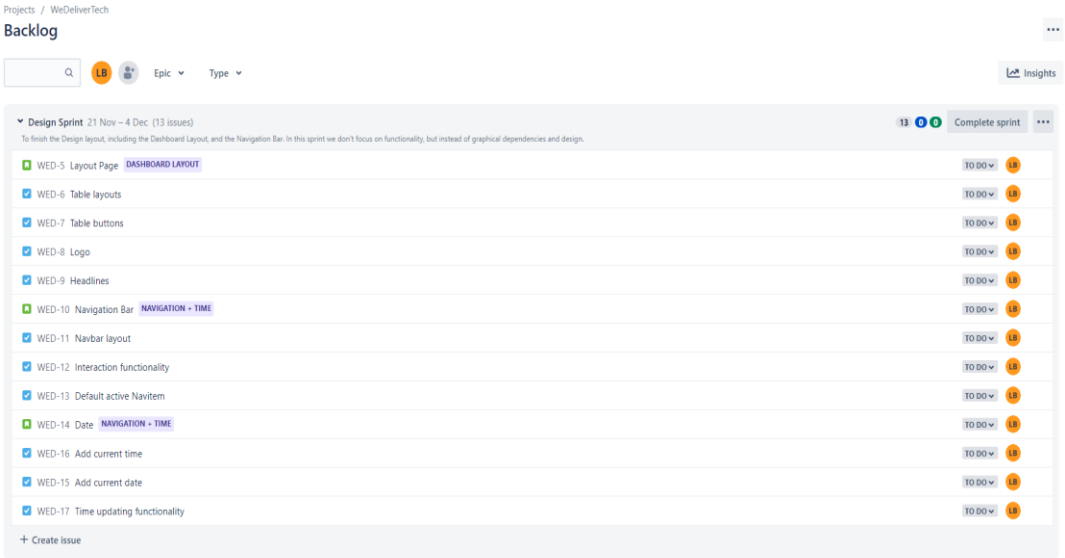
Reflection Report – WDT

The initial strating point included making a roadmap of the Epics that we wish to pursue.



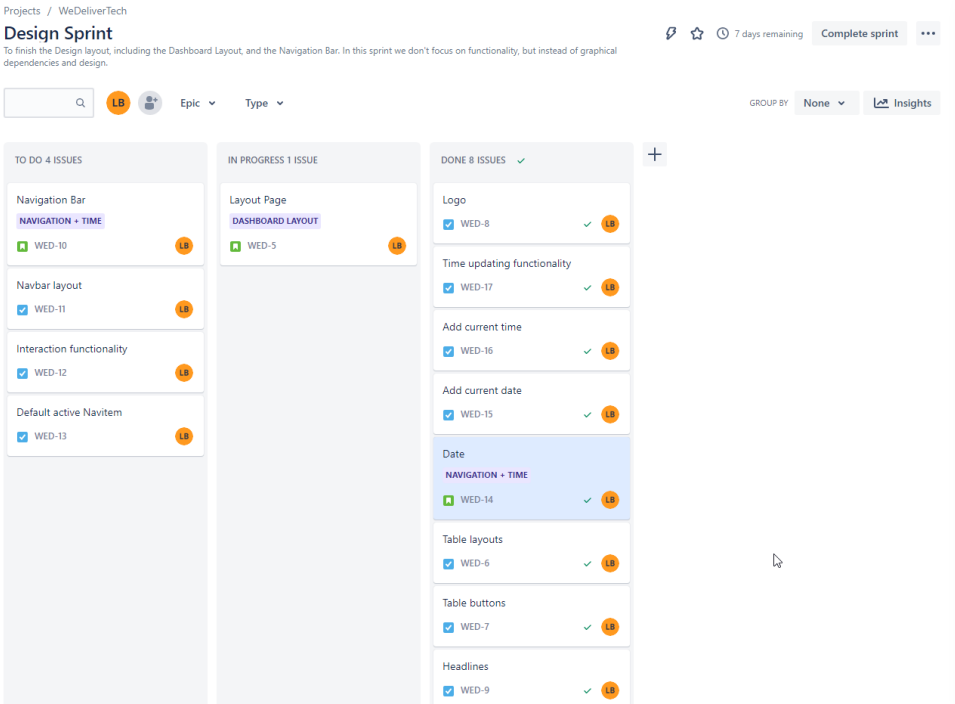
The Dashboard Layout, Navigation + Time, Tracking staff and Tracking deliveries are the Epics for this project. These four Epics are based on the context of each working part. They all work sepraretly and don't rely on each other, but are needed for the overall flow of the application.



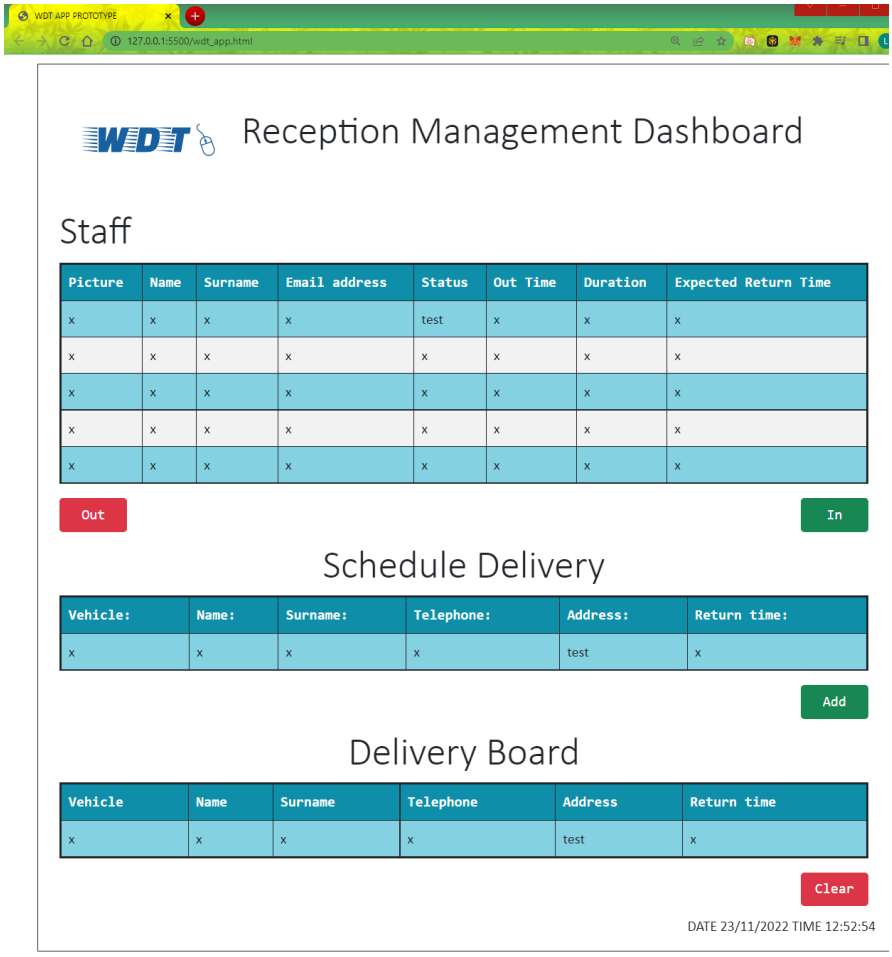
The workload was split into two sprints, namely Design and Funtionality Sprint. The design sprint would focus on the layout and the visual features of the application, while the functionality would give the dashboard the needed functioanlity to operate. This decision was taken as functionality would be easier to implement and test with a working layout to show the results. The design sprint had 3 stories namely Layout Page, Navigationg Bar and Date. These stories had numerous issues assosicated with them. The decision to work on the layout first was to establish the working environment of the application.

As we see in the backlog, the various issues were chosen to fit the story. To do the layout page, we needed to make the table layouts of the various tables, and the corresponding buttons. We also had to add the logo + headlines to finish the layout page story.

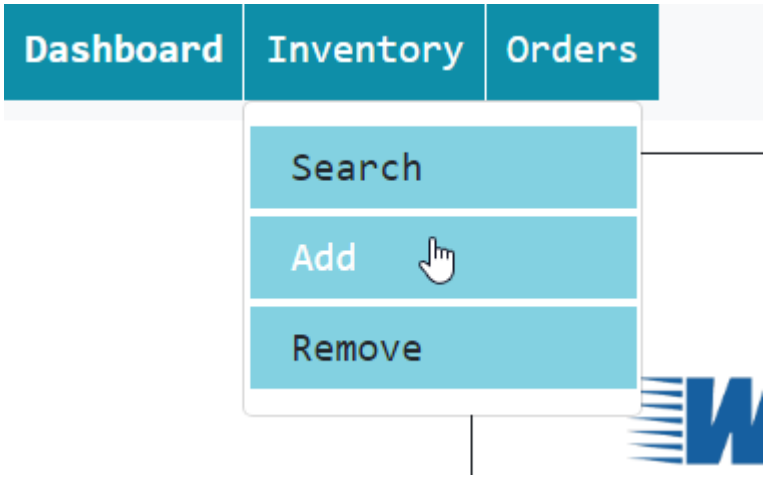
The Design sprint board started with the “Layout Page” Story as the first story to be in progress, however, the first one to be finished was the ‘Date’ Story.



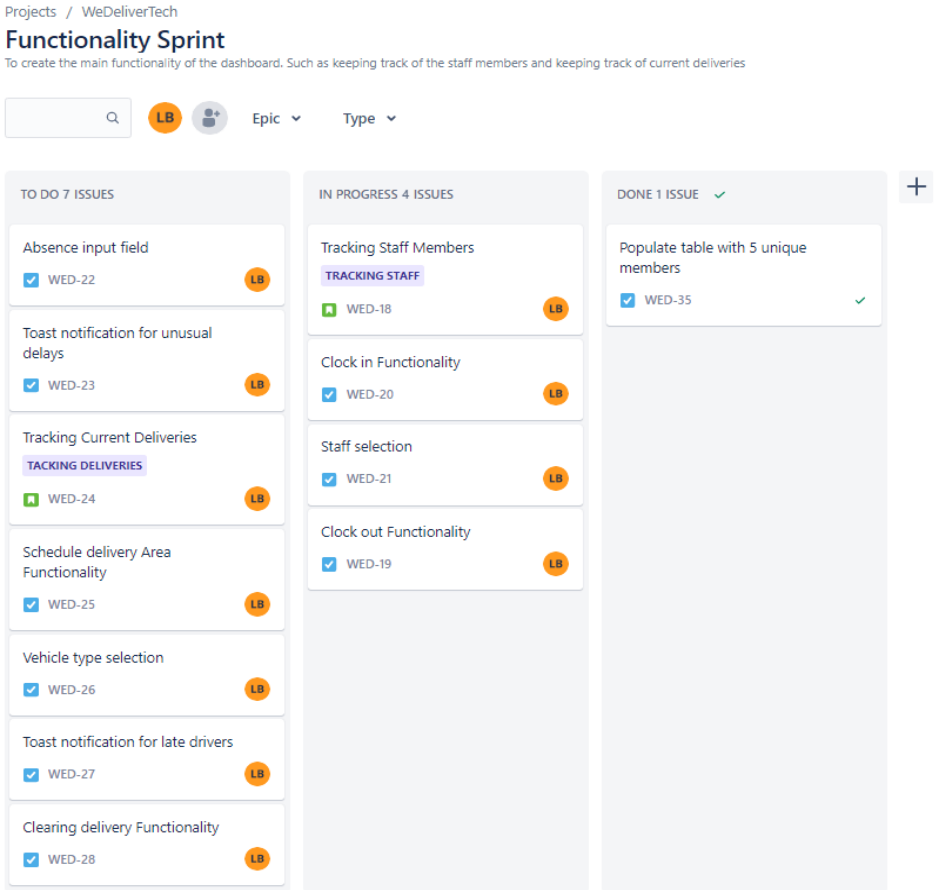
When we approached the end of the “Layout Page” story. We had the following:



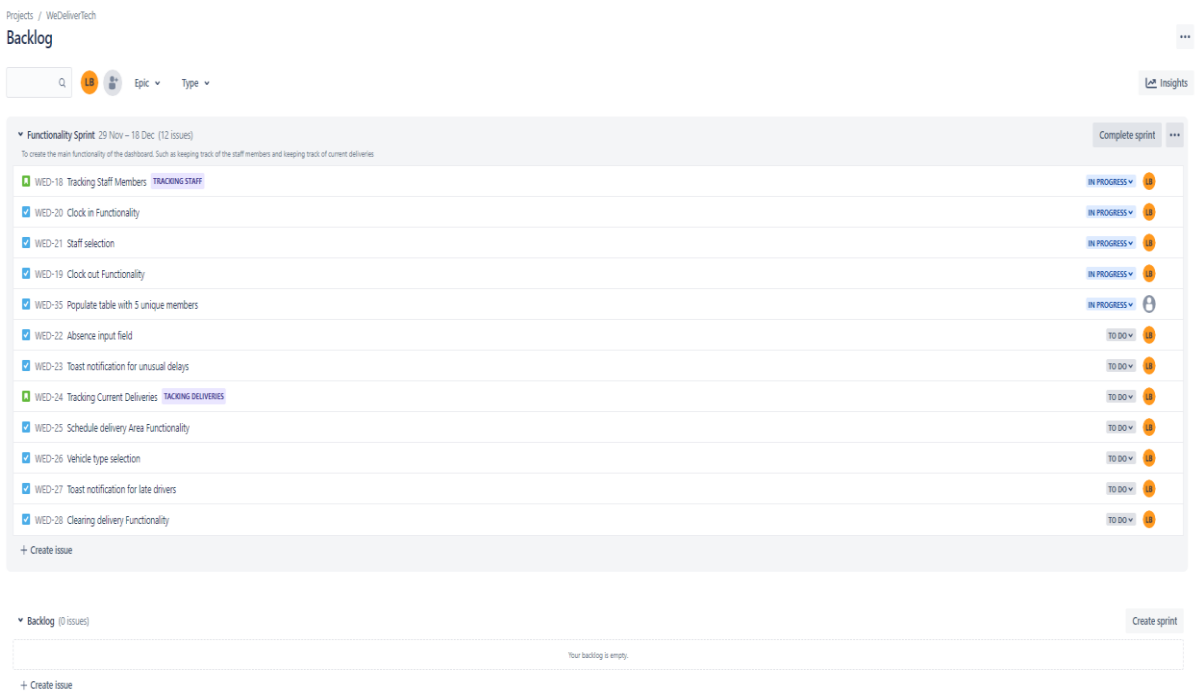
Here we have finished every task of the Layout Page + Date story, therefore they are marked as done. The navigation bar, the final user story in the design sprint, was completed prior to the 2-week timeframe..



The design sprint was completed ahead of schedule, which allowed for additional time to be dedicated to the functionality sprint. The first user story in this sprint focused on tracking staff members, including populating the staff list, adding clock in/out functionality, and selection.

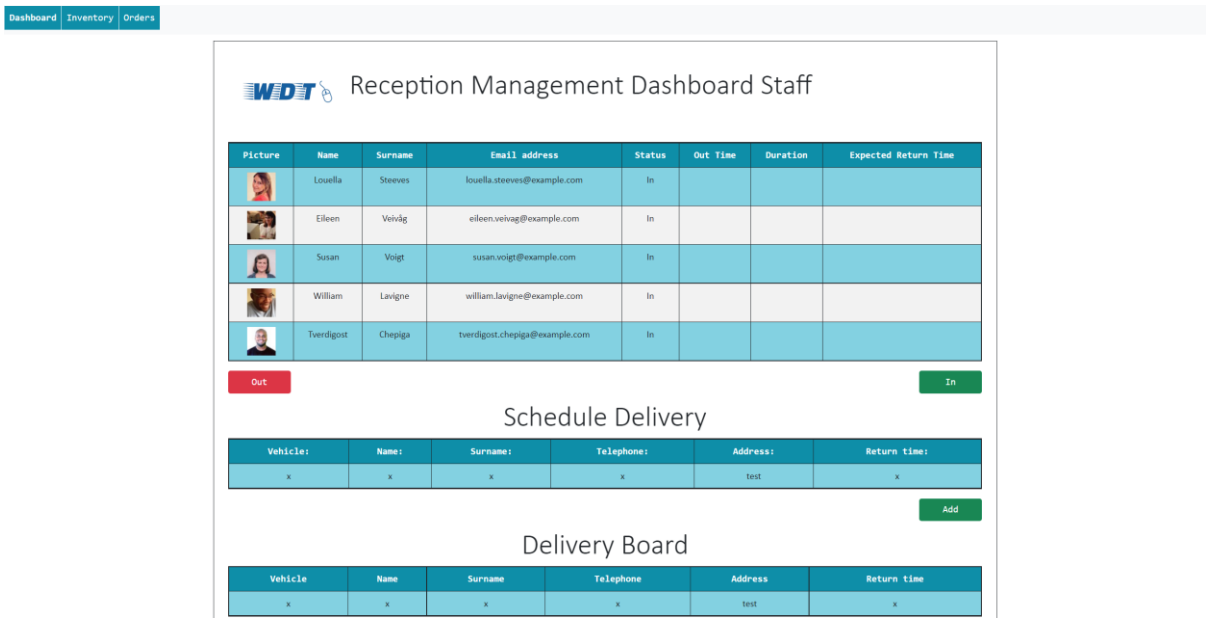


With the following Backlog:



In order to meet the customer's needs, we identified four key areas, or "Epics," to focus on based on the requirements provided in the assignment instructions. We created stories and issues to address these objectives and made progress on the functionality sprint. The specific areas of focus, also known as Epics, included:

1. Tracking staff members who have clocked out of the office
2. Tracking current deliveries of orders to customers
3. Displaying the current date and time at the bottom of the web page, updated every second
4. Implementing a layout including a navigation bar at the top of the screen



The table automatically populates with 5 staff members using the API from

https://randomuser.mehttps//randomuser.me/api/ The soft limit of 5 staff member is easyily changeable inside the source code, at line 116.

```
114 const staffEmployees = []
115 function populateTable(value){
116     if(value==5){ // This fills the table to only 5 for now, changing this
        number adds more/less staff
117         return;
118     }
119     let staffMember = staffUserGet('Staff');
120     staffEmployees.push(staffMember);
121     $("#dashboardBoard tbody").append(`<tr><td><img src=${staffMember.
        picture} /></td><td>${staffMember.name}</td><td>${staffMember.surname}</
        td><td>${staffMember.email}</td><td>${staffMember.status}</td><td></
        td><td></td><td></td></tr>`);
122     populateTable(value+1);
123 };
```

PopulateTable is a recursive function that is initialized with a value of 0 on page load. The function increments the value by 1 each time it is called, until the value reaches 5. At that point, the function terminates and stops populating the table. Changing the if condition inside the function will change the amount of staff members created on load.

Management & Challenges

The focus of the project was on implementing key features such as API calls, staff scheduling, delivery management, and visual effects. One of the initial challenges was selecting staff members efficiently, which was resolved by utilizing the hasClass() function from jQuery to identify the relevant rows. To enable multi-selection, we utilized control and jQuery functions such as hasClass(), siblings(), and removeClass(). This required a thorough understanding of how these functions worked together to accurately update the selection when a new staff member was chosen, unless ctrl was being held.

```
$(this).addClass('selected').siblings().removeClass('selected');
```

This line of code allows the user to select a specific staff member by clicking on the corresponding row. The current row (indicated by 'this') is given the 'selected' class using the addClass function, while the siblings of the current row have their 'selected' class removed using the removeClass function.

By using the jQuery function hasClass(), we can target the staff members whose status we want to change. We can do this by iterating through the board and identifying which rows have the selected class.

```
Array.prototype.forEach.call(board, child => {  // Finds every element that is selected on the table and adds the name value to an array
    if($(child).hasClass('selected')){
        selectedName.push(child.cells.item(1).innerText);
        selectedLastName.push(child.cells.item(2).innerText);
    }
});
```

We can then append the name and last name of the staff member to an array, which we can use to update their status. This knowledge allows us to efficiently target and modify the status of specific staff members. This was the main challenge that was overcome, a lot of other ways were attempted by using ID and targeting, but nothing proved to be as efficient as the various class funcitons by jQuery.

Updating both the staff members' properties and the table simultaneously posed a challenge, as they are separate entities - one being an object and the other a table data cell. To overcome this, we stored all of the staff members in an array and used the updateTable function to update the table each time a change was made. This function took a value as input, which we used to locate the specific index (staff member) that needed to be updated. We were able to find this index by using a for loop to iterate through all of the 'staffEmployees' and search for matches within the staffOut or staffIn functions

```
for (let i = 0; i < staffEmployees.length; i++) {
  const staff = staffEmployees[i];
  if (selectedName.includes(staff.name) && selectedLastName.includes(staff.surname)) { //If there is a match
```

By using this method, we can determine the index of the currently selected staff member using the variable i. We can then pass this value to the updateTable function to update the relevant staff member. This allows us to effectively update the table with the correct information for the selected staff member.

```
return updateTable(i);
```

```
function updateTable(value){
    let memberValues = Object.values(staffEmployees)[value];
    let rows = $("#dashboardBoard tbody").children();
    let currentRow = rows[value].cells;
```

We could now change the rows values with the correct information from the selected staff member.

Conclusion

In conclusion, this project required the coordination of multiple functions in order to be successfully completed. Jira was a useful tool for staying organized and on track, as it allowed us to prioritize and focus on specific tasks on a daily basis. The use of sprints helped to focus on specific goals, such as design or functionality, and the creation of epics provided a clear understanding of the tasks that needed to be completed before the deadline. Challenges were met throughout the process of developing the final product, but with enough ingenunity and using Bootstrap & jQuery, we were able to overcome the tasks at hand.