**REAL-TIME DASHBOARD FOR ANALYZING THE DELAY IN DUBLIN BUS CAUSED BY LOCAL WEATHER**

**ABSTRACT**

Public transportation plays a crucial role in the daily lives of citizens across countries, facilitating their routine activities. Dublin Bus stands as a prominent public transport system within the city of Dublin, greatly facilitating the daily lives of its residents. However, occasional unforeseen delays in bus arrivals may inconvenience passengers, despite the overall utility the service provides. This study aims to analyze the delays experienced by Dublin buses on individual routes and explore any potential correlation with local weather conditions. It seeks to determine whether local weather factors contribute to these delays. The method involves developing a real-time dashboard to showcase delays, calculated using the NTA dataset, which is GTFS and GTFS-R data, in conjunction with local weather data. The dashboard provides valuable assistance to the public by enabling users to identify delays on specific Dublin bus routes at particular times or weather conditions. This functionality aids in planning journeys more effectively, especially during specific time frames or weather scenarios. The summary of the primary findings indicates that local weather significantly contributes to delays in Dublin buses.

**Keywords: Dublin Bus, Delay, Real-time dashboard**

1. **INTRODUCTION**

For any modern country, public transportation plays a crucial role in the daily lives of its citizens. A fast and reliable transportation network is vital for easy access across the country. Time is invaluable and cannot be regained once lost, making punctuality a key aspect of public transport services. One of the significant challenges in public transport is the city service, which is heavily utilized by most people for their daily activities such as commuting to work, school, and other destinations. The reliability and availability of city services, like buses, are crucial, especially in cities like Dublin, where the Dublin Bus system is the primary mode of transportation due to its widespread availability and dependability. Dublin Bus is an Irish state-owned bus operator providing services in Dublin(‘Dublin Bus’, 2024). Dublin bus is the largest bus operator in the city, it carried over 145 million passengers in the year of 2023(‘Dublin Bus’, 2024).

Despite being one of the most widely utilized public transportation systems in Ireland, Dublin Bus occasionally encounters unforeseen circumstances leading to significant delays. At times, passengers experience prolonged waiting periods before boarding a bus, and even upon boarding, journeys may take longer than expected to reach their destinations. Dublin Bus, affiliated with the National Transport Authority of Ireland, offers advanced services including travel schedules, estimated arrival times, travel predictions, vehicle positioning, and real-time updates to passengers. However, despite these provisions, passengers may occasionally experience delays in reaching their destinations within the estimated arrival times. Road transport, especially buses, is vulnerable to delays influenced by factors like traffic, weather conditions, and road infrastructure. Weather significantly affects journey durations. Adverse weather conditions not only affect the operational efficiency of buses but also alter road conditions, leading to increased travel times and extended stops at each station, consequently elongating transit durations. “Weather can affect the total trip duration by increasing the access time, transfer time and the normal trip duration and also by causing schedule disruptions”(Singhal, Kamga and Yazici, 2014). The aim is to examine the influence of weather on Dublin Bus, specifically assessing its impact on travel times and potential delays. This analysis seeks to understand how varying weather conditions contribute to fluctuations in the efficiency and punctuality of bus transportation services.

* 1. **Research Question**

Arrived at research question by initially selecting the broader topic of challenges in public transport in Ireland. Subsequently, honed in on the research problem, focusing specifically on the timeliness of public transport, particularly buses. The purpose statement outlined the objective of analyzing the causes of delays in scheduled arrival times. This process led to the final research question concerning the correlation between local weather conditions and the punctuality of public transport. And the research question is follows:

***“How much does the local weather affect the delay of Dublin buses?”***

Figure 1 Narrowed down to the Research Question

dfff

Problems in Public Transport in Ireland

Timeliness of Public Transport in Ireland, especially Dublin Bus

Analysis on the causes of delays in scheduled arrival time

Question?

* 1. **Research aim(s)**

The aim is to publish the extent to which weather conditions impact delays in public buses, especially Dublin buses, determining which weather conditions are most conducive to bus delays. The study also seeks to publish the average/current duration of delays associated with each distinct weather condition or distinct times of the day in general or from specific bus routes.

* 1. **Research objective(s)**

The National Transport Authority of Ireland provides a transport feed service containing essential information such as trip schedules and real-time updates, including vehicle positioning and arrival times. Utilizing this data along with local weather information from Dublin city allows us to conduct diverse analytical activities to achieve our research objectives. This involves publishing bus delays on specific days under a particular weather condition and identifying which weather conditions contribute most to delays. The objective also includes displaying the current delay, both in general and for specific routes of Dublin Bus.

* 1. **Research hypothesis**

Dublin Bus delays are more probable during days characterized by rain and strong winds. Adverse weather conditions, specifically rain and wind, tend to have a notable impact on the timeliness of bus services, increasing the likelihood of delays during such conditions.

1. **LITERATURE REVIEWS**

“Weather conditions are considered exogenous factors which indirectly influence the demand for transit”(Singhal, Kamga and Yazici, 2014). Bad weather conditions directly affect both the accessibility and the quality of transit services. The study done by Singhal, Kamga and Yazici in 2014 is analyzing the impact of weather on the Metropolitan Transportation Authority-New York City Transit subway ridership. They obtained two years of ridership data and weather information from the National Oceanic and Atmospheric Administration (NOAA) and the Weather Underground website. Subsequently, they developed a model using ordinary least squares regression (OLS).

“Weather conditions and built environment contribute 30.22 and 55.83% to ridership fluctuations, respectively”(Lin *et al.*, 2020). “Adverse weather, such as strong wind, high humidity, or heavy rainfall, has a more disruptive impact on leisure-related areas than on residence and office areas”(Lin *et al.*, 2020). The study of ‘Analyzing the relationship between weather, built environment, and public transport ridership’ done by Lin et al, gathered smart card data in Beijing, China, spanning from February 2018 to January 2019, including both bus and subway usage. The study considered various factors, including daily weather conditions (such as temperature, wind speed, humidity, rainfall, snowfall, and air quality) and built environment factors (like residential and office density, as well as accessibility of public transport infrastructure in a Traffic Analysis Zone). The Light Gradient Boosted Machine (LightGBM) algorithm was utilized to examine the influence of weather conditions (Lin *et al.*, 2020).

“Weather has a huge impact on many aspects of traveler’s travel decision, for example, departure time, route and mode choices”(Khattak and De Palma, 1997; Miranda-Moreno and Nosal, 2011). “All four weather variables, namely humidity, wind speed, rainfall and temperature are found to have statistically significant negative effects on bus ridership”(Li *et al.*, 2015). The study done by Li et al categorizes the ridership data from the smartcard according to route types (RTs) and seasons, and then analyzes the impact of different weather factors on various types of routes separately (Li et al., 2015).

“The effect of weather can be direct, by making people drive more slowly, or indirect, e.g. by people choosing different means of transportation or by increasing the amount of accidents which can then cause even more congestion” (Peltola, 2019). This study was carried out by recording bus link travel times between consecutive bus stops from live location data provided from Tampere regional buses during different weather circumstances (Peltola, 2019).

The research on forecasting irregularities in transit bus arrival times by Alam et al. involved predicting the likelihood of arrival time disruptions. They utilized GPS coordinates data provided by the Toronto Transit Commission (TTC) in conjunction with hourly weather data. Machine learning models, particularly Long Short-Term Memory Recurrent Neural Network (LSTM) models, were employed in this study (Alam *et al.*, 2021).

Turning to studies conducted on Dublin Bus, the research by Akhil Alfons and Kirthy Francis centers on constructing a system that gathers transit feed in real-time and utilizes this data to forecast delays (Kodiyan and Francis, 2020). Another study conducted by Pandurangi et al. focuses on predicting bus travel times through a Segmentation Approach. They developed a user-friendly application that employs machine learning techniques to estimate the travel time of buses and destinations (Pandurangi *et al.*, 2020). The research done by French and O’Mahony utilized Automatic Vehicle Location System (AVLS) data to explore the influence of adverse weather conditions on bus journey times. Rainfall, temperature, and wind speed were utilized as indicators to assess their impact. Additionally, that study aimed to determine if the impacts varied across three types of bus routes: those with bus lanes along the entire route, those with bus lanes along part of the route, and those where buses operate in mixed traffic (French and O’Mahony, 2021).

The rationale is to enhance the public transit planning through real-time analysis of Dublin bus delays and weather. From previous research and studies, the problem becomes evident. Various analytical methods have been employed in previous studies to address the challenges related to timelines faced on public transport like Dublin Bus. However, this research introduces a novel approach by providing a real-time dashboard displaying delay information alongside corresponding weather data. This dashboard enables individuals to access current delay information as well as historical delays, which can vary due to external factors. The significance of a real-time dashboard lies in its ability to assist the public in planning their journeys more effectively, offering additional insights into potential delays they may encounter during their travels.

The research is conducted by collecting and analyzing data using the General Transit Feed Specification (GTFS). The General Transit Feed Specification (GTFS) is an open standard used by most of the public transport providers to publish relevant information about transit systems to riders (‘GTFS: Making Public Transit Data Universally Accessible’, no date). Since there is no existing dataset available for this study, data needs to be recorded in the desired format. This involves utilizing GTFS schedule and real-time data to capture information in a specific format. By integrating this data with weather data, the goal is to establish a robust real-time data analytical model. The objectives include investigating the impact of weather conditions on delays and determining which weather conditions are more conducive to public transport delays, all accessible through the dashboard.

1. **METHODOLOGY**

For any dashboard, the primary focus is on data flow. Data must seamlessly transition from its source to the backend services and ultimately display in the specific format on the dashboard. The key consideration is ensuring this process occurs smoothly, given that the dashboard operates in real-time. Dublin Bus collaborates with the National Transport Authority, disseminating schedule and trip updates to passengers in GTFS format. GTFS, or General Transit Feed Specification, is the standard format for this data. The methodology involves utilizing both GTFS and GTFS-R data along with the open weather data from Dublin city to compile the information for the delay dashboard.

* 1. **GTFS**

The General Transit Feed Specification (GTFS) is an open standard employed to disseminate pertinent details regarding transit systems to passengers. This standard enables public transit agencies to publish their transit data in a format that can be easily utilized by a wide array of software applications. Presently, the GTFS data format is adopted by numerous public transport providers. GTFS consists of two primary components: GTFS Schedule and GTFS Realtime. GTFS Schedule encompasses information pertaining to routes, schedules, fares, and geographic transit particulars, all presented in simple text files. This user-friendly format facilitates simple creation and maintenance without the need for complex or proprietary software. On the other hand, GTFS Realtime includes updates on trips, vehicle positions, and service alerts. It utilizes Protocol Buffers, which serve as a language- and platform-neutral mechanism for serializing structured data (‘GTFS: Making Public Transit Data Universally Accessible’, no date).

* 1. **DATA COLLECTION**

The data collection process entails gathering GTFS static data and creating a code module responsible for extracting delays from all major routes and inserting them into the database. The database used is the GTFS schedule, which primarily consists of static transit information. This data is structured into several text files (.txt) contained within a single ZIP file. Each file within the archive describes a specific aspect of transit information, such as stops, routes, trips, and fares (‘GTFS: Making Public Transit Data Universally Accessible’, no date). The GTFS static data is accessible for download from the National Transport Authority's developer portal and can be imported into a database for further analysis. GTFS Realtime is a feed specification enabling public transportation agencies to deliver real-time updates about their fleet to application developers. It serves as an extension to GTFS (General Transit Feed Specification), an open data format for public transportation schedules and related geographic data. GTFS Realtime prioritizes ease of implementation, seamless interoperability with GTFS, and a strong emphasis on providing passenger information (‘GTFS: Making Public Transit Data Universally Accessible’, no date). The feeds are delivered through HTTP and are updated regularly. Among the plethora of information provided in GTFS-R, the focus is on collecting trip updates. This category primarily includes details about delays, cancellations, and route modifications, all of which are crucial for monitoring the real-time status of transit operations. The TripUpdates services are invoked at regular intervals to calculate delay parameters, and the resulting data can be pushed into the database. This database can be either the same as the one containing the static GTFS data or a separate one.

One challenge encountered when reading GTFS static data is the unpredictable nature of changes resulting from updated route and timetable information provided by operators. These changes, originating from the operator or agency level, can occur unexpectedly. The solution recommended by the NTA help desk is to regularly update the static data to prevent discrepancies between real-time and static GTFS data. Identified this issue during the initial stages and labeled it as the static data refresh problem. Developed a solution at the same stage, which involves creating a dedicated table for mapping bus route IDs to bus route names and indicating active routes. The visual representation of this solution to the static data refresh problem is as follows:

Figure 2 Solution for static data refresh problem.

A diagram of a data processing process

Description automatically generated

This solution can be implemented within the NTA static loader module, responsible for reading static data dumps from the NTA and loading the data into the master database. Originally intended for one-time execution, the static loader module will now be triggered each time a refresh occurs in the static data due to the static data refresh problem. The NTA static loader module can be integrated into the extract engine, which operates at periodic intervals. Whenever an exception arises due to a mismatch between existing route IDs in the master database and route IDs from the latest dump, control is transferred to the NTA static loader module. Here, the module identifies the mismatch and proceeds to clear all static tables before updating them with the latest data from the NTA static dump. Upon completion of the process, the NTA static loader updates the route mapping table, maintaining records of all route IDs to date, and marking the current route IDs as active. The advantage of this solution is that the dashboard will have access to both historic and current delay information.

Another crucial dataset required for the delay dashboard is weather data. Numerous weather services offer forecasts or current weather conditions for any given location. Open-Meteo is one such weather service that provides up to 10,000 API requests per day for free of cost. Open-Meteo collaborates with national weather services to provide open data with high resolution, ranging from 1 to 11 kilometers. The concept is to gather weather data from Dublin city concurrently with the collection of GTFS-R data. The periodic data will encompass various types of weather information at different times of the day. To ensure data consistency, ensure that each delay entry is accompanied by corresponding weather data. This weather data can also be inserted into the same database used for storing the GTFS-R data.

* 1. **DATA FORMAT**

GTFS data adheres to a specific structure, which is also followed by the NTA dataset. However, when it comes to the data requirements for the delay dashboard, the data must be stored in a specific format that can be easily retrieved and processed for displaying on the dashboard. The goal is to maintain data integrity, ensuring that each delay entry is identified as a distinct occurrence. To achieve this, each entry must be assigned a unique identifier, serving as an index. Additionally, every delay entry must include a timestamp indicating when the delay occurred. To display delay information for routes, the route\_id, which serves as the unique identifier for a specific route, is required. Additionally, the direction of the route where the delay information is calculated must also be included. When it comes to the data format of weather information, it is straightforward to utilize the data supplied by Open-Meteo. This data can be paired with the entry ID, serving as the unique identifier for the corresponding delay information. Additionally, the location from which the weather is obtained, and the timestamp of the data entry must also be included. The weather data supplied by Open-Meteo includes temperature, relative humidity, apparent temperature, day or night, precipitation, rain, showers, snowfall, weather codes, cloud cover, pressure, surface pressure, wind speed, wind direction and wind gusts. The weather codes follow WMO (WORLD METEOROLOGICAL ORGANIZATION) Weather interpretation codes:

|  |  |
| --- | --- |
| **Code** | **Description** |
| **0** | Clear sky |
| **1, 2, 3** | Mainly clear, partly cloudy, and overcast |
| **45, 48** | Fog and depositing rime fog |
| **51, 53, 55** | Drizzle: Light, moderate, and dense intensity |
| **56, 57** | Freezing Drizzle: Light and dense intensity |
| **61, 63, 65** | Rain: Slight, moderate, and heavy intensity |
| **66, 67** | Freezing Rain: Light and heavy intensity |
| **71, 73, 75** | Snow fall: Slight, moderate, and heavy intensity |
| **77** | Snow grains |
| **80, 81, 82** | Rain showers: Slight, moderate, and violent |
| **85, 86** | Snow showers slight and heavy |
| **95 \*** | Thunderstorm: Slight or moderate |
| **96, 99 \*** | Thunderstorm with slight and heavy hail |

(\*) Thunderstorm forecast with hail is only available in Central Europe

* 1. **DELAY CALCULATION**

In GTFS-R, trip delays are linked to the TripUpdate API. The TripUpdate API response includes details about all active trips at the current moment, along with timestamps. This information encompasses the scheduled start time, schedule relationship (e.g., SCHEDULED or CANCELLED), direction ID indicating the trip's heading, and route ID specifying the operating route. Additionally, the response includes stop time updates, which detail arrival and departure delays at upcoming stops along the bus route. The objective is to utilize these delay values to compute the average delay of Dublin buses or the average delay on a specific route at any given moment. The average delay of Dublin buses is calculated using the following equation:

Let 𝐷𝑖​ represent the delay value for the 𝑖𝑡ℎ bus trip, where 𝑖 ranges from 1 to 𝑛. The average delay is determined by adding together the delay values from the 𝑛 bus trips and dividing by the total number of bus trips.

The delay for a specific route is calculated by first finding the average delay of each trip, and then determining the cumulative average of trips on that route. Here's the mathematical equation to calculate the average delay of a bus route:

Let 𝑇𝑖 represent the number of trips on the 𝑖𝑡ℎ bus route, where 𝑖 ranges from 1 to 𝑛 and 𝐷𝑖,𝑗 represent the average delay of the 𝑗𝑡ℎ trip on the 𝑖𝑡ℎ bus route, where 𝑗 ranges from 1 to 𝑇𝑖. The equation calculates the average delay for each bus route by taking the cumulative average of average delays of each trip on that route. The calculated delays are stored in the master database, along with unique entry IDs and timestamps, to facilitate easy retrieval for display on the dashboard.

* 1. **DASHBOARD REQUIREMENTS**

The main objective of the dashboard prioritizes functionality over design. The dashboard should effortlessly retrieve delay information from the database and display it seamlessly. It should be able to present the average delay of Dublin buses across all routes, as well as the current delay status from all routes. Additionally, the dashboard should provide historical delay data and be capable of plotting delays on graphs. Weather parameters should also be included for correlation analysis with delays. Users should have the ability to filter the graphs based on their preferences, selecting either all-time data or data from the current day. Furthermore, the dashboard should provide insights into delays through visualizations such as delays occurring at different times of the day, on different days of the week, and in various weather conditions. It should also display the distribution of weather conditions across Dublin city. These visualizations should be user-friendly, allowing users to filter the data based on parameters such as all routes, specific routes, all-time data, current day data, or selecting a particular date.

When it comes to development of the dashboard, simplicity and efficiency are paramount. The dashboard should operate seamlessly without any downtime and deliver data with all possible haste. To achieve these objectives, the methodology involves using Python Dash for dashboard development. Dash is the original low-code framework for rapidly building data apps in Python (‘Dash Python User Guide’, 2024). It's designed to make it easy to create complex, web-based data visualizations and interactive dashboards using Python, HTML, and CSS. Dash allows users to create interactive, web-based data visualizations and applications entirely in Python, without needing to know JavaScript or HTML. Dash can create interactive dashboards that update in real-time, respond to user input, and can be deployed as standalone web applications or integrated into larger web applications. It's particularly popular in data science and analytics circles for creating interactive data visualizations and dashboards that can be shared and accessed through web browsers.

1. **DESIGN**

The solution design prioritizes efficiency, reliability, and robustness. The main concept is to smoothly transfer data from the sources to the system, ensuring it is in the required format, and retrieve the data efficiently.

Figure 3 Design Diagram

A diagram of a computer system

Description automatically generated

The design diagram provides a clear visualization of the proposed solution. It outlines the seamless flow of data from the source systems to the destination, ensuring delivery in the desired format. Key components of the design include data sources, NTA Static loader, periodic extract engine, format converter, master database, and the dashboard engine.

* 1. **DATA SOURCES**

The data sources consist of NTA services and weather services. NTA services encompass both GTFS and GTFS-R data. GTFS represents static data, constituting a one-time dump containing text files storing trip, route, and stop details. These files are read by the NTA static loader and deposited into the database. To tackle the static data refresh issue, the NTA static loader continuously monitors static files for any alterations. Upon detecting changes, it updates the static data in the master database. The GTFS-R data is extracted by the Periodic Extract Engine at regular intervals and forwarded to the format converter for additional processing. GTFS-R contains delay information for all active trips at the current moment. This data, typically in JSON format, can be pulled using Python scripts at regular intervals. Just as with GTFS-R data, weather data can also be retrieved using Python scripts at the same intervals as GTFS-R data extraction. This weather data encompasses various parameters such as temperature, relative humidity, apparent temperature, day or night conditions, precipitation, rain, showers, snowfall, weather codes, cloud cover, pressure, surface pressure, wind speed, wind direction, and wind gusts.

* 1. **NTA Static loader**

The NTA static loader module is pivotal for incorporating GTFS static data dumps into the master database. It retrieves the static files from the NTA portal, processing each file sequentially, and then inserts the data into the master database. These files encompass essential information such as agency details, calendar schedules, route shapes, and stop times. To address the static data refresh problem, which arises from changes made to the static data by NTA, a solution is implemented. This involves maintaining a separate mapping between route names and route IDs. Upon detecting changes in the static data, the NTA static loader module updates the master database, accordingly, ensuring synchronization between the data sources and the database.

* 1. **Periodic Extract Engine**

The Periodic Extract Engine plays a crucial role in retrieving delay information and weather data at regular intervals. The Python code for the Periodic Extract Engine is designed to request Trip Updates services from NTA for GTFS-R data. Upon receiving the response, it processes the data with the help of the delay calculator module, ensuring efficient processing. The delay information for each route is then calculated and forwarded for further processing, ultimately being stored in the master database. Simultaneously, weather data is fetched from Open-Meteo services. The extracted weather parameters are processed and directly sent to the database. Ensure that both datasets are synchronized with the same timestamp to maintain data consistency throughout the process. The Extract engine's configuration allows it to execute at specified intervals, typically set to 30 minutes for optimal performance considering the data load and available computational resources. However, this interval parameter can be adjusted for more precise data or if higher computational power becomes available.

* 1. **Format Converter**

The format converter module plays a crucial role in ensuring compatibility between source and destination data types. While some data types from the GTFS-R format may differ from those in the master database, the format converter ensures that they are transformed into the appropriate format for insertion. Upon examination, it was noted that only timestamp fields require conversion, while other data types remain unchanged. Timestamps from both GTFS-R and weather services are in Unix time format. The Unix timestamp is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970, at UTC. Therefore, the Unix time stamp is merely the number of seconds between a particular date and the Unix Epoch (‘What is the unix time stamp?’, 2014). Therefore, the format converter handles the conversion of these timestamps to ensure consistency in the data. To achieve this, the code can be implemented to convert the Unix timestamp to a local timestamp compatible with the SQL database. This conversion involves localizing the timestamp based on the time zone of Dublin, ensuring that the entry time reflects the local time in Dublin without conflicting with the time zone of the database. Additionally, this approach resolves issues related to different times due to daylight saving changes.

* 1. **Database**

As per the design, the database should be a relational database. Relational databases are preferred in this scenario because they offer structured storage of data, with each piece of information associated with specific tables. This ensures data integrity throughout the data. This structure allows each entry to be treated as individual objects, facilitating efficient retrieval of data using SQL queries when required. The GTFS schedule data can be stored as it is without making many changes. The format for each static table is as follows:

* Agency

A screenshot of a computer

Description automatically generated

The agency table contains various agencies operating in association with the National Transport Authority. Dublin Bus is one of these agencies and is assigned a unique agency\_id along with other agency details. In this table, the agency\_id serves as the primary key.

* Calendar

A screenshot of a calendar

Description automatically generated

The calendar table contains a unique service\_id, serving as the primary key, along with columns representing various days of the week. A value of 1 indicates that the service operates on a particular day, while 0 indicates otherwise. Each entry is associated with a start date and end date for the service.

* Calendar\_dates

A screenshot of a computer

Description automatically generated

The calendar\_dates table explicitly enables or disables service by date. The service\_id serves as the primary key, and the exception\_type column specifies various types of exceptions for the date.

* Feed\_info

A screenshot of a computer

Description automatically generated

The feed\_info table contains publisher details, which is National Transport Authority.

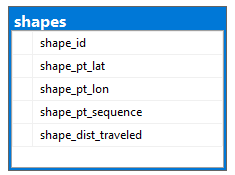
* Routes

**A screenshot of a computer

Description automatically generated**

The routes table contains specific details about bus routes operated by the agencies. It includes the route\_id, which serves as a unique identifier and is the primary key. Additionally, this table includes information such as route\_short\_name, route\_long\_name, and other route-specific details.

* Shapes



Shapes represent the trajectory that a vehicle follows along a route alignment and are outlined in the shapes table. These shapes are linked with trips and comprise a series of points that the vehicle traverses sequentially. While shapes don't necessarily intersect with the exact locations of stops, all stops on a trip should be relatively close to the shape's trajectory, often falling along straight line segments connecting the shape points (‘GTFS: Making Public Transit Data Universally Accessible’, no date). The columns include shape latitude, longitude, shape sequence, and the distance traveled along that particular shape.

* Stop\_times

A screenshot of a computer

Description automatically generated

The stop\_times table contains the arrival time and departure time of each trip at specific stop\_ids. In this table, trip\_id and stop\_id form the composite primary keys.

* Stops

A screenshot of a computer

Description automatically generated

The stops table contains stop details such as stop code, stop name, stop location, etc. Each stop is identified by a unique stop ID, which serves as the primary key of the table.

* Trips

A screenshot of a computer

Description automatically generated

The trips table stores details of individual trips or services provided by the agency. Each trip has a unique trip ID, and the service ID and route ID indicate the route the trip is operating on. Both route ID and service ID are foreign keys in the table. Additionally, columns such as trip head sign, trip short name, direction ID (indicating the direction the trip is heading), block ID, and shape ID are included.

* Route\_mapping

In addition to the 9 static tables, the design includes another table in the static database called route\_mapping. This table is responsible for storing the mapping between route short names and route IDs. The static data may be updated by transport operators when they add or modify services, leading to the static data refresh problem. To address this problem, a route mapping table was created to store the latest mapping along with previous mappings of bus routes and route IDs. The current mapping is indicated by 1, while previous mappings are indicated by 0. This route\_mapping table only includes route IDs operated by the agency Dublin Bus, as Dublin Bus is the sole focus of the research.

A screenshot of a computer

Description automatically generated

The table consists of route\_id and route\_short\_name, which together form the composite primary key. The column is\_active indicates whether the route\_id is active or not.

In addition to the static tables, the design incorporates four operational or dynamic tables: delays, route\_delays, weather, and weather\_codes. When the extract engine is executed, data from various sources is pushed into these operational tables.

* Delays

A screenshot of a computer program

Description automatically generated

The delays table is responsible for storing the average delay generated by all active buses in the service. The delay calculator module calculates the delay value, which is then entered into this table. Each Delay entry is identified by a unique entry\_id, generated by an SQL sequence, and is used across all operational tables. Entries in the delays table include the entry\_id, delay values, and the current timestamp.

* Route\_delays

A screenshot of a computer program

Description automatically generated

The route\_delays table is designed to store delay values generated by buses from specific bus routes. In addition to the columns used in the delays table, the route\_delays table includes route\_id and direction\_id within the route. The composite keys in this table consist of entry\_id, route\_id, direction\_id, and entry\_timestamp.

* Weather

The weather table is designed to store the values of weather parameters that obtained from the weather services. It also has the unique entry\_id used across the operational tables, geo location indicating where the weather is calculated, and entry\_timestamp, along with all the various weather parameters. Although some of the weather parameters might not be relevant in the dashboard, the design includes all weather variables that can be fetched in real-time.

A screenshot of a computer

Description automatically generated

* Weather\_codes

A screenshot of a computer code

Description automatically generated

To facilitate the retrieval of weather conditions, a weather codes table was designed to store the information of weather codes. This table contains a fixed number of rows corresponding to the WMO weather codes. Each code is accompanied by a description referenced in the methodology. Additionally, a SQL Trigger function was created to update the weather\_codes table when every time a new entry is made in the weather table. The code\_count column in the weather\_codes table is incremented by one whenever a weather entry with a particular weather code is recorded. This allows for easy understanding of which weather conditions occur more frequently across locations based on the weather codes.

* 1. **Dashboard Engine**

The dashboard engine is designed to be highly available and effective in always displaying the data. Data must flow seamlessly from the database to the client without any intermediary layers. To achieve this streamlined approach, a Python framework was chosen for creating the dashboard. Dash, a low-code Python framework, is selected for its ability to create interactive dashboards with excellent graph plotting capabilities. Python Dash offers numerous advantages for creating interactive simple dashboards. It provides a highly flexible and easy-to-use framework, allowing to create visually appealing and dynamic web applications with minimal code. Dash integrates seamlessly with Plotly, enabling the creation of rich, interactive graphs and charts. Additionally, it supports a wide range of components and customizations, making it suitable for diverse data visualization needs. Its capability to handle real-time data updates made the technology selection easier.

Figure 4 Dashboard Engine architecture.

A diagram of a dash app

Description automatically generated

The architecture of the dashboard engine is straightforward and similar to most Dash apps. When a user makes a request, the front-end Dash app calls the data fetching module to retrieve data from the backend, which is the SQL master database. The data fetching module uses Pyodbc APIs to communicate with the SQL database. The dashboard is designed to be user-friendly, accepting user inputs to display or filter data. This is achieved through Dash callbacks. Callback functions are automatically called by Dash whenever an input component's property changes, in order to update some property in another component (the output) (‘Dash Python User Guide’, 2024). The data visualizations, created using basic Dash components, help users gain insights into the delay and weather data. The design includes features to display average delay, current delay, average weather conditions, and current weather conditions across Dublin buses or specific routes of Dublin buses.

Apart from this, the architecture includes securing the endpoints using ngrok secure endpoints. Ngrok provides secure tunnels to localhost, which allows to expose the dashboard to the internet securely. This ensures that the data and the dashboard are protected against unauthorized access and potential threats, providing an additional layer of security. Also, according to the dashboard design, the backend services, including the periodic extract engine, are deployed on an Azure virtual machine, while the front-end dashboard engine is deployed on AWS. The idea is to keep both front-end and back-end services separate for better performance. There is no special requirement to use Azure or AWS for a particular service; this separation is intended to optimize the performance and scalability of the system.

The design layout of the dashboard prioritizes simplicity and ease of use for all users. The web page is designed to be a straightforward dashboard with basic functionalities, focusing more on functionality than aesthetics. To achieve this, the design includes simple user input fields such as dropdown buttons and radio buttons. Users should be able to select the bus route to retrieve data for the respective route and filter the data based on the current day or all-time. Visualizations are an integral part of the dashboard and include various plots such as bar charts, line charts, and pie charts. Additionally, other information can be displayed using tables and gauge charts. These plots and components can be easily created using Dash and Dash Core Components. For basic styling of Dash components, Dash Bootstrap Components can be utilized.

1. **IMPLEMENTATION**

The motive behind the development and implementation of this project was to complete it within the given timeframe while ensuring that the implementation is both efficient and robust. The focus was on delivering a functional and reliable system that meets the project requirements without compromising on performance or stability.

* 1. **Technical Requirements**
     1. **NTA Services**

The NTA services require a subscription token from the developer portal of the National Transport Authority (NTA) to access the data. The fair usage policy includes a limit of 5000 requests per day.

* + 1. **Open-Meteo services**

Open-Meteo is the data source for weather information. It does not require a subscription key. Open-Meteo is an open-source weather API and offers free access for non-commercial use.

* + 1. **Database**

The database requirement specifies the use of a SQL database. Initially, development was conducted using a MSSQL Server installed on localhost and connected via SQL Server Management Studio 19. For deployment, MSSQL Server 2022 running on an Ubuntu virtual machine was chosen, and it was connected from the client using SQL Server Management Studio 19. The SQL scripts for creating the database and tables are provided in the Appendix.

* + 1. **Python libraries**

The list of the Python libraries required for the development of both the NTA static loader and the periodic extraction engine, are given below:

* pyodbc: For connecting to and interacting with SQL databases.
* requests: For making HTTP requests to fetch data from the NTA services and Open-Meteo API.
* json: For parsing and working with JSON data.
* csv: For reading and writing CSV files.
* zipfile: For extracting files from zip archives.
* time: For time-related functions.
* datetime: For manipulating date and time objects.
* pytz: For handling timezones and converting between different timezones.

The list of the required Python libraries apart from the pyodbc libraries for the Dash app are given below:

* dash: Core Dash library for creating the Dash app.
* dash\_html\_components: For creating HTML elements in the Dash app.
* dash\_core\_components: For creating interactive components like graphs, dropdowns, etc., in the Dash app.
* dash\_table: For creating tables within the Dash app.
* dash.dependencies: For creating callbacks in the Dash app.
* plotly.express: For creating simple and intuitive plots.
* pandas: For data manipulation and analysis.
* plotly.graph\_objects: For creating more complex and customizable plots.
* plotly.subplots: For creating subplot layouts in plots.
* dash\_bootstrap\_components: For styling the Dash app with Bootstrap components.
  + 1. **Infrastructure Requirements**

The system prerequisites encompass the virtual machines necessary for deploying the database, extraction engine, and Dash application. Azure virtual machines are selected as the preferred cloud platform for installing the SQL database and deploying the periodic extraction engine. The Azure virtual machine is configured with the following specifications: Standard B2s (2 vCPUs, 4 GiB memory), running the Ubuntu 22.04 operating system, deployed in the UK South region (Zone 1).

The Dash application is deployed on an AWS EC2 instance, running Canonical Ubuntu 22.04 LTS (amd64) server type t2.micro, with 8 GiB of storage. The deployment is in the eu-west-1 region. This EC2 instance connects to the database installed on Azure. The rationale for utilizing two separate virtual platforms is to leverage additional computational power without incurring extra costs. A port forwarding program, ngrok, is configured on the EC2 machine to deliver the application through a secure port. The infrastructure requirements include an authToken generated from the ngrok website, which is necessary to configure ngrok on the EC2.

* 1. **TOOLS USED**

The development process involved the use of several tools for analysis, coding, and testing:

* Postman: Utilized for testing and analyzing various web services such as GTFS-R and Open-Meteo.
* SQL Server Management Studio 19: Employed for creating and accessing tables from the master database.
* Visual Studio Code: Used as the primary code editor.
* Git Desktop: Facilitated version control and pushing of code to GitHub.
* Notepad++: Utilized for initial analysis of various file types, including CSV and SQL files.
  1. **Development Process**

The development process followed a straightforward and systematic approach, progressing through each module in a linear fashion. It began with the creation of the NTA static loader module, responsible for uploading NTA static files into the master database and establishing route mappings with route IDs and route short names. Subsequently, the extraction engine was developed, followed by the format converter. The extraction engine encompassed the retrieval of delay values from real-time GTFS-R data and the calculation of delay values using the delay calculator module. Additionally, weather parameters were fetched and stored in the master database. The Dash app was then created, starting with the creation of an initial Dash app, and subsequently adding plots to the application. Dash callbacks were implemented to enable user interactions on the dashboard.

* 1. **Version Handling**

Throughout the process, version control and code backup were maintained using Git. Git allows you to track changes made to your codebase over time. Git helps to easily undo changes or revert to previous versions of the code. This capability provides a safety net, allowing to experiment freely without fear of irreversible consequences. If a mistake is made, it's usually straightforward to roll back to a well-known state. The GitHub Desktop tool was utilized for committing and pushing code changes to the repository. The Git repository link can be found in the appendix.

* 1. **Deployment**

Deploying the code to a cloud-based system was deemed necessary due to the requirement for periodic execution of the extraction engine in every 30 minutes to fetch real-time weather and delay data. Additionally, the database needed to be deployed on the cloud, which was accomplished by setting up a SQL server database on an Azure virtual machine. It was crucial to expedite this deployment process to ensure that the dashboard could utilize the data available post-deployment of the database and extraction engine.

The extraction engine automatically triggers the NTA static loader module under two conditions: firstly, if it's the initial execution of the extraction engine, and secondly, if there's a static data refresh problem happened. The extraction engine is scheduled to run using cronjobs. The cron command-line utility is a job scheduler on Unix-like operating systems. Cronjobs will help to schedule jobs (commands or shell scripts), to run periodically at fixed times, dates, or intervals (‘Cron’, https://en.wikipedia.org/wiki/Cron).

The Dashboard application is deployed on an AWS EC2 instance by installing the necessary prerequisites and cloning the code from GitHub. The procedure involved enabling network traffic on both HTTP and HTTPS ports of the network interface of EC2 instance. To ensure the application runs on a secure port, a port forwarding program is set up. ngrok is a globally distributed reverse proxy that secures, protects, and accelerates the applications and network services, no matter where the application is running. ngrok is a unified ingress platform because it combines all the components to deliver traffic from the services to the internet into one (‘ngrok’, 2024). To configure the application port to a secured port, the pregenerated authToken must be configured on the EC2 instance. Then, the secured tunnel can be started by providing the pregenerated domain name and the application port, ensuring that the application runs securely in the given domain name.

* 1. **Testing**

Testing is a critical phase in the implementation of the project. Each module is thoroughly tested end-to-end after development to ensure proper functionality as per the design. Only after a module is tested and verified as working correctly does the development of the next module begin. The initial testing of the NTA Static Loader was successful. However, during the soak testing of the Periodic Extraction Engine, a static data refresh problem was identified. This problem arose when changes occurred in the NTA static dataset, typically due to route changes by the operator. To address this issue, an additional table, `route\_mapping`, was added to the database. The `route\_mapping` table stores the mappings between route short names and route IDs, ensuring the system can handle updates in the static data seamlessly.

Another significant bug identified during the testing phase was a time zone error. This issue emerged after the deployment of the Extraction Engine and Dashboard, where the system was using the local time instead of Dublin time. Additionally, during testing, modifications were made to the format converter to enhance its precision.

From the user's perspective, the dashboard underwent comprehensive testing with numerous test cases. The rigorous testing during development itself was advantageous, allowing most issues to be addressed simultaneously. Some UI glitches and data format issues were identified and fixed through this thorough testing of the UI dashboard.

1. **RESULTS**

After the development and deployment process, successfully created a dashboard that encompasses various data and plots. These visualizations provide numerous insights and details, allowing users to estimate the correlation between weather parameters and delays directly from the dashboard. Users will be able to understand the delays occurring in Dublin bus services under various weather conditions and times and can analyze this information based on specific routes as well.

* 1. **Dashboard**

On the initial load, the dashboard displays a line chart showing the current day's delays, plotted using readings taken every 30 minutes by the extraction engine. Alongside this, it includes line plots of delay values along with weather parameters such as temperature and wind speed.

Figure 5 Dashboard screenshot with line graph.

A screenshot of a computer

Description automatically generated

The dashboard refreshes automatically every 5 minutes to fetch the latest data. The dashboard is made user-friendly, allowing users to select specific bus routes and route directions. Users can also choose to display either the current day's values or all-time data on the dashboard.

Figure 6 User operations on dashboard.

A screenshot of a computer

Description automatically generated

Another important feature of the dashboard is the addition of gauge charts. These charts display the average delay across routes and the current delay across routes, providing a clear and immediate visualization of the bus service performance.

Figure 7 Gauge charts.

A close-up of a diagram

Description automatically generated

The gauge charts provide an immediate understanding of how the current delay values range and their status at the current time. The gauge displays the delay range, with 0 on the left side and the maximum delay value on the right end. The current delay value is represented by the gauge needle, plotted in green. The delay value is displayed in seconds. Below the actual value, previous values are shown with color-coded indications to highlight whether the current value is greater than or lower than the previous value. If this value is displayed in green, it indicates that the current delay value is lower than the previous value. Conversely, if it is displayed in red, it signifies that the current delay value is higher than the previous delay value. This provides users with an understanding of the delay range across routes and how the current delay compares to previous values. These plots for a specific route can be generated when the user selects a route from the dropdown menu.

Figure 8 Plots for a specfic bus route.

A screenshot of a computer

Description automatically generated

When the user selects a specific route, they can also choose the direction within that route. The direction selection dropdown presents three options: "both-sides" and two different directions. If a particular direction is chosen, only the values from that direction will be fetched from the backend. Otherwise, if "both-sides" is selected, values from both directions are included. The diagrams are then updated accordingly based on the user's selection of route and direction.

Figure 9 Top delayed routes.

A screenshot of a route

Description automatically generated

The dashboard presents the top delayed routes in a Dash table, ordered by delays. The routes with the highest delays appear at the top, descending downwards. If the user selects the "Today" button, the table displays the top delayed routes from the current day. Otherwise, if the user chooses another option, the top delayed routes from the entire dataset are shown.

Figure 10 Bar graph plotting delay with weather conditions

A blue squares with white text

Description automatically generated

The bar graph illustrates delay values corresponding to weather conditions. The weather condition with the highest delay is highlighted in red. According to the available data the most delays weather condition is ‘slight rain showers’.

A graph of a graph

Description automatically generated with medium confidence

The bar graph above illustrates delay values based on days of the week. The day with the highest delay is highlighted in red. According to the available data, Saturday has the highest delay.

Figure 11 Pie chart of Dublin weather.

A pie chart with numbers and text

Description automatically generated

In addition to these plots, for gaining insights into Dublin's weather, a pie chart is plotted to display the distribution of weather data. At the very bottom of the dashboard, the last updated time of the dashboard is displayed. The dashboard will automatically update every 5 minutes.

* 1. **Findings**

Project work tracking – Trello, github, sprints

CHALLENGES FACED – static data refresh problem and time zone bug

ETHICAL CONSIDERATION

CONCLUSION

-limitations future study

Appendix –

Git link, Table queries, NTA links, Open-meteo links, deployment link, Trello link