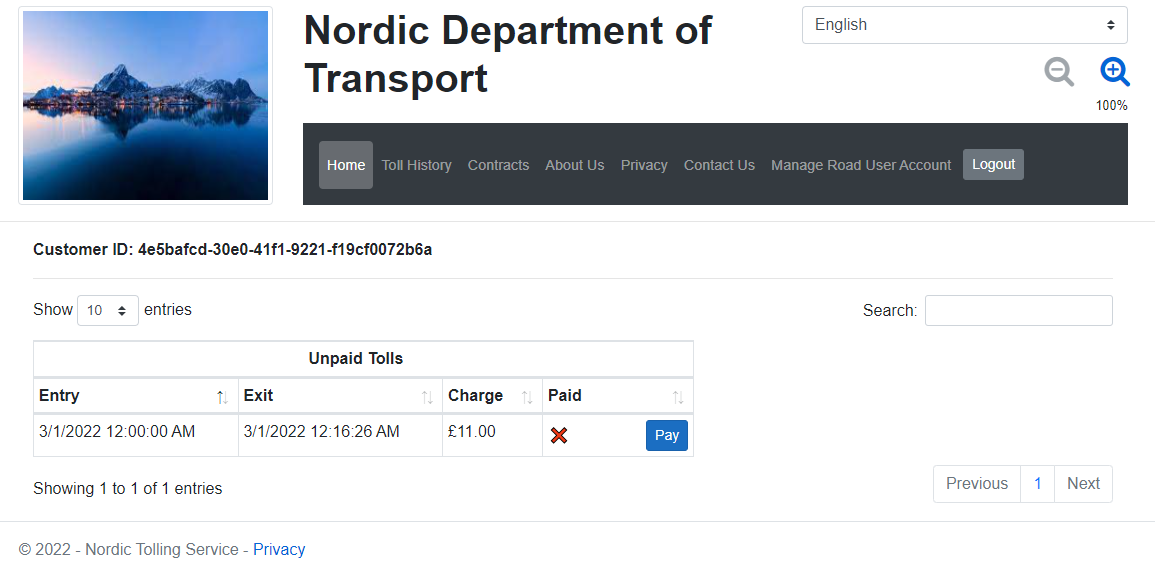
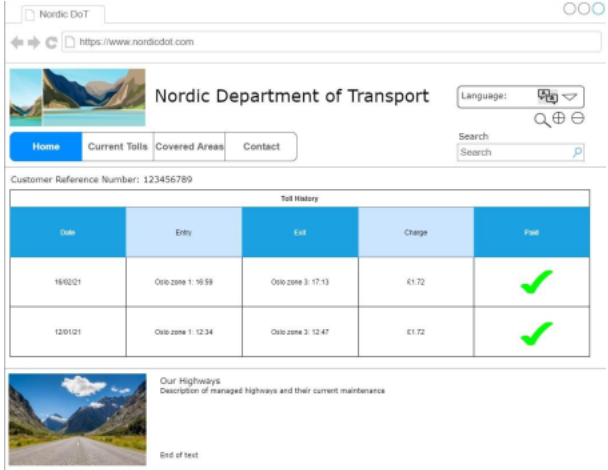
Critical Reflection Report:

## Differences in design

Our first change to the UI design of the system was to make the navigation bar include all the different needs for our site and including some oversights in our prior design and tailor it to each user of the system (road user and toll operator), for example we changed the current tolls page to be the home page for both road users and toll operators so that they can see the most important information such as unpaid tolls as soon as they login to the system. In our original wireframes we did not anticipate the number of separate pages we would need to ensure there wasn’t an overabundance of irrelevant information on any one page, to remedy this we added more pages, thus more navigation options to the navbar. We also decided that having the navbar under the header created an unnecessary section of white space so moved the bar to make, we believe, a more readable and concise header and navigation because it is an integral part to the website.

Another change we made in the interest of readability was moving certain UI elements and adding text to inform the user on what changes they are making to the system. For instance, the keyword search bar being in the header made it slightly ambiguous as to where and what you were querying, so we moved the element to the main body beside the table where it was more clear you were querying the table's data rather than a generic search across the page. Furthermore, we opted to change the magnification element to be two elements instead of three to help convey the use of the buttons without superfluous additions, these two elements are prolific on many high-traffic sites and so will lend themselves to improving universal understanding. Then we added a small display text below the buttons to tell the user the current scale of the magnification.

The final alteration we implemented was to improve the database and the applications access to the data within. We altered our static model to add three additional attributes to the Invoice class - *Entry Time*, *Exit Time*, and a *Paid* boolean. This was to aid in the tracking of the cost of the invoice, whether it has been paid and how the charge is invoked. In addition to these changes the class now required access to an *InvoiceService* - to manage and access the data within the class and help ensure information was updated and available for the application during runtime. We also chose to deprecate certain classes, such as *VehicleRegistrationDataService*, *SystemManager*, *RFIDService*, *AccountManager* and *AdminAccountManager* as these were surplus to requirement, as the functionality they contained had already been included in other classes, to simplify the system. The TranslationService was also removed, because we opted to utilise the standard .Net core localisation service, coupled with Azure translator to expedite our implementation. We also chose to add a *PreferenceService*, as this made more sense than encapsulating that functionality on the Preference class itself.



## Project management

We started the project using blazer framework as we believed it would be perfect to recreate our prior designs in, however quite a few problems were encountered and a few weeks into development we decided to switch framework. Some of the key problems came from our lack of understanding of Blazor as we all had zero experience using this framework, one key problem was interoperability between *.razor* and *.cshtml* files making development difficult this could have been solved using online resources and documentation however most of useful resources were for a more modern version but because of limitations with installed software at university we were unable to use any version higher than 3.1 which didn't have many useful resources to help us with our problems.

After moving away from the Blazor framework we decided to use pure ASP.NET Core MVC which as a group we had more experience using, this made development a lot smoother, we used GitHub for version control, this let us all collaborate as well make sure we had a place that kept important releases (when key parts of the system were tested and working correctly) we also made branches to save our old code from the Blazor framework as it still contained useful code, our branches were separated so code could be saved and some problems could be worked on together. Furthermore we also used separate feature branches for key parts of the system whilst they were being worked on - for example when reorganising how the simulation data was generated (*feat/simulation-tests*) and adding the navbar (*feat/navbar*).

### 

### Scrumban method - Trello plus weekly meetings

At the start of development we created a Trello board that separated out the system into manageable tasks for each member of the team to work on, early on in development we met once a week for a meeting and to work on the project then as we got closer to the deadline we increased it to 2-3 meetings a week to make sure we were on schedule and could meet the deadline. We set up different categories to organise tasks:

* Backlog - Tasks that are not key parts of the system
* Remaining - Key parts of the system that still needed to be worked on
* In Progress - Tasks that have been picked up and are being worked on
* Complete - Tasks that have been tested and are finished
* Obsolete - Tasks that are not needed thanks to changes in design or other factors.

The Trello board helped us to all have a task that could be currently worked on and made sure we didn't all work on the same thing unless it was given a high priority or one member of the group was having issues completing it.

