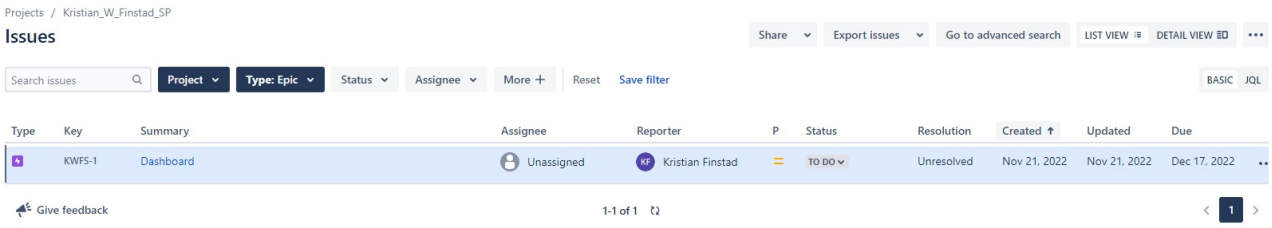
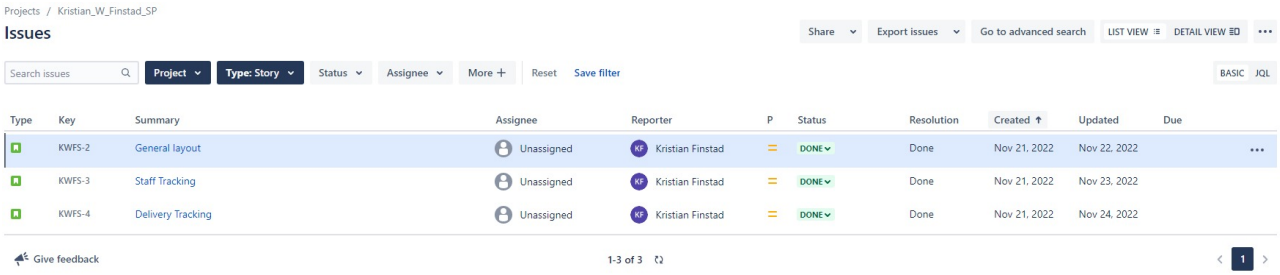
Reflection Report

Planning

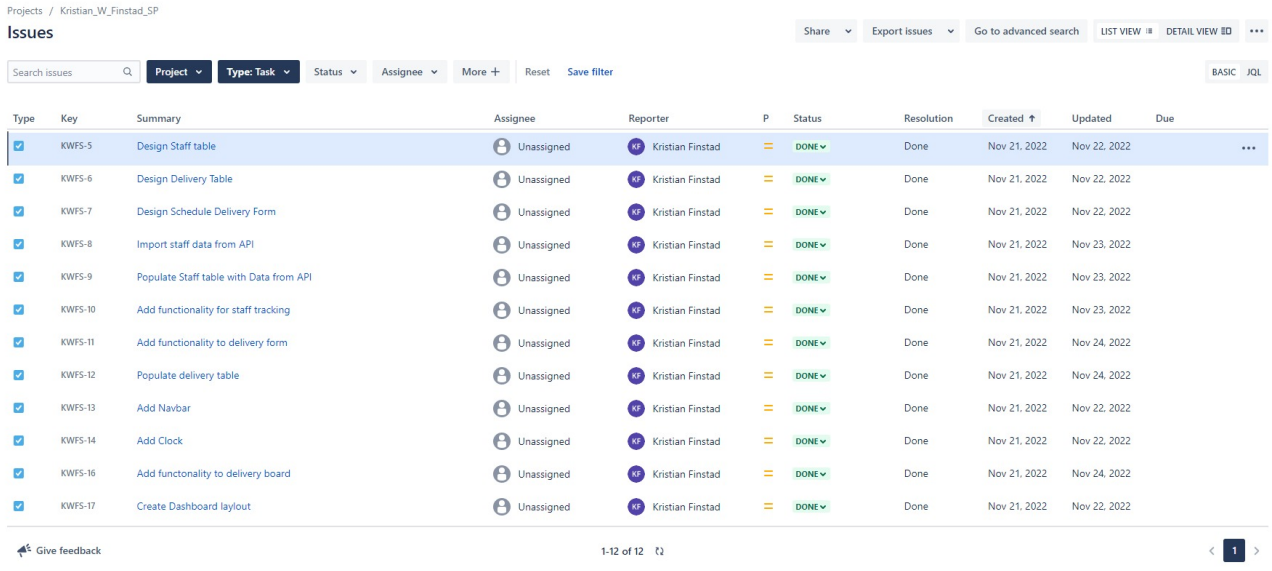
I created a single epic for this project with the goal of creating the dashboard as described in the semester project assignment.



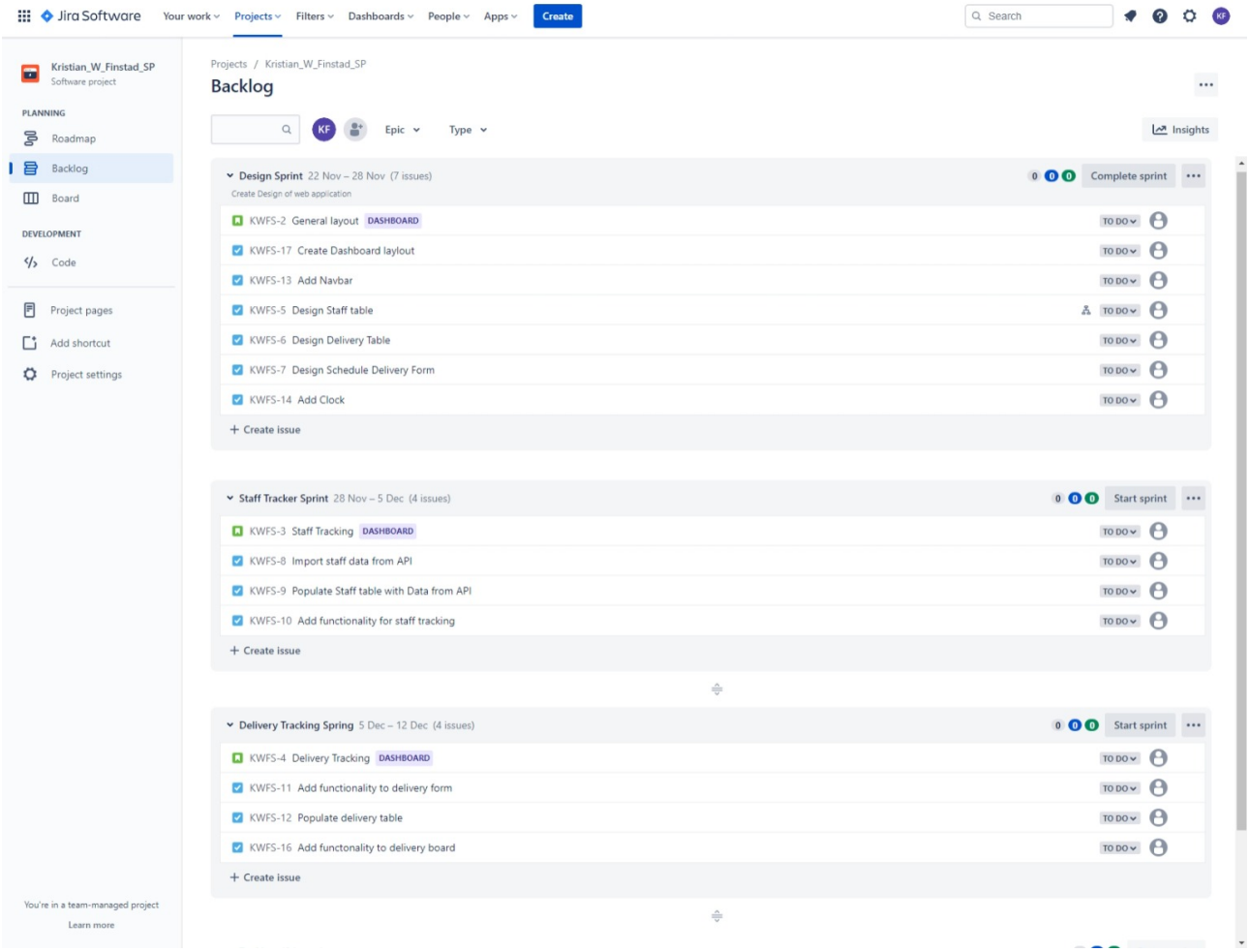
I created three stories that representing a fully functional feature, adding them as child issues to the epic.



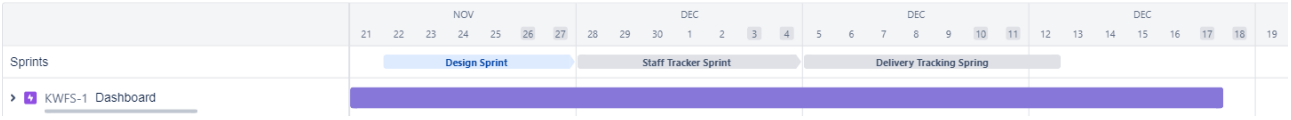
After breaking up the features in three categories, i broke these down into smaller tasks representing the stages of each story. I did not wanted to micromanage the project by making each required function a Task. These tasks could be added as child issues in the Tasks created.



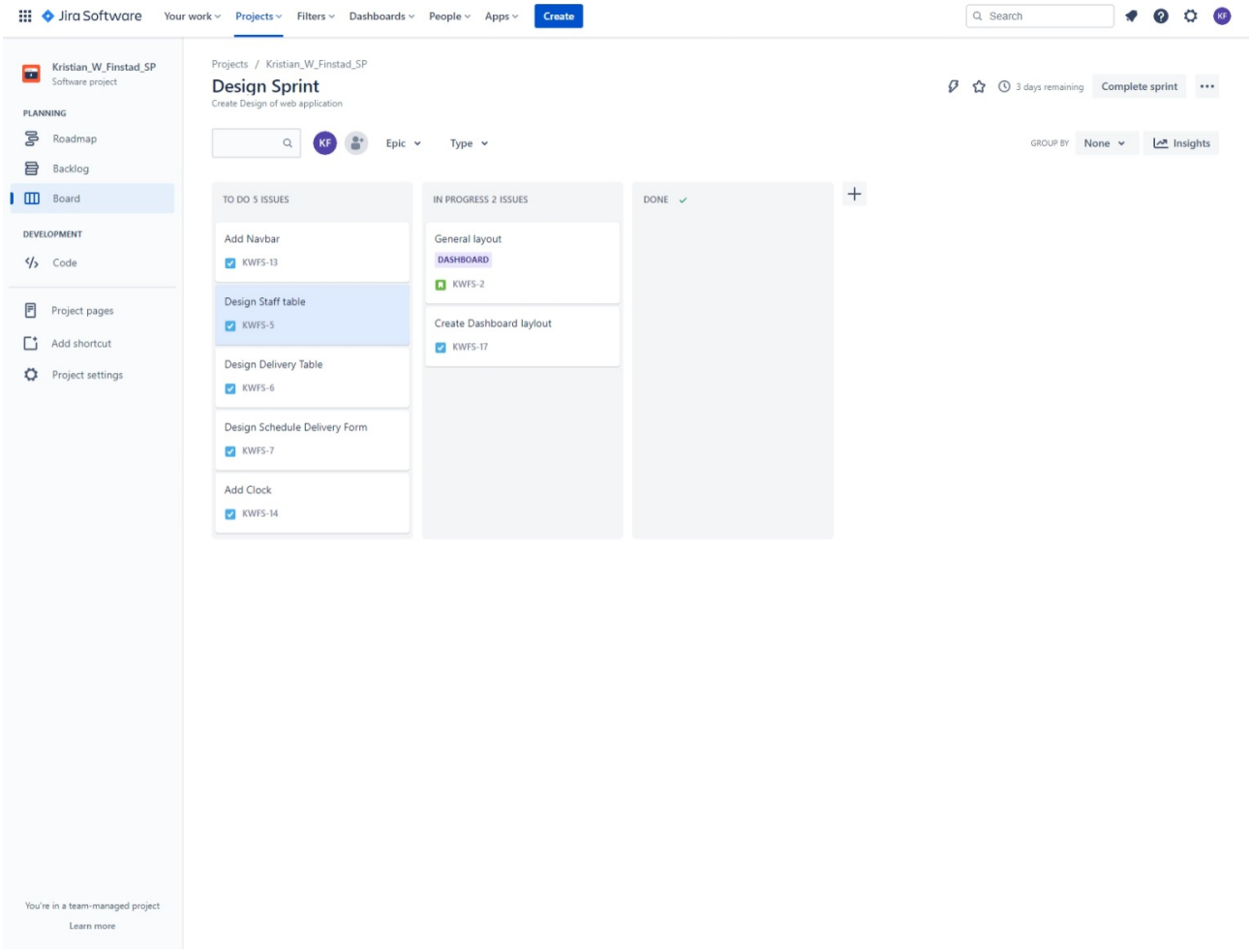
In backlog I created 3 sprints each lasting a week. Each sprint have the goal of adding a functional feature to the application, with tasks assigned accordingly.



The roadmap shows how the Epic of creating the dashboard have the due date of the semester project. Each sprint occupy a week representing the planned progression of the project. The last week is deliberately left open in the planning phase for any unforeseen issues or delays that may occur.



My Jira board only have the three default columns of To Do, In Progress and Done. In this project I want to mainly focus on the work and less on managing the project. Other columns could have been added, but would probably just be skipped for this specific project.



Design

I started creating the general layout of the page following the provided mock-up and following the branding guidelines and adding the features not described by other stories. I did not add any extra to the design, though a background colour to the navigation bar would have been nice.

A zebra striped table would have been nice, but since no secondary colours was provided it would have broken the branding profile. On the same premiss I used white background on hover with a 0.1s transition for the hover effect and to show selected table row. Rounded border lines for tables using Bootstrap can be quite painful, so I left it out since it is of a minor aesthetic concern.

I used the table as container for the delivery driver form to ensure the design was consistent for all elements on the dashboard as shown in the mock-up. I also added a custom error style for the inputs unsure if the Bootstrap error handling would fit the branding profile.

I made the footer with white background, low opacity and with a fixed position at the bottom and added the clock element to ensure clock is visible.

I also added Bootstrap modals for user input since they look nicer than the default browser *prompt* and *confirm* dialogues. I used the table header colours for modal and toast header to ensure consistent design.

Programming

Since the application is supposed to object oriented, I wanted to make a main Application class to hold values and methods regarding functionality. Using encapsulation would ensure only the run method would be publicly available. After consulting a teacher, I wrote all functions as standalone since it was implied that was expected of the functions mentioned in the grading criteria. But I did write the functions in regard to easy transition to methods inside the application class.

I tried to keep all code regarding the grading criteria within the scope of each function, extracting only what was required for consistency in separate functions.

I made the API call with an old fashion **XmlHttpRequest()** using a query string to limit the data only to what was required. I would prefer using **fetch()** in a later version for easier handling of more complex API calls.

I do not quite understand what is meant by *“Inheritance is used in object creation, using the data from the API call“*, but the StaffMember class do inherit the Employee class. After being encouraged in a previous assignment, only used a single object as input parameter for the class constructors. I would have preferred using a single parameter for each value defining the value type in the doc string due to the data transfer object not being defined locally. In addition using an object literal on instantiating the object feels a bit like boilerplate code. The data was stored in a private field inside the application class for later handling. This data is currently available outside the Application instance due outside functions requiring access.

Since there is no id’s is included the data models, I used the email address for staff member objects, and telephone number for delivery objects, as unique identifiers when searching in data arrays.

I created a simple unique id generator for creating ids for generated table rows and toasts with the purpose of easier document queries.

In the form, I used a try/catch/finally block to avoid page reload on submit or in the occurrence of an error. I also extracted the value from each field individually for easier handling of validation and validation error.

The validation of the time field format was a bit redundant due to the default format provided by the input field. But it’s added as an extra safety, plus it is always fun to play around with RegExp.

I do not entirely understand what is meant by *“Inheritance is used in the Delivery Driver object creation”*, but the DeliveryDriver class do inherit from the Employee class. This class also have a single object as input parameter on the same premiss as the StaffMember class. Object literal is also used when instantiating due to the nature of the input values.

I stored the SVG icons as text in wdt_app.js for easier access and configuration.

As an extra feature, I added double click for the table rows for set out time for staff member, and clearing deliveries on the delivery board.