

# **Urban Bike Real-Time Dashboard Application Documentation**

**Release 1.0**

July 01 , 2024

## **Authors**

Sangeetha Ramesh

Pradyumna Shivanandaiah

## **Contact Information:**

Github

## **Document Purpose**

This document provides comprehensive information about the Urban Bike Real-Time Dashboard Application, including user guides, deployment instructions, and technical details.

## Table of Contents

Project Overview .....	3
Objectives.....	3
Scope .....	3
In-Scope.....	3
Out-of-Scope.....	3
Requirements .....	3
Functional Requirements.....	3
Non-Functional Requirements .....	4
Deliverables .....	5
Technical Details.....	6
Architecture .....	6
1. Frontend .....	6
2. Backend.....	7
3. Data Extraction and Transformation .....	7
4. Data Visualization .....	7
5. Deployment .....	7
6. External Services.....	7
Interaction Flow .....	8
User Authentication Flow .....	8
Technology Stack .....	9
Deployment Guide .....	10
Prerequisites .....	10
Installation Steps .....	10
Environment Configuration.....	10
Deployment Steps .....	11
Docker Deployment on Coolify Self-Host Server with Oracle VM .....	11
Running Locally without Docker .....	11
Maintaining the Deployment .....	12
Troubleshooting Deployment Issues .....	12
Detailed Project Description .....	13
Appendix .....	17
Glossary .....	17
References.....	17

---

# Project Overview

The Dublin Bike Real-Time Dashboard Application is a Flask-based web application designed to provide real-time visualizations of Dublin bike data. It integrates a Power BI dashboard that is updated daily with data pulled from the Power BI Web. The application includes a data table for managing station data, a contact form, and information about the project.

## Objectives

- To provide a real-time visualization of Dublin bike data.
- To allow authorized users to manage station data through a web interface.
- To offer a contact platform for users to reach the developers.
- To present project details, including its vision, mission, and goals.

## Scope

### In-Scope

- **Dashboard:** Displays a Power BI dashboard with data updated daily from the Power BI Web.
- **Data Table:** A table displaying data from the SQL database with options to add, edit, and update station information.
- **Contact Us:** A form for users to send messages to the developers and contact information.
- **About Us:** Information about the project's vision, mission, goals, and details.

### Out-of-Scope

- Advanced user authentication mechanisms (beyond hardcoded user authentication).
- Integration with other external APIs or data sources beyond the provided Dublin bike API.

# Requirements

## Functional Requirements

### 1. User Authentication

- **Description:** Implement a basic hardcoded user authentication system to restrict access to the data table management feature.
- **Details:**
  - Only authorized users should be able to access and make changes to the data table.
  - The authentication system will include a login form where users can enter their credentials.
  - User credentials will be hardcoded in the application for simplicity.

- Upon successful login, users will be redirected to the dashboard or data table management page.
  - Unauthenticated users attempting to access restricted areas will be redirected to the login page.
2. **Data Refresh**
- **Description:** Ensure that the data displayed in the Power BI dashboard is automatically refreshed daily from the Power BI Web.
  - **Details:**
    - Set up a daily auto-refresh for the Power BI dashboard.
    - Ensure that the dashboard fetches the latest data from the Power BI Web during each refresh.
3. **CRUD Operations**
- **Description:** Provide a web interface for authorized users to create, read, update, and delete (CRUD) station data stored in the SQL database.
  - **Details:**
    - **Create:** Users can add new station records through a form. The form will include fields for all necessary station details.
    - **Read:** Display the existing station data in a tabular format on the web interface.
    - **Update:** Users can edit existing station records. An edit form will be pre-filled with the current data for convenience.
    - **Delete:** Users can delete station records. Implement a confirmation prompt to prevent accidental deletions.
    - All CRUD operations will be restricted to authenticated users only.
    - Changes made through the web interface will be immediately reflected in the SQL database.
4. **Contact Form**
- **Description:** Provide a contact form for users to send messages to the developers.
  - **Details:**
    - The form will include fields for the user's name, email address, subject, and message.
    - Upon submission, the form data will be sent to a specified email address or stored in a database for the developers to review.
    - Implement form validation to ensure all fields are filled out correctly.
    - Display a confirmation message to the user upon successful submission of the form.

## Non-Functional Requirements

### Performance

- **Description:** Ensure the application performs efficiently under expected user loads.
  - **Details:**
    - Optimize the ETL process to minimize the time taken for data extraction, transformation, and loading.
    - Ensure the Power BI dashboard loads and updates quickly with the latest data.
    - The web interface should respond promptly to user actions, including CRUD operations and form submissions.
-

- Use indexing and query optimization techniques in the SQL database to speed up data retrieval.

## Security

- **Description:** Implement basic security measures to protect the application and user data.
- **Details:**
  - Use HTTPS to encrypt data transmitted between the client and server.
  - Store user credentials securely and avoid hardcoding them in the codebase (consider using environment variables).
  - Implement input validation to prevent SQL injection and other common security vulnerabilities.
  - Ensure that only authenticated users can access and modify sensitive data.

## Usability

- **Description:** Design the application to be user-friendly and easy to navigate.
- **Details:**
  - Create a clean and intuitive user interface with clear navigation links to different sections of the application.
  - Ensure forms are easy to fill out and provide helpful error messages for validation issues.
  - Make the data table sortable and searchable to help users find specific records quickly.
  - Provide clear instructions and feedback for user actions, such as form submissions and CRUD operations.

## Scalability

- **Description:** Design the application to handle increased data volume and user load without significant performance degradation.
- **Details:**
  - Use a scalable infrastructure (e.g., AWS) to deploy the application, allowing for easy scaling of resources as needed.
  - Implement caching mechanisms to reduce the load on the SQL server for frequently accessed data.
  - Design the database schema to handle large volumes of data efficiently.
  - Ensure the ELT process can be scaled to handle increased data extraction and transformation needs.

## Deliverables

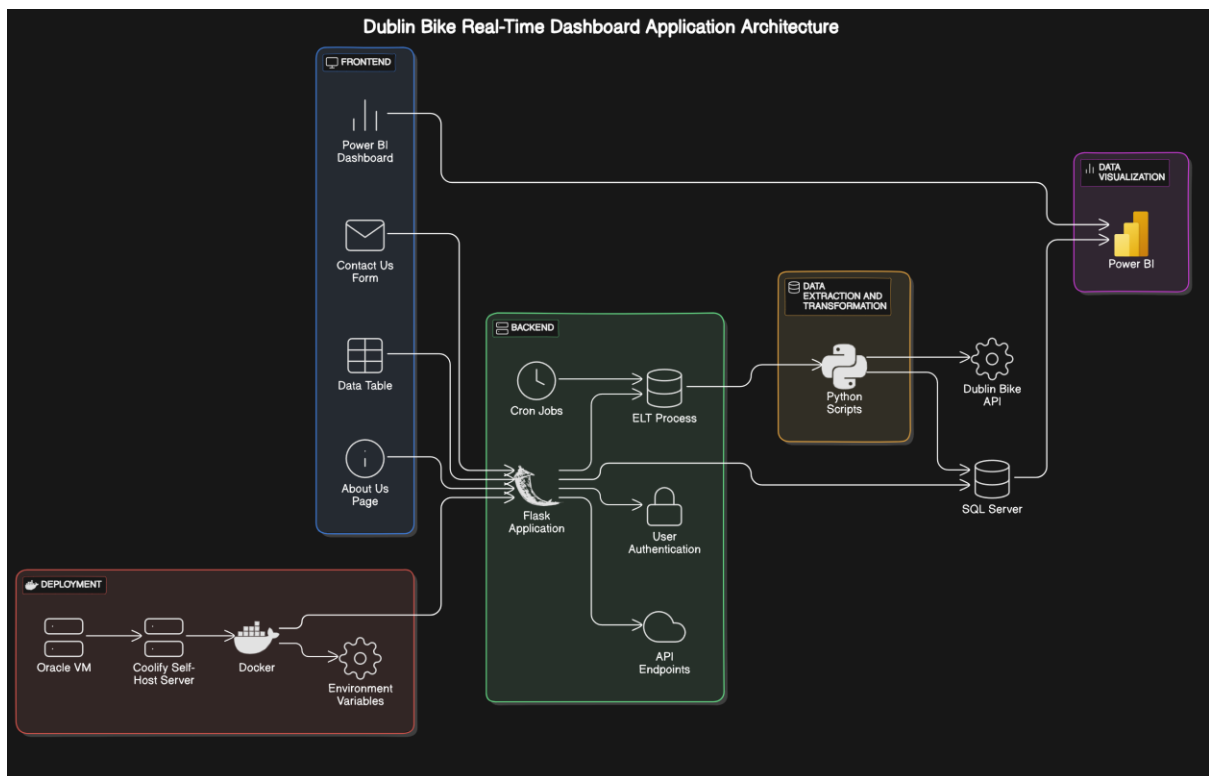
- **Code Repository:** [Link to the repository]
  - **User Manual:** Instructions for users on how to use the application.
  - **Technical Documentation:** Detailed technical documentation for developers.
  - **Deployment Guide:** Steps for deploying the application in different environments.
-

# Technical Details

## Architecture

- **Overview:** The application is built using Flask for the backend, with data manipulation and ETL processes handled by Python scripts. The front end is developed using HTML, CSS, and AJAX.
- **Components:**
  - **Backend:** Flask application.
  - **Frontend:** HTML, CSS, AJAX.
  - **Database:** SQL Server for storing and managing data.
  - **API Integration:** Data pulled from the Dublin bike API.

Architecture Diagram:



The diagram provides a comprehensive view of the components and interactions within the Dublin Bike Real-Time Dashboard Application. The architecture is divided into four main sections: Frontend, Backend, Data Extraction and Transformation, Data Visualization, and Deployment.

### 1. Frontend

- **Power BI Dashboard:** This component is responsible for displaying real-time visualizations. It directly fetches data from Power BI Web with a daily auto-refresh setup.
- **Contact Us Form:** Provides users a way to send messages to the developers.

- **Data Table:** Displays the data pulled from the SQL database and allows for CRUD (Create, Read, Update, Delete) operations.
- **About Us Page:** Contains information about the project, including its vision, mission, and goals.

## 2. Backend

- **Flask Application:** Acts as the core backend server, handling requests from the frontend, including routing and API endpoints.
  - **User Authentication:** Implements basic hardcoded user authentication to restrict access to certain features.
  - **API Endpoints:** Handles various API requests from the frontend for different functionalities like data table management.
  - **Cron Jobs:** Manages scheduled tasks, specifically running the ELT process daily to ensure data is up-to-date.
  - **ELT Process:** Includes scripts that extract data from the Dublin Bike API, transform it to fit the required schema, and load it into the SQL Server database.

## 3. Data Extraction and Transformation

- **Python Scripts:** These scripts are responsible for the ELT (Extract, Load, Transform) process.
  - **Extract:** Pulls data from the Dublin Bike API.
  - **Transform:** Cleans and structures the data to match the application's schema.
  - **Load:** Inserts the transformed data into the SQL Server database.

## 4. Data Visualization

- **Power BI:** Embedded in the frontend to provide real-time data visualization by fetching data directly from Power BI Web.

## 5. Deployment

- **Oracle VM:** Virtual machine used for hosting the Dockerized application.
- **Coolify Self-Host Server:** Platform for deploying Docker containers.
- **Docker:** Containerizes the Flask application for consistent deployment across different environments.
- **Environment Variables:** Configures SQL server connection details and other necessary settings for the application.

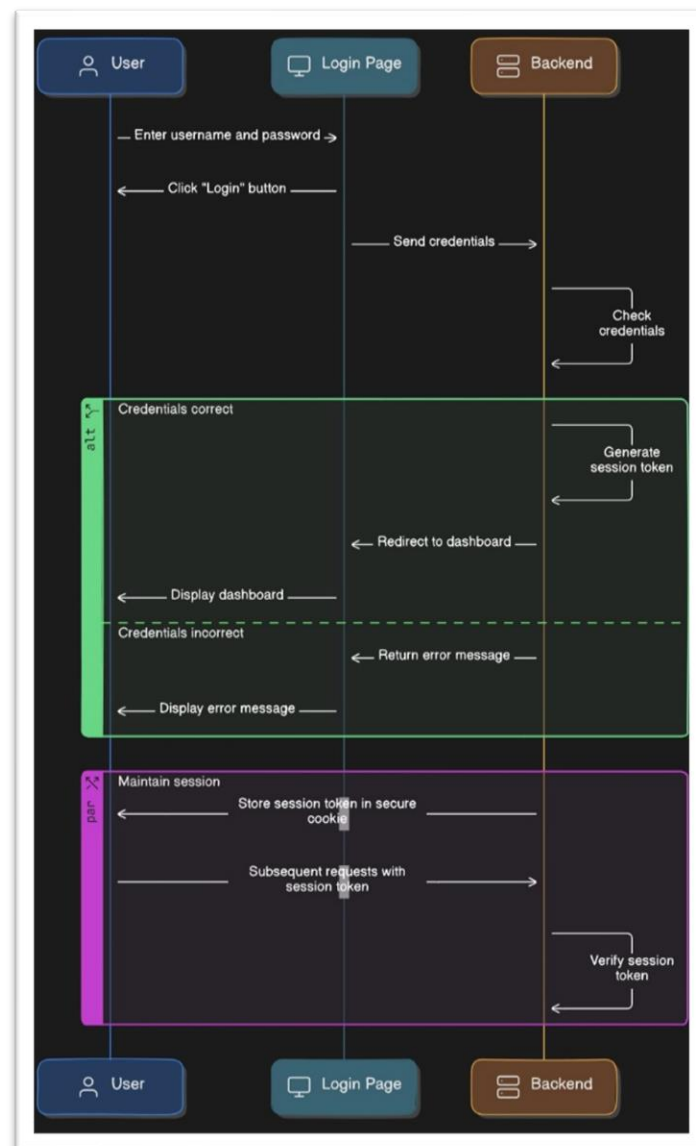
## 6. External Services

- **Dublin Bike API:** The external API source that provides real-time bike data, which is essential for the ELT process.
-

## Interaction Flow

1. **Data Extraction and Transformation:** Python scripts extract data from the Dublin Bike API, transform it, and load it into the SQL Server database.
2. **Backend Processing:** The Flask application handles various backend processes, including user authentication, API requests from the frontend, and running the ELT process via cron jobs.
3. **Data Visualization:** The Power BI dashboard in the frontend fetches real-time data directly from Power BI Web and displays it to the users.
4. **Frontend Interaction:** Users interact with the application through the frontend components such as the Power BI dashboard, data table, contact us form, and about us page.
5. **Deployment:** The application is containerized using Docker and deployed on a Coolify self-host server running on an Oracle VM. Environment variables are used to configure necessary settings.

## User Authentication Flow





1. **User Interaction:**
  - **Enter username and password:** The user enters their credentials on the login page.
  - **Click "Login" button:** The user submits the credentials by clicking the login button.
2. **Login Page:**
  - **Send credentials:** The login page sends the entered credentials to the backend for validation.
3. **Backend:**
  - **Check credentials:** The backend checks the provided credentials against stored user information.
  - **Generate session token:** If the credentials are correct, the backend generates a session token.
4. **Credential Validation:**
  - **Correct credentials:**
    - **Redirect to dashboard:** The user is redirected to the dashboard.
    - **Display dashboard:** The dashboard is displayed to the user.
  - **Incorrect credentials:**
    - **Return error message:** The backend returns an error message.
    - **Display error message:** The login page displays the error message to the user.
5. **Session Management:**
  - **Maintain session:**
    - **Store session token in secure cookie:** The session token is stored in a secure cookie on the user's browser.
    - **Subsequent requests with session token:** Future requests from the user include the session token.
    - **Verify session token:** The backend verifies the session token to maintain the user's authenticated state.

## Technology Stack

- **Backend:** Flask, Python
  - **Frontend:** HTML, CSS, AJAX
  - **Database:** SQL Server
  - **APIs:** Dublin bike API
  - **Data Manipulation:** Python scripts for ELT processes
-

# Deployment Guide

## Prerequisites

Before deploying the Dublin Bike Real-Time Dashboard Application, ensure you have the following tools installed:

1. **Docker Desktop for Windows:** Install Docker from [here](#).
2. **Git for Windows:** Install Git from [here](#).
3. **Coolify:** Set up Coolify for self-hosting Docker containers. Refer to the [Coolify documentation](#).

## Installation Steps

Follow these steps to set up the project locally:

### 1. Clone the Repository

```
1. git clone [YOUR GITHUB REPO URL]
2. cd [YOUR REPO DIRECTORY]
```

### 2. Set Up a Virtual Environment (Optional for Local Development)

```
1. python -m venv venv
2. .\venv\Scripts\Activate
```

### 3. Install Dependencies

```
1. pip install -r requirements.txt
```

## Environment Configuration

Configure the necessary environment variables for your application. Create a config file in the root directory and add the following variables:

```
1. SQL_SERVER=<your_sql_server>
2. SQL_DB=<your_db_name>
3. SQL_USER=<your_db_user>
4. SQL_PASSWORD=<your_db_password>
```

---

# Deployment Steps

## Docker Deployment on Coolify Self-Host Server with Oracle VM

### 1. Build the Docker Image

- Ensure you are in the root directory of the project where the Dockerfile is located.

```
1. docker build -t dublin-bike-dashboard .
```

### 2. Run the Docker Container

- Configure environment variables or modify configuration files to point to your SQL server.

```
1. docker run -d -p 5000:5000 --name dublin-bike-dashboard -e SQL_SERVER=<your_sql_server> -e SQL_DB=<your_db_name> -e SQL_USER=<your_db_user> -e SQL_PASSWORD=<your_db_password> dublin-bike-dashboard
```

### 3. Deploy on Coolify

- Access your Coolify dashboard.
- Create a new application and choose "Docker" as the deployment method.
- Provide the necessary Docker configuration, including the Docker image (dublin-bike-dashboard) and environment variables.

### 4. Set Up Scheduled Task for ELT Process in Coolify

- Set up a scheduled task to run the ELT script daily:
  1. Task Name: ELT Daily Update
  2. Schedule: Daily at midnight (or any preferred time)
  3. Command:

```
1. python /path/to/your/ELT.py
```

## Running Locally without Docker

If you prefer to run the application locally without Docker, follow these steps:

### 1. Clone the Repository

```
1. git clone [YOUR GITHUB REPO URL]
2. cd [YOUR REPO DIRECTORY]
```

### 2. Set Up a Virtual Environment

```
1. python -m venv venv
2. .\venv\Scripts\Activate
```

### 3. Install Dependencies

```
1. pip install -r requirements.txt
```

---

#### 4. Set Up Environment Variables

- Create a config file in the root directory and add the necessary environment variables.

```
1. SQL_SERVER=<your_sql_server>
2. SQL_DB=<your_db_name>
3. SQL_USER=<your_db_user>
4. SQL_PASSWORD=<your_db_password>
```

#### 5. Run the Application

```
1. python run.py
```

## Maintaining the Deployment

#### 1. Updating the Application

- Pull the latest changes from the repository:

```
1. git pull origin main
```

- Rebuild and restart the Docker container:

```
1. docker build -t dublin-bike-dashboard .
2. docker stop dublin-bike-dashboard
3. docker rm dublin-bike-dashboard
4. docker run -d -p 5000:5000 --name dublin-bike-dashboard -e SQL_SERVER=<your_sql_server> -e
SQL_DB=<your_db_name> -e SQL_USER=<your_db_user> -e SQL_PASSWORD=<your_db_password> dublin-bike-
dashboard
```

#### 2. Scaling the Deployment

- Use Docker Compose or a similar tool to manage multiple instances of the application if needed.
- Configure load balancing to distribute traffic among instances.

#### 3. Regular Maintenance Tasks

- Monitor application logs for errors or issues.
- Regularly back up the SQL database.
- Ensure the Task Scheduler job for the ELT process is running correctly.

## Troubleshooting Deployment Issues

#### 1. Docker Container Fails to Start

- Check the Docker logs for error messages:

```
1. docker logs dublin-bike-dashboard
```

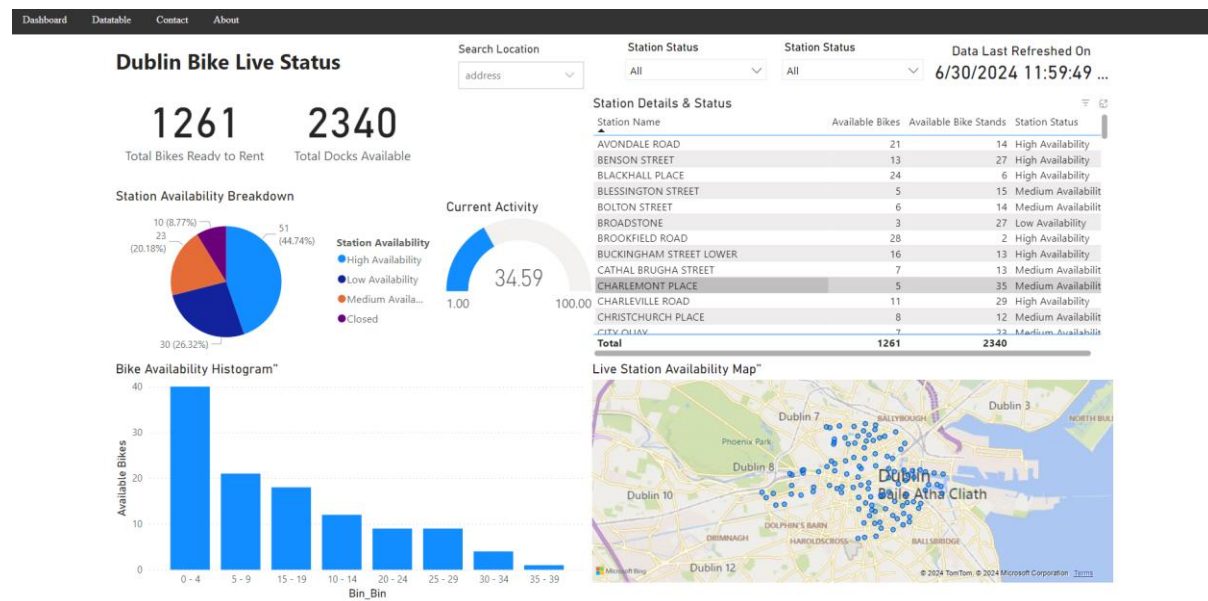
#### 2. Database Connection Issues

- Verify that the SQL server credentials and connection details are correct.
-

- Ensure the SQL server is accessible from the Docker container.
- 3. **Task Scheduler Job Not Running**
  - Check the status of the task in Task Scheduler.
  - Verify the Task Scheduler configuration and ensure the ELT script path is correct.
- 4. **Power BI Dashboard Not Updating**
  - Ensure the Power BI dashboard is set up to auto-refresh daily.
  - Verify the connection between the Power BI dashboard and the data source.

## Detailed Project Description

Dashboard:



The dashboard integrates a Power BI visualization that updates in real-time. Data is pulled daily from the Power BI Web and displayed in an easy-to-understand format.

Login Page:

Login

Username:

Password:

Login

Get Access

Return to Dashboard

The login page act as a secured layer while accessing the data table content. Current we have hardcoded the user credentials. Once user authenticates it they will be able to access the content.

Data Table

Station Data Table

Add New Station

Show 10 entries

Search:

Number	Name	Address	Position	Banking	Bonus	Status	Contract Name	Bike Stands	Available Bike Stands	Available Bikes	Last Update	Actions
108	AVONDALE ROAD	Avondale Road	Lat: 53.359405, Lng: -6.276142	No	No	OPEN	dublin	35	22	13	2024-07-01 19:51:15	<div>Edit</div> <div>Delete</div>
90	BENSON STREET	Benson Street	Lat: 53.344153, Lng: -6.233451	No	No	OPEN	dublin	40	40	0	2024-07-01 19:50:29	<div>Edit</div> <div>Delete</div>
88	BLACKHALL PLACE	Blackhall Place	Lat: 53.3488, Lng: -6.261637	No	No	OPEN	dublin	30	7	23	2024-07-01 19:56:34	<div>Edit</div> <div>Delete</div>
2	BLESSINGTON STREET	Blessington Street	Lat: 53.356769, Lng: -6.26814	No	No	OPEN	dublin	20	20	0	2024-07-01 19:56:00	<div>Edit</div> <div>Delete</div>
3	BOLTON STREET	Bolton Street	Lat: 53.351182, Lng: -6.269859	No	No	OPEN	dublin	20	7	13	2024-07-01 19:58:17	<div>Edit</div> <div>Delete</div>
116	BROADSTONE	Broadstone	Lat: 53.3547, Lng: -6.272314	No	No	OPEN	dublin	30	13	17	2024-07-01 19:54:46	<div>Edit</div> <div>Delete</div>
84	BROOKFIELD ROAD	Brookfield Road	Lat: 53.339005, Lng: -6.300217	No	No	OPEN	dublin	30	10	20	2024-07-01 19:59:22	<div>Edit</div> <div>Delete</div>

The data table presents station data pulled from the SQL server. Users with the appropriate access can add, edit, and update station data. Changes are protected by a basic hardcoded user authentication system.

## Contact Us

### Contact Us

Welcome to the UrbanWheel: Dublin Bike Availability in Real-Time Project. If you have any questions or feedback, please don't hesitate to reach out to us using the form below or via the following contact details:

#### Contact Form

**Name:**

**Email:**


**Subject:**

**Message:**

Send




#### Developer Profiles

**Developer 1**




7 years of steadily increasing expertise in the IT industry, with in-depth working knowledge of all stages of the project's software development lifecycle.

GitHub Profile






**Pradyumna**



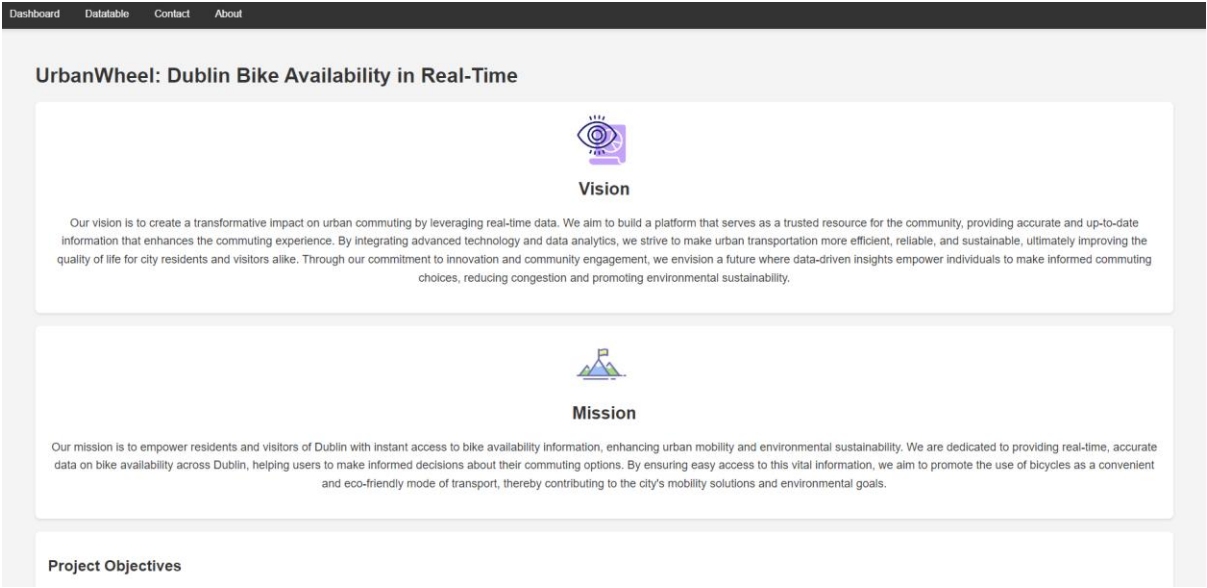
I am a data-driven Business Analyst with an MSc in Business Analysis from DBS and four years of diverse experience in product building, process automation, and business analysis across various industries.

GitHub Profile



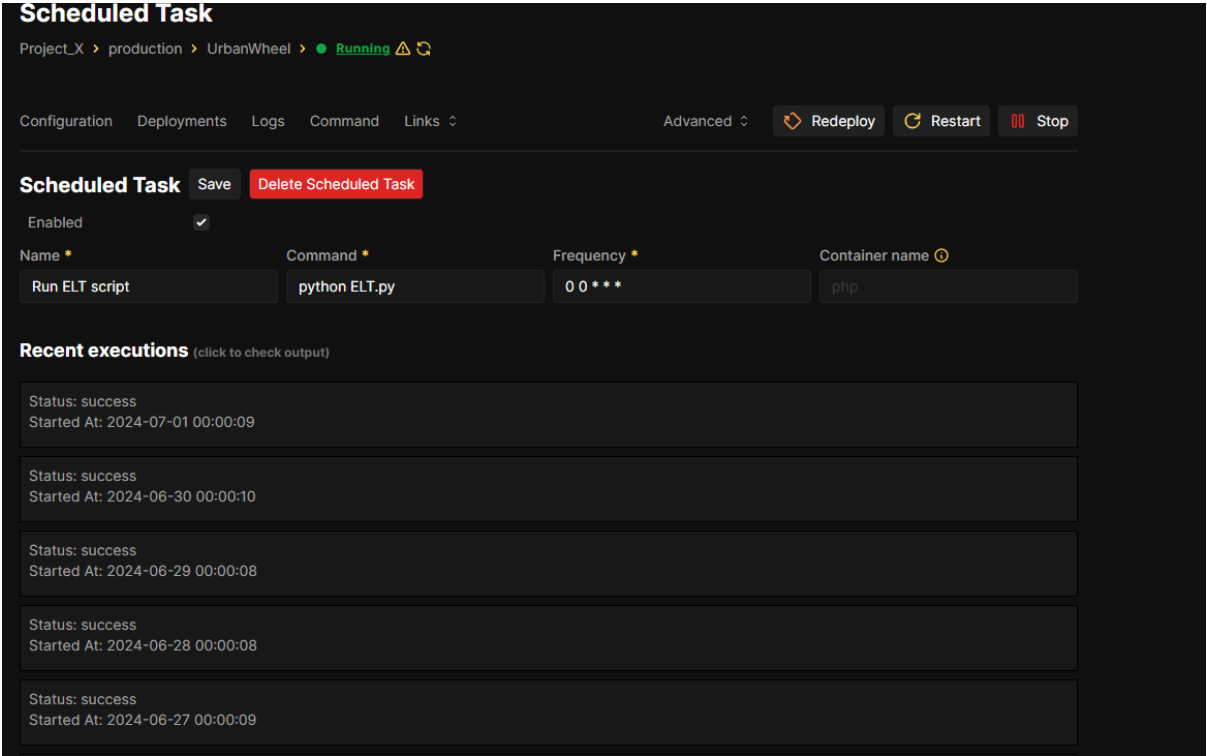
This section provides contact information for the developers and includes a form for users to send messages. The form collects user queries and feedback, which are then directed to the development team.

## About Us



The 'About Us' section outlines the project's vision, mission, and goals. It also includes background information about the project, providing context and explaining its significance.

## ELT Process



The ELT (Extract, Load, Transform) process is implemented using Python scripts. Data is extracted from the Dublin bike API, transformed to meet application requirements, and then loaded into the SQL database.



## ELT Script Details

- **Extract:** Data is pulled from the API at regular intervals using a scheduled cron job.
- **Transform:** The data is cleaned and transformed to fit the application's schema.
- **Load:** The transformed data is loaded into the SQL server for use by the application.

## Cron Job

A cron job is set up to run the ELT process daily, ensuring that the dashboard displays the most recent data.

## Code Structure

- **app:** Contains the Flask application code.
- **cronjob:** Contains scripts for setting up and running the cron job.
- **Dockerfile:** Configuration for containerizing the application.
- **ELT.py:** Python script for the ELT process.
- **requirements.txt:** Lists the dependencies required to run the application.
- **run.py:** Entry point for running the Flask application.

## Appendix

### Glossary

- **ELT:** Extract, Load, Transform
- **CRUD:** Create, Read, Update, Delete
- **API:** Application Programming Interface

## References

- [Dublin Bike API Documentation](#)
-