Dude Where's My Bike?

Website: http://dudewheresmybike.ie/

GitHub: Already shared with @aonghus

Team Members: Tomás Kelly / Jörg Striebel / Will O'Donohoe

Student-IDs: 21207079 / 21205773 / 21200503

Overall Contributions:

Tomás Kelly: 36%

Jörg Striebel: 32%

Will O'Donohoe: 32%

COMP30830 Software Engineering

UCD School of Computer Science

15th April 2022

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

Project Overview

Introduction

The project objective for every team was to come together, using the skills developed throughout our course to produce an interactive website that integrates technologies from various modules to realise this goal.

It is well known that collaboration with other team members in software projects is paramount for software engineers. Effective teamwork lays the foundation for efficiently sharing high-level skills to complete complex software projects on time and within budget. Conducting our software engineering group project – developing a standalone web application for Dublin Bike users – aimed not only to develop and deepen the skills and knowledge in various fields of computer science but also to learn how to work collaboratively together on a common goal.

One central part of our project was project management. Two well-known project management methodologies, namely Waterfall and Agile, are commonly applied in software engineering projects. However, unlike the traditional Waterfall methodology, agile project management frameworks such as Scrum have become more and more popular in recent years since it allows software engineering teams to adapt quickly to changing requirements, a common phenomenon in software development. Hence, becoming familiar with the terminology and workflow of Scrum played a crucial role in our daily routine while collaborating on this project.

Another essential part of conducting this group project was to apply the knowledge gained during the first and the second semester, ranging from front-end web development to more advanced back-end applications. Github – a distributed version control and source code management tool – was used to facilitate collaboration and keep track of our development process. As all team members used to be unfamiliar with using Github, applying the Github workflow turned out to be a challenge at first.

Overall, this project offered an excellent opportunity to effectively collaborate with other team members and realise our first larger software project. In addition, we could apply the knowledge gained throughout this program and broaden our horizons in various computer science fields.

Objectives of the Application

The standalone web application “Dude Where’s My Bike” aims to help Dublin Bike users find the closest station with bikes available to start their journey. In addition, it informs users at the end of their cycling tour through Dublin about stations that have free bike stands available so that they can return their bikes.

What’s more, the application does not only display relevant and valuable current information about bike station occupancy and current weather in Dublin, but it also provides a future

trend about the availability of bikes. This occupancy prediction for the upcoming hours and days is based on a Machine Learning model fed by past occupancy and weather information, allowing users to see the occupancy trend up to 48 hours in the future.

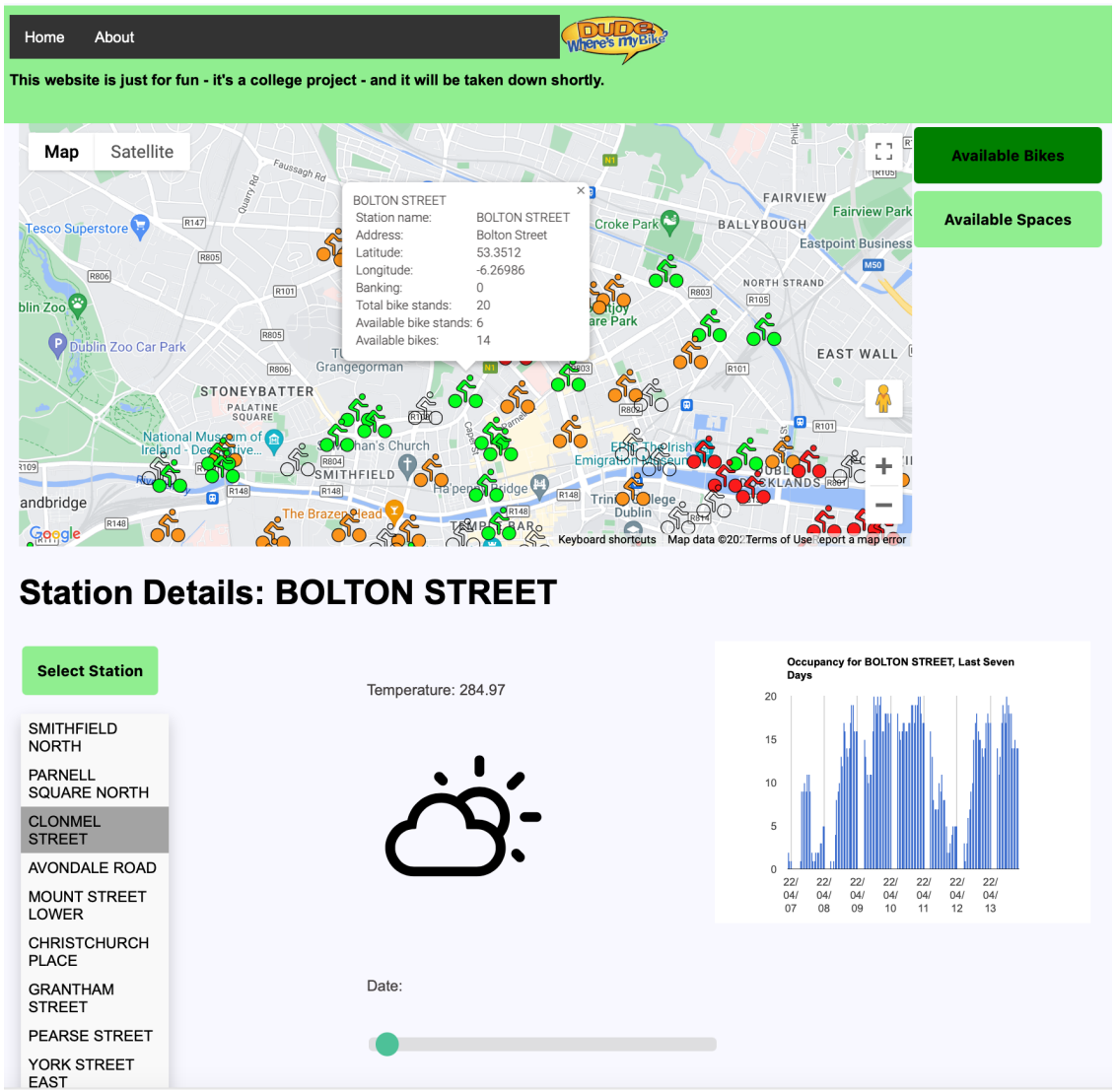

Figure 1: Main page of “Dude Where’s My Bike”

Target Users of the Application

The target users of our “Dude Where’s My Bike” web application are tourists and locals who want to travel environmentally friendly – whether for daily commuting or merely leisure activity – around the heart of the Dublin city centre.

While the current version is targeted at desktop users rather than mobile users, we could imagine optimising this app so that it can be conveniently used on mobile devices. The necessities for a responsive web design have already been considered during the design phase.

The service will be freely available, and no registration process will be required to use the service. However, it could be imagined to provide a log-in account for privileged users at

some point in the future, offering more sophisticated features to improve the overall user experience.

Structure and Features of the Application

The primary focus on designing a single paged front-end in HTML and Javascript was simplicity and usability - following strictly the motto: less is more. To begin with, a relatively large and zoomable Google map allows the user to see all bike stations in the Dublin city centre at a glance. What's more, the user can choose between two different modes to either show the occupancy for available bikes or available spaces.

The distinct feature, however, is indicating the occupancy by coloured bike markers rather than displaying overloaded heatmaps. While, in our opinion, heatmaps are a great way to illustrate larger areas on a map such as the weather forecast, utilising coloured bike markers was considered to be a more suitable approach in our use case. More specifically, in contrast to visualising heatmaps on the map, the coloured markers approach retains the visibility of the map itself, meaning streets and buildings in the background will still be visible.

Another remarkable feature is the range slider, enabling the user to look up occupancy weather predictions up to 48 hours in the future. While it would have been feasible to utilise a so-called data & time picker window, which is commonly used for date and time selection on booking platforms, a range slider appeared to be more user friendly as the range slider can simply be dragged from left to the right without the need of clicking on any sort of input windows.

Last but not least, instead of simply displaying weather information in plain text on the screen, illustrating weather descriptions in the form of graphical weather icons has further enhanced the usability of this application. The decision to visualise the weather information graphically follows the slogan "a picture is worth a thousand words".

Architecture

With a constrained time frame for project completion and a new team we decided very quickly to stick to the general architecture laid out in lectures. The 'big-picture' view of which is as follows:

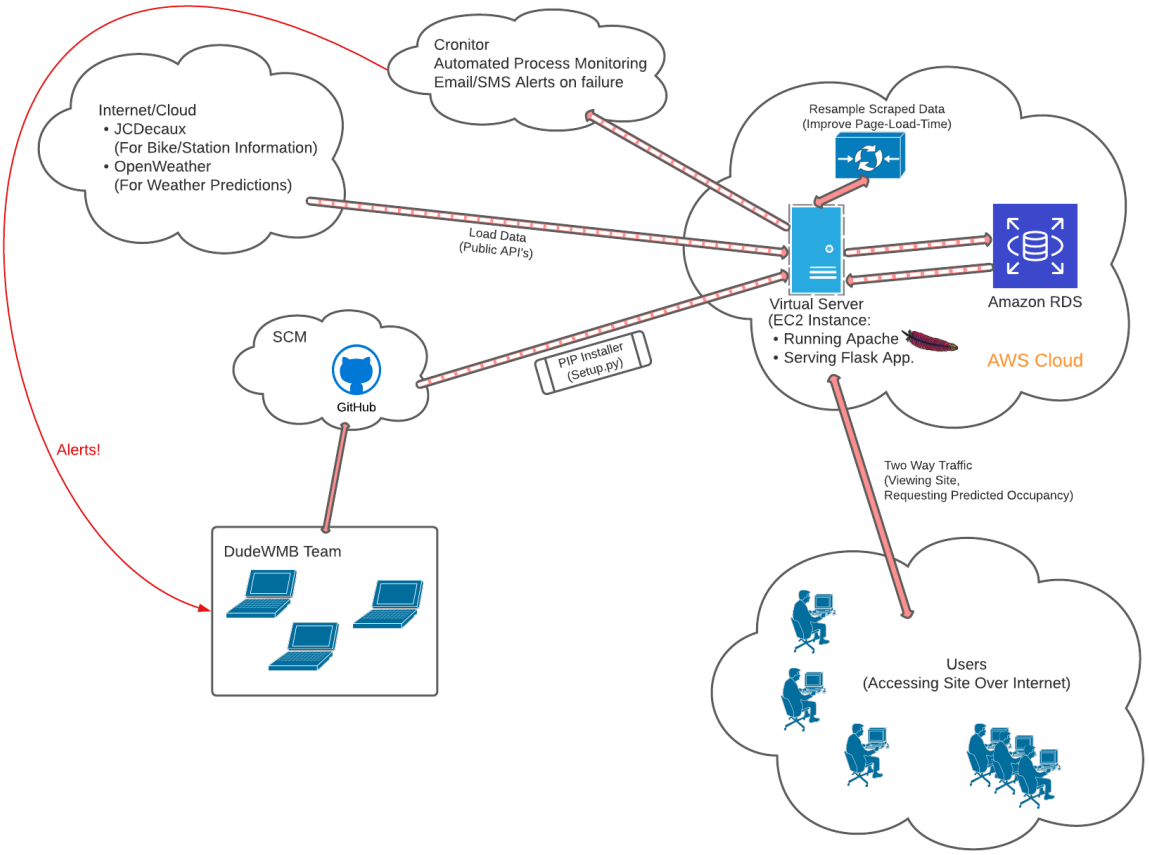

Figure 2: Overall Architecture of the DudeWMB System

The DudeWMB team used a number of technologies to bring our web-application to life.

Back-end

DudeWheresMyBike.ie runs as a Python Flask application, which is served using Apache WebServer. Apache uses the “mod_wsgi” package, which implements a simple to use Apache module to host our Python web application (which supports the Python WSGI specification, thanks to Flask and some basic configuration).

Our python application is packaged and distributed with Distutils, which is the standard for distributing Python Modules. Our ‘setup.py’ allows us to install our application directly from our GIT repository, using standard commands like “pip install”. Installation in this method guarantees that all the dependencies specified in setup.py are met, each time we install. It also allows us to define “entry points” for our programs (essentially convenience aliases) that allow us to run our programs from any working directory. Configuring this took some time during the first and second sprint, but once it was configured and running it saved us a lot of time. This gave confidence in a repeatable, reliable installation method.

The Python application / Apache all run on an Amazon AWS EC2 Instance, running Ubuntu. Running on Linux gives us ready access to schedulers etc. and we use Cron to automate running of our background processes:

- The data scheduler is a Linux script that allows us to stop/schedule start/show the state of our background jobs easily. We defined an entry point for this script so we can inspect the state of the scheduled jobs from anywhere
    - The data loader is a python process running every two minutes that calls the JCDecaux API and then the OpenWeather.org API in turn, parsing the received data and storing it in the back-end database
    - The data resampler is a python process running every hour that processes the last hour's data and produces a single 'hourly' mean statistic to represent that hour. The sole purpose of this process is to give us a smaller data source that is more manageable to display. The process recovers automatically if it ever fails. It simply looks for "the last hourly record" produced and then resamples/analyses every hour from that point forward until it has caught up

The scheduled processes use the cloud-based service "Cronitor" (https://cronitor.io/) for monitoring and reporting. As each job starts it records the "process start" event by calling the cronitor API. Errors can be registered using the same technique (as required). When the process completes, completion is also registered in the same way. If our background process ever failed for any reason (and it did!) we were notified by email immediately. The cronitor service is free for up to five jobs and afforded us an excellent solution for automated monitoring.

All data collected and processed is stored on an Amazon RDS database. The cloud-based database gives us the reliability we're looking for, as we build a large data source to provide a good input for our predictive model.

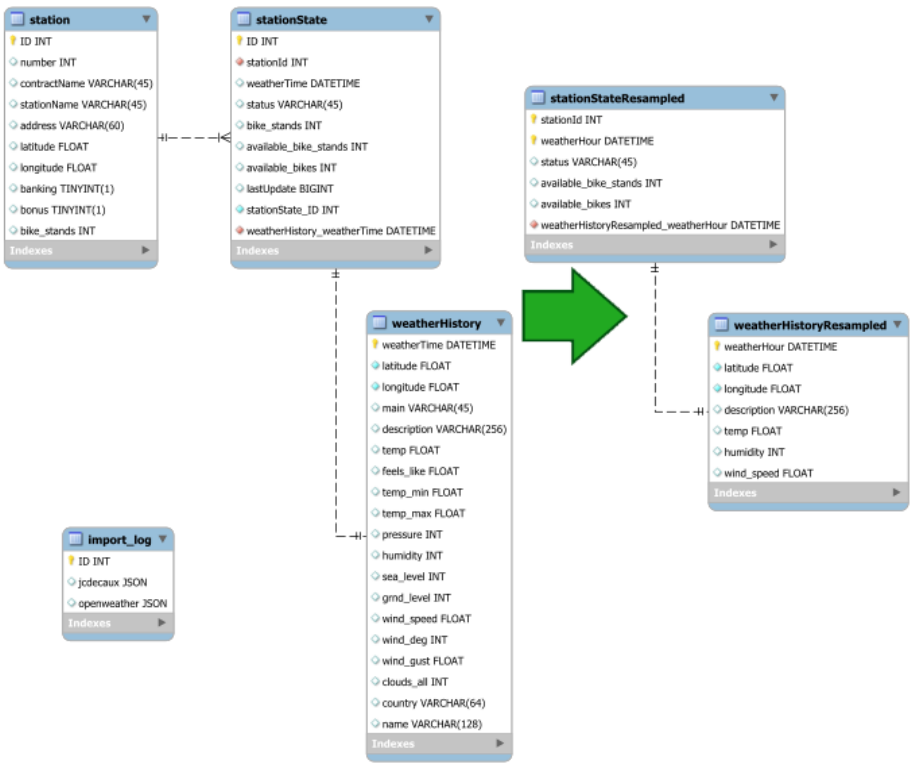

Figure 3: DudeWMB Data Model: Loaded Data -> Resampled Data

Front-end

The front-end of the application is done entirely in Html, CSS and Javascript. The data driving the front end is supplied from the back-end in JSON format. The google visualisation API is used for presenting/charting historic (last two weeks) occupancy data per station for the end user. And of course the Google Maps API is used for the main display, showing station locations on the map across Dublin. Additional layout tools (Bootstrap etc.) were dismissed early in sprint one as none of our team has any experience in them.

Data Analytics

All of the model development and analysis was performed in Jupyter Notebooks using pandas, matplotlib and sklearn to clean and prepare the data (transforming some features), analyse the data collected (looking for correlations), train (fit) the models and then evaluate the results. A variety of data models were considered (Linear Regression, Logistic Regression, Random Forests) and the model with the highest accuracy was chosen.

If there were more time it would be interesting to test and benchmark the impact the use of each model had on the page load time of the application. Time constraints made such an analysis impossible, but it is definitely something that we would like to explore. It's quite possible that accepting a slightly weaker model would be a better choice - if choosing that model had a measurable impact on page load time.

Development Environment

SCM

We used GIT as the VCS/SCM for the project. It is a tool our team took time to grow accustomed to - but forms an excellent open platform for collaboration on a mixed technology project like a web application, as it is (for the most part) content agnostic.

IDE's

Our team used VS Code as the main development environment for the project. It is available for both PC and Mac, it has plugin-based support for GIT, for Python, for Html, Javascript and CSS. It supports both local and remote connections making it a suitable choice for this project where a significant portion of the work took place on a cloud-based AWS EC2 instance running Ubuntu. We were happy with the choice of IDE - it supported our collaboration nicely. One team member experienced some difficulty with the configuration of the integrated terminal. It does seem likely a remove/reinstall would resolve those issues. So our team would be confident recommending the IDE to future teams taking on this module. Of particular use are the advanced GIT branch, merge, sync, stash tools etc. allowing us to work on the same object simultaneously with the confidence no effort would be lost.

OS

Our team used a mix of PC and Mac as the development platform. This was not without issues. For some python modules it was not possible to get exact version matches across the two platforms. We struggled greatly (i.e. it was very time expensive) to use Anaconda environments to synchronise our development environments across the two OS's. PIP/PIP requirements proved a slightly better choice in terms of ensuring consistency among team members. If our honest opinion was sought - we think for a small team on a tight timeframe project like this, manually managing inclusions/exclusions from the development environment and simply choosing a single tool (PIP in our case) was the preferable course of action. For larger projects that clearly wouldn't be an option. But if we could claw back the time spent in the first half of sprint 00 on this issue, we would do so without hesitation.

Functionality

The DudeWMB application performs acceptably as it stands. Page load times - a key metric for a web application - are a touch slow at around 2-3 seconds for the active home page. Our initial thoughts - with no experience in using the Amazon EC2 t2.micro environment - were to design as 'current/live' a website as possible. Our intent was to load all data on demand.

The application depends on data scraped from open API's by JCDecaux and OpenWeather.org to function. We are scraping data every two minutes so have a plentiful data source to work from, when it comes to training our models. The catch-22 of this lucky situation is that when we attempted to display - for example - the last two weeks occupancy data on the page (as an occupancy chart), we were attempting to retrieve and display circa 10,000 rows. This quickly pushed us to introduce a second process to resample the data so we would have data sources of more manageable sizes to drive the front-end.

Removing Live Predictions

One of the reasons for our initial poor page load time is the fact we are running all predictions for all 110 stations every time a future date is selected by the end user. This is clearly not scalable.

We have discussed a number of ways to address this issue. Our front-end only currently allows the end user to request predictions in four-hour windows moving forward through the next 48 hours (the limit of our weather forecast data). These predictions could easily be produced by a scheduled process in the background. If we modified our "StationStateResampled" table to include a record type, then we could have 'real' records representing actual data scraped from the open API's and 'predicted' data generated by a process running once each hour, which predicts the occupancy for each station at each interval using our model and the weather forecast data. The front-end would then simply display the data as requested, which should result in a dramatic reduction in page load times.

Choices such as the above were not evident to us at the start of the process but as our experience with the platform has grown and our understanding of the

technologies/techniques involved has matured, we begin to see new opportunities for how to realise our goal.

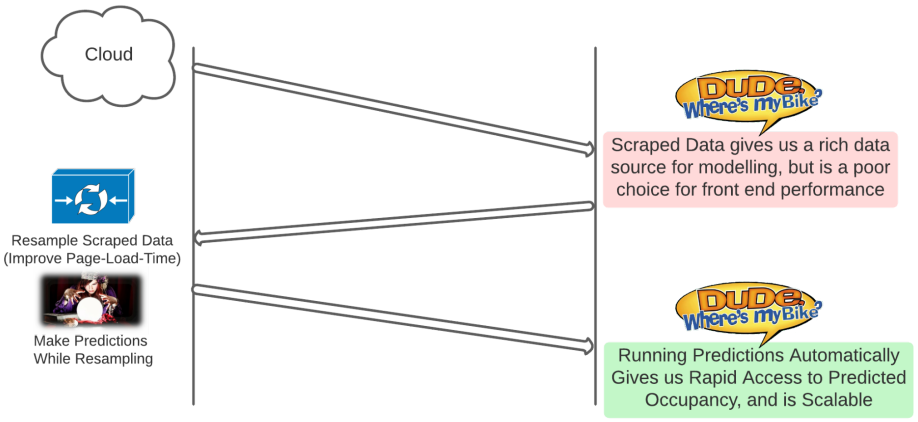

Figure 4: Offline Predictions - Best of Both Worlds?

Data Analytics

Predictive Model:

Our objective within analytics was to gather relevant data sources to help predict future bike availability at each station, as well as future bike stand availability in order to return a bike. We compiled both historical station statistics on usage and total stands and also the weather at those times. The hypothesis being that weather can impact usage, such as fewer bikes rented when it is raining.

Data Collected:

- Station State: from JCDecaux API
- Weather History: Open Weather API
- Resampled Station State: JCDecaux API resampled inside RDS
- Resampled Weather History: Open Weather API resampled inside RDS
- Weather Forecast: Open Weather (One Call) API

From there we used jupyter notebook to build and test several predictive models, with the hope of obtaining a model with a high R-squared, ideally greater than 0.7. The R-squared value is the proportion of variance in the dependent variable (i.e. bikes available & bike stands available) that can be explained by the independent variables (i.e. weather, time of day, etc.). We started with a more basic model - multiple linear regression - and increased in complexity to random forest regressor. As we progressed, we improved the data sources and altered the features included to ultimately improve the model performance. During the process, we debated about going through with logistic regression as well, but decided against it because we knew logistic regression was not going to outperform random forest. Table 1 below outlines the model iterations and performance results.

In the beginning, the Multiple Linear Regression models were performing very poorly. In class, the idea of modelling each station was discussed, and piqued our curiosity to run linear regression for a single station (station 1). This improved the results dramatically,

although still a low R-squared from our 3rd Model to our 4th, R-squared 0.0455 to 0.2123, respectively. At this point we decided to test the Random Forest model, still using a single station (station 1). This further improved the R-squared from our 4th to 5th model, 0.2123 to 0.4752, respectively. We then decided to run Random Forest on all stations again, rather than only one station at a time, and added a few more features for day of month and month of year. From our 5th to 6th model iteration, the R-squared improved from 0.4752 to 0.7321. Our last model, the 7th version which we used towards the product, ended with an R-squared of 0.6165. This is less than our most accurate predictive model (version 6), but we decided to trade model accuracy for performance integrating into our application. For example, page load time was 40 seconds, but we were able to decrease this load time to around 2-3 seconds by moving to model 7.

Table 1: Predictive Model Iterations

Model Type	Model	Data	Features	Performance	Results
Multiple Linear Regression (1st Model)	Data_Model_Prep_ Linear_Reg	Weather History	Temperature	Mean Absolute Error (MAE)	**7.3618**
		Station State	Feels like Temperature	Root Mean Square Error (RMSE)	**8.967**
			Humidity	R-squared Score	**0.000826**
			Wind Speed		
Multiple Linear Regression (2nd Model) With added features	Multi_Linear_Reg_ Model	Weather History	Station ID	Mean Absolute Error (MAE)	**7.2607**
		Station State	Number of Bike Stands	Root Mean Square Error (RMSE)	**8.7963**
			Temperature	R-squared Score	**0.0385**
			Feels like Temperature		
			Humidity		
			Wind Speed		
Multiple Linear Regression (3rd Model) With Resampled Data and Added Features	Resampled_Multi_ Linear_Reg_Model	Resampled Weather History	Station ID	Mean Absolute Error (MAE)	**7.199**
		Resampled Station State	Number of Bike Stands	Root Mean Square Error (RMSE)	**8.7218**
			Weather Hour (Int Hour Number)	R-squared Score	**0.0455**
			Weather Description (Encoded to Int Number)		
			Temperature		
			Feels like Temperature		
			Humidity		
			Wind Speed		
Multiple Linear Regression (4th Model) For Single Station (station 1) using Resampled Data	Single_Station_ Resampled_Multi_ Linear_Reg_Model	For station 1 only:	Station ID	Mean Absolute Error (MAE)	**5.8396**
		Resampled Weather History	Number of Bike Stands	Root Mean Square Error (RMSE)	**7.049**
		Resampled Station State	Weather Hour (Int Hour Number)	R-squared Score	**0.2123**

			Weather Description (Encoded to Int Number) Temperature Feels like Temperature Humidity Wind Speed		
Random Forest Regressor (5th Model) For Single Station (station 1) using Resampled Data	Single_Station_ Resampled_Random _Forest_Model	For station 1 only: Resampled Weather History Resampled Station State	Station ID Number of Bike Stands Weather Hour (Int Hour Number) Weather Description (Encoded to Int Number) Temperature Feels like Temperature Humidity Wind Speed	Mean Absolute Error (MAE) Root Mean Square Error (RMSE) R-squared Score	**4.3866** **5.5832** **0.4752**
Random Forest Regressor (6th Model) All Stations using Resampled Data with Added Features	Station_Resampled_ Random_Forest_ Model	Resampled Weather History Resampled Station State	Station ID Number of Bike Stands Weather Hour (Int Hour Number) Weather Day (Int Number Day of Month) Weather Month (Int Number Month of Year) Weather Description (Encoded to Int Number) Temperature Feels like Temperature Humidity Wind Speed	Mean Absolute Error (MAE) Root Mean Square Error (RMSE) R-squared Score	**3.444** **4.5969** **0.7321**
Random Forest Regressor (7th Model) Using Resampled Data Looped for Each Station	LoopedStation_ Resampled_Random _Forest	Resampled Weather History Resampled Station State	Number of Bike Stands Weather Hour (Int Hour Number) Weather Day (Int Number Day of Month) Weather Month (Int Number Month of Year) Weather Description	Mean Absolute Error (MAE) Root Mean Square Error (RMSE) R-squared Score	**3.7216** **4.7544** **0.6165**

	(Encoded to Int Number) Temperature Feels like Temperature Humidity Wind Speed

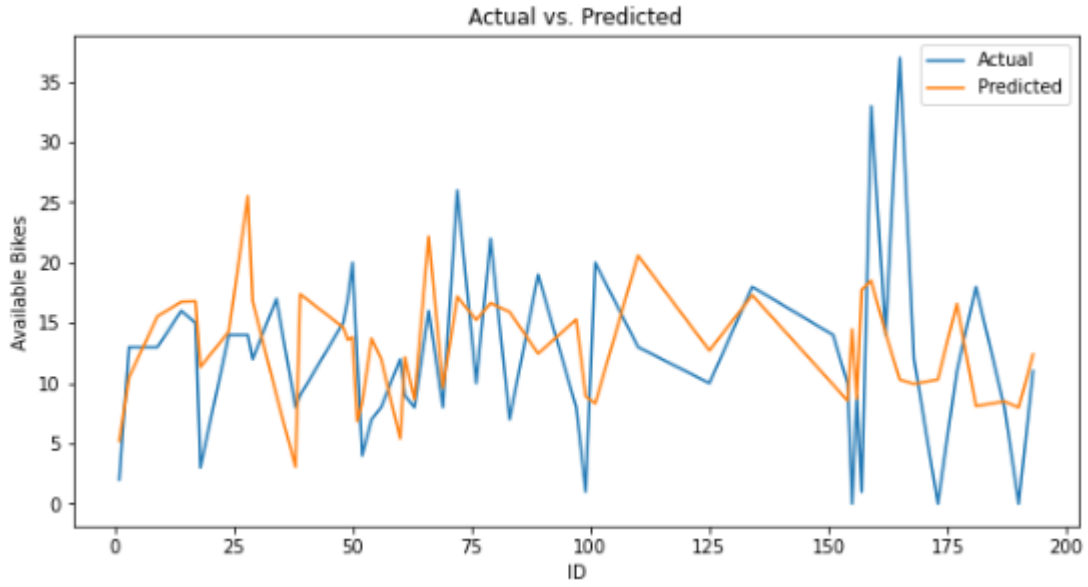

Figure 5: Actual vs. Predicted Best Model

The above figure is a sample of the actual bikes available versus the predicted bikes available from our best evaluated model (Model #6). After evaluating the model graphically, the model lost accuracy where there was extreme high/low availability. We are confident that over time with continued data the model may be better at detecting these swings.

Predictions:

Our planned delivery of the predictions to the user was to have a future forecast of the bike availability in a bar chart form when the user chooses a station.

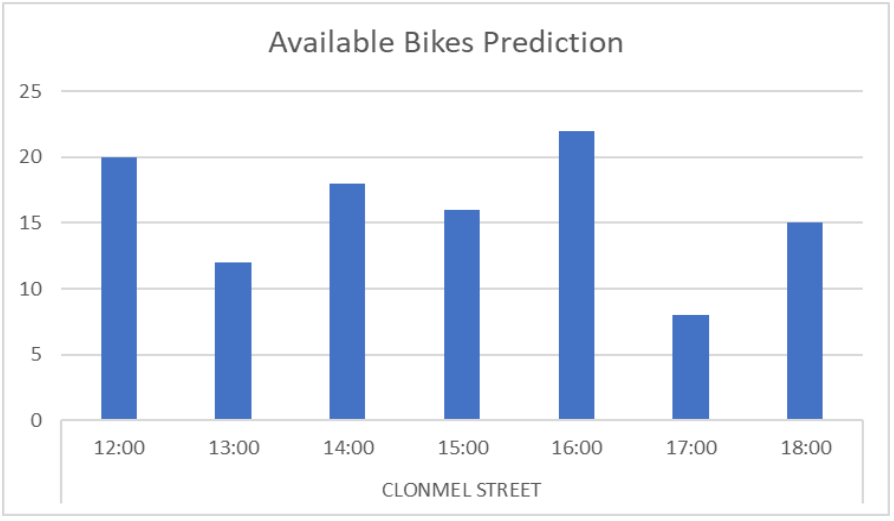

Figure 6: Example Prediction Chart for User

We currently have the model in place and the ability to produce a prediction given the time it was called. The issue is dealing with the load of the calls on the refresh for the page. This issue is in our future work plan and when solved will be able to deliver that future forecast chart.

Model Integration:

Once we chose the model to run for our Application, we utilised jupyter notebook to loop through all of the stations with their respective data, trained models for each, and created a pickle file for each. With the pickle files in the application's code base we created a 'predict' app that would match the user's choice of station to the appropriate pickle file and run the model. The prediction would then be displayed for the user in chart and text form.

Design

Concept

The idea behind DudeWheresMyBike was to provide a simple, at-a-glance, single screen way for a Dublin Bikes user to assess predicted availability of bikes at any given station at a future date and time. The predictions would take into account past usage patterns and predicted weather conditions, using predicted weather sourced from OpenWeather.org.

We decided to focus on ease of use by giving the user a slider on screen that can be used to quickly flick through the predicted availability at all stations in the upcoming days, without having to fill in forms which might provide a barrier to entry.

Realisation

In technical terms, all the components we envisaged in the project have been realised. We have a functioning, publicly hosted web application. We are loading and storing sufficient data for a robust predictive model. The model has been successfully integrated with the web application and can predict future availability/occupancy at the various Dublin bike stations in the city.

However, in terms of delivering a feature rich client to the user to explore the data we have collected and prepared we have fallen a little short. A lack of experience working on and producing web-front ends became immediately apparent and has hampered our ability to translate our initial design goals to reality. We believe additional development time could very quickly lead to large improvements in the front end. Small changes like improved on-screen hints to prompt the user to action would help direct the user to explore what is a very simple site. Help text, improved accessibility support, improved interactivity, some time refining the theme of the website would all help make the end product more coherent.

User Flow

The interaction between user and the system is shown in the form of a UML class diagram as illustrated in Figure 7.

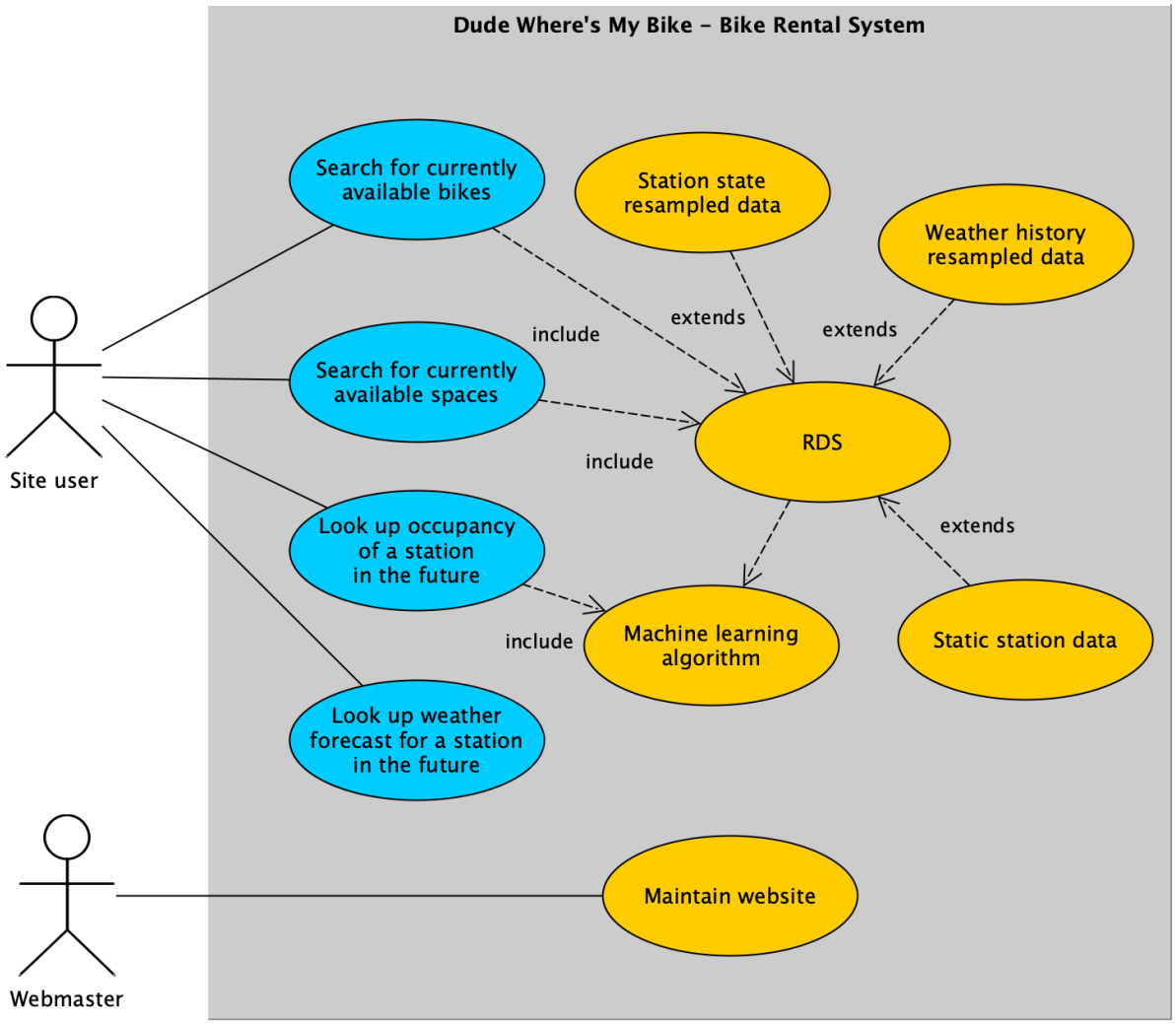

Figure 7: UML use case diagram

Functionality and Features of the Application

To improve the usability of the “Dude Where’s My Bike” website, the focus on the front-end has been to include essential functionality while also excluding fancy features most users probably would never use. Hence, all functionality needed to control and display the application fitted on one single website, resulting in a user-friendly design by avoiding any awkward navigation between multiple web pages.

A relatively large and zoomable Google map allows the user to see all bike stations in the Dublin city centre at a glance. One feature of this website is that the map displays the location and current/predicted occupancy state by utilising coloured bike icons as markers, representing bike stations themselves. In contrast to using so-called heatmaps to display the current/predicted availability of bikes and spaces, the coloured bike icons facilitate indicating the occupancy in a more neatly arranged way. In other words, it avoids overloading the map itself with flashing colours as it may appear when taking the heatmap approach.

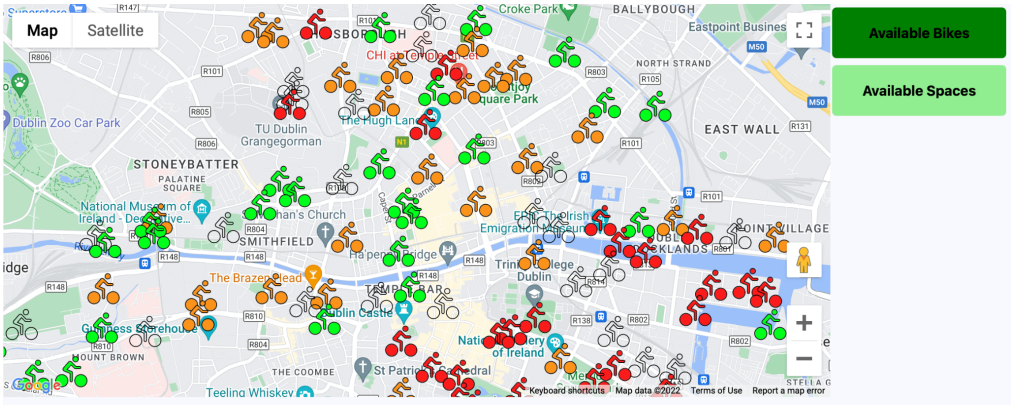

Figure 8: Google map with coloured bike icons and user mode selection

Two buttons on the right-hand side of the map allow users to switch between two different user modes, namely “Available Bikes” and “Available Spaces”. The active user mode is indicated by changing the background colour of these buttons respectively. Every time the user mode is changed, new data is being queried from the back end of the application to update the coloured bike icons according to the selected user mode (available bikes / available spaces).

As outlined in the paragraph above, the coloured bike icons indicate either available bikes or available spaces depending on the selected user mode. In addition, four different colours have been chosen to distinguish between the following occupancy states:

Table 2: Bike Icon Legend

Bike icon colour:	**State:**
Black	Bike station is closed
Green	Availability of bikes/spaces is above 70%
Orange	Availability of bikes/spaces is between 10% and 70%
Red	Availability of bikes/spaces is lower 10%

By clicking on a bike marker, an information window displays current/predicted information about the selected station, as shown in Figure 9.

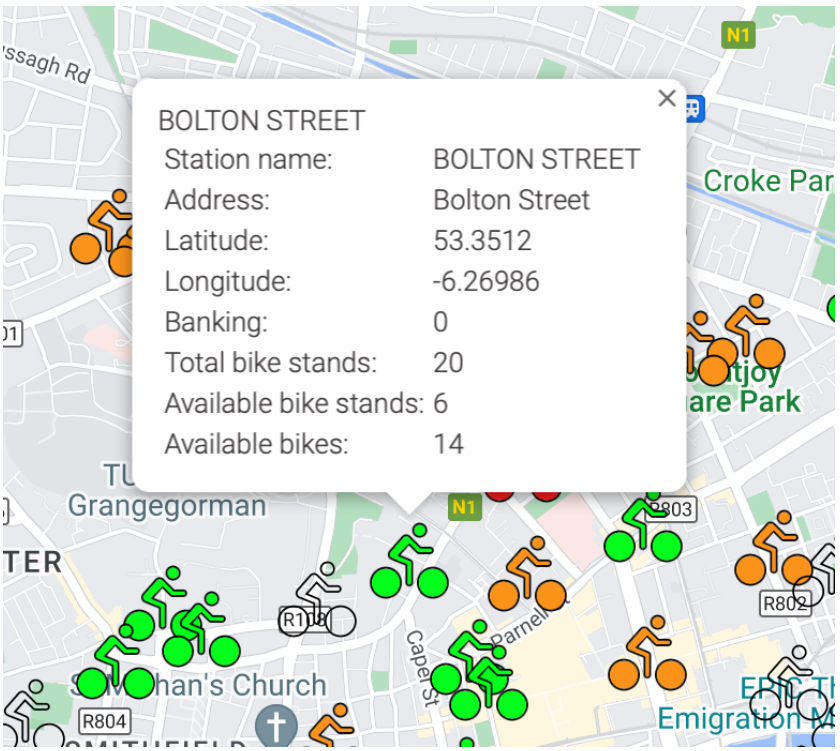

Figure 9: Bike markers displaying details about a bike station

The displayed data of the selected station is comprised of the following:

- Station name
- Station address
- Latitude
- Longitude
- Banking (0 == no cash machine available / 1 == cash machine available)
- Number of total bike stands
- Number of available bike stands
- Number of available bikes

Details of the station, such as occupancy and weather information, are shown below the map, as illustrated in Figure 10. Selecting a station and updating its details can be triggered by two events: clicking on a marker in the map or choosing a station in the dropdown menu on the left-hand side.

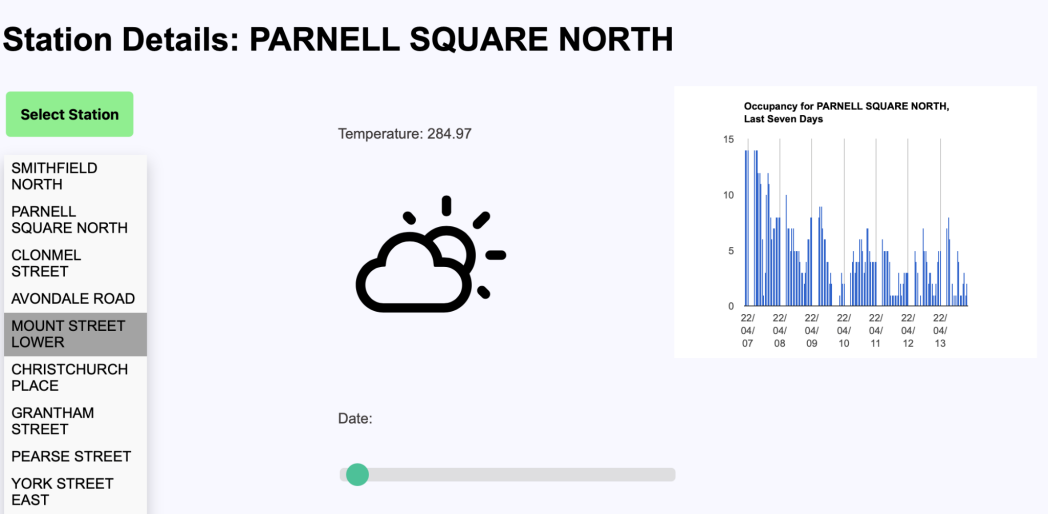

Figure 10: Displaying current/prediction details about a bike station

Another feature illustrated in Figure 10 is the range slider, allowing users to display occupancy and weather prediction of bike stations up to 48 hours in the future. The slider has been deliberately chosen to improve the usability of the prediction feature. For example, instead of awkwardly filling out/selecting the date and time on a so-called data & time picker window, the prediction time can be chosen by simply moving the slider from left to the right where the most left position marks the actual time. Furthermore, this slider also facilitates optimising this application for the use on mobile devices.

As shown in Figure 10 above, the current/predicted weather description is visualised by fifteen weather icons, ranging from clear sky to heavy rain. Utilising weather icons rather than simply printing plain text on the screen makes the weather prediction more appealing to the user. Moreover, the user is also informed about the temperature which is displayed right above the weather icon.

The chart in Figure 11 shows the occupancy of the past seven days for the selected station. As already mentioned, the station can be either selected by the station dropdown menu on the right or by simply clicking on a marker on the map.

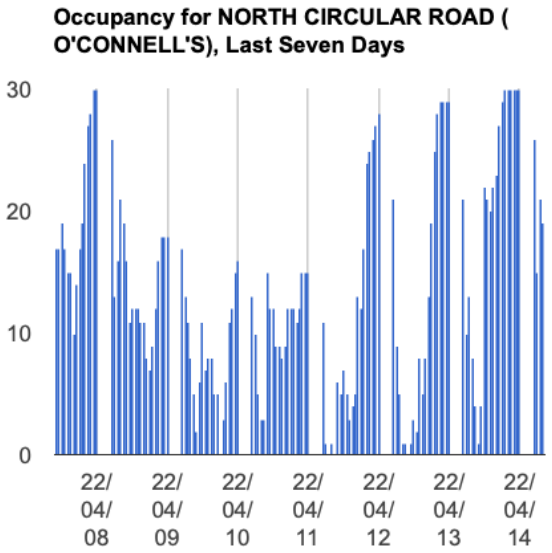

Figure 11: Occupancy chart of selected bike station

Back-end

Scalable: Weather is passed to the front-end per station. At present only a single weather prediction location is included, but if more stations were added (nationally) then the data model sent to the front-end can support that.

Process

Sprint 00:

Sprint Planning:

See Appendix A.

Beginning of Sprint 00 Storyboard:

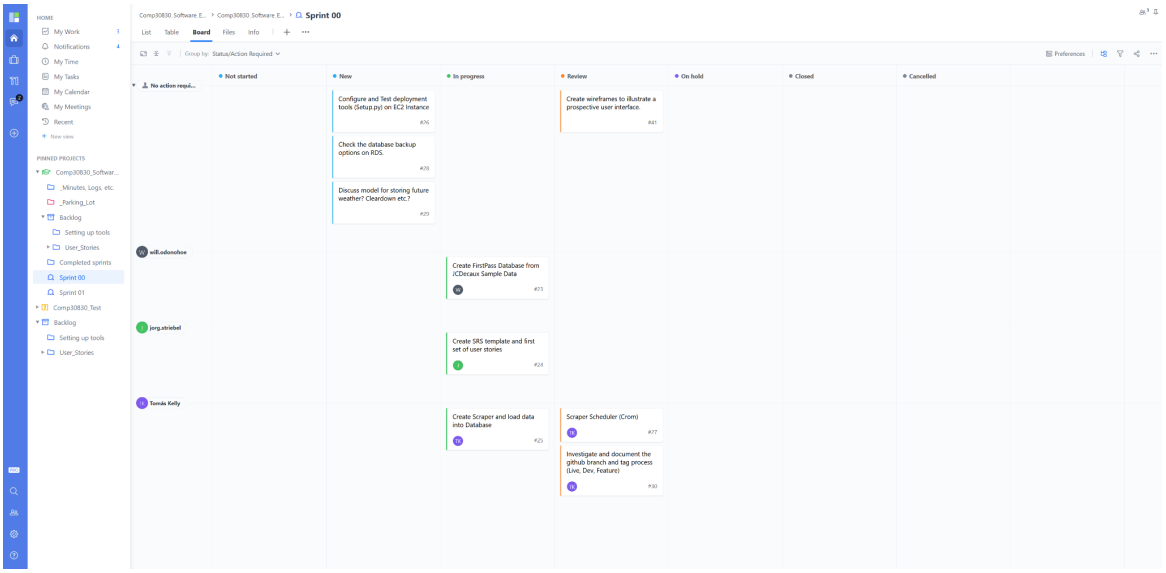

Figure 12: Beginning of Sprint 00 Storyboard

End of Sprint 00 Storyboard:

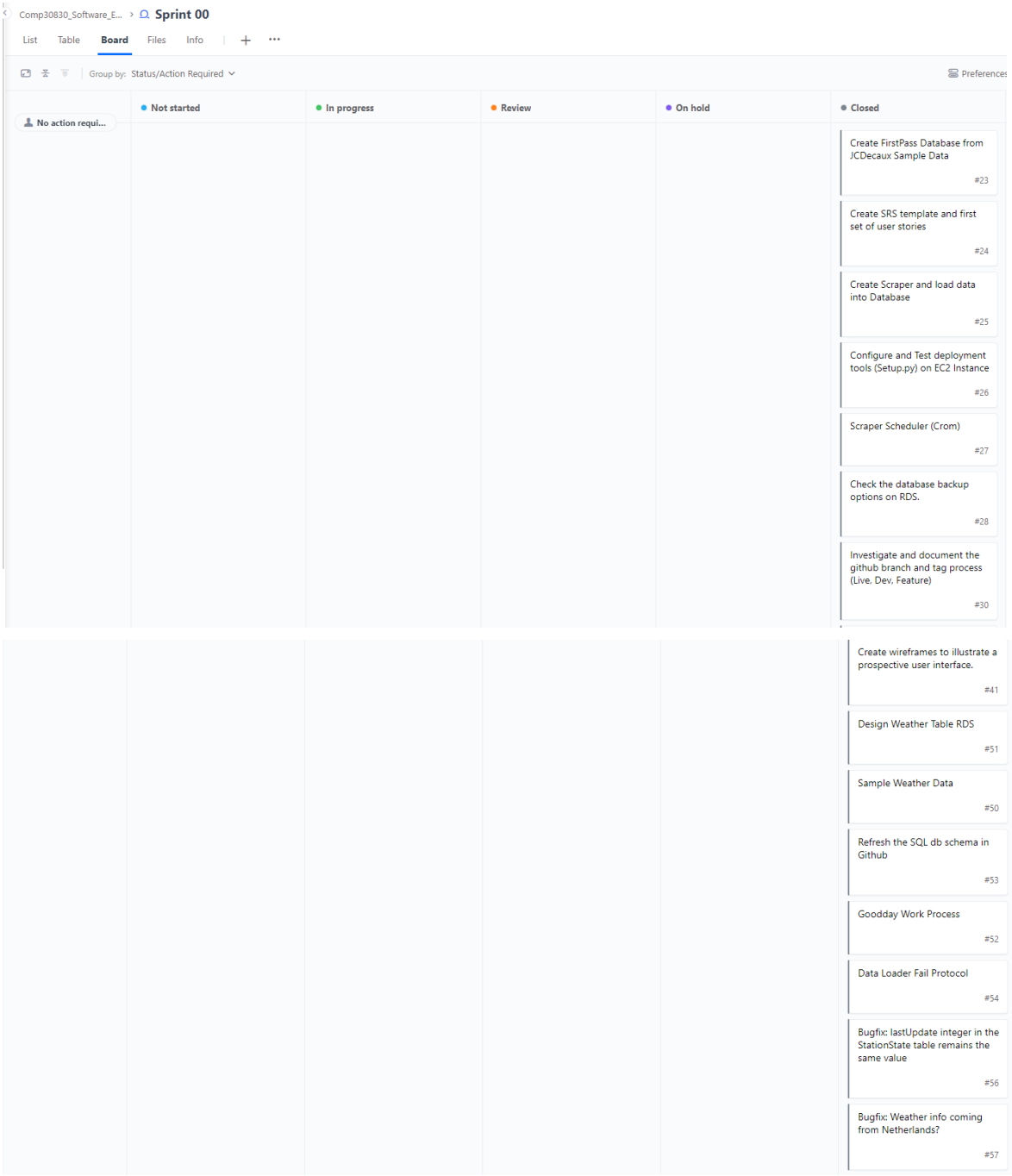

Figure 13: End of Sprint 00 Storyboard

Features & Product Backlog:

Table 3: Feature & Product Backlog Sprint 00

Feature	Description	Product Backlog	Sprint 00
Overall Function	The back-end and structures that make the application run.	Wireframe structure of App	**X**
		Scraper	**X**
		Scheduler	**X**

		Data Loader	
		RDS	**X**
		Flask	
		Design App	
		Create HTML/JS/CSS Structure	
		RDS Database Structure/Implementation	**X**
Map	An interactive map where the user can see where bike stations are located in Dublin. The station markers will have station information	Google Maps API	
		JCDecaux API for bike data	**X**
		Display Dublin on Map	
		Display markers for station location	
		Display Available Bikes	
		Display Available Spaces	
		Display Station Information	
		Display information in map markers	
Available Bikes/Spaces Buttons	Two buttons where the user can choose between bikes or spaces depending if they are looking for a bike or looking to return a bike.	Functionality for Available Bikes	
		Functionality for Available Spaces	
Station Dropdown	An interactive dropdown where the user can choose which station to focus on. Will update necessary data/information.	Create Dropdown	
		Grab Station Information	**X**
		Use Station Name for Dropdown	
Date Slider	An interactive slider for the user to choose to focus on a time in the future. Will update necessary data/information/charts.	Create Slider	
		Open Weather API	
		Open Weather Forecast (one call) API	
		Connect data to slider	
Past Availability Chart	A chart that shows the user past bike data	Grab data from RDS	
		Display bike data	
		Create past availability chart	
Future Availability Chart	A chart that shows the user future predictions of bike/space availability based over past bike and weather data, and weather forecasts.	Data Handling	
		Data Cleaning	
		Model Planning	
		Model Comparisons	
		Integrating Model into App	
		Output Predictions into chart	

Meeting Logs:

See Appendix B.

Burn Down:

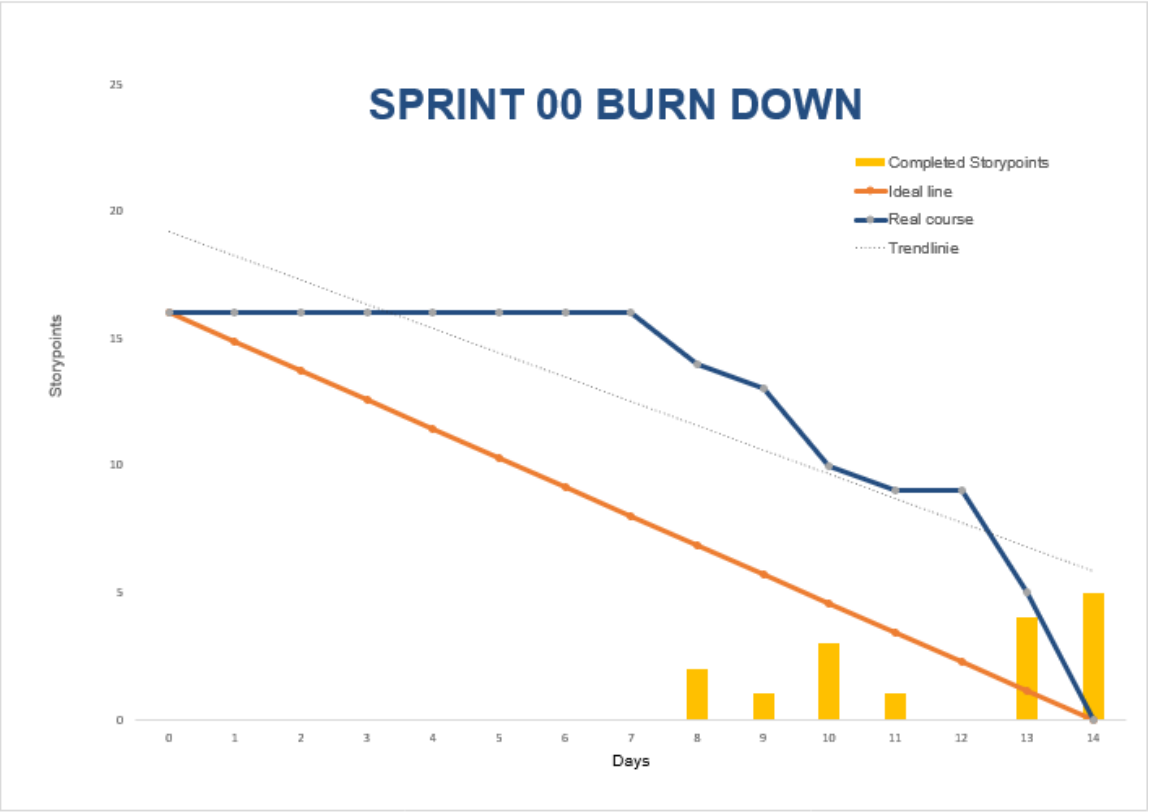

Figure 14: Sprint 00 Burn Down

Issues & Resolutions:

EC2 Setup:

We faced some difficulties getting all three of our EC2 instances up and running. Most significantly with connections to our RDS databases. We tended to research any issues that arose and worked together to solve them.

Virtual Environment:

Syncing our virtual environments took significant time and resources. These issues began in the setup week of the project and continued in the first week of Sprint 00. After an exhausting amount of research and problem-solving, we resolved to the decision to manually keep in-sync with regards to virtual environments.

Scrum Meetings:

At the beginning we struggled adhering to the brevity of daily standups. We tended to devolve into too much depth or trying to show what we have been doing, as opposed to adhering to the three brief questions of the daily standup.

To address this issue we implemented these following solutions:

- Voiced concern about being too detailed
- Emphasised sticking to 1) Do Yesterday; 2) Do Today; 3) In Our Way
- Moved daily standup earlier so that day is still ahead of us
- Started having separate meetings to talk about issues in depth

Project Management Tool:

Other than settling on a specific project management tool, an issue we struggled with in Sprint 00 was getting used to the project management tool and the lack of a usable burndown chart from our project management tool Good Day.
The learning curve felt steep at times learning the intricacies of Good Day. To solve this one team member took on the research and management of the tool. This helped to the extent we needed for this project. Good Day, in hindsight, was actually too robust of a project management tool for this project. Our team could see the benefits it could provide if that was the tool of choice for the organisation we might have worked at, but may have been too much for this project.
That discovery actually led to our burndown chart issue. Good Day had a robust analytics area with all the charts we needed, but they seemed to populate in unexpected ways. The one team member attempted to research and reach out to Good Day for demos or aid, but ended up deciding to make the burndown charts themselves. The team member made a template for the burndowns in excel and created one for each sprint.

Communication:

Communication was not an issue per se, but it was a possible issue in the future if not addressed. So for the avoidance of communication issues we instituted some frameworks to facilitate healthy communication.

- Set up “Masters of Scrum” WhatsApp group text
    - Only used for quick updates or meeting setups
- Set up shared Google Drive
- Any complex or difficult issues were best addressed with in person communication & work sessions

Sprint Review:

For Sprint 00 the main focus was planning and setting up the right structures for the future sprints. We finalised our overall scrum process, got used to our project management tool, planned and set up various structures for the application, established team conventions, and dealt with some ongoing issues from the initial setup process.
As well as set up and planning, we did begin creating the scraper and scheduler for our future data and process needs. The initial issues we encountered with environments affected our ability to move forward on more tangible output, but with perseverance we solved those issues and ended up ready to continue building our application.

Sprint 01:

Sprint Planning:

See Appendix A.

Sprint 01 Storyboards:

See Appendix C.

Features & Product Backlog:

See Appendix D.

Meeting Logs:

See Appendix B.

Burn Down:

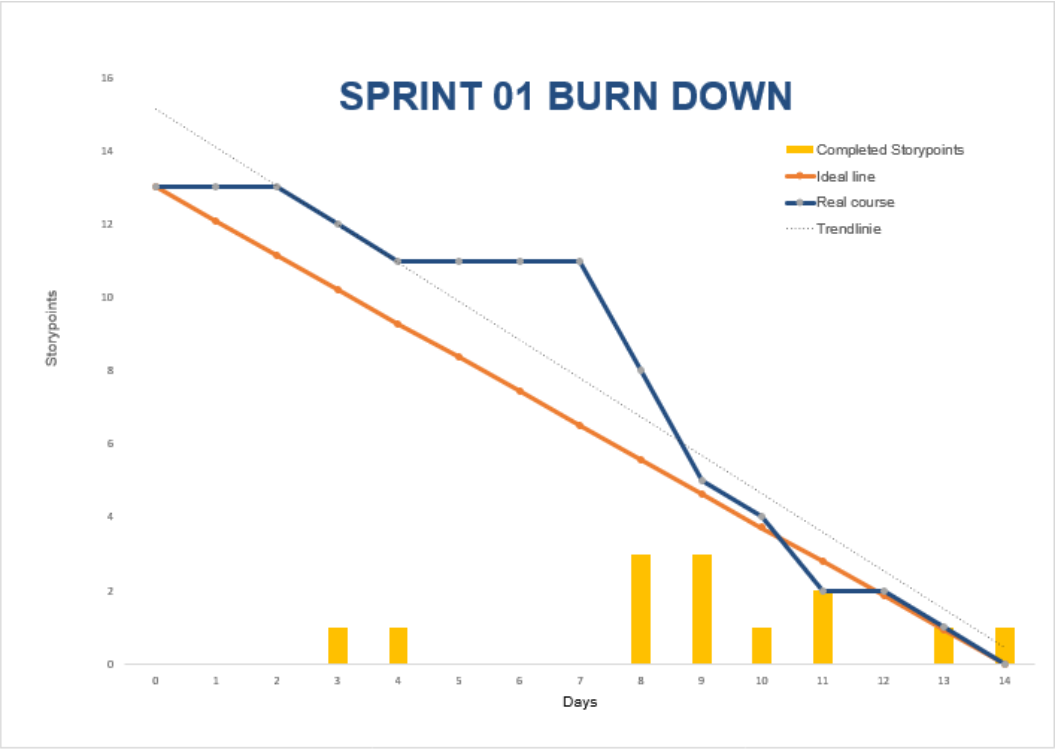

Figure 15: Sprint 01 Burn Down

Issues & Resolutions:

Getting on the same page in regards to Version Control:

It was not so much an issue, as it was that we were all inexperienced using Git. For this reason we spent significant time getting used to Git and branching strategies. It may have slowed our output, but we felt it was important to get it right before it was too late.

Google Maps API Issues:

We successfully obtained a Google Maps API key, but when implementing the map it had “for development purposes only” over the map. The functionality of the map was not affected, but it was not a good look for our website. After many attempts by one team member, we found that their Google developer account could not be separated to setup with a UCD connect account. To move forward we just had another member setup a fresh account and API key, which solved the issue.

Sprint Review:

For Sprint 01 our team continued building out the back-end of the application, started to create the basic front-end vision, and started working with the data in the RDS databases to prepare for our model predictions. This sprint had far less issues to deal with which aided our progress for this sprint.

Sprint 02:

Sprint Planning:

See Appendix A.

Sprint 02 Storyboards:

See Appendix C.

Features & Product Backlog:

See Appendix D.

Meeting Logs:

See Appendix B.

Burn Down:

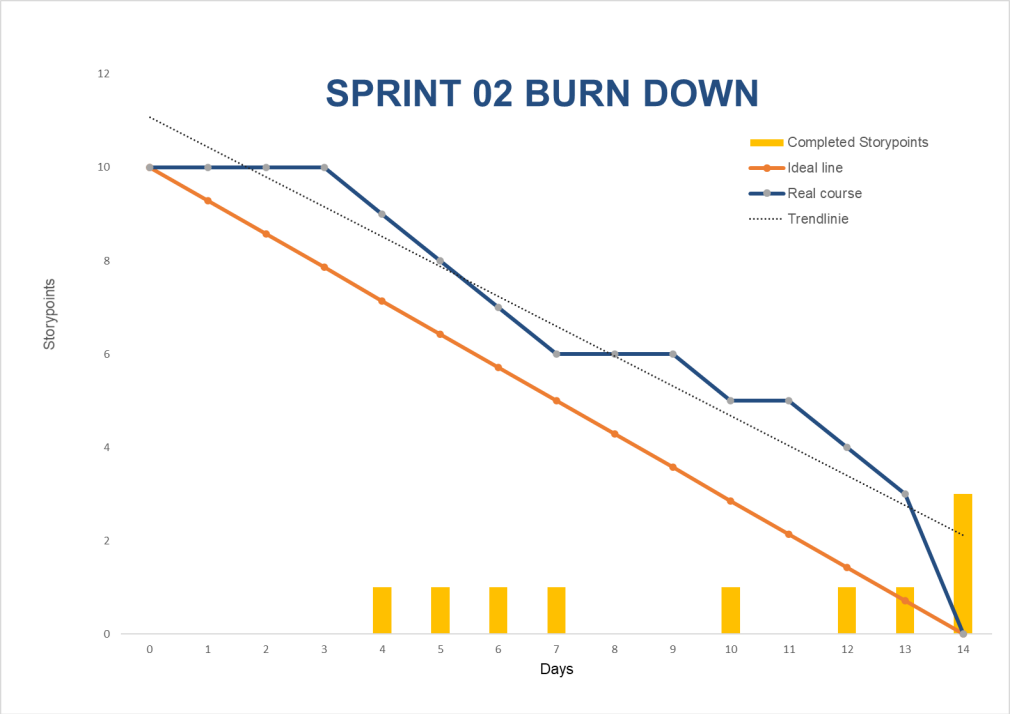

Figure 16: Sprint 02 Burn Down

Issues & Resolutions:

Data Resampler:

We experienced some delay while trying to handle the data for resampling the weather and station date. The hours around a day change had some complexity that required the team to work together on, which solved the delay.

Page Loading Issues while running Predictive Model:

When the model was completed and pickled, we found that page load time was significantly impacted. It would take about 40 seconds to load the page. The team brainstormed together and agreed on ways to lower the load the model would have on page load time.

Sprint Review:

This sprint we mainly focused on refining the model we planned on using for our application and features on the front-end of the application. It was a fairly productive sprint, as we felt like the team was working well together and everyone knew what we needed to do for a successful sprint. Some issues explained earlier crept up, but we handled them well as a team and picked up slack when needed.

Sprint 03:

Sprint Planning:

See Appendix A.

Sprint 03 Storyboards:

See Appendix C.

Features & Product Backlog:

See Appendix D.

Meeting Logs:

See Appendix B.

Burn Down:

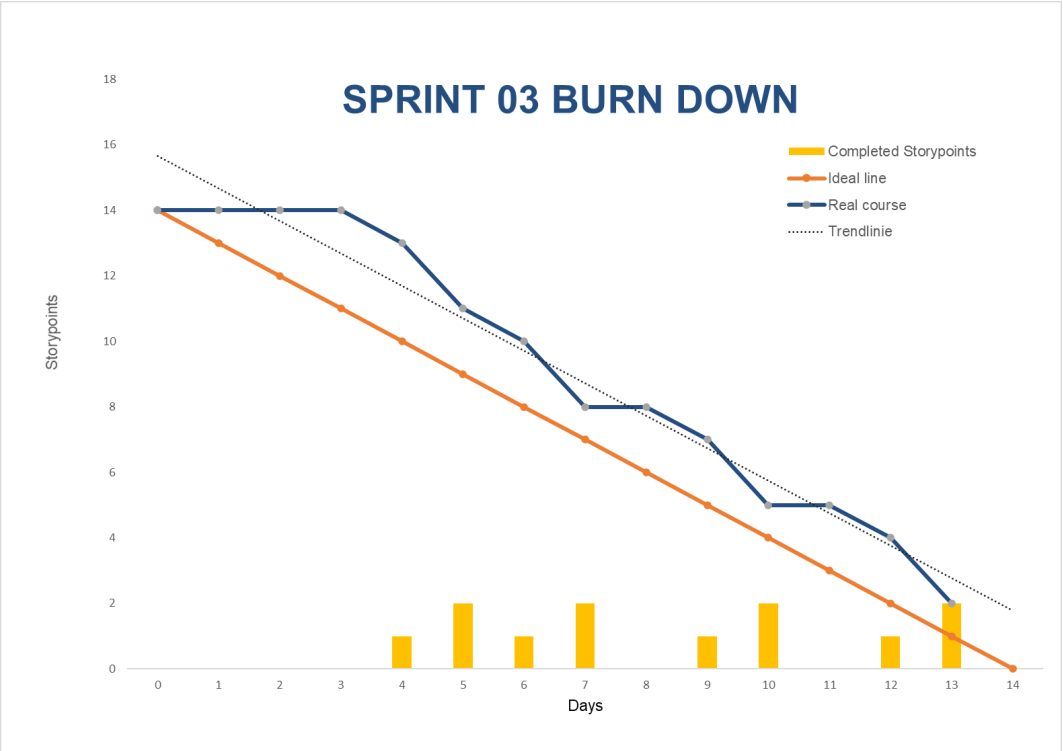

Figure 17: Sprint 03 Burn Down

As you can see we did not meet our intended goals for this sprint. We needed to push two tasks into future work.

Issues & Resolutions:

EC2 Instance Crashed/Corrupted:

In the final week, our EC2 instance crashed twice. It added significant time and stress to get it back and running. The team member who had the most experience with that process and was there for both crashes handled debugging. In the end we created a new fresh EC2 instance and was able to host our application just in time.

End State Focus:

The biggest struggle for the final sprint was deciding what was our finish line. In a project this size, there is always more that can be done to improve existing features or add new exciting features. To help with this issue we held a few meetings that had the general goal of looking at what time we had left, what resources we had, and what we still wanted to get accomplished. We then continually agreed and assessed where our end state was.

Sprint Review:

This sprint was heavily impacted by our two EC2 crashes. Those two incidents brought our momentum to a complete stop for a few days. We worked through it and were able to salvage the last week of the sprint to deliver some more of the features we planned on.
On the positive side, the updates and practises we implemented for how our team worked together aided us greatly in being able to handle an un-ideal situation that happened in this sprint.

Retrospective

What Worked Well

GIT: There was a steep learning curve, but our team found GIT to be an excellent tool for SCM and VCS. It was invaluable in version control, and we did not have many problems working on our individual items and merging. A modification we would make in our workflow going forward is to integrate GIT's built-in issue tracking feature.

Consistent Tools: Using a homogenous set of tools (VS Code, Goodday, etc.) among team members worked well. It reduced some of the difficulties in synchronising work on a project of this nature.

Prototypes: Creating prototypes of our site using whiteboards/mockup tools was very useful as the concepts gave us a common design vision. Without these, the work at hand would have felt more nebulous, and our team would have inevitably experienced confusion in terms of where we were going as a project.

Documentation: One way to ensure our team never hit the same obstacle twice was robust documentation. Our team produced documentation on coding conventions, common git procedures, installation of our environment, etc. While these documents are by nature very project specific, they did form a small knowledge base we built up as the project progressed.

This knowledge was vital, for example when halfway through sprint 03 we were forced to re-install our project on a fresh EC2 instance.

What Was Problematic

Timeframe: While we all appreciate that the timeframe for this particular project is dictated by the school timetable, meeting expected sprint deadlines was extremely challenging. The time required to explore each technology and gain familiarity with it was not available (in the context of a trimester where this module is only one in six).

Unfamiliar Technologies: Planning how to use and deploy technologies that we had not used before was extremely challenging - even at the micro “per sprint” level. It was difficult to share some tasks and expertise when the only way to understand the task at hand was to attempt it and see what worked and what failed.

Meeting Logs

For the most part, we held weekday meetings in person with some on zoom when needed. All members attended group meetings. We tended to keep sprint retrospective meetings more informal with an emphasis on brainstorming. The benefits of those meetings fed into our sprint planning meetings where we would decide on the focus of the new sprint. See full details of daily meetings log in Appendix B.

Future Work

Improvements

- Journey Start/Endpoint
  Allow the user to select start and end stations for their journey by clicking on map (plan is to add a ‘start journey here’ button to the information window for each map. A similar plan should allow the user to select the end of their journey. This approach removes the needs for complicated forms on screen.
- Improved Logging
  Currently our application background process logs are being written to a file in the filesystem on the host EC2 instance. This file grows and grows. We have been manually clearing it down. It would be relatively easy to change the scheduled processes to write directly to the RDS databases they are reading from. This would allow us to trim old log records much more easily and remove any of the risk writing to a static log file introduces.
- Future Occupancy
  Our current occupancy chart shows occupancy over the last week for the selected station. If we changed our implementation - as discussed at the end of the architecture section - to generate predicted data in the background then there would be no impediment (bar the availability of weather predictions) to producing forecasts for the next seven days and displaying ‘predicted future occupancy’ in this chart. It would provide a nice counterpoint between displaying present/past data on the map

and chart when the user arrives at the page, and displaying future data on the map/chart when the user selects a future time using the slider.

- More Statistics
  As future features are added to the application (e.g. journey planning) it will become possible to include more statistics in our front end display, helping users to choose the best stations to start and finish their journeys.
- Populate Station Opening Hours from Loaded Information
  Introduce population of an “Opening Hours” table per station to allow us to predict station status (open/closed) when predicting occupancy levels. Currently when predicting occupancy our application doesn’t also predict if the station in question will be open or closed at the time of the prediction. Population of an opening hours table per station should be trivial enough, allowing us to include expected station status along with our predictions.

Reflection

To summarise, realising such a fairly complex application turned out to be a real conundrum at times, but it also offered a great opportunity to apply the skills learnt over the course of this module. Each team member deepened their technical skills and added to their tool belts. Having said that, the learning outcome for each individual team member would have most certainly been better if we had not been faced with constant time pressure throughout this project. The time constraints were mainly imposed by the fact that we had to study and work for five other modules concurrently, increasing the pressure to achieve the goals set while also ensuring that fixed deadlines are being met.

They do say that hindsight is 20-20. If this had been a real-world project we could have spent far longer on the initial design and UI planning. The more detail and information that can be described in the planning stage the better. However, this is often easier said than done. When a software project is started from scratch without knowing all the technical details of how to implement certain features, it leads to a prototype project where the main focus tends to be on getting functionality rather than making it aesthetically pleasing.

In addition to “hard” technical skills required for this project, we experienced how “soft” inter-personal and management skills also play a very important role in the overall success. It supports alignment across the team members in vision, motivation, and prioritisation of both project-specific work, as well as work for this project versus time required for other courses.

Appendices

Appendix A: Sprint Planning

Product Backlog = All features/products decided by team to complete our App (at the time)
Sprint Focus = Features/products team is focusing for each specific sprint

Sprint 00:

Monday February 7th:

Product Backlog:

- Scrum Process
- Project Management Tool
- Planning for structure of App (wireframe)
- SRS
- Issues with environments
- Scraper
- Scheduler
- Data Loader
- RDS
- Flask
- API for Bike Data
- API for Map
- API for Weather
- API for Weather Forecasting
- Display Available Bikes
- Display Available Stands
- Display Station Information
- Display Map of Dublin
- Display Weather Data
- Display locations of stations on map (markers)
- Display needed data in map markers of stations
- Predictive Model
- Display Predictions
- Display Prediction Chart(s)

Sprint Focus:

- Scrum Process
- Project Management Tool
- Planning for structure of App (wireframe)
- SRS
- RDS
- Scaper
- Scheduler
- API for Weather
- API for Bike Data
- Issues with environments

Sprint 01:

Monday February 21st:

Product Backlog:

- Data Loader
- Flask
- API for Map
- API for Weather Forecasting
- Display Available Bikes
- Display Available Stands
- Display Station Information
- Display Map of Dublin
- Display Weather Data
- Display locations of stations on map (markers)
- Display needed data in map markers of stations
- Predictive Model
- Display Predictions
- Display Prediction Chart(s)
- Design plan for App
- Overall User features
- Create basic HTML/CSS layout
- Working with data from RDS

Sprint Focus:

- Flask
- Data Loader
- API for Map
- Display Map of Dublin
- Display locations of stations on map (markers)
- Create basic HTML/CSS layout
- Working with data from RDS
- Design plan for App
- Overall User features

Sprint 02:

Monday March 21st:

Product Backlog:

- API for Weather Forecasting
- Display Available Bikes
- Display Available Stands
- Display Station Information
- Display Weather Data
- Display needed data in map markers of stations
- Predictive Model
- Display Predictions

- Display Prediction Chart(s)
- Data Grab/Handling
- Model Planning
- Initial Front-end features

Sprint Focus:
- API for Weather Forecasting
- Display Station Information
- Data Grab/Handling
- Model Planning
- Predictive Model
- Initial Front-end features

Sprint 03:

Monday April 4th:

Product Backlog:
- Display Available Bikes
- Display Available Stands
- Display Weather Data
- Display needed data in map markers of stations
- Display Predictions
- Display Prediction Chart(s)
- Front-end improvements/finishings
- Finalise Model/Predictions
- Planning for report

Sprint Focus:
- Display Available Bikes
- Display Available Stands
- Display Weather Data
- Display needed data in map markers of stations
- Display Predictions
- Display Prediction Chart(s)
- Front-end improvements/finishings
- Finalise Model/Predictions
- Planning for report

Appendix B: Meeting Logs

The Three Magic Questions:
"Yesterday, Today, Blockers"

1. What did you do yesterday?
2. What will you do today?
3. Anything blocking your progress?

Closing Questions: "Anyone looking for work?", "Anything Else?"

Week 1:

2022/02/07 Mon, 11:30am:

All:

Informal Meeting, agree on use of "goodday.work" as PM tool

2022/02/08 Tue, 08:50:

Tom:

1. Planning, 2. Getting started with GitHub, 3. No.

Will:

1. Planning, 2. Getting started with GitHub, 3. No.

Jörg:

1. Planning, 2. Getting started with GitHub, 3. No.

2022/02/09 Wed, 08:50:

Tom:

1. Getting started with GitHub, 2. Starting out with AWS, RDS, etc., manage access issues, etc., 3. No.

Will:

1. Getting started with GitHub, 2. Starting out with AWS, RDS, etc., manage access issues, etc., 3. No.

Jörg:

1. Getting started with GitHub, 2. Starting out with AWS, RDS, etc., manage access issues, etc., 3. No.

2022/02/10 Thu, 08:50:

Tom:

1. Planning/Code Management/Scheduler, 2. Planning/Code Management/Scheduler, 3. No.

Will:

1. Planning/Documentation/Database Design, 2. Planning/Documentation/Database Design, 3. No.

Jörg:

1. Planning/Documentation/SRS, 2. Planning/Documentation/SRS, 3. No.

2022/02/11 Fri, 11:15:

Tom:

1. Planning/Code Management/Scheduler, 2. Planning/Code Management/Scheduler, 3. No.

Will:
1. Planning/Documentation/Database Design, 2. Planning/Documentation/Database Design, 3. No.

Jörg:
1. Planning/Documentation/SRS, 2. Planning/Documentation/SRS, 3. No.

Week 2:

2022/02/14 Mon, 12:45 (Walked the board):

Tom:
1. Github documentation, 2. Data Loader, 3. No.

Will:
1. Database Implementation, 2. Add sql schema to git, 3. No.

Jörg:
1. Planning/Documentation/SRS, 2. Planning/Documentation/SRS, 3. No.

2022/02/15 Tue, 12:15 (Round Robin):

Tom:
1. Started the data loader, 2. I plan to resolve the db connection issue with the loader, 3. No.

Will:
1. Created the station table/database, backup freq changed, 2. Git / Virtual env's, backup instructions, 3. No.

Jörg:
1. New User stories, SRS, wireframe in SRS, file to drive 2. Git / Virtual env's and resolve issue with virtual environment on mac , 3.No

2022/02/16 Wed, 15:45 (Round Robin):

Tom:
1., 2., 3. No.

Will:
1. Created JSON ImportLog Table, 2.Exported SQL Schema/Shared RDS Info, 3. Using right branch for SQL Schema upload (will working on tomorrow

Jörg:
1. Still trying to resolve the incompatibility issues with the virtual environment export/import (yml/txt) across different platforms (MacOS and Windows).

2022/02/17 Thurs, (Round Robin):

Tom:
1.Database RDS look/DataLoader(WIP)/Eniv Issues, 2.Word Doc of Installed/Database Talk/Weather Talk/ML Talk/Update Database, 3. No.

Will:
1.Exported SQL Schema/Shared RDS Info, 2.Database Talk/Weather Talk/ML Talk/Sample Weather Data, 3. No.

Jörg:
1.Still trying to resolve the incompatibility issues with the virtual environment export/import (yml/txt) across different platforms (MacOS and Windows). , 2. , 3.No.

2022/02/18 Fri, (Round Robin):

Tom:
1.Word Doc of Installed/Database Talk/Weather Talk/ML Talk/Update Database, 2.Installer Merge with Data Loader into Scheduler, 3. No.

Will:
1.Database Talk/Weather Talk/ML Talk/Sample Weather Data/Goodday Research/Weather Sample Data/Committed Sample Weather Data, 2. Exported and Committed New Version of Sql Schema/Goodday Research, 3. No.

Jörg:
1.Dataloader Planning/Dataloader, 2. Implement get station-ID into data loader 3.No.

Week 3:

2022/02/21 Mon, (Round Robin):

Tom:
1. Sprint Retro & Review, Create weather storage table (key), add timestamp to stationState, get weather data from API, stored data in database, merge all branches (and removed them), installed latest to EC2, tested the scheduler, fix issues with virtual environment, , 2. Sprint 1 planning, 3. No.

Will:
1. Sprint Retro & Review/Goodday Research & Communication, 2.Goodday Research/Prepare Next Sprint Scrum Techniques, 3. No.

Jörg:
1.Sprint Retro & Review, Get station-id integrated into dataloader.py 2. Update datetime format in RDS, 3.No.

2022/02/22 Tue, (Round Robin):

Tom:
1. Fixed bug with lastUpdate timestamp and location weather, 2. Sprint Planning, 3. Project Management tool because we're facing issues with the goodday burndown chart.

Will:
1.Project Management Tool Research/Scrum Planning, 2.Planning/Project Managment Tool, 3. Yes: Bugs/Research of Project Management Tool.

Jörg:
1. Datetimeformat updated in RDS, 2. Sprint Planning/, 3. No.

2022/02/23 Wed, (Round Robin):

Tom:
1. Sprint Planning/Installer/Flask, 2. Sprint Planning - yes - still on Wednesday, even more discussion on our wireframes. , 3. No.

Will:
1.Project Management Tool Research/Sprint Planning/Installed Installer, 2.Sprint Planning, 3., No.

Jörg:
1. Creating and adding user stories to goodday, 2. Sprint Planning, Gathering ideas to draft first wireframe /, 3. No.

2022/02/24 Thu , (Round Robin):

Tom:
1. Bash script to manage scheduler. Complete but needs to be installed., 2. Updating the Setup.py to create an endpoint for the management script. and testing., 3. No.

Will:
1.Sprint Planning, 2.Sprint Planning/Installing, 3., No.

Jörg:
1. Collecting ideas to draft the first wireframe, 2. Sprint Planning, drawing first wireframe, 3. No.

2022/02/25 Fri, (Round Robin):

Tom:
1. Installing the scheduler management script, 2. Sorting out the issue with our virtual environment not being available to root when we run sudo, 3. No.

Will:
1.Sprint Planning, 2.Goodday Research/Burndown options, 3. Yes: Goodday Burndown capabilities.

Jörg:
1. Drawing first wireframe/DB Copy, 2. Implementing first basic HTML/CSS layout, 3. No.

Week 4:

2022/02/28 Mon , (Round Robin):

Tom:
1. DB Copy, getting Setup.py to create an endpoint for our scheduler script. 2. Looking into web servers for the EC2 Instance, 3. No.

Will:
1.Goodday Research/Burndown options, 2.Sprint Burndown Template Creation/EC2 Costs, 3., No.

Jörg:
1. Implementing first basic HTML/CSS layout, 2. Still ongoing - implementing first basic HTML/CSS layout, 3. No.

2022/03/01 Tue, (Round Robin):

Tom:
1. Flask Server (Apache Server), 2. Flask Server (Apache Server), 3. Maybe.

Will:
1.Sprint Burndown Template Creation/EC2 Costs, 2.Sprint 0 Burndown/Google Maps API, 3., No.

Jörg:
1. Implementing first basic HTML/CSS layout, 2. Enhancing first basic HTML/CSS layout , 3. No.

2022/03/02 Wed, (Round Robin):

Tom:
1. Apache server - investigating issues with WSGI, 2. Apache server - investigating issues with WSGI (Mod_WSGI compiled with wrong python), 3. Yes

Will:
1. Sprint 0 Burndown/Google Maps API, 2.Google Maps HTML/CSS/JS code structures, 3., No.

Jörg:
1. Enhancing first basic HTML/CSS layout, 2. Further enhancing first basic HTML/CSS layout, 3. No.

2022/03/03 Thu, (Round Robin):

Tom:
1. Apache server - investigating issues with WSGI (Mod_WSGI compiled with wrong python), 2. Finally resolved the issues - our python (flask) application is live! Woots!, 3. No.

Will:
1. Google Maps HTML/CSS/JS code structures, 2. Embedding Map and Markers for Web App, 3., No.

Jörg:
1.Further enhancing first basic HTML/CSS layout, 2. Still ongoing - further enhancing first basic HTML/CSS layout, 3. No.

Week 5:

2022/03/21 Mon, (Round Robin):

Tom:
1.Integrating maps, styles, basic html, javascript, etc. 2. Update the release (setup.py) file, merge dev into main, and then do an install 3. No

Will:
1.Embedding Map and Markers for Web App 2. Sprint planning, Collecting ideas how to implement SQL query to get current weather/occupancy displayed on the front end 3. No

Jörg:
1.Further enhancing first basic HTML/CSS layout2. Sprint planning, Collecting ideas how to implement SQL query to get current weather/occupancy displayed on the front end 3. No

2022/03/22 Tues, (Round Robin):

Tom:
1. Really just admin, do a release to the website, 2. <Tom out sick today>, 3. No

Will:
1.Sprint planning, Collecting ideas how to implement SQL query to get current weather/occupancy displayed on the front end, 2. , 3. No

Jörg:
1.Sprint planning, Collecting ideas how to implement SQL query to get current weather/occupancy displayed on the front end, 2. , 3. No

2022/03/23 Wed, (Round Robin):

Tom:
1. <Tom out sick today>, 2. <Tom out sick today>, 3. No

Will:
1. , 2. , 3. No

Jörg:
1. , 2. , 3. No

2022/03/24 Thu, (Round Robin):

Tom:
1. <Tom out sick today>, 2. Add basic tables for resampled data to the database to support Jörg and Will, 3. No

Will:
1. , 2. Starting to design a basic model for ML , 3. No

Jörg:
1. , 2. Thinking about the data input for the machine learning model, 3. No

2022/03/25 Fri, (Round Robin):

Tom:
1. Just basic admin - add tables to database(s) for ML model, 2. Looking at sprint review and sprint retrospective., 3. No

Will:
1. Starting to design a basic model for ML, 2. Implementing a basic version of the ML model , 3. No

Jörg:
1. Thinking about the data input for the machine learning model, 2. Starting to implement the resample collected data (occupancy, weather) into a more suitable format for the ML model, 3. No

Week 6:

2022/03/28 Mon, (Round Robin):

Tom:
1. Added colored icons for the bikes on the map, 2. Adding an end-point for the occupancy data , 3. No

Will:
1. DataPrep/Created a basic model for ML, 2. Sprint planning , 3. No

Jörg:
1. Having started resampling the weather/occupancy data, 2. Continue to resample the weather/occupancy data, 3. No

2022/03/29 Tues, (Round Robin):

Tom:
1. Small server tweaks, adding the end-point for the occupancy data, 2. Including the doesn't-yet-exist data_resampler into the scheduler and installer, 3. No

Will:
1. Sprint Planning, 2. Created Pickle from Model/Deserialized Pickle for dwmb.py, 3. No

Jörg:
1. Sprint Planning, 2. Querying occupancy data from the RDS using pandas , 3. External factors have prevented me from progressing: The urge to install JavaFx and set up project collaboration using Eclipse/Github to get up and running with the Java group project.

2022/03/30 Wed, (Round Robin):

Tom:
1.Integrating the data_resampler into the scheduler and installer, 2. Tidying up, refactoring scheduler , 3. No

Will:
1. Created Pickle from Model/Deserialized Pickle for dwmb.py, 2. Logistic Model, 3. No

Jörg:
1.Querying occupancy data from the RDS using pandas , 2. Start implementing data resampler in Jupyter Notebook, 3. External factors have prevented me from progressing: The urge to install JavaFx and set up project collaboration using Eclipse/Github to get up and running with the Java group project.

2022/03/31 Thur, (Round Robin):

Tom:
1. Tidying up, refactoring scheduler, 2., 3. No

Will:
1. Logistic Model, 2. Logistic Model, 3. No

Jörg:
1. Started implementing data resampler in Jupyter Notebook, 2. Implementing function "station state hourly" - data resampler, 3. No

2022/04/01 Fri, (Round Robin):

Tom:
1., 2. Front-end slider work, 3. No

Will:
1. Logistic Model, 2. Predict Route/Forecast Weather API/Working with Forecast Weather Return, 3. No

Jörg:
1. Implementing function “resample station state hourly” - data resampler, 2. Testing function “resample station state hourly” - data resampler, 3. No

Week 7:

2022/04/04 Mon, (Round Robin):

Tom:
1., 2. Fix google maps API Key for live, release version live (includes resampler), 3. No

Will:
1. Predict Route/Forecast Weather API/Working with Forecast Weather Return, 2. Sprint Planning/Sprint Review, 3. Deadline moved up two days per Prof. Lawlor

Jörg:
1. Testing function “resample station state hourly” - data resampler, 2. Implementing function “resample weather state hourly” - data resampler, 3. No

2022/04/05 Tue, (Round Robin):

Tom:
1.Release, Maps, 2. Fix bug in resampler, rename ‘dl_loader’ to ‘scheduler’ and re-install, 3. No

Will:
1. Sprint Planning/Sprint Review, 2. Prototyped the results for the Predict Route, 3. No

Jörg:
1. Implementing function “resample weather state hourly” - data resampler, 2.Keep on implementing function “resample weather state hourly” - data resampler, 3. No

2022/04/06 Wed, (Round Robin):

Tom:
1. Fix bug, rename scheduler, 2. Work on using ORM in python back end for data loading (basic queries, joins, filters (aka where clauses), 3. No

Will:
1. Prototyped the results for the Predict Route, 2. Updated and/or created Multi Linear Reg & Random Forest Models/Create Skeleton for Project Report, 3. No

Jörg:
1 .Keep on implementing the function “resample weather state hourly” - data resampler, 2. Testing function “resample weather state hourly” - data resampler, 3. No

2022/04/07 Thu, (Round Robin):

Tom:
1. SQL Alchemy ORM, 2. flask endpoint for prediction model, flask endpoint for occupancy, 3. No

Will:
1. Updated and/or created Multi Linear Reg & Random Forest Models, 2. Evaluated Models/Worked on pickle files and their size issues, 3. No

Jörg:
1.Testing function “resample weather state hourly” - data resampler, 2. Refactoring prototyping code of data resampler from Jupyter Notebook into clean Python code , 3. No

2022/04/08 Fri, (Round Robin):

Tom:
1. Flask endpoints, 2. Worked with Will on splitting the prediction model from single ‘monolith’ to model per station, focusing on reduced size to reduce load times from current 55s to circa 5, 3. No

Will:
1. Evaluated Models/Worked on pickle files and their size issues, 2. Worked on modifying pickle file to lower the load of running it on the app with Tomas, 3. No

Jörg:
1.Refactoring prototyping code of data resampler from Jupyter Notebook into clean Python code, 2. Implementing search mode ‘available bikes’ & ‘available spaces’ in Javascript front-end, 3. No

Week 8:

2022/04/11 Mon, (Round Robin):

Tom:
1. gviz api - introduced and done (for occupancy charts), complete server rebuild (from scratch) on new EC2 instance, fix map markers not zooming and not displaying correctly , 2. <nothing! - mad java push today>, 3. No

Will:
1. Worked on modifying pickle file to lower the load of running it on the app with Tomas, 2. Grabbed weather icons for weather display on app/Created logic and function to dynamically display weather icons due to weather forecasts, 3. No

Jörg:
1.Implementing search mode ‘available bikes’ & ‘available spaces’ in Javascript front-end, 2. Start implementing ‘getBikeIconUrl’ in Javascript front-end, 3. No

2022/04/12 Tue, (Round Robin):

Tom:
1. Install hotfix live - was a couple of days overdue. Issues remain, but closer, 2. Project report, 3. No

Will:
1. Grabbed weather icons for weather display on app/Created logic and function to dynamically display weather icons due to weather forecasts, 2. Writing Data Analytics & Process for Report , 3. No

Jörg:

1.Start implementing ‘getBikeIconUrl’ in Javascript front-end, 2. Completing the implementation of ‘getBikeIconUrl’ in Javascript front-end. Creating the function ‘createMarkers’ in the front-End, 3. No

2022/04/13 Wed, (Round Robin):

Tom:

1. Dealing with outstanding issues, 2. Project report (Architecture section), 3. No

Will:

1. Writing Data Analytics & Process for Report, 2. Continued writing and editing for report, 3. No

Jörg:

1.Completing the implementation of ‘getBikeIconUrl’ in Javascript front-end. Creating the function ‘createMarkers’ in the front-End,, 2 Implementing range slider for date&time selection in Javascript front-end., 3. No

2022/04/14 Thu, (Round Robin):

Tom:

1. Project report, 2. Update front end to display predicted data on the map when slider changes (was not triggering), 3. No

Will:

1. Continued writing and editing for report, 2. Updated some model details/Finishing first draft of data analytics and process for report/Report editing, 3. No

Jörg:

1.Implementing range slider for date&time selection in Javascript front-end, 2. Bug Fixing Javascript front-end, 3. No

2022/04/15 Fri, (Round Robin):

Tom:

1. Getting predicted data to display properly on map, 2.Final updates to the front end, add map legend, remove unnecessary data loads , 3. No

Will:

1. Updated some model details/Finishing first draft of data analytics and process for report/Report editing, 2. Final Burndown/Writing for reflection and meeting log section of report/Title page, table of contents, appendix, formatting, and final editing/PDF report & Submit report, 3. No

Jörg:

1.Cleaning up dead code from Javascript front-end, 2. DONE!, 3. No.

Appendix C: Sprint Storyboards

Sprint 00

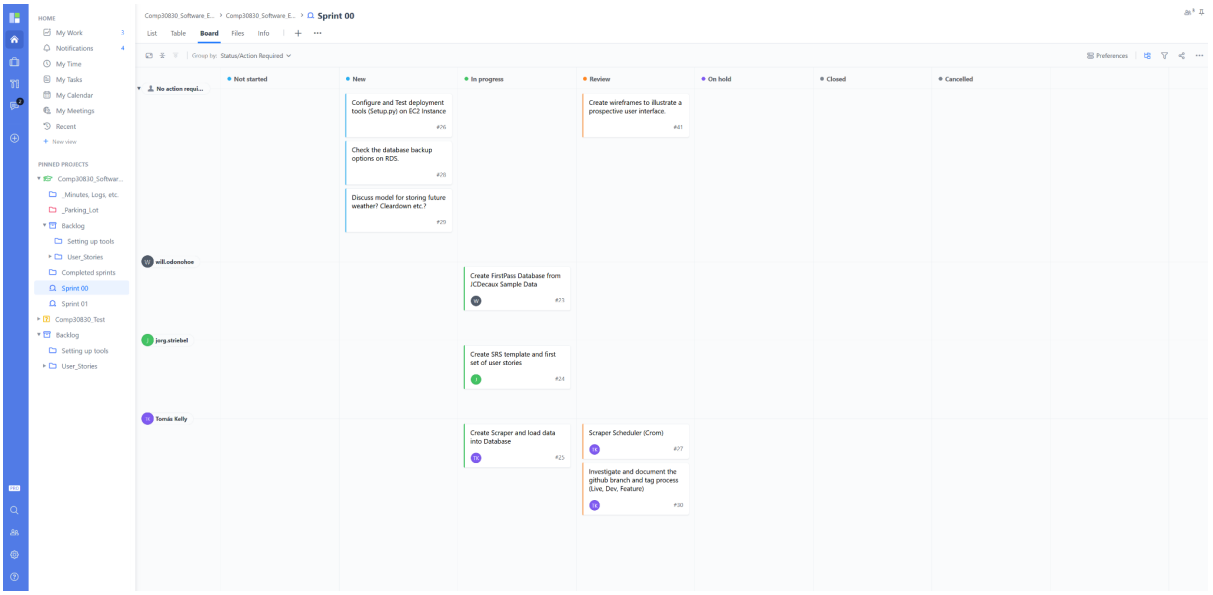

Figure 18: End of the first week of Sprint 00 - Storyboard

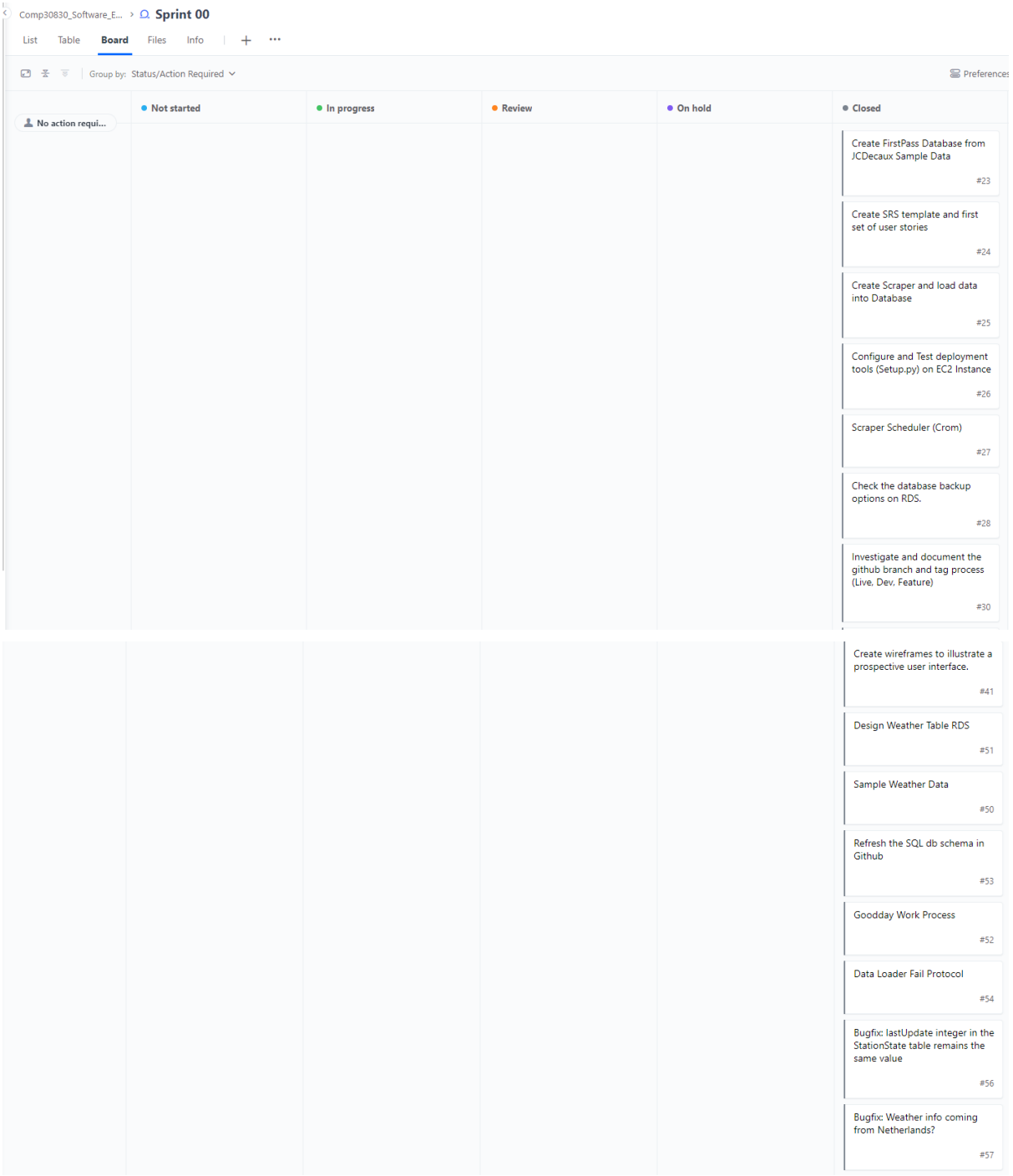

Figure 19: End of Sprint 00 - Storyboard

Sprint 01

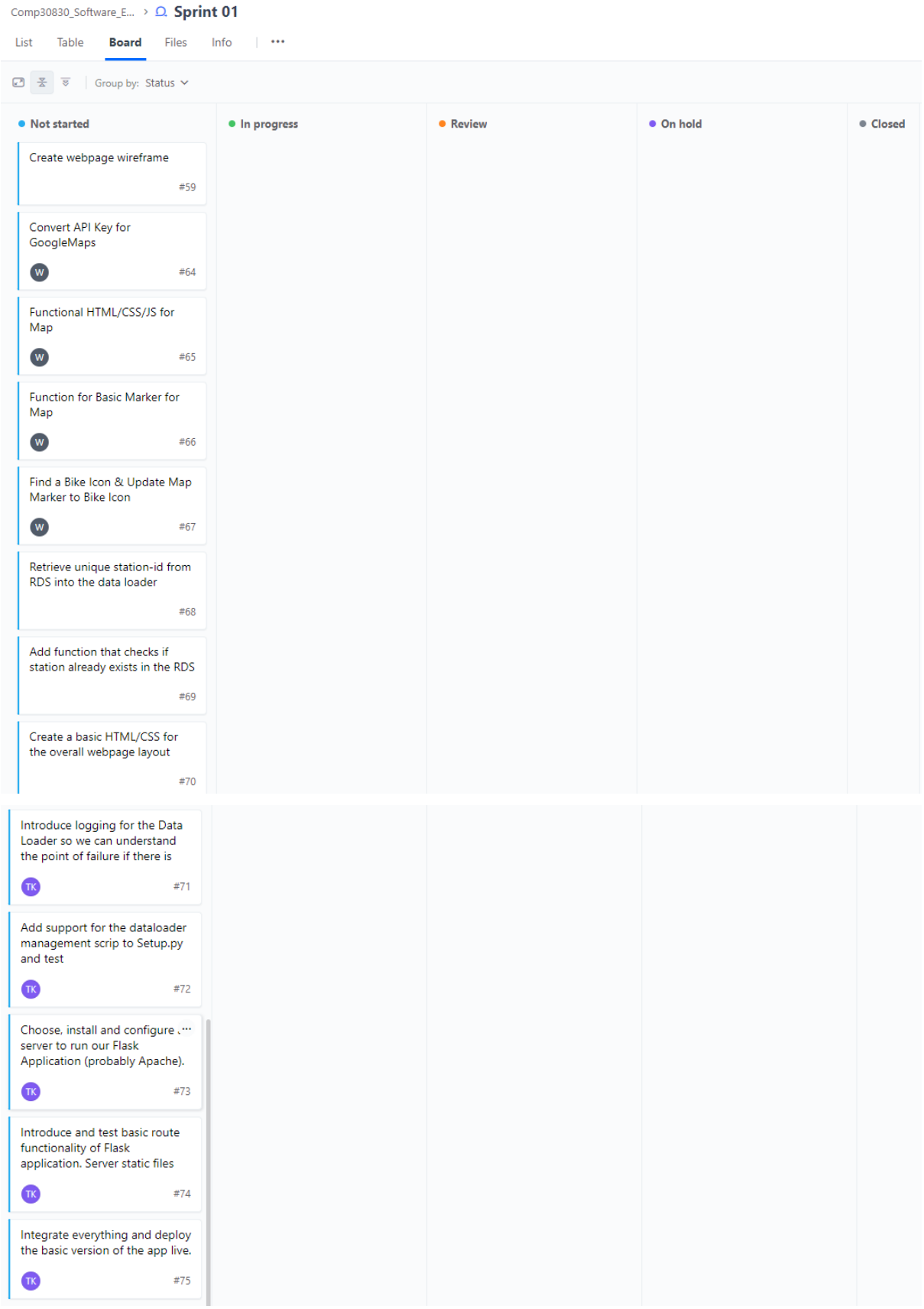

Figure 20: Start of Sprint 01 - Storyboard

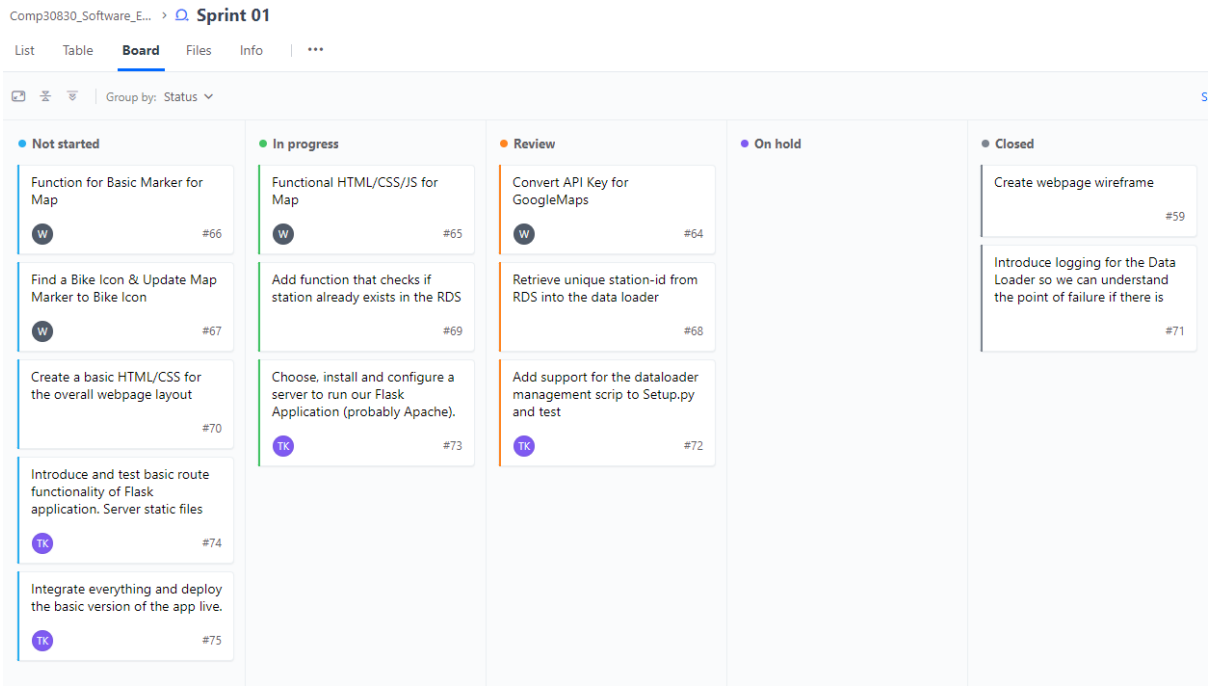

Figure 21: End of the first week of Sprint 01 - Storyboard

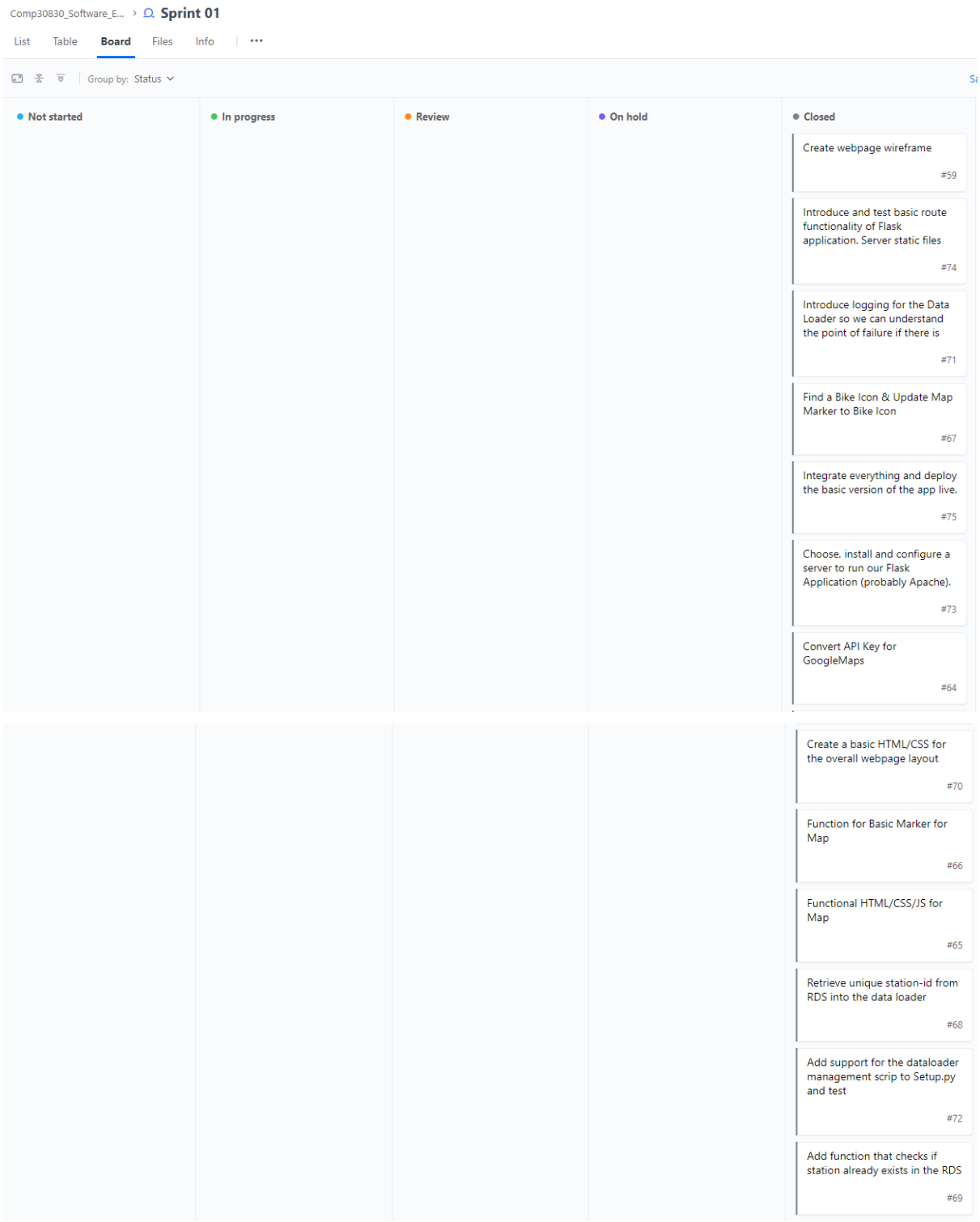

Figure 22: End of Sprint 01 - Storyboard

Sprint 02

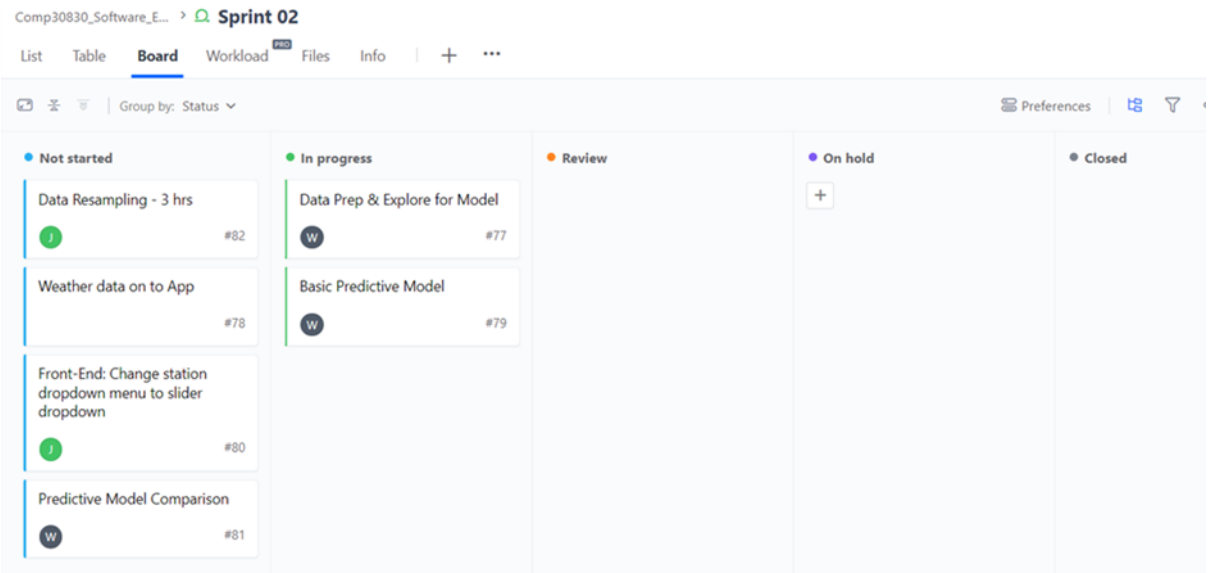

Figure 23: Start of Sprint 02 - Storyboard

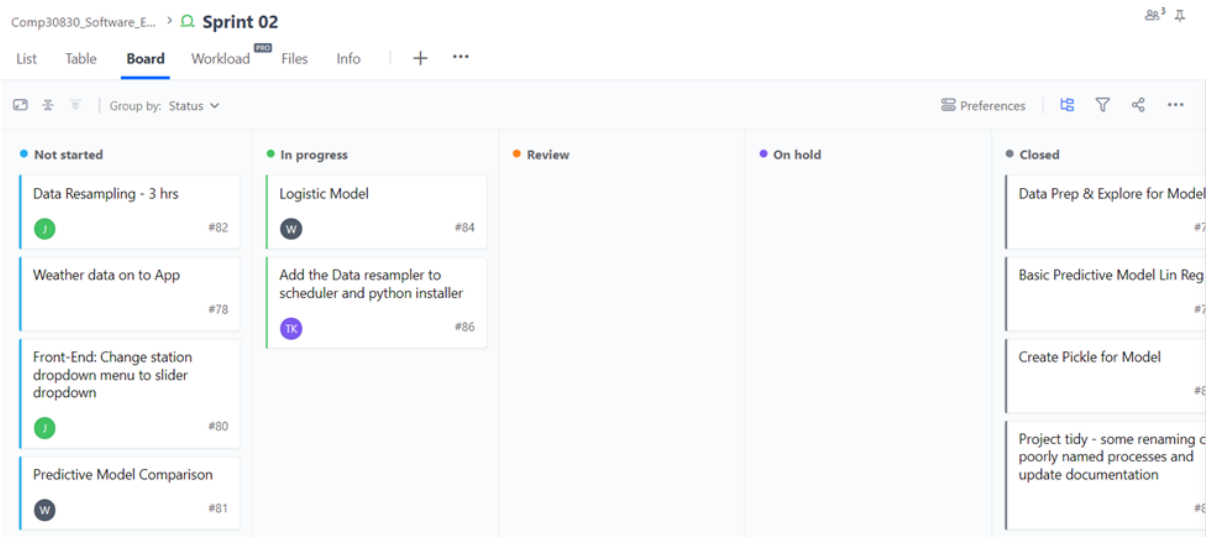

Figure 24: End of the first week of Sprint 02 - Storyboard

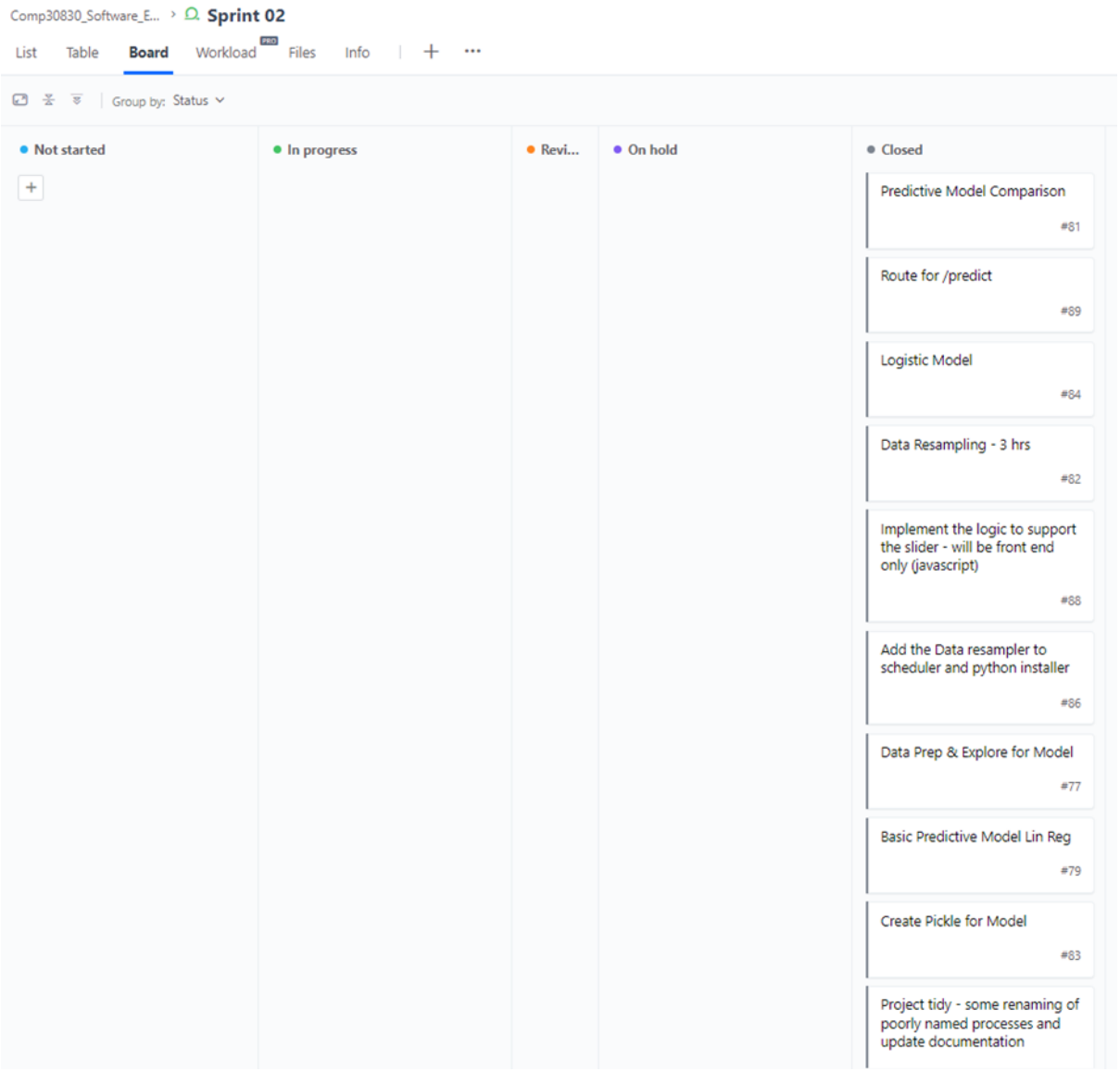

Figure 25: End of Sprint 02 - Storyboard

Sprint 03

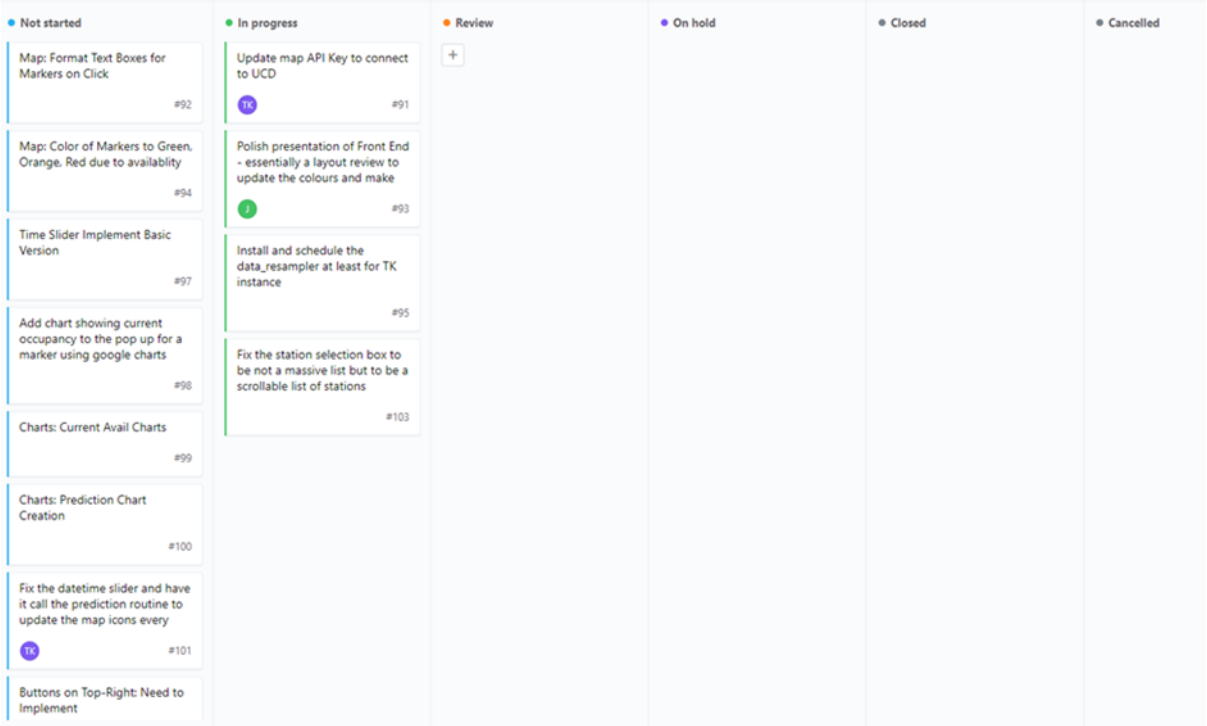

Figure 26: Start of Sprint 03 - Storyboard

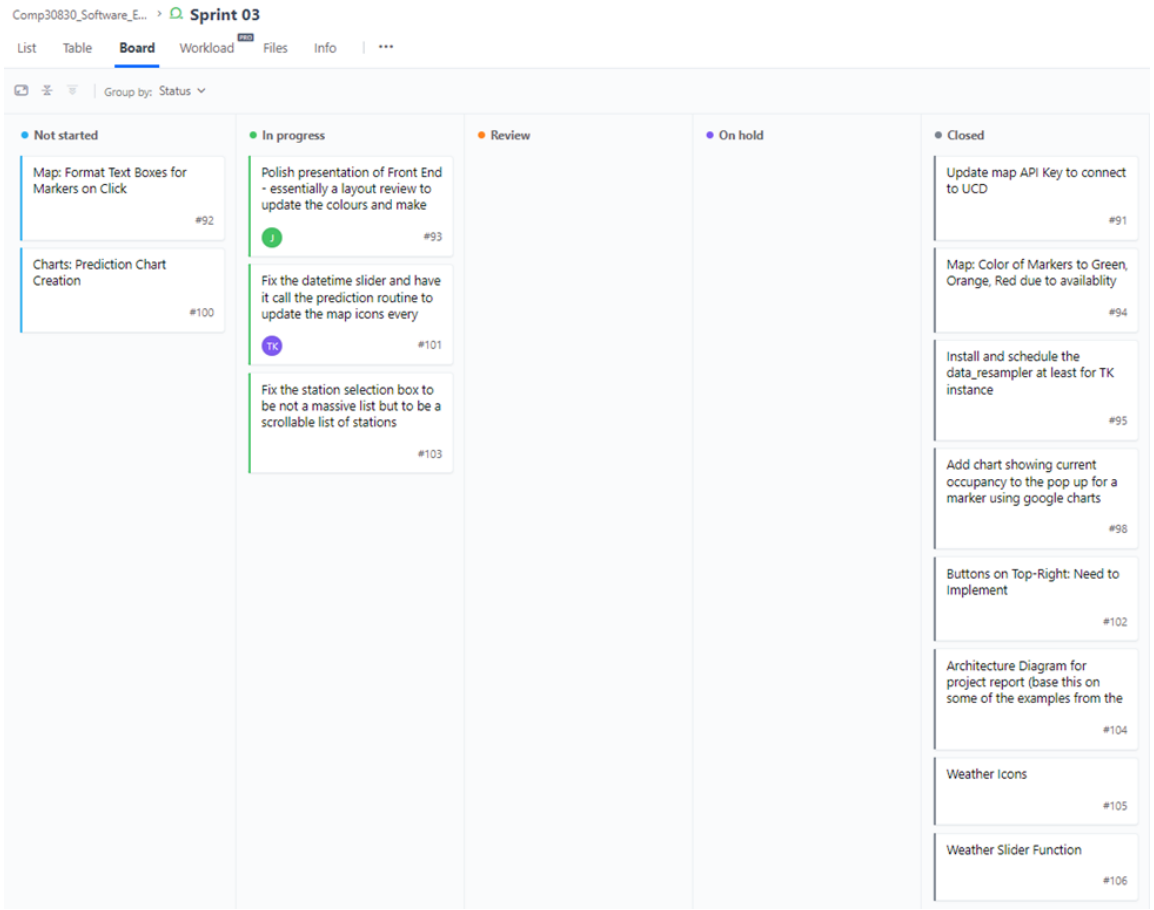

Figure 27: End of the first week of Sprint 03 - Storyboard

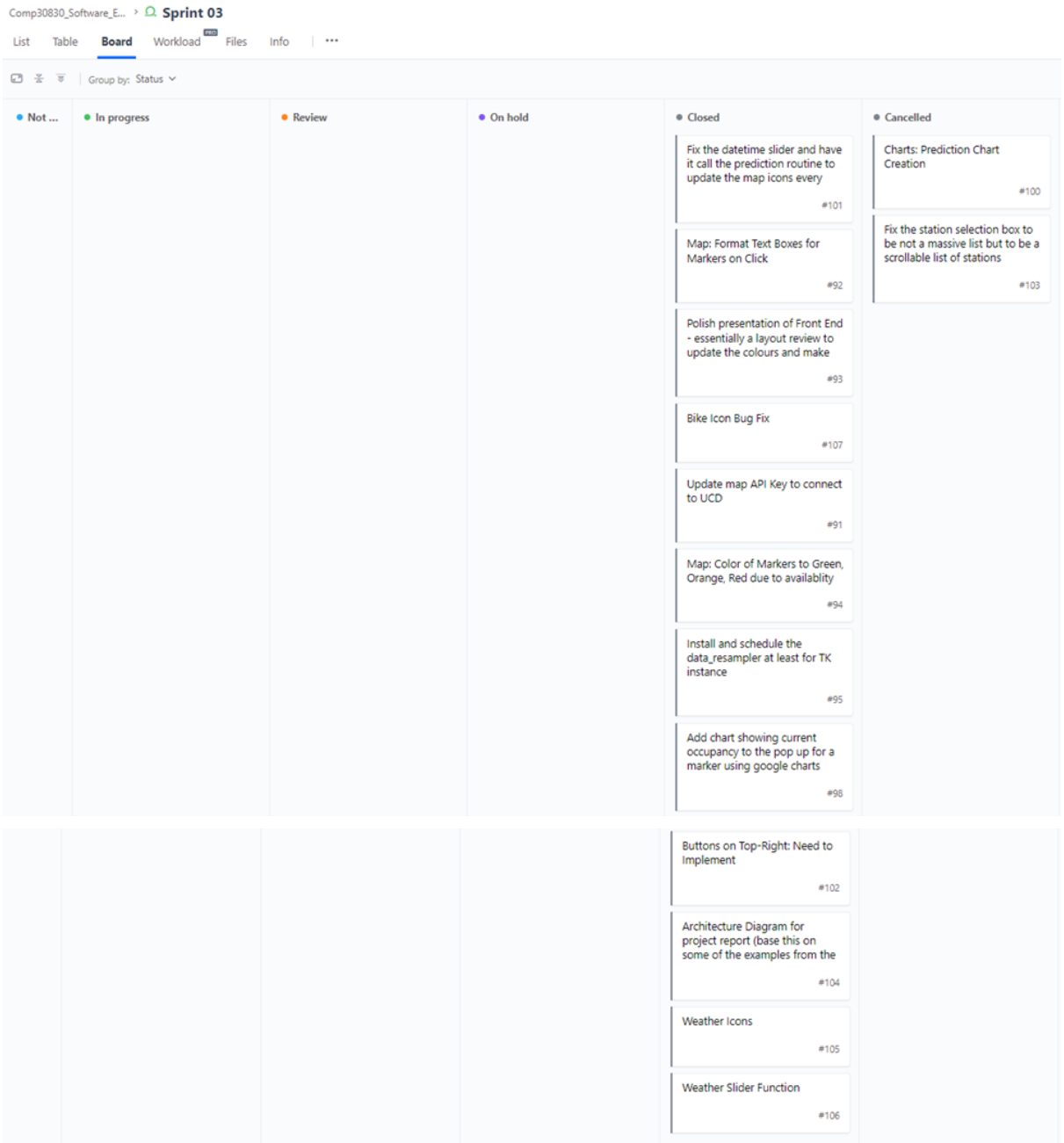

Figure 28: End of Sprint 03 - Storyboard

Appendix D: Features & Product Backlog

An “X” indicates in which sprint the backlog item was addressed.

Table 4: Feature & Product Backlog Across All Sprints

Feature	Description	Product Backlog	Sprint 00	Sprint 01	Sprint 02	Sprint 03
Overall Function	The back-end and structures that make the application run.	Wireframe structure of App	X			
		Scraper	X			
		Scheduler	X			
		Data Loader		X		
		RDS	X			
		Flask		X		
		Design App		X		
		Create HTML/JS/CSS Structure		X		
		RDS Database Structure/Implementation	X			
Map	An interactive map where the user can see where bike stations are located in Dublin. The station markers will have station information	Google Maps API		X		
		JCDecaux API for bike data	X			
		Display Dublin on Map		X		
		Display markers for station location		X		
		Display Available Bikes			X	
		Display Available Spaces			X	
		Display Station Information			X	
		Display information in map markers			X	
Available Bikes/Spaces Buttons	Two buttons where the user can choose between bikes or spaces depending if they are looking for a bike or looking to return a bike.	Functionality for Available Bikes				X
		Functionality for Available Spaces				X
Station Dropdown	An interactive dropdown where the user can choose which station to focus on. Will update necessary data/information.	Create Dropdown		X		
		Grab Station Information	X			
		Use Station Name for Dropdown		X		

Date Slider	An interactive slider for the user to choose to focus on a time in the future. Will update necessary data/information/charts.	Create Slider			**X**	
		Open Weather API			**X**	
		Open Weather Forecast (one call) API			**X**	
		Connect data to slider				**X**
Past Availability Chart	A chart that shows the user past bike data	Grab data from RDS		**X**		
		Display bike data				**X**
		Create past availability chart				**X**
Future Availability Chart	A chart that shows the user future predictions of bike/space availability based over past bike and weather data, and weather forecasts.	Data Handling			**X**	
		Data Cleaning			**X**	
		Model Planning			**X**	
		Model Comparisons			**X**	
		Integrating Model into App				**X**
		Output Predictions into chart				**X**