



MSc in Computing
Advanced Software Development | Data Science

Team Project

Final Report

Magpie
Services at a Glance

Group 3

Saul Burgess	C19349793	Andreas Kraus	D23125112
Kaustubh Trivedi	D23124940	Jessica Fornetti	D23124588
Anais Blenet	D22127697	Yuanshuo Du	D22125495

December 20, 2024

Table of Contents

Table of Contents	1
List of Figures	3
List of Tables	6
List of Listings	7
1 Introduction	8
2 User Scenario	8
2.1 Who is our target user?	8
2.2 Why are they important?	10
2.3 What problem are we solving?	10
2.4 Market analysis	10
3 Technical Problem	15
3.1 Why does our system exist?	15
3.2 The core technical problem	16
3.3 A review of similar systems	17
3.4 Conclusion	19
4 Technical Solution	20
4.1 System Architecture	20
4.2 System Overview	21
4.3 Data Sources	24
4.4 Machine Learning	28
4.5 Frontend	48
4.6 Backend	72
4.7 Conclusion Technical Solution	99
5 Software Management	100
5.1 Why is Software Management important?	100
5.2 What is DevOps?	100
5.3 Continuous Integration	101
5.4 GitHub Actions	107
5.5 Continuous Deployment	109
5.6 Devcontainers	112
6 User Evaluation	113
6.1 User experience design & testing	113
6.2 Non-Expert User evaluation	115
6.3 Domain Expert evaluation	123
6.4 UI/UX Expert Review	129
6.5 Accessibility Expert Review	138
6.6 User evaluation summary	143
7 Future Work	144
7.1 Introduction	144

7.2	Machine Learning	144
7.3	Frontend	145
7.4	Backend	146
8	Conclusion	149
8.1	Project Management Strategy	149
8.2	Challenges and Solutions	149
8.3	Key Contributions	150
8.4	Lessons Learned	150
8.5	Strengths and Weaknesses of Final System	150
8.6	Final Remarks	151
9	Appendix	152
9.1	Data Sources Appendix	152
9.2	User Evaluation Appendix	152
	Bibliography	161

List of Figures

1	User persona - Michael O'Brien	9
2	User persona - Sarah Thompson	9
3	Market analysis - User Sector Distribution	11
4	Market analysis - User County Distribution	11
5	Market analysis - User B Amenity Data Type Distribution	12
6	Market analysis - User A Preferred device & Usage frequency	12
7	Market analysis - User B Tool type & Usage frequency	13
8	Market analysis - User B Tool satisfaction & Usage frequency	13
9	Market analysis - User A Thoughts on Magpie	14
10	Market analysis - User B Thoughts on Magpie	14
11	Market analysis - User B Rating Additional Features	15
12	Example of Parkopedia	17
13	Example of ArcGIS	18
14	System Architecture Diagram	20
15	Magpie's Technology Stack	21
16	Magpie Data Stack	24
17	Dublin Area Delimitation for Parking data	26
18	Single vs Two-Stage Object Detection Process	28
19	YOLO Model training graph	29
20	Non-OBB labels (left) versus OBB labels (right)	30
21	Different Iterations of the Road Mask	31
22	Classification of cars as on the road or parked	32
23	Empty Parking Spot Detection	34
24	Classification of all spots into private, public and parking lot	36
25	Road mask classification test set images being annotated in Label Studio	39
26	Images from the Road Mask Classification Test Set	40
27	Empty detection test set - True label vs Predicted label	42
28	Empty detection test set - True label vs Predicted label	43
29	Classification of spots test set - True label vs Predicted label	44
30	Parking zones and cost defined by Dublin City Council	45
31	Frontend Architecture Diagram	48
32	Site map - magpie	54
33	Landing page - Header	55
34	Landing page - Hero section	55
35	Landing page - About section	56
36	Landing page - Features section	57
37	Landing page - Use Case section	57
38	Landing page - CTA section	58
39	The Signup Page	58
40	The Login Page	60
41	Homepage	62
42	History Page	64
43	Evolution of interface design from initial card-based layout to consolidated dashboard approach	67
44	Wireframe-home	67
45	Wireframe-login/signup	67
46	v1_Home Page	69

47	login/signup	69
48	v2_Home Page	70
49	v3_Home Page	70
50	v3_History	70
51	v3_onboarding	70
52	v4_Home Page	71
53	v4_Landing Page	71
54	Backend Architecture Diagram	72
55	Database Workflow Diagram	79
56	Visualisation of a bcrypt hash	80
57	Example of a JWT used by Magpie for authentication	81
58	Database Schema for the minimum viable Product	83
59	Final Database Schema	84
60	Routing in the Backend: from Request to Handler	85
61	Example of logs produced by the public backend server	91
62	Excerpt from the Error Code Documentation for the Backend	95
63	The environment variables available for configuring the backend	96
64	Comments created by custom GitHub action for executing backend unit tests	98
65	Excerpt from the hidden section on a coverage comment (see Figure 64), providing detailed, function level coverage	98
66	The flow of work in <i>Magpie</i>	101
67	An example of a GitHub Issue	102
68	An example of a GitHub Pull Request	104
69	An example of a GitHub Kanban Board	106
70	An example of a GitHub Roadmap	106
71	User Evaluation Process	113
72	User Evaluation - UI General Score Average	122
73	User Evaluation - Bryan Boyle information	123
74	User Evaluation - Dr. Sarah Rock information	125
75	User Evaluation - Odran Reid information	127
76	User Evaluation - UI Score Odran Reid	128
77	User Evaluation - UI Target Score Average	128
78	V.0.1 Magpie Dashboard & Map when 2 amenities are selected	131
79	Error message when using inexistant username and password	131
80	UX review 1 - UI score	132
81	UX review 1 - Q8 from Information Architecture	132
82	UX review 1 - Overall score	133
83	Magpie V.0.12 - Landing page sign up button	134
84	Magpie V.0.12 - New Dashboard	134
85	Magpie V.0.12 - Amenity Icons grouping relative to zoom level	135
86	V.0.12 Magpie Search Failing	135
87	V.0.12 Magpie Search Working	136
88	V.0.12 Magpie Saved Location	136
89	UX review 2 - UI score	137
90	UX review 2 - Overall score	137
91	WCAG 2.1 Scope, Guidelines & Exemptions	138
92	V.0.1 Magpie Onboarding Steps	141
93	V.0.1 Magpie Profile & Onboarding icons	141
94	Accessibility review Overall Score	142

95	UX score - Summary all users	143
96	User Evaluation - Satisfaction survey questions	152
97	User Evaluation - UI Score Brendan	153
98	User Evaluation - UI Score Anonymous 1	153
99	User Evaluation - UI Score Paul	154
100	User Evaluation - UI Score Livia	154
101	User Evaluation - UI Score Ben	155
102	User Evaluation - UI Score Jakub	155
103	User Evaluation - UI Score Bryan Boyle	156
104	User Evaluation - UI Score Dr. Sarah Rock	156
105	User Evaluation - UI Score Odran Reid	157
106	UX review 1 - Information Architecture score	157
107	UX review 1 - Technical Performance score	158
108	UX review 2 - Information Architecture score	158
109	UX review 2 - Technical Performance score	159
110	Accessibility review - UI Components Score	159
111	Accessibility review UI Design & Language Score	160
112	Accessibility review UI Interaction Score	160

List of Tables

1	YOLOv8-OBb model results	37
2	Performance Metrics for Road Mask Classification	39
3	Performance Metrics for Empty Parking Detection	42
4	Performance Metrics for Parking Spot Classification	43
5	Process Flow Table: Design and Development Stages	66
6	Usability testing Tasks - Paul	117
7	Usability testing Tasks - Livia	118
8	Usability testing Tasks - Ben	119
9	Usability testing Tasks - Jakub	120
10	Usability testing Tasks - Non-Expert Users Summary	121
11	Usability testing Tasks - Bryan	124
12	Usability testing Tasks - Dr. Sarah Rock	126
13	Expert Review Questionnaire	130
14	Scenario for the Accessibility review	139
15	General tasks score for the Accessibility review	140

List of Listings

1	Python script to obtain training images for ML model	25
2	YOLOv8-OBB model training settings	30
3	An example of a SQL query with annotations used by sqlc	74
4	An example of a sqlc configuration file with two targets with separate query inputs and type replacement	76
5	An example of a Go binding generated by sqlc from the SQL query in Listing 3	77
6	An example of how routing is configured in the backend	86
7	An example of a DTO including data validation used by the backend . . .	93
8	An example of a response DTO used for the successful retrieval of user details	94
9	An example of a response DTO transmitting a custom error condition . .	94
10	An example of a test case for a table-based unit test of the private points handler	97
11	An example of an issue template used in <i>Magpie</i>	103
12	An example of a pull request template used in <i>Magpie</i>	105
13	An example of a GitHub Actions workflow that will not work	107
14	An example of a GitHub Actions workflow that will work	107
15	A github action that checks the branch name (this was edited for brevity)	108
16	Example of a Kubernetes Deployment for the public backend	111

1 Introduction

Urbanization brings increasing demands for efficient city planning and sustainability. **Magpie – Services at a Glance** is a geographical information service designed to address these challenges, by offering a streamlined platform for accessing and analysing public amenities. Developed as part of the MSc in Computing (Advanced Software Development | Data Science) group project, **Magpie** combines cutting-edge machine learning, intuitive visualization, and robust data aggregation to provide a high-level overview of public infrastructure.

Targeting urban planners, **Magpie** simplifies the traditionally complex and fragmented processes of data collection and analysis. By integrating satellite imagery, automated detection techniques, and multi-layered data visualization, the system identifies parking spots, bike infrastructure and more – all at a glance. Unlike existing tools that rely on manual input or advanced GIS expertise, **Magpie** automates the extraction and fusion of real-world data, ensuring accessibility and ease of use.

This report details the technical challenges, innovative solutions, and real-world applications of **Magpie**. From the architecture and machine learning methodologies to the user-centric frontend and robust backend infrastructure, each component contributes to a powerful tool that empowers informed decision-making and urban development.

2 User Scenario

2.1 Who is our target user?

Magpie's primary target is Urban Planners. Urban Planners are professionals responsible for the development of cities and towns, focusing on the efficient use of land, infrastructure planning, and the creation of sustainable and resilient communities. (Fischler, 2012) Our secondary target is any non-expert user who is interested in amenities planning, and this includes, but is not limited to the following: Sustainability Advocates, Commuters, Event Planners, Journalists, Political Advisors, and Parking Companies.

These personas were developed on basis of a combination of real-life people who are known to our team, as well as based on research into the goals and needs of people in these professions.

2.1.1 Primary Persona - Michael O'Brien

Mr. O'Brien is a 48-year-old Urban Planning Specialist at Dublin City Council, deeply committed to enhancing his hometown of Lusk. He specializes in sustainable urban development, integrating smart technologies with traditional planning principles, and has contributed to notable projects like Dublin's cycle lane network. He represents our primary target user.

Michael is currently working on the expansion of Dublin's cycle lane network southbound. He has facing some challenges because the maps provided by Dublin City Council are static and do not show crucial information such as public facilities and amenities. The datasets he's been accessing seem out of date and not comprehensive at all.

Michael needs a tool that allows him to interact with a map that contains key data on public facilities and amenities, as well as options to extract that information for analysis and planning.

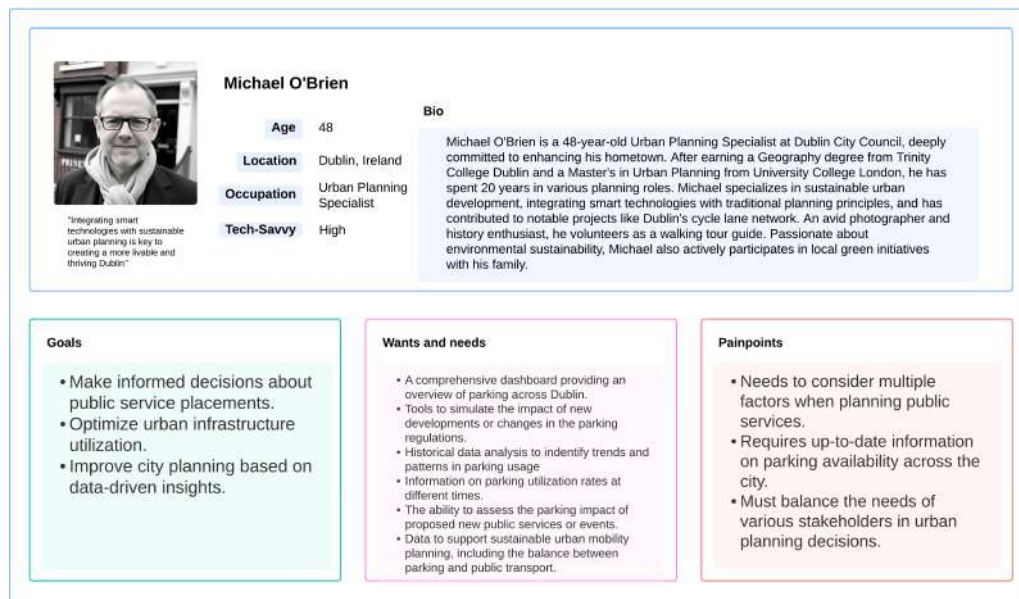


Figure 1: User persona - Michael O'Brien

2.1.2 Secondary Persona - Sarah Thompson

Sarah Thompson is a 35 year old CEO of an event's planning company based in Dublin, Ireland. Currently Sarah specializes in corporate events and weddings, and is known for creating memorable experiences while addressing key logistical challenges such as venue selection, catering, accessibility and transportation. She represents our secondary target user.

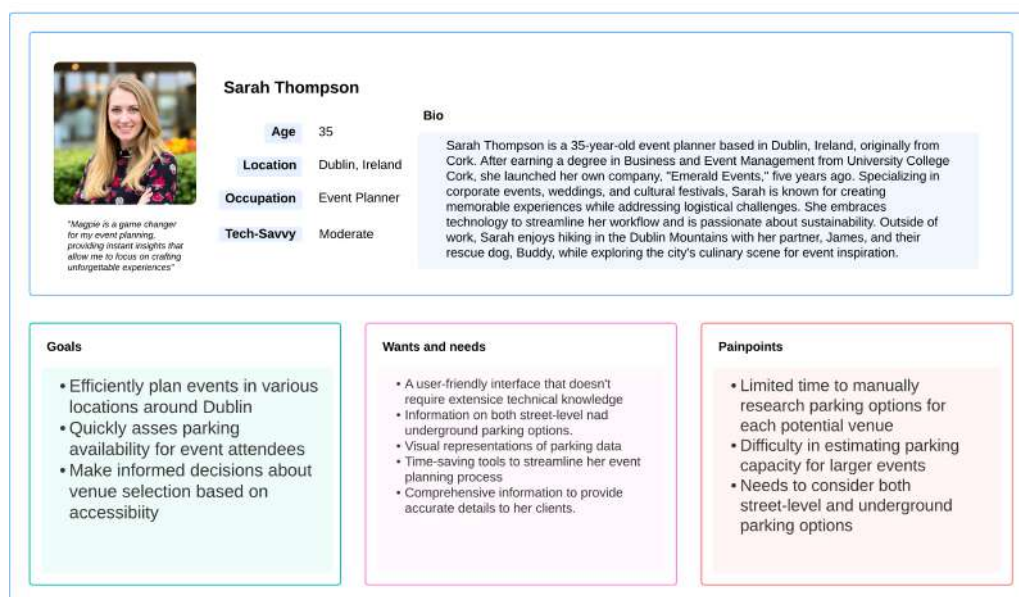


Figure 2: User persona - Sarah Thompson

Currently, she faces many challenges planning events within Dublin city. She has limited time to manually research transport and parking options and to visually assess the accessibility of a possible venue. She's looking for a tool that will give her a visual overview of transportation and parking options in an area, with additional information on their location & their quantity. She also needs something quick and easy to use.

2.2 Why are they important?

Urban Planners are crucial for the holistic development of cities and towns. (Jha et al., 2021) They ensure that urban development is sustainable, balancing economic growth and environmental protection. (Lei, Flacke, and Schwarz, 2021) Urban planners guide cities towards more efficient uses of land, better transport systems, and overall increasing the quality of life of residents. (Janpavle and Īle, 2022)

By planning for current and future needs, Urban Planners help mitigate urban challenges such as congestion, pollution, and lack of infrastructure. Their role is essential for ensuring that urban areas can meet the demands of growing populations, while maintaining standard of living and environmental sustainability.

2.3 What problem are we solving?

Urban Planners often face challenges with accessing and analysing data. Data is typically siloed, making it difficult to access and analyse. (Duivenvoorden et al., 2021)

We aim to provide Urban Planners with a tool that allows them to access this information in a single, easy to use platform. This will allow Urban Planners to streamline the initial phases of their work, decreasing the time spent on data collection and increasing the time spent on analysis and decision making.

2.4 Market analysis

The identification of our target user was made possible through exploratory work done at the beginning of the project timeline. A research survey was conducted using Microsoft Forms, sent to county councils, planners, firms and the TUDublin mailing list, to answer key demographic & product questions:

1. Who is our primary target user?
2. What kind of amenity data do they access and how?
3. What devices/tools do they primarily use?
4. Are they satisfied with those tools?
5. Would they consider Magpie useful in filling the gaps in their toolset?

2.4.1 Demographic data

This first section of the survey covered demographic data, to allow us to build a profile for the respondents. Personal identifiable information was not collected for the purposes of anonymity.

We had a total of 118 respondents, which we divided into 2 categories:

User A: Respondents who did not use amenity data in their work = **88 users**. *User B:* Respondents who used amenity data in their work = **30 users**. Splitting the respondents allowed us to better analyse the answers in the context of if they belong to our primary or our secondary target user group.

Both user groups were evenly distributed in employment sector and counties they work in, as illustrated in the figures below.

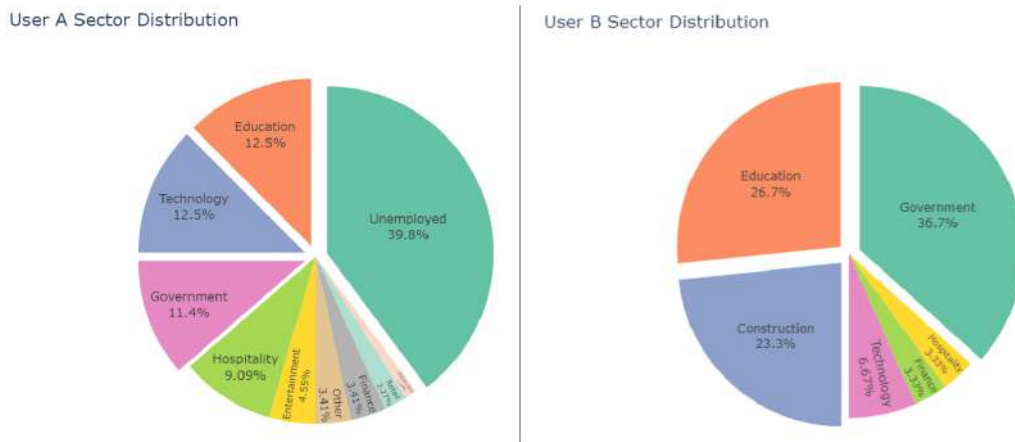


Figure 3: Market analysis - User Sector Distribution

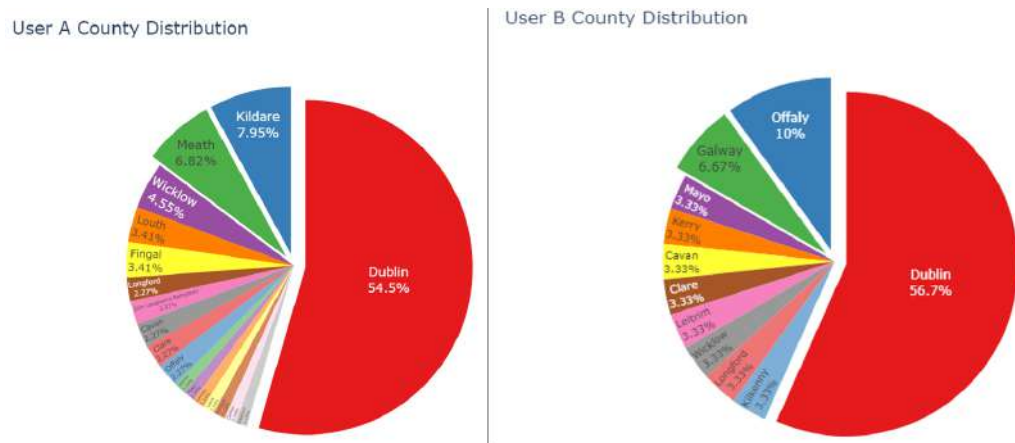


Figure 4: Market analysis - User County Distribution

2.4.2 Amenity data

This section specifically questions respondents in the User B group on their use of amenity data in their work.

We can see that the most common amenity data accessed is transportation.

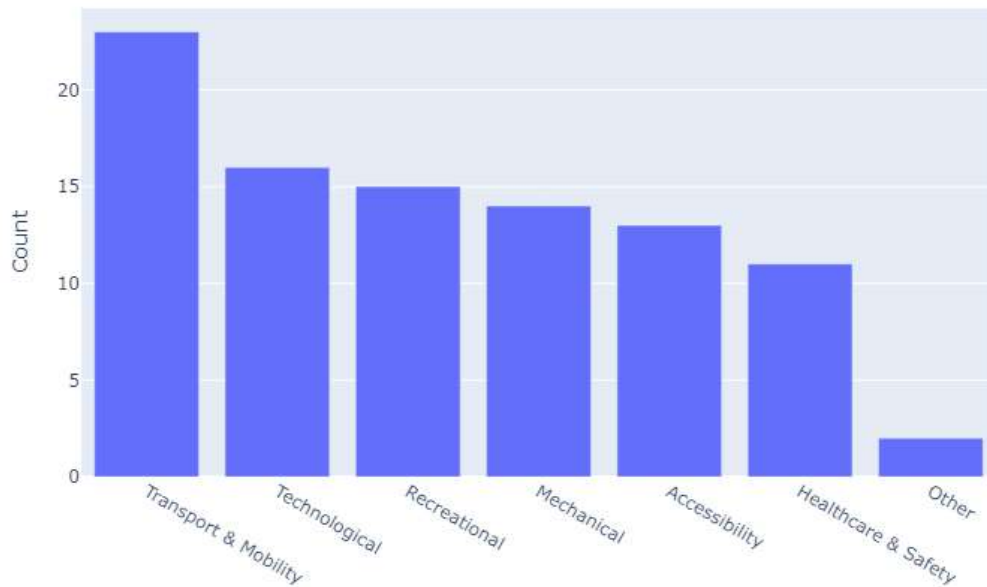


Figure 5: Market analysis - User B Amenity Data Type Distribution

2.4.3 Devices & Tools

This section looks at the devices used by both users groups, as well as the applications or tools they are using to access amenity data.

User A was specifically questioned on which device they use for navigation and at what frequency.

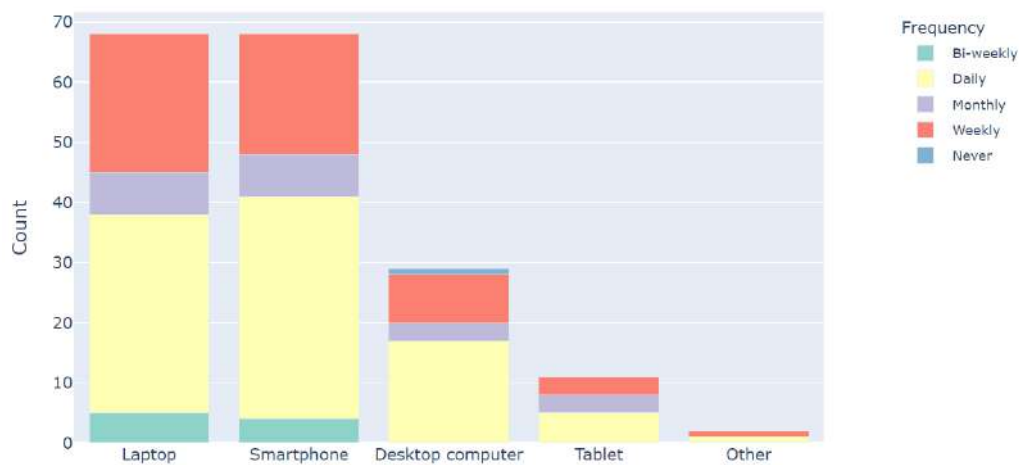


Figure 6: Market analysis - User A Preferred device & Usage frequency

User B was specifically questioned on which device they use for their day-to-day work, as well as what tool they are currently using to access amenity data.

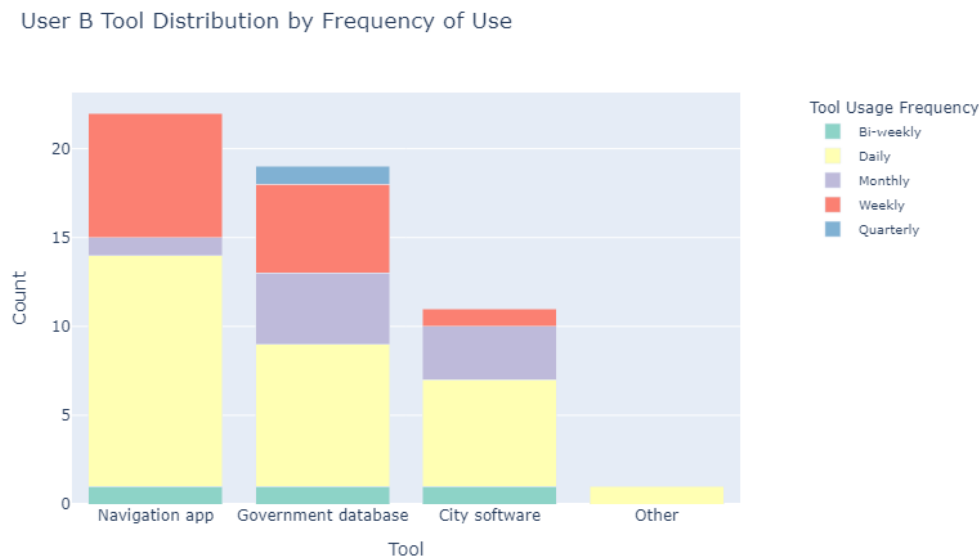


Figure 7: Market analysis - User B Tool type & Usage frequency

User B were also questioned on their satisfaction with their current tool. More than 50% of respondents said they weren't completely satisfied, citing incomplete information, lack of user friendliness and poor speed as reasons.

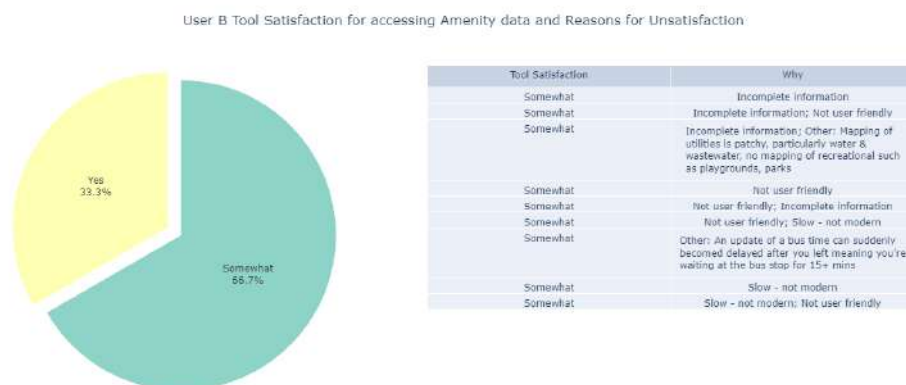


Figure 8: Market analysis - User B Tool satisfaction & Usage frequency

2.4.4 Thoughts on Magpie

Lastly, this section covers questions regarding Magpie's potential to bridge the gap in modern visualizing solutions.

Both user groups found that Magpie would be useful to access amenity data, less than 5% citing it would be impractical because either they did not require access to this information, or other tools such as Google Maps already satisfied their needs.

User B Tool Distribution by Frequency of Use

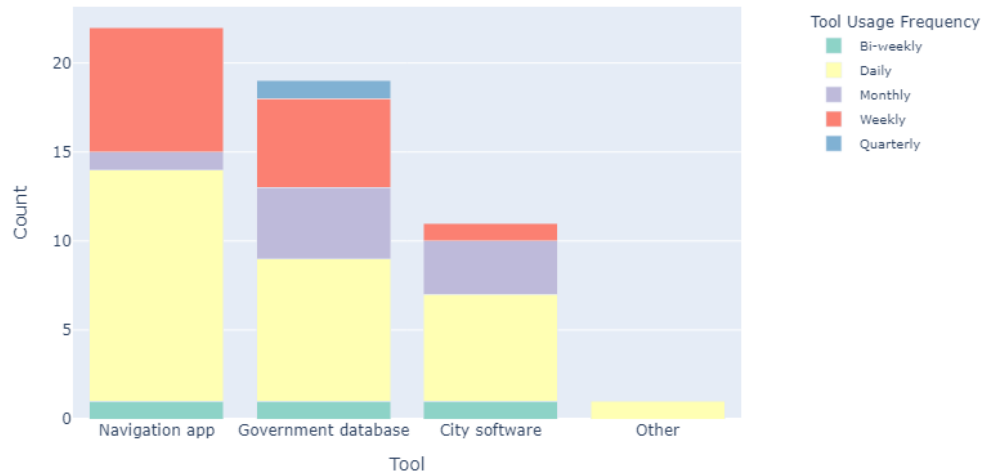


Figure 9: Market analysis - User A Thoughts on Magpie

User B Tool Distribution by Frequency of Use

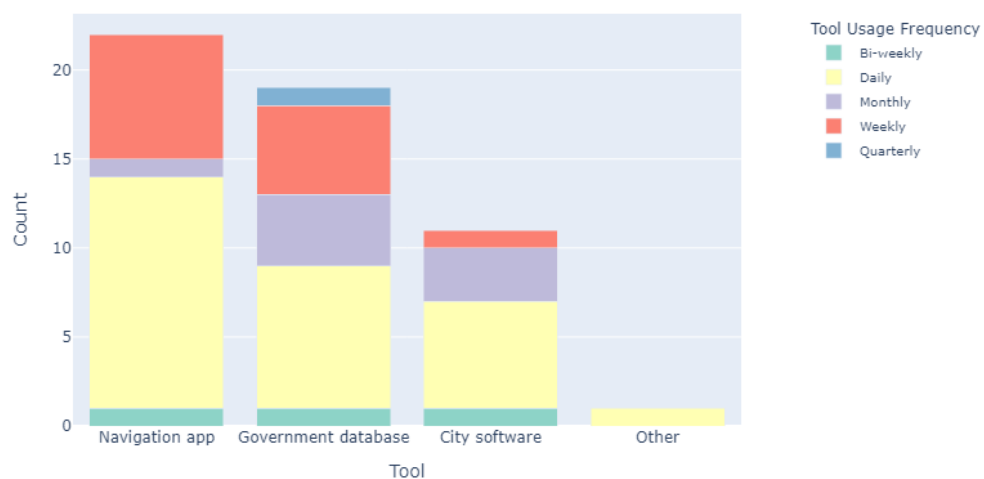


Figure 10: Market analysis - User B Thoughts on Magpie

Users from Group B were also asked what additional features they would like to see on Magpie. The highest rated ones were a **search functionality** and **filters**.

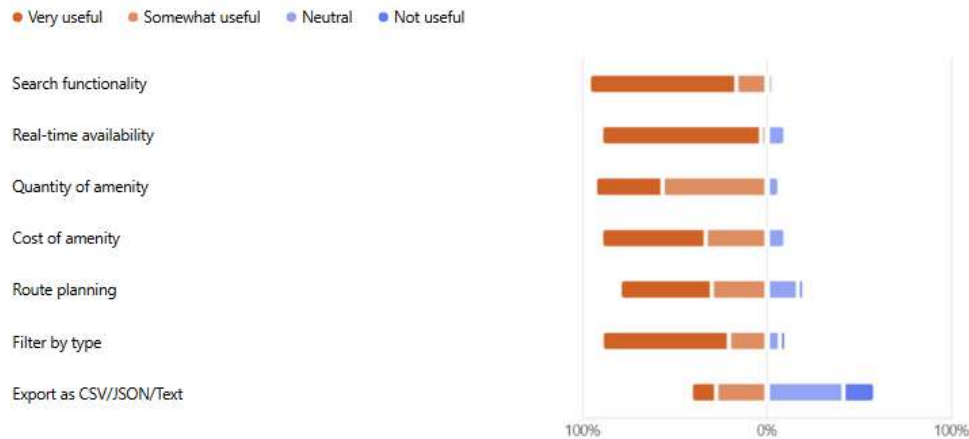


Figure 11: Market analysis - User B Rating Additional Features

Overall, this exploratory work allowed us to confirm our primary and secondary target users as *urban planners* and *Non-Expert Users* respectively.

Additionally, it gave us insight into a feature list to implement throughout development, as well as competitors, into which, we would research.

3 Technical Problem

3.1 Why does our system exist?

In our research, we couldn't find a singular system that allows users to gain a general overview of public amenities, for example, parking, bike infrastructure, or public transport. In Ireland and the United Kingdom, the prevalence of proper digitised records in county administrations varies wildly (Lynn et al., 2023). Some make use of state-of-the-art geographical information systems (GIS) while others rely on spreadsheets which are manually kept up to date. (McGuirk and MacLaran, 2001)

A system that would allow users to quickly inspect a combined dataset grounded in automatically generated, real-world data could accelerate processes like planning permissions, urban development, or resource allocation. This applies especially to smaller counties that may not have personnel experienced in this area. (Clark et al., 2002)

Improving how governments allocate resources could lead to improvements in efficiency, making public services more attainable, even for smaller communities. Such as, having accurate data on public transport and bike infrastructure encourages more people to choose sustainable modes of transport, helping reduce carbon and noise emissions (Mugion et al., 2018). It could also make it easier for people in underserved communities to access important services, such as housing, thus helping to bridge social gaps. (Allen, 2015)

Currently, solutions are fragmented, focusing on just one type of amenity, like transport or housing, without giving a full picture. Even the more advanced GIS systems, while powerful, usually need a lot of manual input or additional programming, which smaller municipalities or organizations might not be able to handle. For example, to find parking spots in an area, the user needs to either bring their own dataset or they have to perform computer vision analysis themselves. While *ArcGIS* includes tools to run deep learning on raster images, the process is far from straight forward. Many of these GIS tools are also expensive, or require specialised training, making them out of reach for smaller boroughs or counties. (Kaufmann et al., 2022)

3.2 The core technical problem

The problem at the heart of our project is to find a way to collate and visualize useful data about public amenities, without the need for any existing information or manual data entry. This would allow the solution to be useful, no matter the state of the users digital records. Relying purely on existing data would do nothing to reduce the divide between counties with more data and counties with less data. This necessitated the creation of an interconnected series of modules that would form a pipeline, for the automatic extraction and fusion of data.

In order to present users with a map containing information about the number of public amenities, we first need identify them. Our biggest technical challenge was the extraction of these data points. Different forms of public amenities exist in satellite imagery, we combined this with other data sources, interweaving them, which produced a better result.

Detecting public amenities from overhead views is a non-trivial process, as satellite imagery suffers from low resolution, occlusion, seasonal changes, or cloud cover. This means classifying specific objects can be complicated, and can take a lot of time and processing power. For instance, the task of quantifying on-street parking is dependent on reliably detecting cars that are not actively participating in traffic (in aerial images). Not only that, but the detected points in the images must be mapped back onto real-world coordinates to be of any utility.

Ensuring consistency between the extracted data, and other data sets, can have its own pitfalls. For example, compensating for:

- different precision in geographical alignment
- varied update frequencies
- different data formats

Not only can data be geographically misaligned, but since satellite data is only a snapshot in time, it can lead to temporal misalignment as well. Even the appearance of amenities can change depending on their location. While cars look more or less the same anywhere on Earth, the same cannot be said for public transport links, housing or hospitals, which could lead to difficulties achieving the same level of accuracy in different regions. Processing a large amount of data can be difficult and finding a strategy to do this efficiently is necessary.

3.3 A review of similar systems

Parkopedia

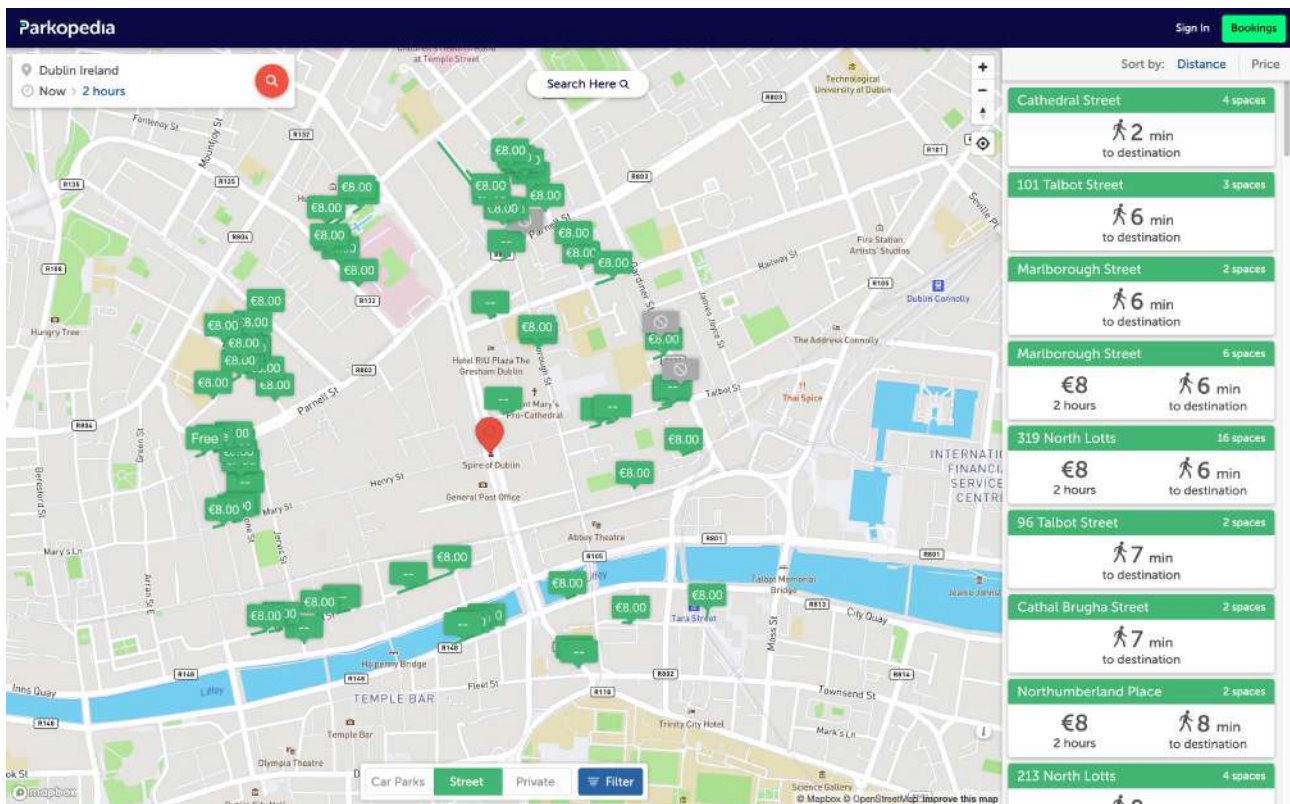


Figure 12: Example of Parkopedia

*Parkopedia*¹ is a web application that allows users to find parking in many cities around the world. They offer location and pricing data on parking garages, on-street parking and parking on private property.

It is similar to *Magpie* in the sense that it allows users to gain a quick overview of all parking options in their area.

Our system differentiates itself from *Parkopedia* in three main ways:

- **Data fusion:** Our system integrates data from publicly available sources, combining multiple datasets to provide a more comprehensive overview of public amenities – not just parking.
- **Aggregated data:** Our system offers a high-level overview of the amount of available amenities in a certain radius. This would be possible with the data from *Parkopedia*, but the user would have to manually retrieve the data they wish to analyse.
- **No reliance on manual data entry:** Unlike *Parkopedia*, which relies on user-submitted information, we automatically acquire and fuse datasets. This pipeline allows us to keep information up-to-date and adapt our system to new regions without the need for manual data input.

¹<https://www.parkopedia.com/>

ArcGIS

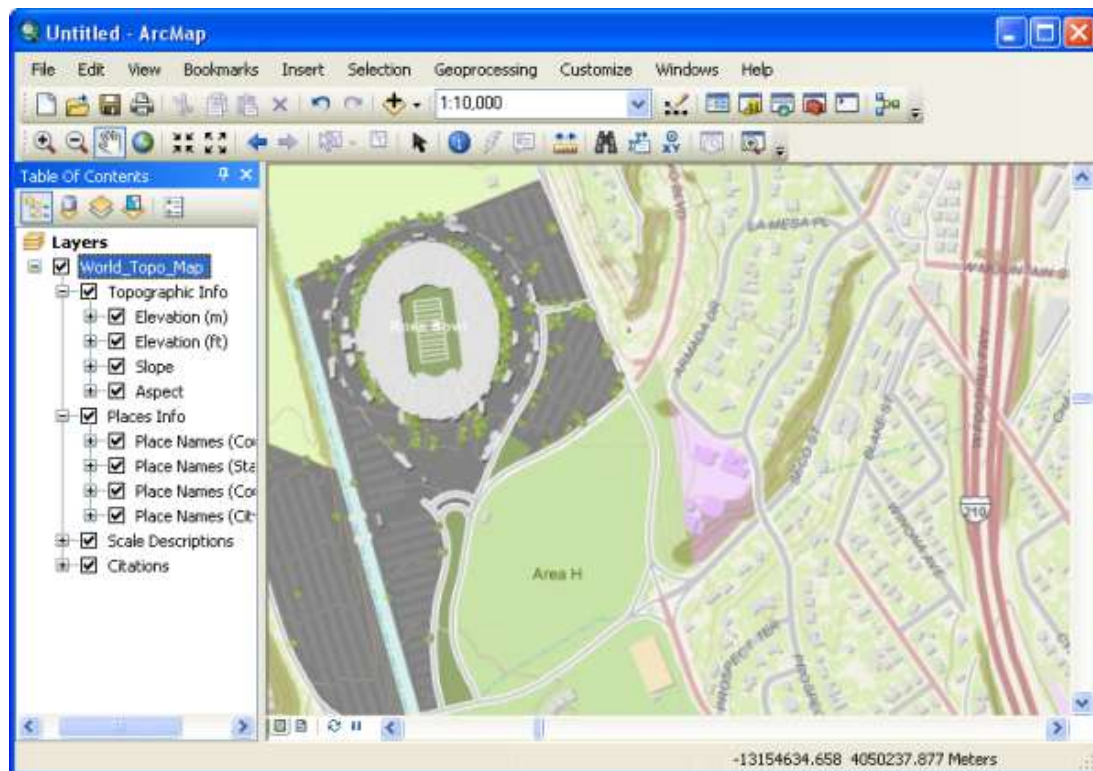


Figure 13: Example of ArcGIS

Systems like *ArcGIS*² target mainly expert users like cartographers and researchers. These systems are powerful and feature-rich tools for professionals. While *ArcGIS* is able to produce the same overview offered by our system, it is different in these key aspects:

- **Ease of use:** The goal of our system is to cater to a broad range of users, not just experts. Systems like *ArcGIS* require specialised domain knowledge, as it's aimed at professional users. While *ArcGIS* offers powerful tools, it can be overwhelming for the more Non-Expert User who needs a quick overview of public amenities, without engaging with advanced features.
- **Accessibility:** GIS's are a robust tool, but their access is often restricted by a lack of technical expertise. Compared to *ArcGIS*, our system is less versatile. However, our system is more streamlined to offer- at a glance- an easily accessible way of gauging information about public amenities. This makes sure that users from all backgrounds can utilize our system without the need for specialised training.

²<https://www.arcgis.com/>

3.4 Conclusion

The need for *Magpie* arises from the fragmented state of public amenity data and the barriers to accessing it efficiently. By addressing these challenges, our system offers a combination of automated data extraction, fusion, and visualization that caters to users across technical skill levels. Unlike existing solutions, which are either too specialized or demand significant expertise and resources, our system prioritizes inclusivity and accessibility.

Through the integration of real-world data we aimed to bridge gaps in resource allocation, improve efficiency in urban planning, and support sustainable development. By lowering the entry barriers for smaller municipalities and underserved regions, the system democratizes access to critical geographical insights and also contributes to fostering equitable and sustainable communities.

This project lays the groundwork for a more informed approach to public planning and resource distribution, with the potential to evolve into a versatile tool that supports diverse stakeholders in addressing the complexities of modern urban and regional development.

4 Technical Solution

4.1 System Architecture

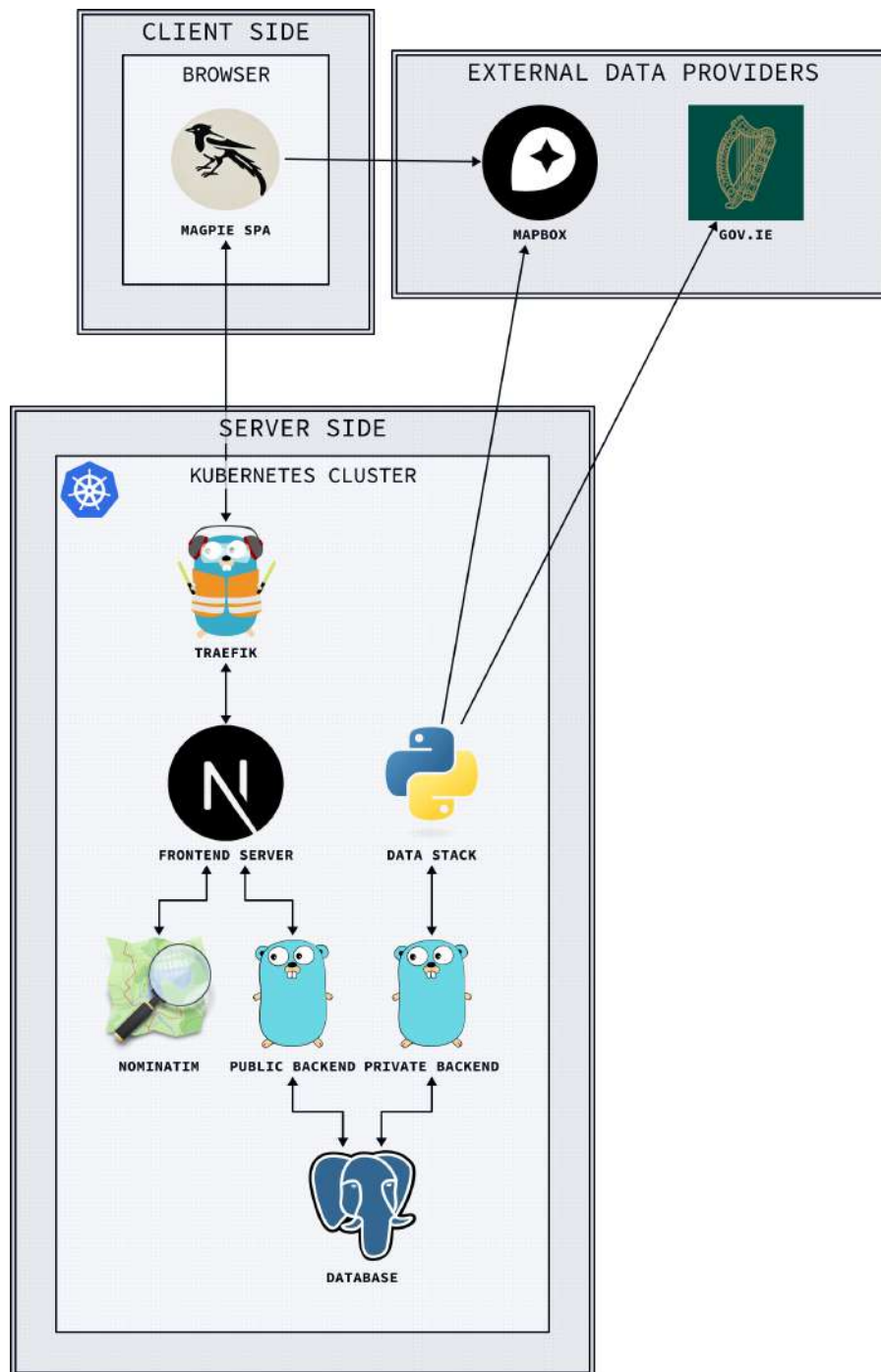


Figure 14: System Architecture Diagram

- The client-side portion of this diagram is discussed in the Frontend section.
- The server-side portion of this diagram is discussed in the Backend section.
- The external data providers are discussed in the Data Sources section.

4.2 System Overview

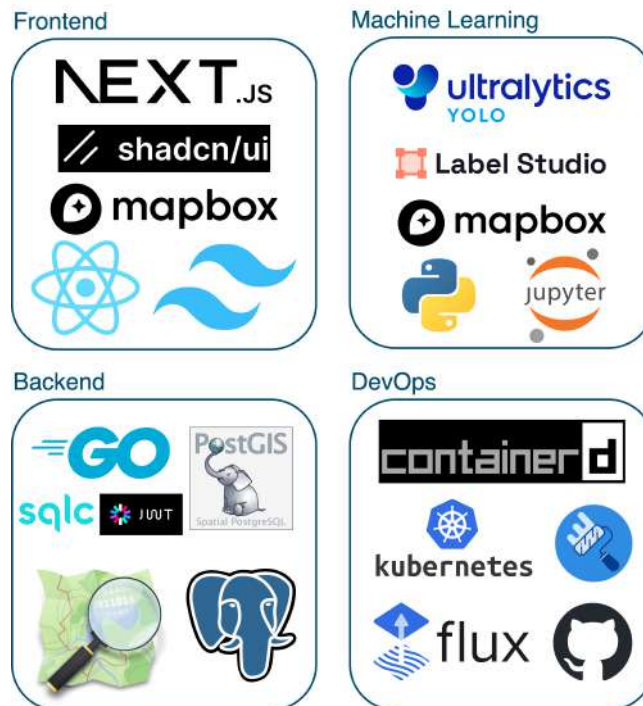


Figure 15: Magpie's Technology Stack

4.2.1 Frontend Technologies

The frontend stack plays a critical role in providing a smooth and interactive user experience for the application. It is built on top of modern web technologies, each serving a unique purpose:

- **Next.js:** This React framework offers server-side rendering (SSR) and static site generation (SSG), which improve performance, SEO, and user experience.
- **React:** As the foundation of the frontend, React allows for component-based development, making the interface scalable and easy to maintain.
- **TailwindCSS:** A utility-first CSS framework that simplifies styling by providing ready-to-use classes, enabling rapid UI development.
- **shadcn/ui:** A collection of reusable UI components that work seamlessly with Next.js and React, enhancing productivity and consistency across the application.
- **Mapbox:** A powerful mapping platform that enables interactive, customizable maps. It helps visualize geospatial data effectively, which is key for location-based services, in this project.

These technologies combine to create an engaging user interface that is both functional and visually appealing, allowing users to interact with location-based data and services efficiently.

4.2.2 Machine Learning Technologies

The machine learning stack adds intelligent capabilities to the system, enabling advanced features such as object detection. The following is a breakdown of the tools in this section:

- **Ultralytics YOLO:** An object detection model that helps identify specific objects in images, enhancing the system's ability to recognize and interpret visual data.
- **Label Studio:** A flexible tool for annotating datasets, enabling easy labeling of images, text, and audio. This is crucial for creating high-quality datasets to train machine learning models like YOLO.
- **Python:** The primary language employed to develop scripts and run machine learning models. Python supports image processing, model training, and analysis.
- **Jupyter Notebooks:** An interactive platform that allows data scientists to experiment with code, visualize results, and document their machine learning processes.

These technologies allow the application to process and analyse data intelligently, enabling features such as object recognition in geographical data and personalized insights.

4.2.3 Backend Technologies

The backend stack ensures the application can handle large volumes of data, manage user requests, and perform complex computations. Some of the key technologies include:

- **Go (Golang):** A highly performant programming language ideal for building scalable backend services.
- **PostGIS:** An extension of **PostgreSQL**, PostGIS adds geographic object support to the database, making it easier to work with spatial data like maps, coordinates, and distance calculations.
- **sqlc:** A SQL code generator that helps developers write type-safe database queries in Go, reducing the chances of errors during database interactions.
- **JWT:** A compact, URL-safe token format used for securely transmitting information between parties, often for authentication and authorization purposes.

By leveraging these tools, the backend is capable of managing complex spatial data, performing queries on geospatial data, and ensuring smooth user interactions through highly responsive APIs.

4.2.4 DevOps Technologies

The DevOps stack ensures that the infrastructure supporting the application is reliable, scalable, and easily deployable. Key technologies in this section include:

- **Containerd:** A core container runtime that manages containerized applications, ensuring consistency across different environments (development, staging, production).
- **Kubernetes:** An orchestration platform for automating the deployment, scaling, and management of containerized applications. It allows the system to scale effortlessly based on demand and ensures high availability.
- **Flux:** A tool for continuous delivery and GitOps, Flux automates deployments by integrating with Git repositories. Changes made to the codebase are automatically reflected in the deployed infrastructure, ensuring consistency and reducing manual intervention.
- **GitHub:** The widely-used platform for version control and collaboration. GitHub supports seamless development workflows by enabling pull requests, code reviews, and version tracking.

Together, these tools create a robust DevOps pipeline that automates deployment, enhances reliability, and ensures that the application can handle scaling demands as it grows.

4.2.5 Integration and Synergy

The combination of frontend, backend, machine learning, and DevOps technologies ensures that the system functions as a seamless whole. The following illustrates the different parts work together:

- The **frontend** presents an intuitive interface to users, allowing them to interact with location data, maps, and insights.
- The **backend** handles the heavy lifting, managing the spatial data, processing queries, and supporting dynamic content through APIs.
- **Machine learning** capabilities add intelligence to the system, enabling features like real-time object detection and personalized data analysis.
- **DevOps** practices ensure that the entire application can be deployed and scaled efficiently, with smooth continuous integration and deployment pipelines.

In conclusion, this tech stack forms a cohesive, scalable, and intelligent system. From object detection and efficient spatial data management to automated deployment pipelines, the technologies complement each other, resulting in a high-performance, feature-rich application.

4.3 Data Sources

Throughout project development, several datasets from various sources have been used to fulfil Magpie’s project goal. Below is an overview of Magpie’s data stack:

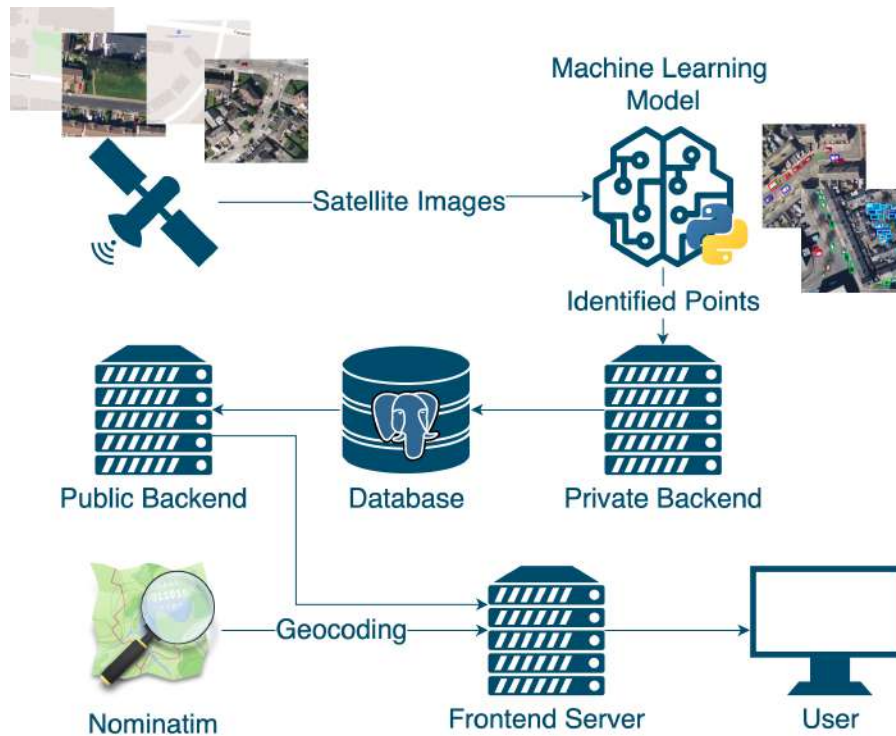


Figure 16: Magpie Data Stack

4.3.1 Data sourcing & Collection

Three main categories of data have been sourced for Magpie:

1. For the *Machine learning models*
2. For the *Amenity data*
3. For the *Search functionality*

1.1 Machine learning models - Car detection The Mapbox Static Images API was used to retrieve all the images used for the machine learning models of this project.

First, 250 images were downloaded using CSV data from the following sources:

- **Data.gov.ie:** Online portal containing thousands of publicly available datasets about Ireland
- **Smart Dublin:** Founded by Dublin local authorities, their goal is to look for innovative technological solutions for local environmental, social, mobility, government and living issues. They also have hundreds of publicly available datasets
- **Kaggle:** a data science and machine learning platform part of Google Cloud where competitions are hosted, in addition to thousands of publicly available datasets

These CSV files contained information on the location of certain facilities such as rentals, coach parking, public parks. This location was latitude and longitude which we needed to input into the script to download images, as well as the street name of that location which was used to name the downloaded image.

```
# CSV file containing location information
df = pd.read_csv("2020-coach-parking-dcc-1.csv")
location_data = df.to_dict('records')

# function to download image using location information from CSV dataset
def get_images(name, latitude, longitude):
    url = f'''https://api.mapbox.com/styles/v1/mapbox/satellite-v9/
            static/{longitude},{latitude},18,0,0/400x400'''
    response = requests.get(url)
    print(response)
    if response.status_code == 200:
        img = Image.open(BytesIO(response.content))
        if not os.path.exists('output'):
            os.makedirs('output')

        output_folder = 'output'
        output_path = os.path.join(output_folder, f'{name}_coach-parking.png')
        img.save(output_path)
        print(f"Image saved to {output_path}")

# function to extract location & name location from CSV dataset
for place in location_data:
    name = place["Street Name"]
    if ("/" in name):
        name = name.replace("/", "-")
    latitude = place["Latitude"]
    longitude = place["Longitude"]
    get_images(name, latitude, longitude)
```

Listing 1: Python script to obtain training images for ML model

Following successful model training and tuning, a further 19,536 satellite images were downloaded spanning Dublin City area illustrated in the figure below. The machine learning model was then applied to those images to detect the cars.

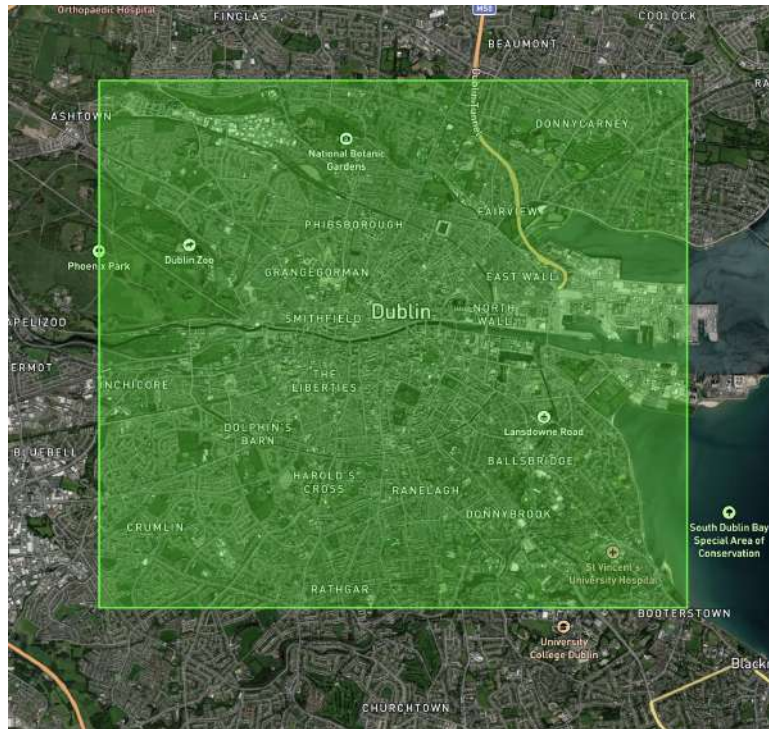


Figure 17: *Dublin Area Delimitation for Parking data*

1.2 Machine learning models - Parking detection

An additional 19,536 map-view images corresponding to the satellite images used for car detection were downloaded using the MapBox API to create the custom mask used for detecting parked cars. Our project has two main Python scripts, `parking_detection.py` and `parking_detection_local.py` to localize parking spots. In `parking_detection.py`, the satellite images (Mapbox Satellite) and corresponding road mask images (Mapbox Streets) are retrieved for a specific area contained within a bounding box, defined by its top-left and bottom-right coordinates, directly from the Mapbox API. From the coordinates of the specific bounding box, the center coordinates of all the images necessary to make up that area are calculated, and the images are then retrieved. While in `parking_detection_local.py`, the original parking detection script is adapted to run on a database of locally stored Mapbox Satellite images and the corresponding Mapbox Streets images, covering the entirety of Dublin city.

Extensive testing was done at all the different stages of development of the scripts, to identify the optimal thresholds and parameters using test images from Mapbox, in a variety of scenarios including edge cases or cases prone to causing issues.

2. Amenity data CSV files were collected from multiple sources mentioned above, notable Data.gov.ie and Smart Dublin. Below is a list relative to each amenity currently present on Magpie, linked in 9.1:

- Parking meter
- Bike stand
- Public Wifi
- Library
- Multi-storey Car park
- Drinking water fountain
- Public toilet
- Bike sharing station
- Car parking
- Accessible parking
- Public bins
- Coach parking

Some data cleaning and processing had to be done to obtain a new csv file with longitudes and latitudes in the right format to send to the backend using our send points script.

For example in 2 datasets, the coordinates were in a local coordinates system (easting/northing) and not the universal longitude/latitude requiring us to write a script to convert them properly. And certain datasets were only in geojson format so we had to write a script to handle the conversion to csv. For one file, the geometry field, which contains the point data, the coordinates were not being read into a pandas dataframe correctly necessitating that we parse it using json and extract the fields instead.

To complement this data visually on the map, SVG icons were used from **Icons8**, a UX design agency that provide high quality, free-to-use icons for personal and commercial web projects.

3. Search functionality Nominatim is a geocoding solution created by the OpenStreetMap Foundation. Magpie uses a self-hosted instance of Nominatim for geocoding and reverse-geocoding tasks. Using the Nominatim API, the frontend server can request the coordinates associated with a place name. This is used in the search functionality of the frontend.

When the user saves a location, the frontend will also query Nominatim, but this time with a set of coordinates. Nominatim will then return the closest place name to the given coordinates, which is then sent to the database alongside the location data.

4.3.2 Data storage

Machine learning models data The 250 training images are stored locally. The 19,536 satellite and 19,536 map-view images are also stored locally.

Amenity points The CSV data obtained from sources named above are stored on the Postgres database server. The amenity icons were downloaded locally then stored on the Magpie frontend server.

Search functionality Any actions which involve data querying, requests and storage are handled by Nominatim, and therefore not stored on any of Magpie's servers.

4.4 Machine Learning

Early in the Project planning, car parking was considered an amenity that had to be included on Magpie’s interface. Through searching for publicly available datasets, no information on public, on the street, or private parking spaces in Ireland was found.

Thus, creating our own dataset of car parking spots in Dublin using a novel machine learning approach was agreed. This approach is divided into 3 main steps:

1. Car detection on satellite images
2. Parking detection using custom road mask
3. Classification & evaluation of the parking spots

4.4.1 YOLO model used for car detection

YOLO (You Only Look Once) is an object detection and image segmentation model launched in 2015 which quickly gained popularity for its high speed and accuracy.

The YOLO model was introduced to us on Label Studio, an image labelling software used for image recognition & detection tasks. There is an extension called ML Backend on the platform which can be used to automate the labelling process using an integration of the YOLOv5 model. However, this extension was not working as intended, which caused us to pivot towards Ultralytics, the company behind the YOLO models.

Several papers focusing on object detection cited YOLO over other popular models such as (Fatma M. Talaat, 2023) and (Mehrshad Lalinia, 2023). A key challenge that influences the choice of model for the object detection task is effectively managing fluctuations in image resolutions and aspect ratios, as well as the computational resources required to run image detection models.

There are two main types of models for object detection tasks: single stage and two-stage. The figure below from Poirson et al., 2016 illustrates the process for both.

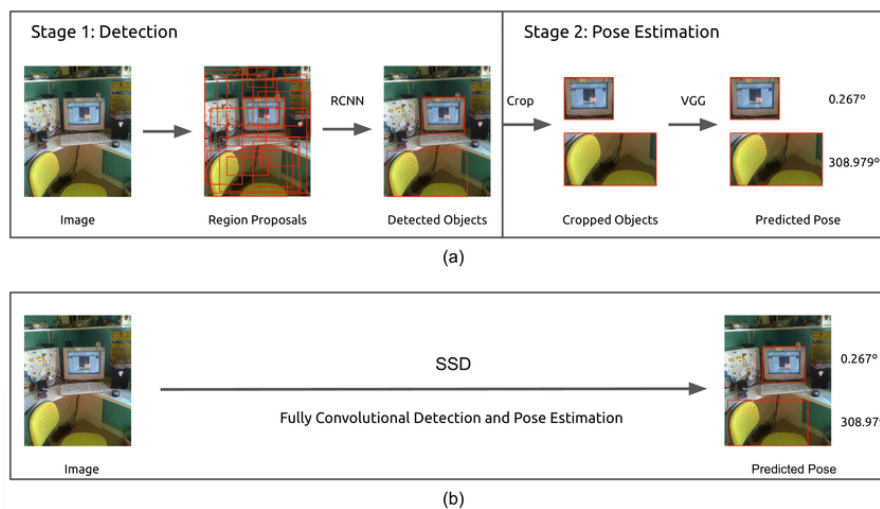


Figure 18: Single vs Two-Stage Object Detection Process

Single stage object detector models processes an image through a feature extractor using a CNN (Convolutional Neural Network) and then directly uses the extracted features for classification and regression of bounding boxes. These models are very fast which is why they are popular for real-time object detection tasks; however their accuracy performance can sometimes be poor. Examples of single stage models are SSD, YOLO and RefineDet (Hussain, 2024.)

Two stage object detector models divides the process into two main steps. First, it extracts features from the image using a CNN, then selects regions of interests that will only be used for classification and regression of bounding boxes. These models are much more accurate than one stage detectors and are used for tasks where accuracy is prioritized over speed in the medical field for example RCNN, Fast-RCNN and Faster-RCNN(Takur, 2023).

For the scope of this project, YOLO models were chosen for their speed, their scalability and their lack of computational resources.

Different versions of the YOLOv5 model were experimented with, slowly levelling up the versions up to version 8. The YOLOv8 model has been pre-trained on the COCO (Common Object in Context) dataset, a popular large scale dataset with 200,000 images across 80 object categories commonly used for benchmarking computer vision models (Ultralytics, 2024).

Metrics such as mAP (mean Average Precision), F1-score and the false positive rate were monitored. The figure below shows the metrics of the model iteratively experimented up the version chain.

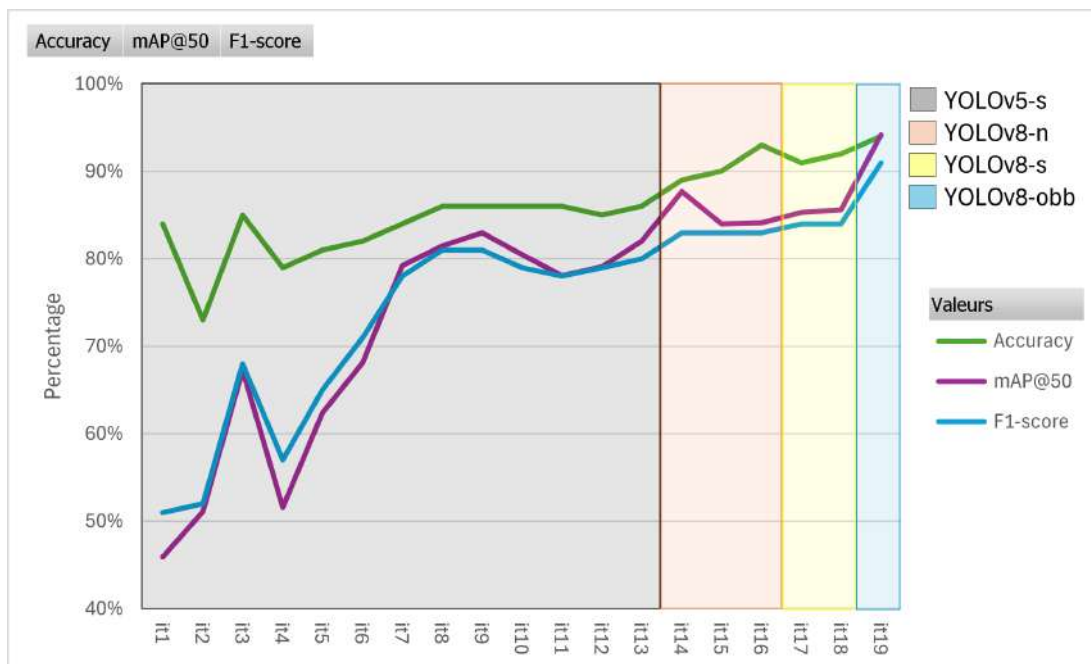


Figure 19: YOLO Model training graph

The object detection modelling training ended with the YOLOv8 - Oriented Bounding Boxes (OBB). Oriented bounding boxes are a newer type of bounding box where the bounding box captures the orientation of the object providing a more accurate fit of the object. They are defined by 5 parameters (center xy, width, height and rotation angle) and capture the object's spatial orientation more precisely, reducing overlap and improving the robustness of the object detection model (Kapoor, 2024).

Below the difference is displayed in bounding boxes between the labelled image used for the non-OBb YOLOv5 & YOLOv8 iterations versus the OBb YOLOv8.



Figure 20: *Non-OBb labels (left) versus OBb labels (right)*

Lastly, the YOLOv8-OBb model was trained on the settings below:

```
train_results_obb = YOLO8s_obb_model.train(data=data_config,
    epochs=35,
    patience=15,
    optimizer="AdamW",      # Adam + weight decay for less overfitting
    val=True, # validate during training
    seed=1,
    imgsz=416,
    batch=16,
    cache="disk",
)
```

Listing 2: *YOLOv8-OBb model training settings*

The weights of the YOLOv8-OBb model were then saved and used to identify parked cars on over 18,000 satellite images spanning Dublin City. These images are then used to identify parking spots.

4.4.2 Parking spot detection

Subsequently, to identify the parking spots in each satellite image, the cars detected by the YOLO model are classified into on the road or parked, based on a road mask.

Many different iterations of the road mask were used, the main iterations and their differences are explained and shown below in Figure 20.

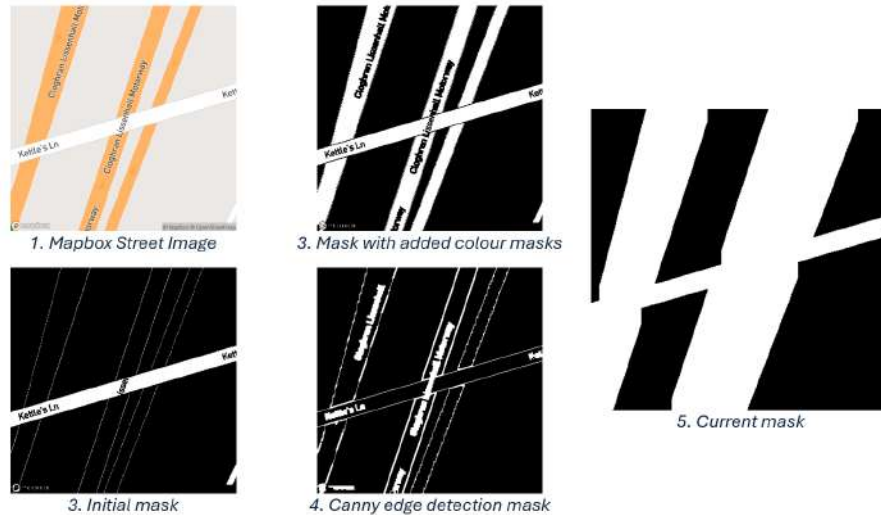


Figure 21: *Different Iterations of the Road Mask*

The original mask was a very simple binary mask which identified the road pixels by their color, as the majority of roads were depicted in white and darkened the rest of the pixels in the image.

Through testing and thoroughly analysing the Mapbox Streets images, motorways and national roads were discovered to be depicted in yellow or orange. Therefore, two additional color masks (for orange and yellow) were concatenated to the original binary mask through bitwise operations.

To refine the mask, and reduce misclassification errors, additional annotations contained on the Mapbox Streets images such as street names and white dotted lines which denote a multitude of things (footpaths, planter boxes, pedestrian crosswalks, football fields delimitations, etc.) were removed through morphological operations, leaving only a plain white line for each road. A number of different kernel sizes, different contour thresholds and other parameters were tested to achieve the optimal removal of the street names and additional annotations.

A different approach using Canny Edge Detection was tested, however due to the nature of the Mapbox Streets images, the edges picked up were not significant and the overall performances was much worse than the current mask at the time.

A final improvement to the mask was made to avoid certain misclassification due to the Mapbox Streets road width not correctly reflecting all the lanes of the road and therefore classifying on the road cars as parked. This issue was most prominent for motorways and national roads and was resolved by enlarging those roads using a dilation technique. Multiple different kernel sizes, different number of iterations and kernels of different sizes applied consecutively, were tested to find the optimal solution.

4.4.3 Parking spot classification

The process of classifying the parking spots has been split into 3 steps:

1. Classify initial spots as *on the road* or parked
2. Identify additional *empty* parking spots
3. Classify all identified spots in 3 *classes*

Step 1 - Classifying into on the road or parked

The cars detected by the YOLO model are then sorted into on the road or parked based on the road mask previously generated, which is explained in more detail below.

The model's predictions which are lower than the confidence threshold of 0.4 are discarded as they are more likely to be misclassifications, such as chimneys or roof pieces which were often misclassified as cars, which can be seen below on the second image of Figure 22.



Figure 22: Classification of cars as on the road or parked

A bounding box for each of the model's predictions is computed from the center pixel coordinates, the width and the height returned from the model. The overlap between the bounding box and the road pixels is calculated, and if the overlap is higher than the threshold of 0.5, the prediction is discarded, keeping only the parked cars. Different thresholds were tested as well, While different thresholds were tested as well, overall, a harsher threshold works better given that these detections are used later on for the empty parking detection.

For each of the model's predictions for parked cars, the center pixels coordinates are mapped to geographic coordinates (longitude and latitude), and the width, height, rotation and orientation based on the angle, are saved. The horizontal orientation is assigned for spots that have a rotational angle in degrees ranging between -45 and 45 or between 135 and 225, while the vertical orientation is assigned to the remaining spots that do not fit the criteria.

For testing and debugging purposes, all the bounding boxes of the cars found by the model are drawn onto the satellite image- on the road cars are drawn in blue while parked cars are drawn in red as seen above in Figure 22.

Step 2: Detecting unoccupied parking spots

Occupied parking spots were identified, therefore it is necessary to identify unoccupied ones to offer a more comprehensive view of car parking in Dublin city.

Unoccupied parking spots are detected between parked cars in each image, based on the parked cars detected by the YOLO model and classified using the road mask.

A number of different variations and calculation techniques were tested out, however only the final version will be explained below.

Average parking spot width and length in pixels are calculated for each image, to draw the empty spots more uniformly. Average parking spot width and length in meters are set to 3.05 as found to be the optimal value through testing, accounting, for the variations in size.

Setting this value avoids misclassifications, as previously the average width and length in meters were calculated dynamically, however in cases where the averages were lesser than 3, spots tended to overlap.

The parked cars detected are then sorted by latitude for horizontally aligned cars and by longitude for vertically aligned cars, to identify gaps between consecutive cars. For each pair of consecutive spots, the distance in meters in between them is calculated. The gap is adjusted to account for the half cars on both sides, as the coordinates of the parked car represent the center of the car. For the gaps smaller than the maximum gap threshold of 12 meters and larger than the average size of a car, where there is enough space to fit 1 or more car, and where the angle deviation is smaller than a threshold of 35 degrees, to ensure the spots are sufficiently aligned, the number of cars that fit into the gap is calculated.

Based on the number of unoccupied spots found, the coordinates for those empty spots are calculated to be aligned with the 2 consecutive cars and then added to a list of empty spots detected.

Afterwards, spots where the distance between them is smaller than a threshold of 1 meter are removed.

Additionally, spots coinciding with cars identified by the YOLO model, where the distance between them is smaller than 1.25 meters are removed from the unoccupied spots found. Spots that overlap with the road are filtered out in a similar manner to the classification done by the road mask.

Following the detection of all the unoccupied spots, the bounding boxes corresponding to each spot are drawn onto the satellite image in **green** as shown in Figure 23.



Figure 23: *Empty Parking Spot Detection*

Step 3: Classification of all spots detected Both the parked cars identified by the model and the unoccupied parking spots are then classified into 3 different classes: *public* (on the street parking), *private* (residential parking) and parking lot, based on their proximity to the nearest road.

Public & private parking spots

Parking spots are classified as *public* if the proximity to the nearest road is less than a pixel threshold of 30, otherwise the parking spots are considered *private*.

This optimal threshold was found through extensive testing on a test set containing various edge cases.

Parking lot spots

Parking spots are classified as parking lots, when clusters of 18 or more spots are identified in an image. Larger clusters are prioritized as parking lots to minimize the risk of misclassifying residential areas as parking lots.

Multiple different clustering approaches were tested out to cluster spots together, such as DBSCAN, HDBSCAN and OPTICS.

Overall DBSCAN performed the best and the quickest out of all the clustering algorithms, given that some images contained very few parking spots, meaning clustering was only significant in cases with a minimum number of parking spots.

Through testing, the optimal hyperparameters found for DBSCAN are `eps : 55` and `min_samples : 5`, which allow the correct identification of the clusters, avoiding misclassifications of residential areas with many cars as parking lots, which was a common misclassification initially, when the hyperparameters and thresholds were not set correctly.

`eps` refers to the maximum distance between two points for them to be considered as neighbors and `min_samples` refers to the minimum number of points required to form a cluster.

Using HDBSCAN and OPTICS required a minimum number of samples larger or equal to 2. However, not all images contained at least 2 parked cars therefore clustering using those algorithms was not possible in all cases. A value of -1 was assigned to the spots belonging to no clusters similarly to the default behaviour of DBSCAN.

Clustering using HDBSCAN gave slightly lower results to DBSCAN, while OPTICS only found smaller clusters which were not as significant and not particularly helpful to classify the spots into the parking lot category.

Afterwards, clusters and classification labels are drawn onto the satellite image. The clusters are color coded and denoted as a circle at the center of the bounding box, while the classification label is written to the right of the bounding box. The private class is written in red, public in green and parking lot in white as seen below in Figure 24.



Figure 24: Classification of all spots into private, public and parking lot

4.4.4 Evaluation of each submodule of the parking detection model

Each major section of the parking detection model was evaluated individually in order to assess the overall performance. The results will be presented for each section below.

Evaluation of the YOLOv8-OBB model

The YOLOv8-OBB model was evaluated on a validation set of 51 images, where precision, recall, accuracy, mean average precision (mAP@50) and F1-score were computed. These were the scores below:

Model	Accuracy	Precision	Recall	F1-score	mAP@50	Conf threshold
YOLOv8-OBB	94%	100%	99%	91%	94%	35%

Table 1: YOLOv8-OBB model results

The model obtains near perfect precision and recall on the validation dataset, meaning it is very good at correctly identifying the majority of true positives and preventing false positives. However, the lower f1 score of 91% highlights the slight imbalance between precision and recall. Additionally, the confidence threshold is based on the F1-score, suggesting that the model performs at a level where precision and recall are balanced best closer to 30%. This indicates that our model is not strict.

Although the overall metrics obtained are quite high, it is important to consider that, the validation and training data are not very large, therefore impacting overall model performance to identify cars in a larger detection set.

Evaluation of the classification of cars into on the road or parked

The classification of cars into on the road or parked by the road mask is evaluated in the `evaluate_road_mask.py` Python script. The classification is evaluated on the following metrics commonly used for object detection tasks: Average Intersection over Union (IoU), Balanced Accuracy and Precision, Recall, F1 Score, Accuracy, Specificity per class.

These metrics are defined in detail as follows.

The IoU is calculated as the overlap between the predicted bounding box B_p and the ground truth bounding box B_g , divided by their union:

$$\text{IoU} = \frac{|B_p \cap B_g|}{|B_p \cup B_g|}$$

The Average IoU is defined as the mean IoU across all detections:

$$\text{Average IoU} = \frac{1}{N} \sum_{i=1}^N \text{IoU}_i$$

where N is the total number of detections.

Precision is defined as the ratio of true positives (TP) and the sum of true positives and false positives (FP):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall is defined as the ratio of true positives (TP) and the sum of true positives and false negatives (FN):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The F1 Score is defined as the harmonic mean of precision and recall:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy is defined as the proportion of correct predictions (TP and TN) out of all the predictions:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Balanced Accuracy is defined as the average of Sensitivity (or Recall) and Specificity:

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

where:

$$\text{Sensitivity/Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

These metrics are then saved in a csv file for each test image as well as the overall average metrics on the entire test set.

A test set of 50 images was carefully selected to include edge cases and difficult cases and then consistently labelled in Label Studio as seen in Figure 25. The labels are drawn without rotation to ensure the maximum overlap, as the labels are exported to txt format and include only the pixel coordinates, width and height of the bounding boxes annotated.

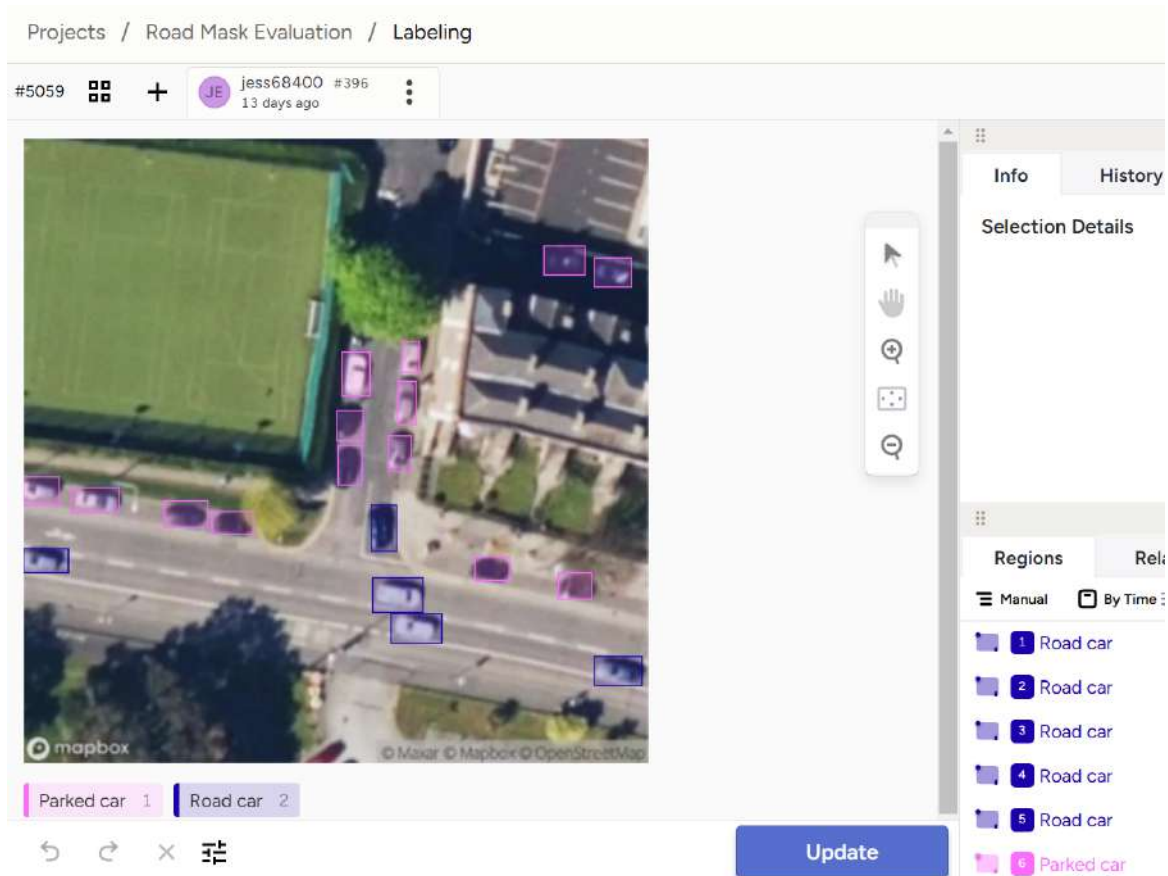


Figure 25: Road mask classification test set images being annotated in Label Studio

The overall results on test set are presented below in Table 2.

Metric	Parked	Road
Average IoU	0.66	0.66
Average Balanced Accuracy	0.58	0.58
Average Precision	0.70	0.90
Average Recall	0.75	0.67
Average F1 Score	0.59	0.54
Average Accuracy	0.69	0.64
Average Specificity	0.74	0.79

Table 2: Performance Metrics for Road Mask Classification

Precision and Recall are relatively high for both classes. Precision for the on the road cars class is extremely high as the majority of the cars in the test set were perfectly horizontally or vertically aligned, and given that the rotation cannot be exported in label studio, the bounding box is drawn less accurately unless if it is perfectly horizontally or vertically aligned.

Similar papers that explore object detection for cars on satellite images evaluate their experiment on more general metrics such as *Average Precision (AP)* and *Mean Average Precision (mAP)*.

AP represents the area under the precision-recall curve and mAP represents the mean of the APs across all the different classes. These metrics provide a broader overview of the model's performance as it is measured across varying thresholds.

The results obtained on our model are not directly comparable, given that precision, recall and f1 score are calculated for a fixed IoU threshold in our evaluation.

Nevertheless, they are in line with Zambanini et al., 2020, who achieve a precision of 0.75, a recall of 0.66 and an f1 score of 0.70 at the IoU threshold of 0.30, for detecting parked car on stereo satellite images.

The current road mask has a few outstanding problems, that are discussed further below.

Firstly, not all street names are completely removed due certain factors such as the Mapbox watermark and the length of the street name, the latter causing identified cars on the road to be misclassified as parked.

Secondly, the Mapbox Streets images used to create the road mask do not correctly reflect the width of the road for roads with multiple lanes, which leads to certain cars on the road being misclassified as parked. This issue was resolved for motorways as they are depicted in orange or yellow, making it possible to distort the mask to cover more surface.

Another problem inherent to Mapbox Streets images is the Dublin Tunnel being marked as above ground even though it is underground, leading to certain misclassifications in that specific area.

Some images from the test set can be seen below in Figure 26. The predicted parked cars are drawn in red while the true labels are drawn in light pink. The predicted on the road cars are drawn in blue while the true labels are drawn in cyan.



Figure 26: Images from the Road Mask Classification Test Set

Evaluation of the empty parking detection

The empty parking detection logic is evaluated in the `evaluate_empty_parking_detection.py` Python script. The classification is evaluated on the following metrics: Average IoU, Average Precision, Average Recall, Average F1 Score, Average Orientation Accuracy, Average Spot Detection Ratio (SDR), Average Spot Detection Error (SDE), Average False Positive Rate (FPR), and Average False Negative Rate (FNR).

The additional metrics are defined as follows.

The Average Orientation Accuracy is defined as the ratio of the number of predictions that correctly identify the orientation (either horizontal or vertical) of the spots on the total number of true spots:

$$\text{Average Orientation Accuracy} = \frac{\text{Correct Orientation Predictions}}{\text{Total True Labels}} = \frac{N_{\text{correct_orientation}}}{N_{\text{true_labels}}}$$

where $N_{\text{correct_orientation}}$ represents the number of predictions with correct orientation, and $N_{\text{true_labels}}$ represents the total number of true labels.

The Spot Detection Ratio is defined as the ratio of the number of predictions and the total number of true labels:

$$\text{SDR} = \frac{N_{\text{predictions}}}{N_{\text{total_true_labels}}}$$

where $N_{\text{predictions}}$ is the total number of detected spots, and $N_{\text{total_true_labels}}$ is the total number of true labels.

The Spot Detection Error is defined as the absolute difference between the number of predictions and the total number of true labels:

$$\text{SDE} = |N_{\text{predictions}} - N_{\text{total_true_labels}}|$$

The False Positive Rate is defined as the proportion of incorrectly predicted positives (FP) out of all the actual negatives:

$$\text{Average FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

The False Negative Rate is defined as the proportion of incorrectly predicted negatives (FN) out of all the actual positives:

$$\text{Average FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}$$

These metrics are then saved in a csv file for each test image as well as the overall average metrics on the entire test set.

Likewise, a test set of 50 images was carefully selected and consistently labelled in Label Studio, without rotation to ensure the maximum overlap. The pixel coordinates, width and height of the annotated bounding boxes are exported. The orientation of each bounding box, is calculated automatically in the evaluation script when loading in the true labels.

Achieving a high IoU was a significant challenge, specifically in this section of the model, as the positioning of the spots can be difficult to label accurately. Indeed, it was difficult to identify if there was sufficient space available to fit a car. To remedy this issue, the IoU threshold was lowered to 0.35, as there was still significant overlap visible when analysing the test images.

The problems linked to the overlap, is equally due to the differences in orientation as the true labels and the predictions are calculated differently. For the true labels, the orientation is based on whether the width or length of the spot is larger (the spots with larger width are labelled as horizontal and the spots with larger length are labelled as vertical), while for the model's predictions the orientation is based on the angle measurement.

The overall results on test set are presented below in Table 3.

Metric	Value
Average IoU	0.59
Average Precision	0.77
Average Recall	0.70
Average F1 Score	0.69
Average Orientation Accuracy	0.64
Average Spot Detection Ratio (SDR)	1.05
Average Spot Detection Error (SDE)	1.44
Average False Positive Rate (FPR)	0.23
Average False Negative Rate (FNR)	0.22

Table 3: Performance Metrics for Empty Parking Detection

The Average Orientation Accuracy being somewhat low of only 0.64 is expected because the orientation labels for the test set are calculated automatically based on the width and length of the bounding box. On the other hand, the orientation associated with the model's prediction is based on the rotation. This is illustrated below, the orientated label in **pink** and the non-oriented one in **green**.

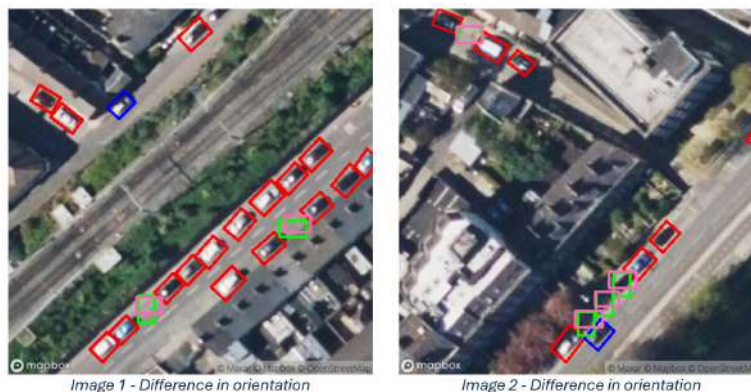


Figure 27: Empty detection test set - True label vs Predicted label

The results achieved are similar to other articles performing car detection on satellite imagery such as Zambanini et al., 2020.

The main reason unoccupied parking spots are not detected is because cars are classified as on the road by the road mask and the empty parking detection logic only applies to parked cars. This is illustrated in the figure below.

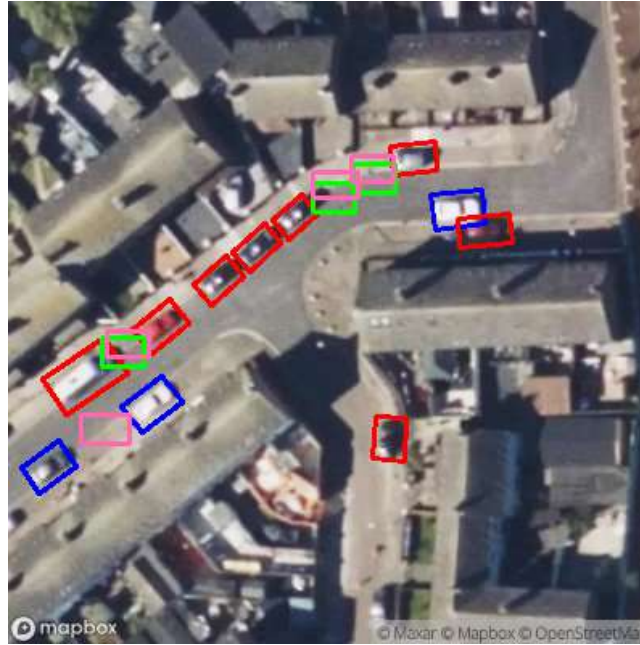


Figure 28: Empty detection test set - True label vs Predicted label

Evaluation of classification of spots into private, public & parking lot The classification of the spots detected into private, public and parking lot is evaluated in the *evaluate_classification_of_spots.py* Python script. The classification is evaluated on the following metrics as defined previously: Average IoU, Balanced Accuracy and Precision, Recall, F1 Score, Accuracy, Specificity per class. These metrics are saved in a csv file for each test image as well as the overall average metrics on the entire test set.

Similarly, a test set of 50 images was carefully selected and consistently labelled in Label Studio, without rotation to ensure the maximum overlap.

The overall results on the test set are presented below in Table 4.

Metric	Public	Private	Parking Lot
Average IoU	0.60	0.60	0.60
Average Balanced Accuracy	0.57	0.57	0.57
Average Precision	0.73	0.65	0.59
Average Recall	0.72	0.75	0.74
Average F1 Score	0.68	0.62	0.61
Average Accuracy	0.67	0.54	0.63
Average Specificity	0.59	0.54	0.41

Table 4: Performance Metrics for Parking Spot Classification

The performance is similar for all 3 classes, however for the parking lot class more false positives are present, given that having many bounding boxes clustered together creates more unpredictability adding more additional spots.

The results presented above are in line with other articles performing car detection on satellite imagery such as Zambanini et al., 2020.

The main outstanding problem for this section is residential areas, which are very rarely misclassified as parking lots. Additionally, in cases where public spots are quite far from the nearest road, the spots are misclassified as private, which is not a major issue.

Another point worth noting is that smaller parking lots with less than 18 spots are not detected though that is intentional and by design in order to focus on larger parking lots and avoid misclassifications.

A few images from the test set are shown below in Figure 29. The predicted public spots are drawn in green while the true labels are in dark green. The predicted private spots are highlighted in red while the true labels are in light pink. The predicted parking lot spots are shown in blue while the true labels are in cyan.



Figure 29: *Classification of spots test set - True label vs Predicted label*

4.4.5 Integration with Magpie system

Once all the parking spots have been identified, their longitude, latitude and class are saved to a csv file. Potential spot duplicates are dropped, and parking zones and associated tariffs are added.

Parking zone and tariff were obtained by manipulating the map shown in the figure below in QGIS, a geographic information system used for working with spatial data. Polygons were drawn on a raster layer, and associated id, zone name and tariff were defined before being exported as a geojson.

Dublin City Council defines different parking zones by color, based on their high to moderate demand which changes the price per zone as seen below in Figure 30.

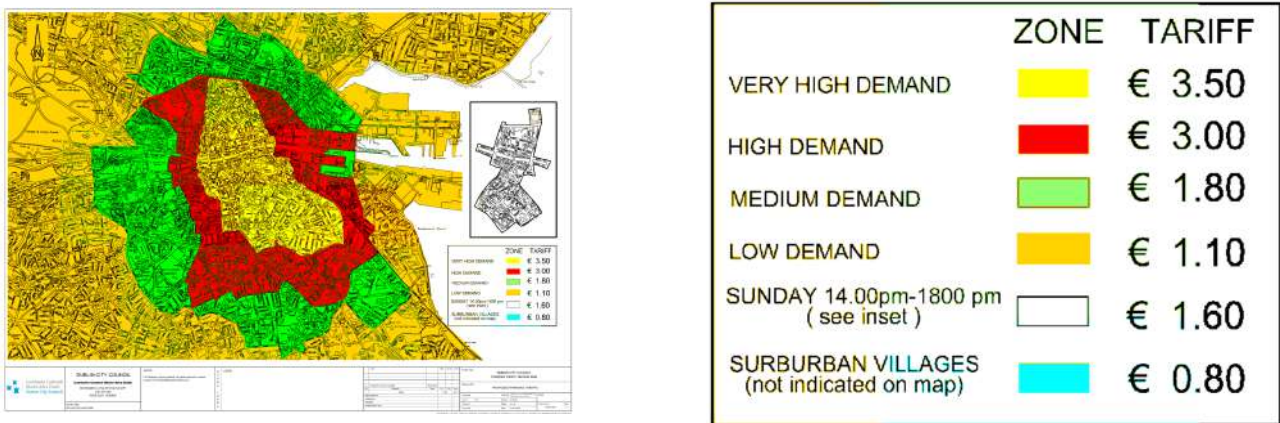


Figure 30: Parking zones and cost defined by Dublin City Council

Lastly, the csv file containing the longitude, latitude, classification label, parking zone and parking cost is then uploaded to the PostgreSQL database, in the backend server using the `send_points.py` script.

4.4.6 Summary of main challenges

Throughout the implementation of our technical solution, there have been many major challenges and changes throughout the project timeline.

Week 1-4

This iteration focused on the startup of the project, exploratory work, machine learning model tryouts and setting up Magpie's environment.

The biggest initial hurdle was finding a labelling software to annotate our satellite images. Initially, Label Studio, a popular labelling software with their integrated ML backend extension was chosen, to use YOLO v5 to automatically label our training images, after only annotating a few images. However, after much troubleshooting, the approach was switched to manually labelling the images and subsequently training the YOLO model on those images.

The next challenge concerned to training the YOLO model to identify cars on the satellite images. An object detection model remains a deep neural network model, which is computationally intensive the more complex it becomes.

Due to limitations in available computational resources, training of the model was difficult. Tuning for 35 iterations would last several hours, and training above a certain number of epochs would not yield better results. Training the model was both computationally expensive and very time consuming.

A further major challenge referred to the low resolution of the satellite images, heavily contributing to the high misclassification rate in the early beginnings of model training. This was stabilised through proper model choosing and tuning however, the biggest factor contributing to the remaining false positive rate is the low number of training data.

Week 5-8

Another major challenge, was the unoccupied parking detection section. The key decisions and changes are recounted below.

In Week 8, the `parking_detection.py` script was updated to use the most up to date trained YOLO model, YOLO v8 obb, that returned oriented bounding boxes. Furthermore, given that change, the size of the bounding boxes became more variable, which allowed the script to be adapted to use average parking spot dimensions.

Week 9-12

Week 9 was decisive in implementing a double pass-through to sort the cars by longitude and the latitude to identify both the horizontal and vertical spots. The `parking_detection.py` script was updated to use the `xywhr`(centre coordinates, width, height and rotation) bounding boxes instead of the `xyxy` (top left, bottom right) bounding boxes initially used. This major change was crucial, adding more modularity and allowing the separation into horizontal and vertical cases in a more consistent manner. Many additional improvement were made that week, such as the removal of overlapping spots and filtering out the empty spots detected on the road.

In week 12, the unoccupied parking detection was completely finalized, resolving the majority of the outstanding issues. Originally 4 cases were considered, cars parked horizontally in a row or in a column, cars parked vertically in a column and cars parked vertically side by side. However, horizontally parked in a column and vertically parked side by side were removed as they led to too many misclassifications.

A final critical issue was the selection and accurate labelling of the test sets to evaluate each subpart of our model. Week 11 marked the beginning of the unoccupied parking detection evaluation. Labelling the test set accurately was a significant challenge, multiple different phases of relabelling and improvements occurred, to ensure the maximum overlap between the true labels and the model's predictions. During Week 12, the two other evaluation scripts were finalized. Similarly, multiple phases of labelling and relabelling occurred. For the classification of spots evaluation, additional images containing private class instances were added to the test set, as the initial lack of enough instances was making that class perform poorly in comparison to the other classes.

4.5 Frontend

4.5.1 Frontend Architecture and Implementation

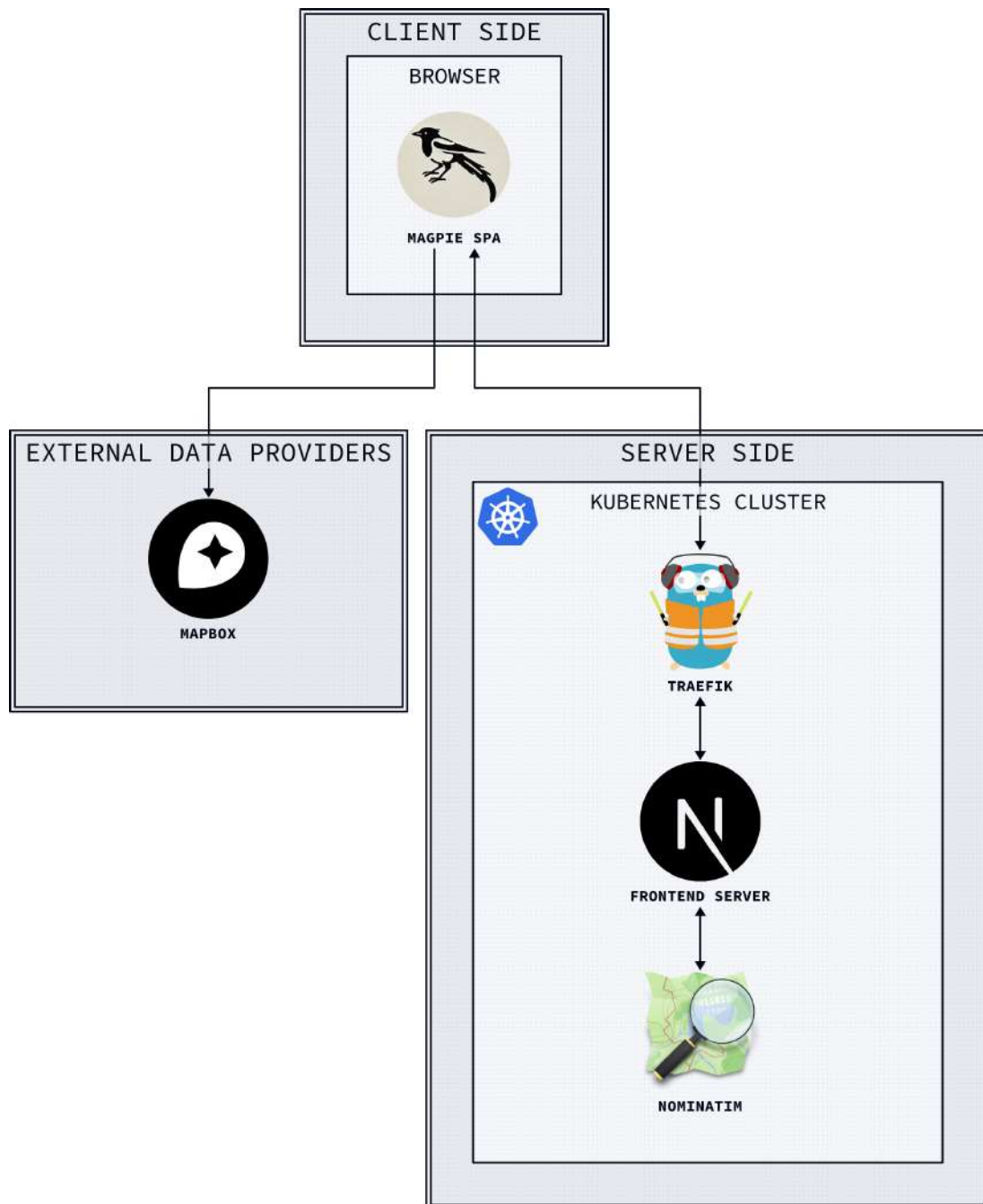


Figure 31: Frontend Architecture Diagram

Introduction

In the development of this project, we adopted a modern web development stack that incorporated a combination of cutting-edge technologies aimed at optimizing performance, scalability, and maintainability. The integration of these technologies facilitated the creation of a highly efficient and user-friendly web application. This section provides a comprehensive overview of the key technologies used in the frontend of the application, outlining their individual roles and how they collectively contributed to the success of the project.

React

React is a widely used and powerful JavaScript/TypeScript library that is primarily designed for building user interfaces. It was chosen for this project due to its ability to facilitate the development of dynamic and responsive web applications. In conjunction with **Next.js**, React allowed for the implementation of a component-driven architecture. This architectural approach enables the reuse of individual UI components throughout the application, which not only streamlined the development process but also ensured greater maintainability in the long term. By breaking down the user interface into smaller, reusable components, we were able to manage and scale the application more effectively. React's virtual DOM mechanism also ensured efficient rendering of changes to the UI, contributing to an enhanced user experience by minimizing the amount of time required to update the interface.

Moreover, React's active community and vast ecosystem of libraries and tools made it easier to integrate other technologies and features into the application, further improving its overall functionality and performance. As a result, React played a crucial role in enabling a seamless, interactive experience for users, while also enhancing the maintainability and scalability of the codebase.

Next.js

Next.js is a robust React framework that extends the capabilities of React by offering features such as server-side rendering (SSR) and static site generation (SSG). These features proved to be essential for optimizing the application's load times, which was critical to the project's success. Server-side rendering allows the server to generate the HTML for the page on each request, as opposed to client-side rendering where the browser must first download and execute JavaScript to generate the page. This process significantly reduces the time required for the page to be displayed to the user, resulting in a faster and more efficient application.

Additionally, Next.js' static site generation capabilities allowed us to pre-render pages at build time, meaning that the application could serve static HTML files for commonly accessed pages, further enhancing performance. The combination of SSR and SSG enabled a smooth and responsive user experience.

Next.js also provided a robust routing system that made it easier to manage the application's navigation. The framework's file-based routing system allowed for simple, declarative routing, which contributed to the overall maintainability of the project. Next.js' out-of-the-box support for features like code splitting and automatic optimization ensured that the application remained highly performant, even as it grew in complexity.

TailwindCSS

For styling the application, **TailwindCSS**, a utility-first CSS framework, was chosen for its ability to enable rapid UI development. Unlike traditional CSS frameworks, which rely on pre-defined UI components and styles, TailwindCSS provides low-level utility classes that can be combined to build custom designs. This approach allowed for greater flexibility in styling the application, as developers were able to apply precise and granular control over the UI without the need for writing custom CSS from scratch.

One of the key advantages of using TailwindCSS was the speed with which we could implement and modify styles. By utilizing utility classes, we were able to quickly experiment with different design configurations and make adjustments on the fly. This approach also minimized the need for additional CSS files, reducing the overall complexity of the styling process.

In addition to TailwindCSS, we incorporated several tools to enhance the framework's capabilities. **Tailwind-merge** was used to intelligently combine conflicting utility classes, ensuring that styles were applied consistently throughout the application. **TailwindCSS-animate** provided utility classes for adding animations, which helped bring dynamic visual elements to life without needing to write custom animation logic. These tools, when used in tandem, allowed for more advanced styling and complex animations while maintaining the simplicity and ease of use that TailwindCSS does well.

Mapbox GL and React Map GL

The integration of interactive maps and advanced visualization features played a key role in the functionality of the project. For this purpose, we utilized **Mapbox GL** and **React Map GL**. **Mapbox GL** is a powerful mapping library that provides advanced capabilities for rendering interactive maps with smooth transitions and animations. It is built to handle large datasets, enabling the display of dynamic geospatial information with high performance and precision. Mapbox GL was used to power the core mapping functionality of the application, allowing users to interact with maps seamlessly.

To integrate Mapbox GL with our React application, we used **React Map GL**, a library that provides a simple wrapper around Mapbox GL, making it easier to embed and control interactive maps within React components. React Map GL allowed us to leverage the full potential of Mapbox GL while maintaining a consistent and efficient React-based architecture.

In addition to basic mapping functionality, we integrated **Deck.gl** for advanced data visualization. Deck.gl is a framework that enhances Mapbox GL by enabling the display of complex data visualizations on top of interactive maps. This was particularly beneficial for our project, as it allowed us to visualize large datasets in an efficient and interactive manner, providing users with a rich and engaging experience.

ShadCN/UI

To streamline the development of UI components, we incorporated **ShadCN**, a component library built on top of **Radix Primitives**. ShadCN combines the flexibility of Radix UI with a more opinionated design system, tailored for projects that require custom styling with the power of TailwindCSS. Radix Primitives offers a set of low-level UI components that are highly customizable, accessible, and composable, making them ideal for building complex user interfaces.

By using ShadCN, we were able to access a suite of pre-built, customizable components that adhered to accessibility standards, ensuring that our application was usable by all users, including those with disabilities. ShadCN's integration with Radix Primitives allowed us to quickly implement accessible and customizable components, reducing the need for additional development and ensuring consistency across the application.

The combination of ShadCN's flexible components and TailwindCSS allowed us to build a visually appealing and highly functional user interface with minimal configuration. Furthermore, ShadCN's tight integration with accessibility guidelines ensured that the application met the required standards for usability, providing an inclusive experience for all users.

Radix Primitives

Radix Primitives played a crucial role in the design and development of our user interface components. Radix Primitives provides a set of foundational, high-quality UI components that serve as the core building blocks for any application. These components are unstyled and highly customizable, giving developers full control over the look and behaviour of the elements while ensuring accessibility and functionality out of the box.

The core of Radix Primitives includes essential UI elements such as buttons, modals, popovers, sliders, tabs, and other interactive components. These primitives are designed to be flexible and composable, allowing developers to combine them in various ways to create complex UI patterns. The components are built to be accessible, ensuring that the user interface remains usable for all users, including those with disabilities, without requiring additional development effort to meet accessibility standards.

One of the standout features of Radix Primitives is its focus on accessibility. The components are designed with built-in features like keyboard navigation, screen reader support, and proper focus management. This allows the application to be fully functional and accessible across a wide range of devices and assistive technologies, without the need for custom implementation of these features. As accessibility is a fundamental aspect of modern web development, Radix Primitives ensured that the project adhered to the highest standards for inclusive design.

In addition to accessibility, Radix Primitives allows for extensive customization. While the components come with sensible defaults, they are unstyled, which provides the flexibility to apply custom styles that match the overall design language of the application. This level of customization allowed the development team to ensure that the UI components blended seamlessly with the rest of the application's design. By combining Radix Primitives with **TailwindCSS**, we could easily customize the components using utility-first CSS classes, speeding up the styling process and ensuring consistency across the entire application.

The composability of Radix Primitives also played a significant role in the project's development. The library is designed to allow developers to create complex user interface patterns by combining simple components in flexible ways. For instance, we could easily integrate modal

dialogs, dropdown menus, and tooltips to form cohesive UI patterns that enhanced the user experience. This approach not only saved development time but also reduced the likelihood of introducing bugs, as we could rely on well-tested, standardized components rather than building these features from scratch.

By leveraging Radix Primitives, we were able to focus on creating a polished and accessible user interface while avoiding the complexity of building low-level components. The ability to use these pre-built, flexible primitives allowed us to deliver a high-quality user experience with minimal overhead. Radix Primitives was an essential tool in ensuring that the application was both user-friendly and accessible, all while maintaining the flexibility to customize and extend the components as needed for the project.

The integration of these technologies into the frontend of the project provided a solid foundation for building a high-performance, scalable, and user-friendly web application. From the server-side rendering capabilities of Next.js to the highly customizable components offered by ShadCN, each technology played a critical role in the overall success of the project. The use of React and Next.js allowed for a modular and efficient development process, while TailwindCSS and its associated tools streamlined the styling process and enabled rapid UI development. The incorporation of Mapbox GL and Deck.gl brought advanced mapping and data visualization capabilities to the application, making it more interactive and engaging for users. Together, these technologies enabled us to deliver an application that met high standards of code quality, user experience, and performance, ensuring its success in achieving the project's objectives.

4.5.2 Client State Management

Introduction

Effective client state management is a crucial aspect of the **Magpie** frontend, as it ensures a seamless and consistent user experience. Given the dynamic nature of the application, with interactive elements such as maps, search filters, and property details, it is essential to maintain synchronization between different UI components while preventing errors and inconsistencies. Managing state efficiently is particularly important when handling large datasets that need to be presented in a user-friendly manner across various parts of the application.

React Context

To achieve this, the application employs state management techniques that allow for centralized control of the application's state. One of the core tools used for this purpose is **React Context**, which provides a simple way to share state across multiple components without having to pass data manually through props. React Context enables the application to manage the user's preferences, search filters, and saved properties in a global state, ensuring that these settings are accessible across various parts of the application. This method of state management is particularly useful for dynamic features, such as filtering search results based on user input or retaining user selections when navigating between different pages or sections of the platform.

Redux

In addition to React Context, the use of more advanced state management libraries such as Redux can be considered for larger-scale applications that require more complex state handling.

Redux facilitates a unidirectional data flow and provides a centralized store that allows the state to be updated in a predictable manner. With Redux, the application can ensure that all components are working with the most up-to-date information, whether that involves real-time data from maps or user-selected property preferences. Redux also allows for more complex actions, such as asynchronous data fetching, and can be used in conjunction with middleware like Redux Thunk for handling side effects, further enhancing the application's performance and stability.

Consistent User Experience

Client state management also ensures that the data displayed to the user remains consistent. For instance, when a user interacts with a map, the system must track the map's current state and the filter options that the user has selected. By maintaining this state across different components, users can switch between different views (such as property details and map views) without losing their previous selections. Additionally, state management ensures that updates to the user interface, such as property filtering or adding a property to the favourites list, are reflected across the entire application in real-time, maintaining data integrity and providing users with a smooth browsing experience.

Conclusion

Overall, client state management plays a vital role in improving the usability and responsiveness of the frontend, reducing redundancy and complexity by centralizing control over the application's data. This centralized management allows for easier maintenance and scalability as the application evolves and new features are added.

4.5.3 UI Walkthrough

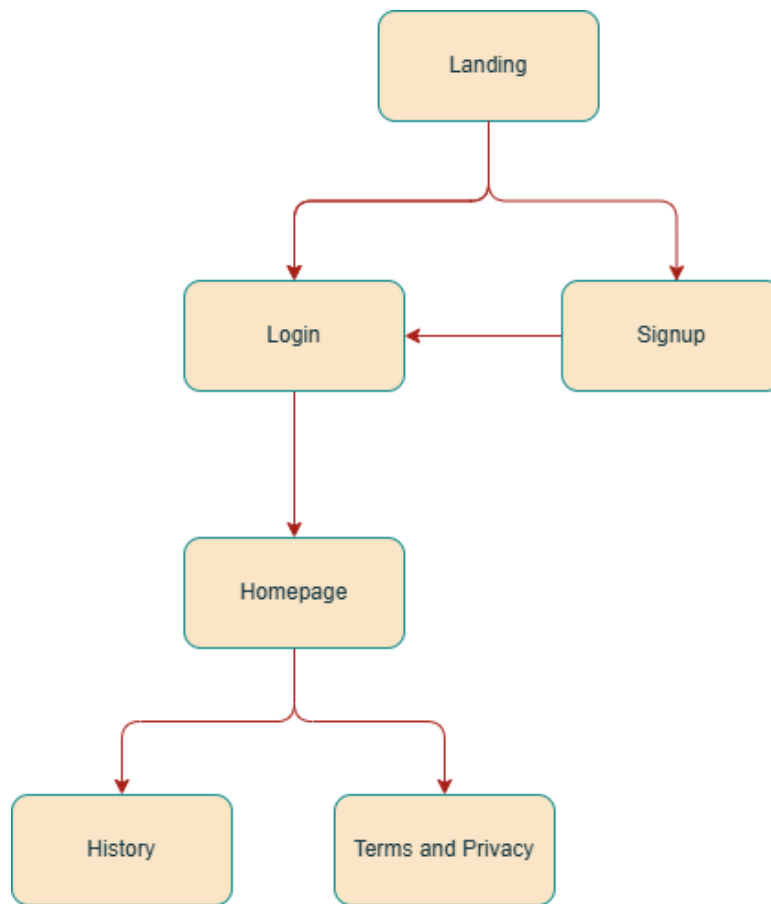


Figure 32: *Site map - magpie*

Landing Page Introduction

Upon accessing the landing page, users are greeted with a clean and modern layout that introduces them to 'Magpie' - a geographical information service providing a glance at available public services in Dublin. The layout is designed to be visually appealing, with prominent calls to action and easily navigable sections.

Header Section



Figure 33: Landing page - Header

- **Navigation Bar:** At the top of the page, the navigation bar features the **Magpie logo** on the left side, offering a simple and recognizable brand identity. To the right, navigation links for **About**, **Features**, **Use Cases**, and **Get Started** provide quick access to key sections. Additionally, a **Sign Up** button stands out in black with white text for clear calls to action.
- **Mobile Menu:** For mobile views, a hamburger menu (three horizontal lines) appears, ensuring that the navigation remains compact and accessible on smaller screens.

Hero Section

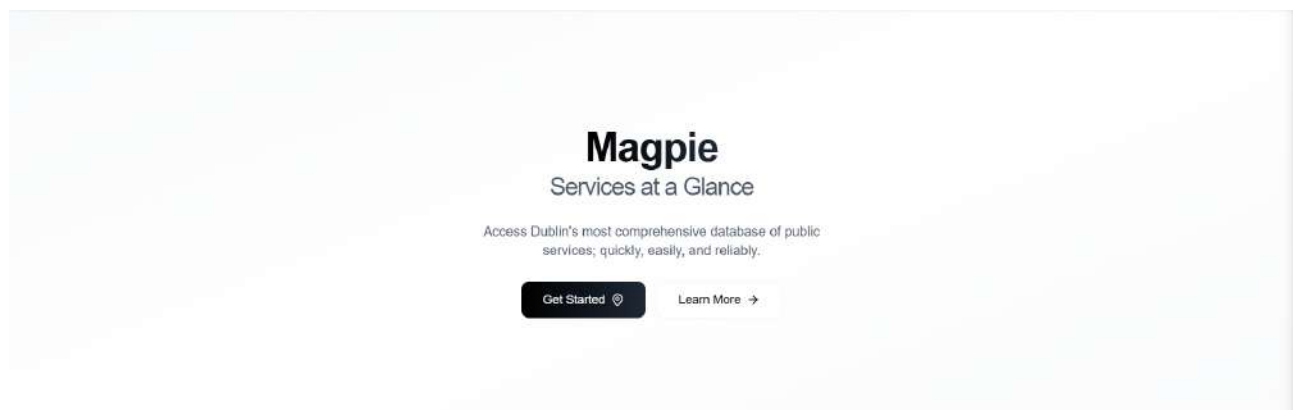


Figure 34: Landing page - Hero section

- **Headline:** The page features a bold, eye-catching headline that reads 'Magpie' in a gradient text style, followed by the tagline 'Services at a Glance.' This is a clear and engaging introduction to the site's purpose.
- **Subheading:** A short description reinforces the value proposition: 'Access Dublin's most comprehensive database of public services; quickly, easily, and reliably.' This emphasizes the platform's utility and target audience.
- **Primary CTA:** A large button titled 'Get Started' invites users to sign up or log in, accompanied by an icon of a map pin to add a visual cue to the call to action.
- **Secondary CTA:** A 'Learn More' button encourages users to explore more details about the platform. It's designed to stand out but with a lighter visual treatment compared to the primary CTA.

About Section

What is Magpie?

Magpie is a cutting-edge geographical information service designed to provide users with a comprehensive view of public services and amenities at a glance. It empowers urban planners, residents, and other stakeholders by offering interactive maps, smart search capabilities, and detailed analysis tools. Whether you're planning infrastructure, exploring local services, or seeking information about your daily routes, Magpie simplifies access to essential data with reliability and precision.

How does Magpie use Machine Learning?

Our project uses machine learning to analyse satellite images and accurately identify parking spaces. Our approach relies on the YOLOv8 OBB Model (You Only Look Once with Oriented Bounding Boxes), a state-of-the-art deep learning model optimized for object detection.

Car Detection using the YOLO v8 OBB Model:

- We fine-tuned the model to detect cars in satellite images.
- The model was trained on our dataset of manually annotated satellite images, allowing it to handle challenges commonly found in satellite images such as low resolutions, shading, and occlusions.
- Once trained, the model outputs precise bounding boxes that capture the position, size, and orientation of cars. These bounding boxes are then converted to geographic coordinates.

Identifying Parking Spaces:

- We apply multiple heuristics to detect parking spaces using the car detected by the model. Key steps include:
 - A road mask is applied to filter out cars on the road, isolating parking lots and on the street parking.
 - Empty parking spaces are identified by analysing gaps between parked cars, considering alignment, orientation, and standard parking dimensions.

Figure 35: *Landing page - About section*

What is Magpie?: This section introduces **Magpie** as a **geographical information service**, explaining its functionality and target users. It emphasizes its usefulness for urban planners, residents, and other stakeholders by providing interactive maps, smart search capabilities, and detailed analysis tools.

How Magpie Uses Machine Learning: This section explains the use of **machine learning** to analyze satellite images and detect parking spaces. It provides insight into the model used (YOLOv8 OBB), detailing how it detects cars and identifies available parking spaces. A breakdown of the process offers technical depth while remaining accessible to a general audience.

Features Section



Figure 36: Landing page - Features section

Unique Features - This section highlights the core capabilities of Magpie:

- **Interactive Map Visualization:** The ability to search, filter, and visualize services through an interactive map.
- **Smart Search & Analysis:** Provides detailed insights with multi-layer data visualizations.
- **Professional Tools:** Features designed to export reports and analyze historical trends.

Each feature is represented by a card that contains an icon, a title, and a brief description. Hovering over these cards gives users more details, encouraging them to explore further.

Use Case section



Figure 37: Landing page - Use Case section

Magpie Works for Everyone: This section presents specific use cases for different user types:

- **Urban Planners & City Authorities:** Features such as optimizing infrastructure, analysing service distribution, and generating reports.
- **Non-Expert Users & Residents:** Services to find nearby amenities, access transport options, and plan efficient routes.

Call To Action (CTA) Section



Figure 38: Landing page - CTA section

- **Ready to Transform How You Plan?:** The page concludes with a strong call to action, urging users to **sign up** and start using Magpie. This section reiterates the platform's value in making data-driven decisions, featuring a large, attention-grabbing button labelled 'Sign Up Now.'
- **Gradient Background:** The footer section features a subtle gradient background, reinforcing the design's sleek and modern aesthetic.

The landing page for **Magpie** is designed to effectively introduce the platform's features, functionality, and benefits through a series of engaging sections. The page focuses on presenting Magpie as a user-friendly tool for urban planners and residents, with clear calls to action and visually appealing design elements. It balances technical information with an approachable layout, ensuring that both potential users and technical stakeholders can easily understand its offerings.

Signup Page

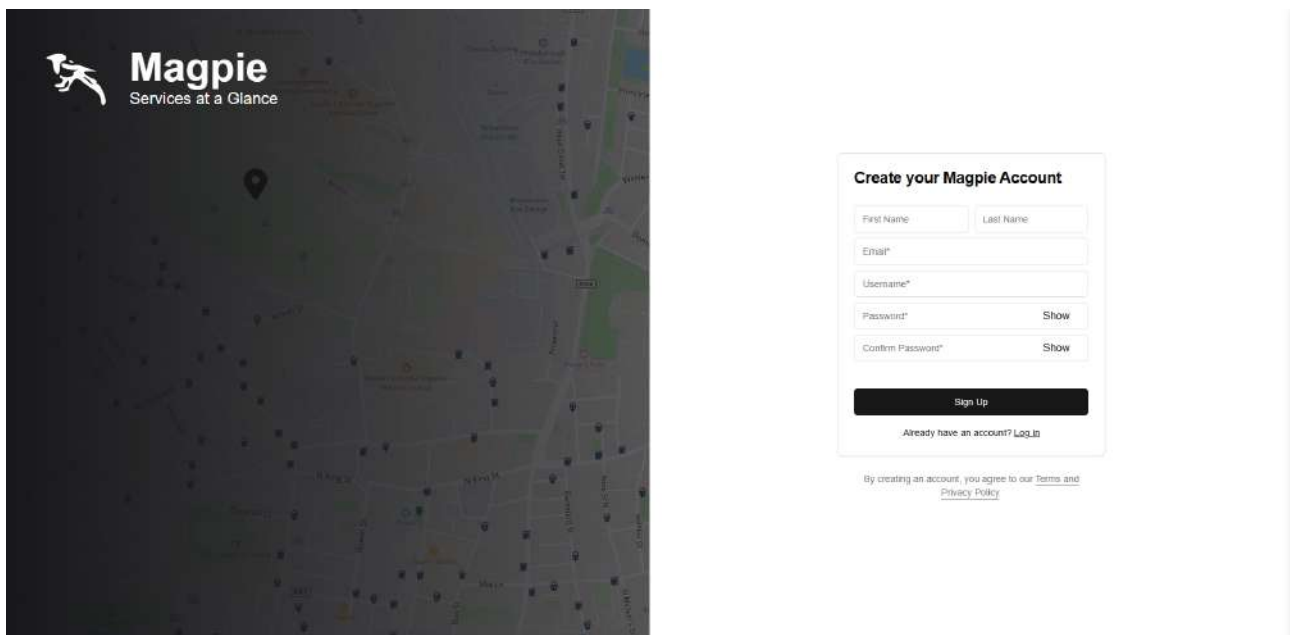


Figure 39: The Signup Page

The **Signup** page serves as the gateway for users to create an account on Magpie. It is designed with a straightforward layout to guide users through the process of entering their personal information securely.

Header Section

The header section plays a crucial role in establishing the identity of the page.

- **Magpie Logo:** Positioned prominently at the top of the page is the **Magpie logo**, providing immediate brand recognition. The logo's size and placement are intentional to make it easy for users to identify the platform and navigate back to the homepage, should they need to.
- **Tagline:** Below the logo, the tagline '**Services at a Glance**' emphasizes the platform's purpose, providing a succinct description of its functionality-offering users quick access to available public services in Dublin.
- **Call to Action:** The headline in the header invites users to **Create your Magpie Account** in large, bold text, making the goal of the page clear and guiding users toward account creation. This is an important call to action that users are encouraged to follow.

Form Inputs

The heart of the Signup page is the **form inputs**, where users input their details to create an account.

- **Required Fields:**
 - **First Name** and **Last Name** fields ensure that the platform has accurate personal information about the user. These fields are essential for building a profile that can be used for various personalization features across the Magpie platform.
 - **Email Address** is also required, which is important for account verification, password recovery, and communication with the platform. Users are reminded that this email will be used for correspondence related to their account.
 - **Username** allows the user to choose a unique identifier for logging into the platform. This is an important step in personalizing the account, as the username will be used alongside the password to access the Magpie dashboard.
- **Password Fields:**
 - The **Password** field is critical for securing the account, and users are encouraged to choose a strong, secure password.
 - A **Confirm Password** field is included for verification to reduce errors and ensure that users input their password correctly.
 - **Password Visibility Toggle:** A key feature for convenience is the **toggle** to show or hide the password as users type. This helps users ensure they've entered the correct password without exposing it to others around them.

Call To Action

- **Sign Up Button:** The page features a prominent **Sign Up** button in the center of the form. The button is large and visually distinct to stand out, ensuring that users know where to click to complete their registration once all fields are filled. The button uses an eye-catching color, making it easy to find.
- **Link to Log In:** A secondary, less prominent link underneath the main form invites users who already have an account to **Log in**. This ensures that returning users do not feel lost and can quickly find their way to the login page.

Terms and Privacy

- **Consent to Terms:** At the bottom of the form, there is an important reminder for users that by signing up for the platform, they agree to Magpie's **Terms and Privacy Policy**. This ensures that users are informed about the platform's legal agreements before proceeding.
- **Link to Terms:** A clickable link is provided for users to read the **Terms and Privacy Policy** in detail. The link opens the full document, so users have access to the complete legal information regarding their data, privacy, and the platform's use.

Overall Design and Experience

The design of the signup page is intuitive and responsive, providing a smooth experience for users on both desktop and mobile devices. It is minimalistic, focusing on the essential information without distractions, while the color scheme and layout are optimized for clarity and usability. The goal of the page is to simplify the signup process, ensuring that users can easily create their accounts while being informed about the necessary legal terms.

This page ensures that users feel confident about creating an account by making the process as straightforward as possible while providing clear information about data privacy and platform policies. The inclusion of visual cues like the password visibility toggle and clearly marked sections for each field enhances usability, contributing to a positive user experience.

Login Page

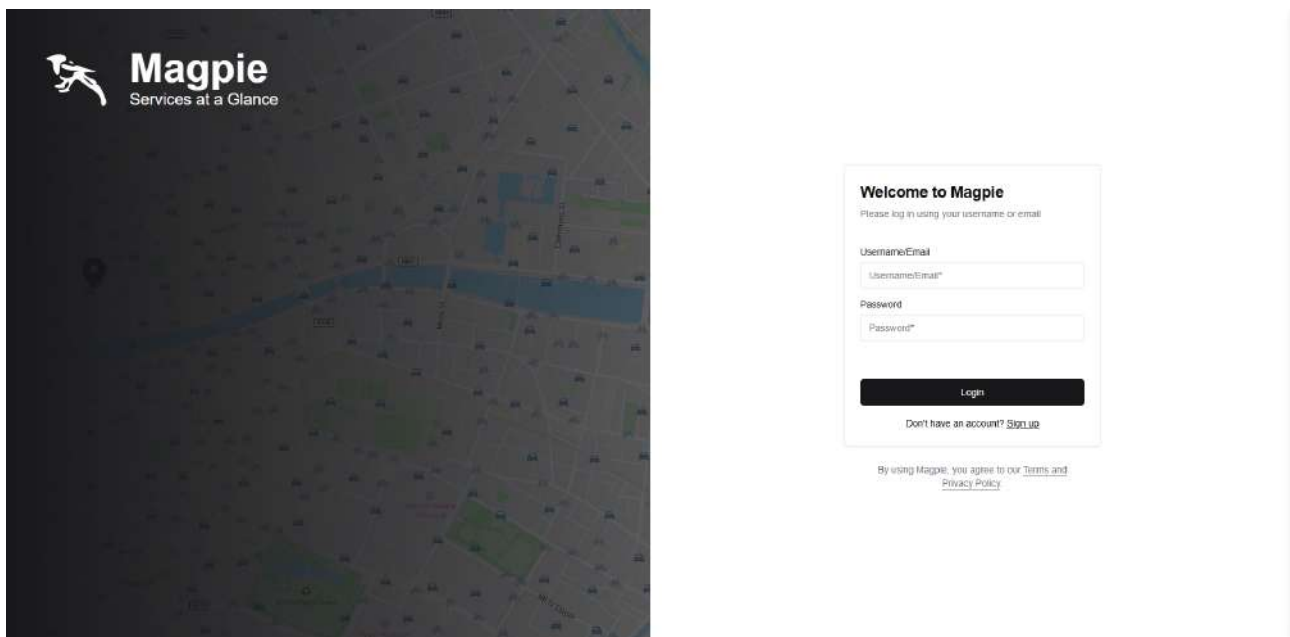


Figure 40: The Login Page

The **Login** page is designed to provide a smooth and efficient experience for registered users to access their Magpie accounts. This page ensures that returning users can log in quickly and securely, reinforcing the brand's focus on user experience and data protection.

Header section

The header section plays a crucial role in maintaining consistent branding.

- **Magpie Logo:** Just like on the signup page, the **Magpie logo** is prominently displayed at the top of the page. The logo serves to create immediate brand recognition and assures users they are on the official platform. It is positioned alongside the tagline '**Services at a Glance**', which reinforces the site's purpose—providing users with an easy and visual way to view available public services across Dublin.
- **Visual Consistency:** The design maintains visual consistency with the rest of the platform, using the same color schemes and font styles, ensuring that users immediately feel familiar with the interface. This consistency across pages helps enhance the overall user experience and navigational ease.

Form Inputs

The core functionality of the page revolves around the **login form**, which requires users to enter their credentials to access their accounts.

- **Username Input:** The form begins with an input field for the **Username**. This field is critical for user identification, allowing the platform to locate the correct account. The label for the input field is clearly marked, and the field is large enough for easy typing, which is particularly important for mobile users.
- **Password Input:** Below the username, users are required to enter their **Password**. This ensures secure access to the platform and protects user data. The password input field features a 'Show' button, allowing users to toggle between obscured and visible text, which reduces errors during login, especially when using complex passwords. This feature enhances user convenience, making it easier for them to confirm they are entering the correct password.

Homepage

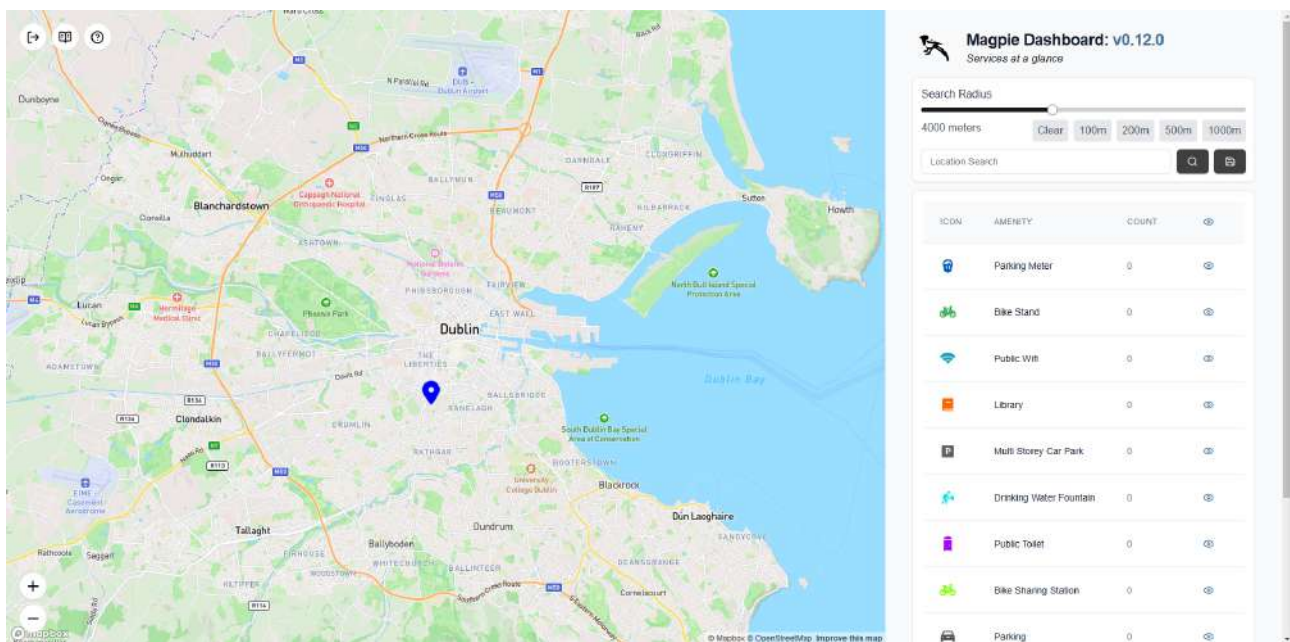


Figure 41: Homepage

Introduction

Upon landing on the page, users are greeted with a map interface centred on Dublin and its surroundings, displaying various public services available in the area. The page provides an interactive map to explore amenities and services in the region, such as parking meters, bike stands, public Wi-Fi, libraries, and more.

Top section

- **Logo and Title:** At the top-right corner, you'll see the Magpie logo and the dashboard title.
- **Search Bar and Radius Control:** Beneath the header, there is a search bar to find specific locations, accompanied by a 'Search Radius' slider. This allows users to set the distance range for viewing available services. The slider can adjust the search radius between 1m and 10km.

Map interface

The main focus of the page is the map, which occupies a large portion of the screen

- **Interactive Map:** The map is interactive, powered by Mapbox, where users can zoom in and out using a mouse or touchpad gestures (pinch to zoom). Location markers represent different public services, such as parking meters or bike stands. These markers provide both a visual and interactive experience for users.
- **Service Markers:** Clicking on any marker will display detailed information about the selected service, such as its name, type, and available count.

Sidebar

- **Interactive Map:** The map is interactive, powered by Mapbox, where users can zoom in and out using a mouse or touchpad gestures (pinch to zoom). Location markers represent different public services, such as parking meters or bike stands. These markers provide both a visual and interactive experience for users.
- **Service Markers:** Clicking on any marker will display detailed information about the selected service, such as its name, type, and available count.

Onboarding Tour:

A guided onboarding process walks the user through the various features of the platform, highlighting key elements such as:

1. **Search Radius:** Adjusting the search range.
2. **Marker Data:** Information displayed upon selecting a service marker.
3. **Selecting Amenities:** The ability to toggle amenity visibility.
4. **Map Interaction:** How to click and zoom on the map to select locations.

Footer and Cookie Consent:

Footer: At the bottom of the page, the user is shown a cookie consent banner, offering the option to accept or decline cookies for the site's functionality.

This page is designed to provide users with an easy-to-use and interactive interface to view and search for public services across Dublin. The combination of a detailed map with customizable filters and an intuitive onboarding process ensures that users can quickly understand and make use of the platform's features. The inclusion of interactive elements like the service markers and radius control further enhances the user experience.

History Page

The **History** page in Magpie provides users with a clear view of their saved location data, allowing them to easily manage and track their previously stored points of interest related to public services. This page is designed to help users interact with their historical data in a seamless and intuitive way.

Header Section

- **Navigation Back Button:** Located at the top left of the page, there is a prominent **Back Button**. This button allows users to quickly navigate back to the previous page. It's styled with an icon of a leftward arrow, making it easily identifiable. Its positioning ensures that users can always return to the previous screen without confusion.
- **Page Title and Description:** The main section of the header includes a title '**Saved Locations**' displayed in a large, bold font. This title makes it clear to the user that they are viewing a list of locations they have saved. Beneath the title, a brief description—'**Here's a list of your saved amenity locations!**'—adds context to the page, helping users understand the purpose of the section.



Saved Locations
Here's a list of your saved amenity locations!

[Delete](#)

<input type="checkbox"/>	Location	Radius	Amenity Types	Date Created	Show on Map
<input type="checkbox"/>	Farnell Street, Dublin Lat: 53.355078 Lng: -6.209925	500 m	Parking Meter: 45 Bike Stand: 51 Public Wifi Access Point: 3 Library: 3 Multistorey Car Parking: 3 Drinking Water Fountain: 0 Public Toilet: 0 Bike Sharing Station: 10 Parking: 1134 Accessible Parking: 43 Public Bin: 209 Coach Parking: 0	12/13/2024 8:19:10 PM	
<input type="checkbox"/>	Knocknashoe, Glastown Lat: 53.200254 Lng: -6.220722	500 m	Bike Stand: 3 Parking Meter: 0	12/16/2024 6:36:38 PM	
<input type="checkbox"/>	Knocknashoe, Glastown Lat: 53.200254 Lng: -6.220722	500 m	Bike Stand: 3 Parking Meter: 0 Public Wifi Access Point: 0 Library: 0 Multistorey Car Parking: 0 Drinking Water Fountain: 0	12/16/2024 6:36:46 PM	
<input type="checkbox"/>	Knocknashoe, Glastown Lat: 53.200254 Lng: -6.220722	1,000 m	Bike Stand: 23 Parking Meter: 0 Public Wifi Access Point: 0 Library: 0 Multistorey Car Parking: 0 Drinking Water Fountain: 0	12/16/2024 6:36:51 PM	
<input type="checkbox"/>	Knocknashoe, Glastown Lat: 53.200254 Lng: -6.220722	1,000 m	Bike Stand: 23 Parking Meter: 0 Public Wifi Access Point: 0 Library: 0 Multistorey Car Parking: 0 Drinking Water Fountain: 0 Accessible Parking: 4 Parking: 0	12/16/2024 6:36:54 PM	
<input type="checkbox"/>	Knocknashoe, Glastown Lat: 53.200254 Lng: -6.220722	1,000 m	Parking Meter: 0 Bike Stand: 23 Public Wifi Access Point: 0 Library: 0 Multistorey Car Parking: 0 Drinking Water Fountain: 0 Public Toilet: 0 Bike Sharing Station: 0 Parking: 0 Accessible Parking: 4 Public Bin: 0 Coach Parking: 0	12/16/2024 6:36:58 PM	

Figure 42: History Page

Saved Locations list

The central feature of the page is a table that displays the user's saved locations along with key details about each entry.

- **Delete Button:** Above the list of saved locations, there is a **Delete button**. This button allows users to remove one or more saved locations from their history. However, the button is disabled in this view, indicating that no deletions are currently available, possibly due to the absence of selected items.
- **Location Table:** Below the **Delete** button, the saved locations are listed in a table format. The table is neatly divided into columns, each containing specific information about the saved location. The columns include:
 - **Location:** The name or description of the saved location.
 - **Radius:** The radius within which services are displayed for the location.
 - **Amenity Types:** A list or description of the types of amenities (e.g., parking meters, bike stands) available at the saved location.
 - **Date Created:** The date when the location was saved to the system, helping users track when the data was added.
 - **Show on Map:** A column that likely provides users with the option to display the saved location on a map.

If there were saved locations, this table would dynamically populate with relevant data.

No Results Found

Empty State Message: If there are no saved locations, the table displays a message in the form of a single row: **'No results found.'** This message is centred in the table, letting users know that there are currently no saved entries to view. This feature ensures that users don't encounter an empty or broken page but rather receive a clear indication of the state of their saved data.

Pagination Controls

- **Previous and Next Buttons:** At the bottom of the saved locations table, there are pagination buttons for navigating through multiple pages of saved locations. These buttons are disabled in the current state, indicating that there is no additional data to paginate through at this moment.
 - The **Previous** button allows users to go back to earlier pages of saved data.
 - The **Next** button lets users move forward in the list.

These controls are hidden when the data table is empty, as no navigation is necessary.

Overall Functionality and User Experience

The **History Page** allows users to manage their saved locations effectively. The ability to track location-based services, adjust the radius, and view amenity types gives users a comprehensive understanding of the locations they have marked. Although the page currently features an empty state, its design allows for easy data manipulation once locations are added, with clear options to delete or view the locations on a map.

This page's clean layout, with intuitive table navigation and clear instructions, ensures that users can easily understand and manage their saved data. The disabled pagination and delete button indicate that users need to take action before those features become available, which helps guide them toward a more active interaction with the page.

4.5.4 Prototyping

A prototype is a useful design tool for testing concepts, clarifying requirements, and starting user interaction and feedback. Prototyping methods can be categorized by fidelity—ranging from low-fidelity sketches to high-fidelity digital mockups.

4.5.5 Prototype methods

We use evolutionary prototyping to continuously update our prototypes. Part of the prototyping process involves dealing with feedback and subsequent revisions. It helps designers test and retest their ideas over and over again. The faster designers are able to test their design concepts and make improvements, the faster they can get to a satisfactory final version. In addition, our team uses Agile development methodologies in prototyping. Agile increases flexibility, collaboration, and rapid feedback cycles to create product prototypes in rapid iterations with continuous ones after collecting feedback and guided ones. At every stage of the prototype design process, user feedback served as a essential guide for our next development steps.

We carried out low-intensity, one-on-one person interviews that allowed us to collect valuable qualitative feedback from users. We focused on collecting remarks about the intuitiveness of the interface and made sure to take note of any issues they encountered during their interactions with the platform.

We created unique tasks for test users to complete, while we observed their usage behaviour and how they navigate through the interface. This technique proved particularly effective in

identifying areas where users were confused, experienced difficulties or hesitated while they interacted with the platform.

To complement our qualitative studies, we applied quantitative testing through carefully designed questionnaires. Those surveys utilized the Likert scale to gauge user satisfaction throughout numerous aspects of the platform and included open-ended questions to capture additional suggestions and remarks. The quantitative statistics collected through those questionnaires enabled us to perform statistical analysis of consumer satisfaction levels and identify common user opinions and pain factors.

Step	Stage	Tasks
1	Requirement Collection and Analysis	<ul style="list-style-type: none"> • Identify core functional requirements through user interviews and research • Analyze the strengths and weaknesses of competitors and existing solutions • Prioritize prototype design and create an iteration plan
2	Prototype Design and Development	<ul style="list-style-type: none"> • Create prototypes using tools like FigJam and Figma • Follow a consistent design system and guidelines • Focus on intuitive and user-friendly interaction design
3	User Testing and Feedback	<ul style="list-style-type: none"> • Conduct user usability testing • Gather qualitative and quantitative user feedback • Analyze test data and develop improvement plans
4	Iterative Optimization	<ul style="list-style-type: none"> • Adjust design solutions based on feedback • Continuously improve the user experience • Ensure the design meets project objectives

Table 5: *Process Flow Table: Design and Development Stages*

4.5.6 Low-fidelity prototypes

- **Sketches**

We used FigJam for the sketch concept, which allowed us to do online brainstorming sessions. We started out with a card format, where the top right side displays the filter and the bottom side displays the rotation of the three icon styles, the top left has the Magpie logo and the bottom left corner shows the cookie component. This made the whole page too complicated, so we updated it so that the filter options and radius slider are placed on the right side of the screen as a dashboard. The logo is also moved to the dashboard, improving brand recognition.

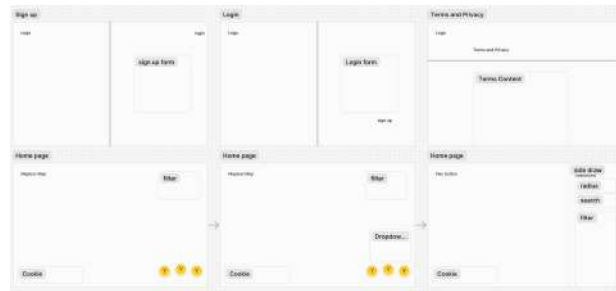


Figure 43: Evolution of interface design from initial card-based layout to consolidated dashboard approach

This evolution in our design approach demonstrates the value of iterative prototyping in achieving a balance between functionality and visual clarity. The final layout creates a more focused user experience while maintaining all essential features in an intuitive, accessible format.

- **Wireframe**

We used wireframes as a low-fidelity prototyping tool. Unlike sketches, wireframes show the structure of an interface design, but do so with reduced detail or colour. We also created wireframes of individual pages, such as the home page and the login and signup pages in Figure 44 and Figure 45, which lay out the structure of the prototype.

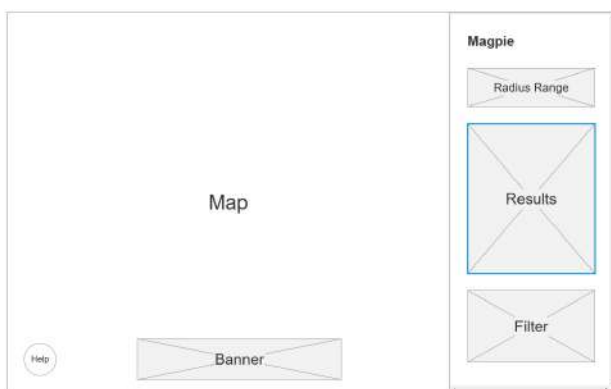


Figure 44: Wireframe-home

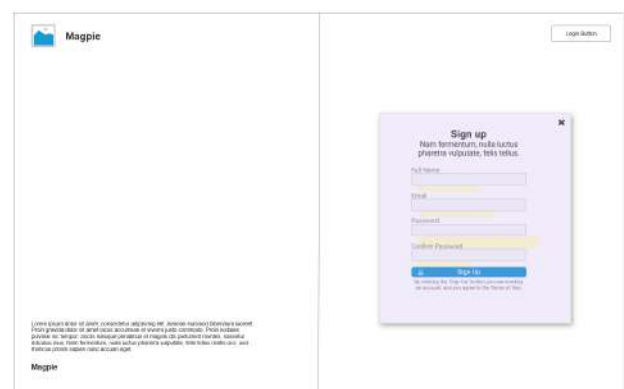


Figure 45: Wireframe-login/signup

4.5.7 Medium Fidelity Prototyping

We moved through the prototyping process from low-fidelity sketches and wireframes to medium-fidelity prototypes. This change allowed us to improve the layout, interactions, and structure while keeping the ability to iterate quickly. Medium-fidelity prototypes are often greyscale designs with simple lettering and placeholders, emphasizing functionality and user flow over visual features.

Medium-fidelity prototypes enabled our team to test crucial design decisions such as navigation structures, button placements, and overall usability without being constrained by aesthetic concerns. We created these prototypes in Figma, where we built interactive flows for tasks such as logging in, signing up, and navigating the dashboard. To simulate user interactions, we used more realistic spacing, component alignment, and basic interactive elements like clickable buttons and form fields. The purpose of medium-fidelity prototyping was to collect input on the primary user path, providing a smooth experience from task start to finish. We saved time on later versions by fixing faults at this stage.

4.5.8 High Fidelity Prototyping

High-fidelity prototypes closely resemble the final product in terms of design and functionality. These prototypes include comprehensive visual aspects including the finalized color scheme, typography, iconography, and branding assets, as well as genuine data and interactive transitions. At this point, our team worked on matching the prototype with user expectations and responding to specific feedback from previous versions.

The high-fidelity prototypes were utilized in advanced user testing to assess visual appeal, responsiveness, and accessibility. We focused on how users interacted with the final design features, such as hover effects, animations, and error states. This level of detail meant that the finished product was intuitive, visually consistent, and satisfied usability criteria.

By the time we moved on to high-fidelity prototyping, the emphasis had switched to improving the user experience and preparing the design for deployment. The feedback gathered during this stage directly influenced the UI iteration process, which is documented in the next section and demonstrates how specific issues were handled and changes were made to obtain the final design.

4.5.9 User Interface Iteration

Our iterative UI process focuses on the overall process involved in the User Interface (UI) and User Experience (UX). For example, the visual and interactive components that users use, including buttons, icons, and layouts, while UX covers the overall journey and emotional response of the user during an interaction. The importance of good UI and UX design lies in the improvement of user satisfaction, which then links to better customer retention and commercial success. It is believed, through studies, that well-thought-out user experiences can increase retention greatly. Ahmed, 2023 We did extensive reviews of each and every aspect in our iterations-from color scheme and typography to layout and how the navigation structures work-to ensure usability and interactivity. In terms of visual elements, we used a single colour scheme of predominantly black to create a cohesive and less intrusive design.

In addition to cultural considerations, the colour black is also inclusive and has a wide audience. The contrast between the accessible text and the background is also stronger, with enough contrast to improve readability for the user.

Iteration 1: Our preliminary prototype revealed several essential pain points through user testing. The interface layout, while intended, presented users with a fragmented experience because of its card-based design. Users encountered problems in dealing with multiple interface elements scattered across the display screen, mainly noting the disconnected placement of filters and radius controls. The login and signup pages, which applied default shadow styling, lacked customization and consistent branding. Additionally, the map view, whilst critical to the Magpie’s functionality, needed refinement in terms of icon visibility and usability.

In response to these early findings, we consolidated the interface elements into a unified dashboard on the right side of the screen. By switching to a more cohesive and intuitive interface we achieved increased user satisfaction. We also reevaluated the design of the authentication journey, but in the end we decided to keep separate login and signup steps. To create visual consistency, a similar, but distinct design was implemented for the authentication pages.

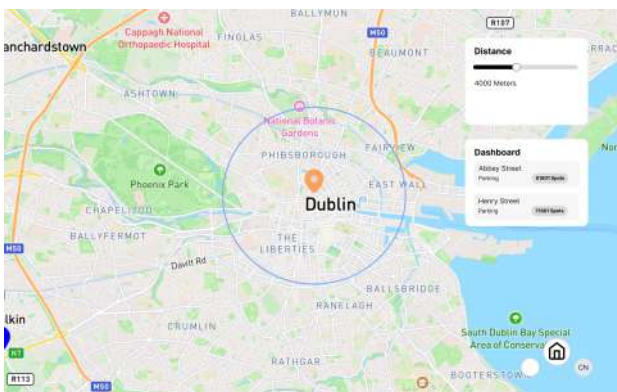


Figure 46: v1 _Home Page

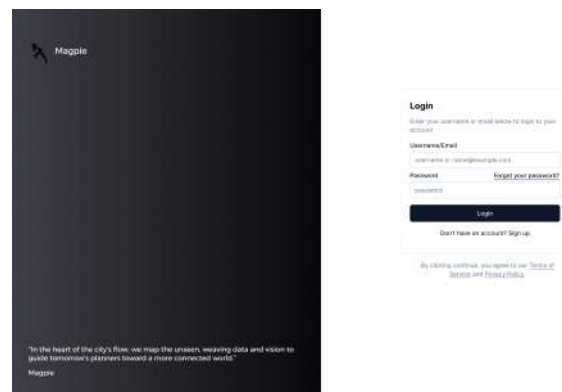


Figure 47: login/signup

Iteration 2: The second iteration aimed on enhancing the core capability and person interaction patterns. An important addition was the introduction of the search bar, which was initially positioned at the top of the screen. But, user testing revealed that this placement, whilst conventional, did not align optimally with the specific workflow of Magpie. Designing the amenities filter as checkboxes proved to be much less intuitive than expected, with users expressing a preference for greater immediately-visible feedback throughout their interactions.

We incorporated that feedback by integrating the search functionality into the dashboard, creating a consolidated control center for all user interactions. The amenity filters have been redesigned to provide clearer visual feedback. However, this iteration highlighted the need for additional refinement in the filter interface design. Those changes were a large improvement in streamlining the user experience, however they also illuminated areas that require extra attention in subsequent iterations.

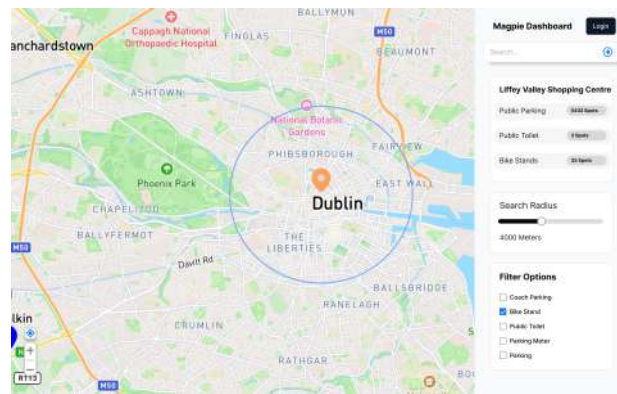


Figure 48: v2_Home Page

Iteration 3: The third iteration included significant improvements to the onboarding experience and interface clarity. We added a comprehensive and interactive tutorial to guide new users through the Magpie's capabilities. We also added pages for interacting with Saved Locations, as well as the Terms and Privacy page. However, user testing revealed that our efforts to streamline the amenity filter function using a dropdown menu was in reality detrimental to usability. Expert users in particular mentioned problems with map icon visibility, highlighting the need for higher contrast between different amenity types.

The feedback from this phase proved especially valuable in figuring out the fine line between making the interface simpler and making it more accessible. The dropdown menu was meant to simplify the interface, but in reality it created an additional layer of complexity that impacted usability. This insight motivated the layout changes in the fourth iteration.

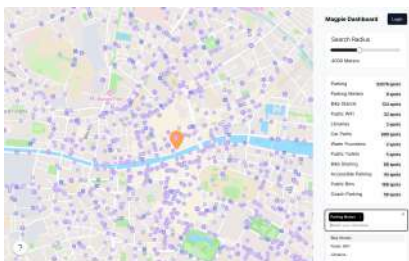


Figure 49: v3_Home Page

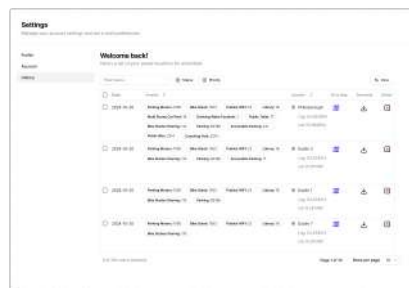


Figure 50: v3_History

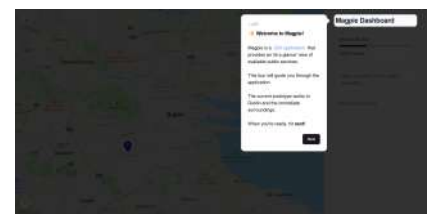


Figure 51: v3_onboarding

Iteration 4: Our fourth and last iteration made comprehensive changes based on the user feedback we gathered. The map icons were completely redesigned, adding contrasting colours for different amenity types. This helped to improve visibility and aids fast recognition. We abandoned the dropdown menu approach in favour of directly available buttons, considerably improving the discoverability of Magpie's features.

Professional user feedback resulted in the addition of quality-of-life features such as zoom controls and a scale ruler. However, a number of features remained in development due to time constraints. The preset buttons for the search radius were relocated to the upper segment of the dashboard for better accessibility, with more options being added (100m, 200m, etc.) as well. The filtering function was redesigned with intuitive toggle controls, allowing users to enable or disable amenities easily.

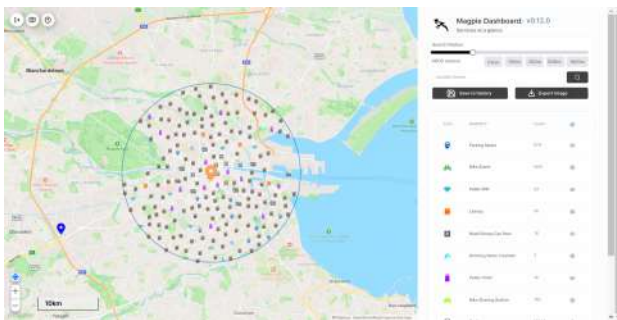


Figure 52: *v4_Home Page*

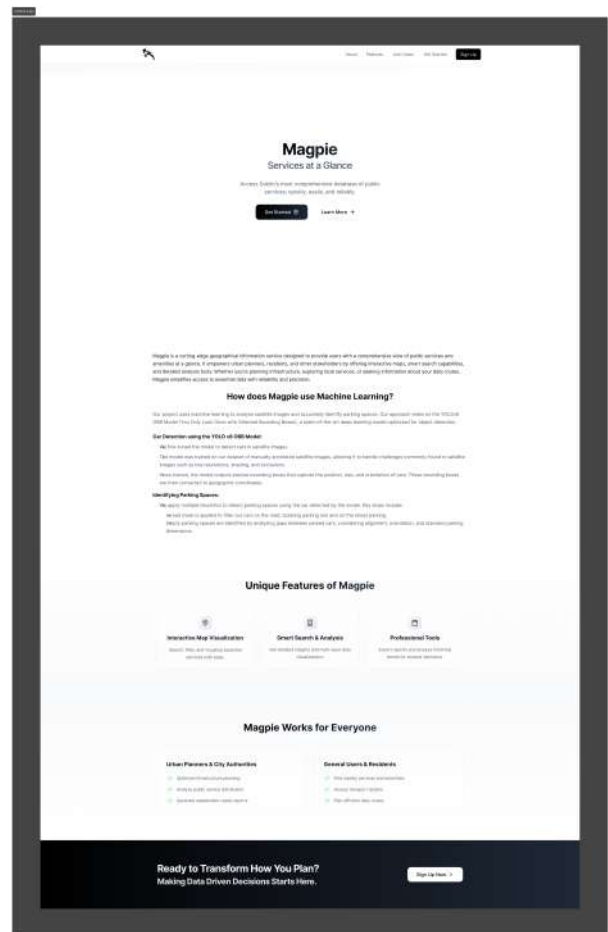


Figure 53: *v4_Landing Page*

Ongoing Development Through these iterations, we identified numerous regions for future enhancements. They include improving cross-browser compatibility, mainly concerning layout issues where dashboard features overlap with the search interface. The accuracy of amenity locations must also be validated through spot checks. The search functionality needs to be updated to find various kinds of location names more consistently. Additionally, expert users have requested more advanced features such as exporting a portion of the map as an image which includes a scale ruler, data export or amenity density visualization capabilities.

This iterative improvement method has validated the value of constant user feedback in creating a user-friendly platform. Each iteration helped us to get a step closer to achieving our goal of creating an intuitive but capable tool for both non-expert users and professionals. Basing Magpie’s development on this design process underlined the importance of continuously refining the product and listening to the needs of users.

4.6 Backend

4.6.1 Backend Architecture

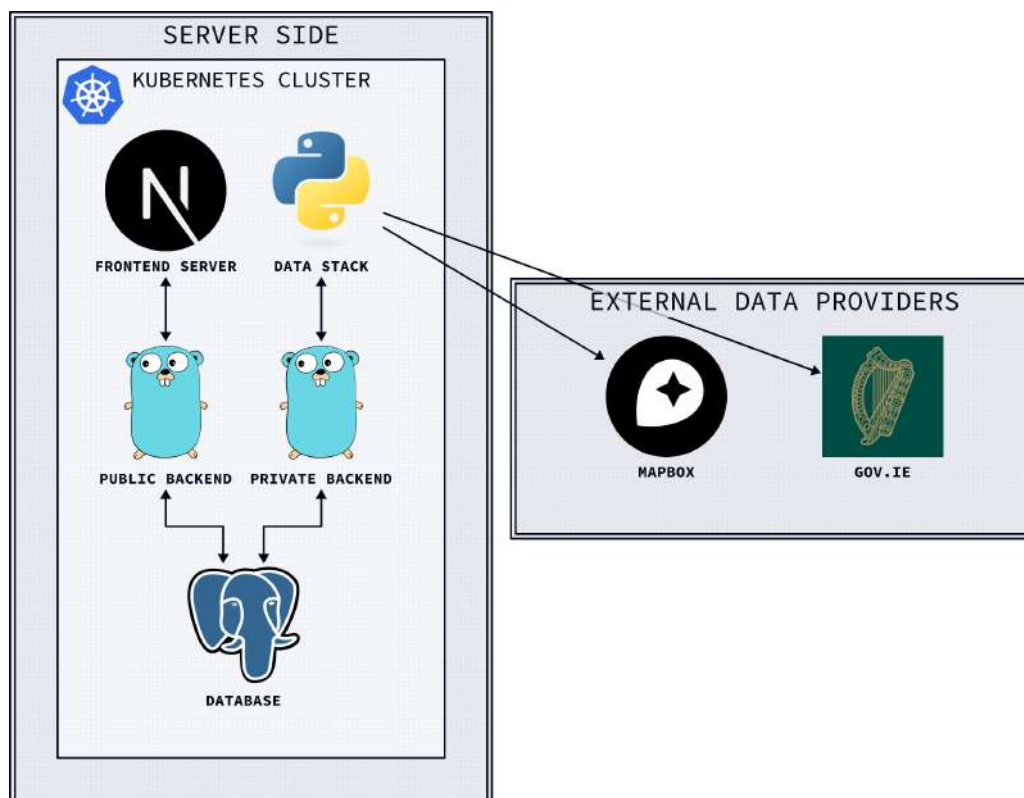


Figure 54: Backend Architecture Diagram

4.6.2 Backend Technologies

Go

At the start of the project, the team deliberated about which technologies to choose for the backend. Options using JavaScript such as NodeJS or Deno with accompanying frameworks like ExpressJS or NestJS were considered, as was Rust and its Rocket framework. These would have been excellent choices as they are well established in the industry and tried-and-tested.

The majority of the team has development experience with JavaScript. However, it was planned instead to deploy this application on a server hosted by one of the team members. Therefore, the usage of computing and memory resources was very important to not strain their Kubernetes cluster more than necessary. Since NodeJS runs on the JavaScript runtime V8, which also powers Google Chrome, our experience indicated that it would be quite resource intensive to run. Findings by Tanadechopon and Kasemsontitum (2023) confirm this.

Due to this, the team shifted towards using compiled rather than interpreted languages, as these are generally more resource efficient. Since small executables and low memory usage was desired, languages and frameworks that run on virtual machines such as Java with Spring or C# with .Net were not deemed to be viable options. As a result, only Rust and Go were seriously considered at this point.

Rust offers a small package size, strong memory safety, an excellent ecosystem and build-tools. The team member that would focus on backend development had recent experience in writing Rust code. However, Rust development can be very tricky and time consuming. Additionally, in our experience Rust has above average compilation times. In a project with a fixed deadline and the expectation of rapid development, choosing Rust would have been detrimental. The team recognised that choosing Rust would come with many drawbacks while offering only few benefits.

This left choosing Go as the logical conclusion. A large drawback that the team identified with Go was the missing experience in the team. Two team members had used Go before, but they last used it a few years back. However, since Go syntax and the languages concepts hadn't changed much since then, it was deemed possible to quickly get up to speed, much more quickly than Rust would have allowed. The final decision fell on using Go, as it offered small binaries, great memory efficiency, a solid ecosystem of libraries and build-tools and a feature-rich, built-in library for creating REST-APIs. Go also removes a lot of the pitfalls that Rust suffers from: it has a simple and approachable syntax and a very low-friction, high-speed development experience. This was also the deciding factor, since it was made clear to the team that getting an MVP up and running as quickly as possible was imperative for the project.

PostgreSQL

The plan for the MVP of Magpie set the simple goal of delivering useful data to users. To achieve this, the data needs to be stored in a way in which it can be efficiently accessed during operation. Since the goal of the project was to provide a tool to professionals, data integrity at every level was an important consideration. Additional requirements for the data storage solution were good support for geospatial data, quick retrieval of a large number of points and ease-of-use.

There is a plethora of database solutions available today. The team considered the most common types: document databases like MongoDB and multi-model databases like PostgreSQL (sometimes called RDBMS).

While document databases like MongoDB have their place in the current development landscape, it soon became clear that SQL-based, multi-model RDBMSs would be best suited for the job. They offer tried-and-tested performance and reliability and they are well equipped to guarantee data integrity. But the deciding reason was the pre-existing experience the lead developer for the backend had with these types of databases.

After some deliberation, **PostgreSQL** was chosen as the database solution for this project. It was combined with the PostGIS extension to add support for geospatial data and queries. The decision was not cut and dried as most established multi-model databases (like Oracle DB or MySQL) offer the functionality that the requirements were asking for. However, the Kubernetes cluster this project was going to be deployed on already had a fully configured PostgreSQL server running on it. Making use of pre-existing infrastructure is the obvious choice in an agile project that had the prime goal of hitting the ground running.

```
-- name: GetPointsInRadius :many
SELECT Id, LongLat::geometry, Type from points
WHERE ST_DWithin(
  LongLat::geography,
  ST_SetSRID(
    ST_MakePoint(@longitude::float, @latitude::float), 4326
 )::geography,
  @radius::float
) AND (
  @types::point_type[] IS NULL OR Type = ANY(@types::point_type[])
);
```

Listing 3: *An example of a SQL query with annotations used by sqlc*

sqlc

In modern software development, there are two common ways for an application to interact with a database via SQL. There is the old school way of writing raw SQL queries, put them into prepared statements and execute them against the database. This gives the developer very granular control over the way their queries are structured and how they run them. But this method requires a significant amount of work in designing and implementing a translation layer between the database and the application. Data that the application wants to send to the database needs to be prepared into a format the database queries expect. Data that the application wants to retrieve from the database needs to be parsed back into the data model that's used by the application.

To make interfacing with databases easier, so called ORMs (Object Relational Models) were developed. These are libraries that the application developer can include. Database queries are not done in SQL, but in the same programming language the application is written in. The ORM provides database interface functions that take in the data in the format the application uses. To actually retrieve any data or make any changes to the database, the ORM runs its internally generated SQL queries on the database using a compatible driver. ORMs provide an abstraction layer, removing the need for directly interacting with the database and providing a data access wrapper in the main programming language of the project. This can make it easier and quicker to develop an application that makes use of a database, but it takes away some of the agency that the developer has.

The team was dissatisfied with both of these solutions, so an alternative called **sqlc** was chosen.

This tool combines the freedom that using SQL gives developers with the productivity increase that ORMs offer. sqlc flips the typical ORM workflow on its head. This means, the developer writes standard SQL queries and schemas, but includes sqlc directives such as the name of the query and the multiplicity of expected results in the comments above the query (see Listing 3). These files are then passed to the CLI of sqlc. In accordance with a configuration file provided by the developer, sqlc then automatically generates type-safe bindings for the queries that were defined in the SQL files (Gray, 2019).

This approach gives the developer more control about the queries that are executed. At the same time, it eliminates the need to write boilerplate wrapper code for accessing the database, just like an ORM would eliminate the need to write SQL queries. The tool treats SQL, which is a structured and typed language, as a source of truth. It also enables developers to reuse their queries across systems as sqlc offers code generation for multiple programming languages like Go, Python or Typescript and databases like PostgreSQL, MySQL or SQLite (Gray, 2024b).

The tool respects best practices to prevent SQL injection attacks. It generates code that only utilises constant strings and parametrised queries (Gray, 2024a). Using this tool, the team was quickly able to connect the backend server to the database. Eliminating the need to update the Go code manually each time the database schema or queries changed was vital for the backend keeping pace with the other developers.

```
version: "2"
sql:
- schema: "sql/migrations"
  queries: "sql/queries_private.sql"
  engine: "postgresql"
  gen:
    go:
      package: "db"
      sql_package: "pgx/v5"
      out: "internal/db/private"
      build_tags: "private"
      emit_json_tags: true
      overrides:
        - db_type: "geometry"
          go_type:
            import: "github.com/twpayne/go-geom"
            pointer: true
            type: "Point"
- schema: "sql/migrations"
  queries: "sql/queries_public.sql"
  engine: "postgresql"
  gen:
    go:
      package: "db"
      sql_package: "pgx/v5"
      out: "internal/db/public"
      build_tags: "public"
      emit_json_tags: true
      overrides:
        - db_type: "geometry"
          go_type:
            import: "github.com/twpayne/go-geom"
            pointer: true
            type: "Point"
```

Listing 4: An example of a sqlc configuration file with two targets with separate query inputs and type replacement

```
type GetPointsInRadiusParams struct {
    Longitude float64    `json:"longitude"`
    Latitude   float64    `json:"latitude"`
    Radius     float64    `json:"radius"`
    Types      []PointType `json:"types"`
}

type GetPointsInRadiusRow struct {
    ID        int64        `json:"id"`
    Longlat   *go_geom.Point `json:"longlat"`
    Type      PointType      `json:"type"`
}

func (q *Queries) GetPointsInRadius(
    ctx context.Context, arg GetPointsInRadiusParams
) ([]GetPointsInRadiusRow, error) {
    rows, err := q.db.Query(ctx, getPointsInRadius,
        arg.Longitude,
        arg.Latitude,
        arg.Radius,
        arg.Types,
    )
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var items []GetPointsInRadiusRow
    for rows.Next() {
        var i GetPointsInRadiusRow
        if err := rows.Scan(&i.ID, &i.Longlat, &i.Type); err != nil {
            return nil, err
        }
        items = append(items, i)
    }
    if err := rows.Err(); err != nil {
        return nil, err
    }
    return items, nil
}
```

Listing 5: An example of a Go binding generated by sqlc from the SQL query in Listing 3

golang-migrate

During the first weeks of the project, the database was defined by a simple initialisation SQL script. This worked fine for the time and allowed all developers to set up their local database instance easily. But once the vertical slice was completed, the team wanted to add additional features that would necessitate changes to the database schema. After adding the first feature, some developers ran into issues where the version of the backend they were running was not compatible with the schema currently deployed on their local database.

Fixing these issues was time consuming and they were likely going to reappear after future database changes. Realising this, the team made the decision to integrate a database migration solution to automate this tedious process and prevent future time losses during development.

The tool used for generating Go bindings from the raw SQL, `sqlc`, does not offer any migration functionality, so another solution needed to be found. The team evaluated all options that were listed as compatible with `sqlc`. These were `atlas`, `dbmate`, `golang-migrate`, `goose`, `sql-migrate` and `tern` (Gray, 2024c). The team was looking for a solution that allowed the migrations to be specified in plain SQL. The solution needed to work both as a command-line interface application as well as a Go library, as it was deemed necessary to apply the migrations to the database automatically. A tool that did not rely on additional configuration files would also be preferred. Multiple options satisfied these requirements. The backend team decided to use **golang-migrate**, due to its straightforward usage and easy integration as a library.

At the time this decision was taken, `golang-migrate` also seemed like it was being developed most actively. This has since changed and at the time of writing, `golang-migrate` is the solution with the least recent update of the options. Nonetheless, using `golang-migrate`, the team was able to integrate automated database migrations into the app, eliminating the need to spend valuable time performing manual database updates each time the schema is changed.

The complete database development workflow for the backend server is visualised in Figure 55.

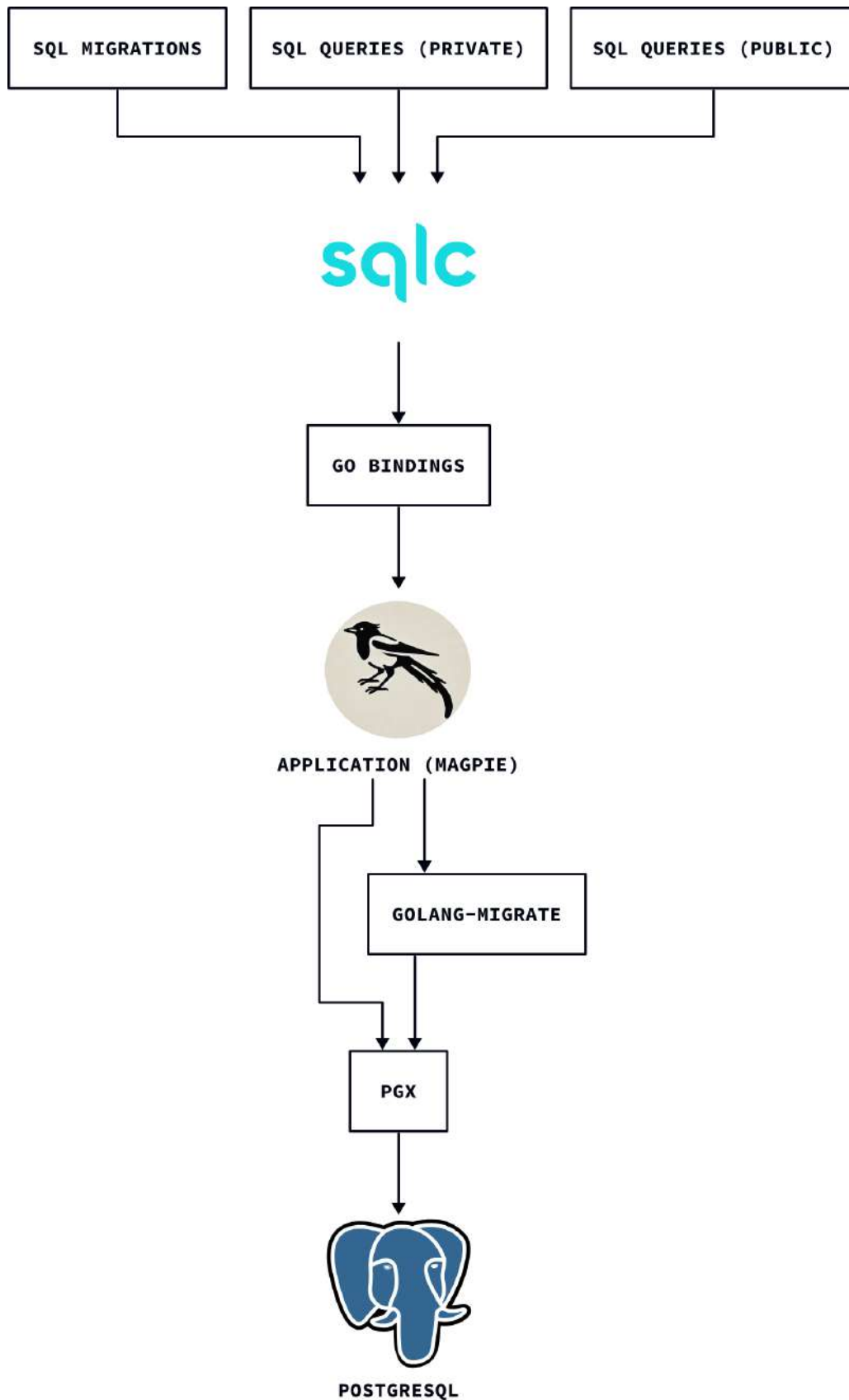


Figure 55: Database Workflow Diagram

bcrypt

To store the users credentials securely, **bcrypt** was chosen to hash them before they are inserted into the database. The resulting hashes (see Figure 56) can then be used to validate if the given password is correct. In accordance with best practices, a work factor of 12 was used when hashing passwords and a password limit of 72 bytes was implemented (Open Worldwide Application Security Project (OWASP), 2024b).

While the Open Worldwide Application Security Project (OWASP) (2024b) recommends other, more recent technologies over bcrypt, they also don't discourage its use in any way. In the case of Magpie, bcrypt was chosen because the team had past experience working with it. This was desirable as the registration and login functionality was deemed to be one of the core features of the MVP. Choosing a technology that the team did not have any experience with could have resulted in slower development, bugs, or even security vulnerabilities as a result of improper implementation.

Since the application was not intended to store highly valuable personal data such as banking information, personal addresses or health care records it was deemed acceptable to use a less recent technology in exchange for more rapid development. Should this project ever evolve into a publicly available and frequently used application, the use of bcrypt would of course need to be reevaluated with a much bigger focus on real-world application security rather than just development speed.



\$2a\$12\$NMjICkysza7OZUx2/yHNeaPLp8s5qfyeAX95bOKJnBAIboHcLUkC

Algorithm Cost Salt Hash

Figure 56: Visualisation of a bcrypt hash

JSON Web Token

JSON Web Tokens were chosen for the authentication of user requests to the APIs. JWTs are JSON objects that are encrypted using a private key on the server and passed to the frontend when a login requests succeeds. The client is then expected to pass the token to the backend as a form of authentication in subsequent requests.

JSON Web Tokens are more than just a simple API token. They can store data about the authenticated user in them, which can be very useful if the handling of the requests needs a reference to the user id for example (see example JWT content in Figure 57). Since the tokens are generated on the backend server using a private secret, JWTs can be checked for integrity during decoding. Should the content not match up with the private secret of the backend server, it can be assumed that the token has been tempered with. In this case, the request would be discarded and an "Unauthorized" error message would be returned instead.

This structure allows for the authentication system to be decoupled between backend and frontend. Using JWTs eliminates the need for the backend to keep track of active sessions, significantly reducing complexity. This and the familiarity the team had with working with JWTs were ultimately chosen as the authentication solution for Magpie.

JSON Web Tokens do however come with their own caveats. For example, if any malicious actor stole such a token from a user, they could pretend to be that user without the system noticing. JWTs do not have a built-in way to revoke potentially stolen tokens, neither from the client nor from the server (Open Worldwide Application Security Project (OWASP), 2024a).

Despite these caveats and some slowdowns during development, the team still feels JWT-based authentication was the right choice for a project of Magpie's scale.



Figure 57: Example of a JWT used by Magpie for authentication

4.6.3 Split Backend

The backend was planned as a one server system, but the team wanted to harden the system as a whole against potential attacks. This meant that some critical functionality could not be included in the server that handles public requests. As a result, the backend is comprised of two standalone REST-API servers. Both of these servers access a shared database, allowing them to cooperate without sharing functionality. The private backend is used to insert datasets into the database, the public backend is used to retrieve that data from the database and pass it to the frontend server.

This means that even if a malicious actor gained access to the public backend server, they could not execute anything that would threaten the integrity of the datasets stored in the database.

Right after the decision to split the backend was taken, a first implementation using two separate codebases was created. This approach was functional, meaning it was able to produce two distinct backends with different feature sets.

But it soon became obvious that this approach was not sustainable. While simply splitting the backend into two codebases was a quick and easy solution, it would almost certainly lead to significantly increased development times in the future. The two backends have a fair amount of identical functionality and differ just in the route handlers and the database queries. Leaving the codebases separate would result in many changes being made twice – once in the private backend and once in the public backend.

After careful consideration and discussions with the team, the decision was made to revert the change that split the backend. The desired result of two distinct backends would have to be achieved in a different way.

To accomplish this, a feature of the Go programming language called **build-tags** was utilised. Usage examples of this feature showcase it by creating multiple binaries that have different feature sets, for instance multiple different payment tiers for a single software. This was a great solution for this problem. It allowed for certain files to be excluded during compilation based on if they were needed for private or public functionality. While the development of a program split by build-tags is more challenging than developing a single binary, it is much more streamlined than keeping functionality identical between two separate projects.

The build-tag system did not fix the problem of code duplication completely however. Some tools in the Go ecosystem did not handle the exclusive split as expected. Most notably, the Go language server that provides code completion and other useful IDE features. It was possible to configure it to recognise files with their accompanying build-tags, but there was no way to specify exclusivity. This meant that two files in the same package could not have identical function definitions even though they would never be included in the binary build process simultaneously. It was possible to work around this restriction, which it lead to some code duplication. But this was still a lot better than duplicating the whole backend.

And while `sqlc` offers integration for the build-tag system (see the config file in Listing 4), it did not offer to just split the query part of the generated bindings. As a consequence, the generated code handling the model bindings and the access to the database had to be generated and included twice.

4.6.4 Database Development

The requirements for the database of the MVP were simple: store points, store users. To achieve this, three tables and one enum were added. The `Points` table is used to store the actual data for Magpie. In the first iteration (see Figure 58), this was only parking spots detected by the machine learning stack.

The tables `Logins` and `User_Details` are used to hold account information. This design is a one-to-one relationship, which is usually discouraged in database design. However, data from the `Login` table and data from the `User_Details` table are never queried at the same time but only separately. While this design still violates database normalisation principles, it makes querying the data more straightforward and was therefore preferred in this time-constrained project.

Encoding the point type in an enum has benefits and drawbacks. Using an enum enforces stricter data integrity, only a point with a type that is present in the enum can be inserted into the database. By extension, this helps to standardise the names of point types accross all layers of the application. On the other hand, storing the point type in an enum makes the database a lot less flexible. Adding a new dataset will almost certainly require changing the point type in the database. This creates a barrier to quickly expanding the number of different datasets in Magpie. Creating what is essentially a custom data type in the database also led to issues with connecting the backend to the database. The driver used for this purpose (`pgx`) expects custom data types to be registered on the database connection instance. Figuring out how to do that acted as a blocker for further backend development. It was still possible to add new datasets, but the decision to store the point type in an enum should be re-evaluated for future projects.

The Schema_Migrations table seen in both the initial database schema as well as in the final database schema is automatically created by golang-migrate to track the current version of the database.

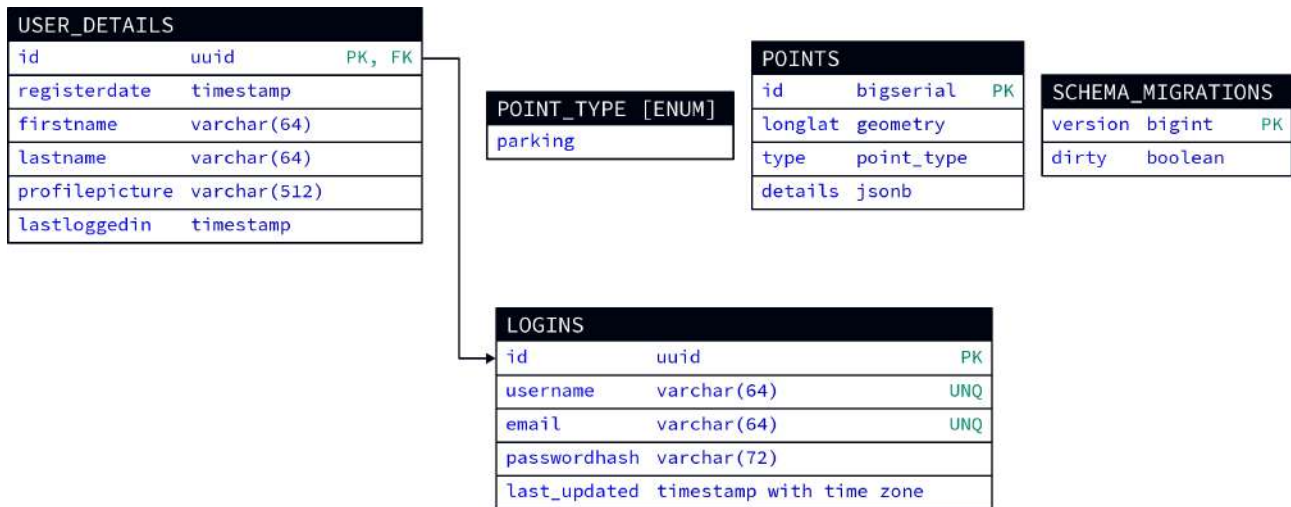


Figure 58: Database Schema for the minimum viable Product

During the course of the project, two significant changes were made to the database. The first one was the addition of more point types to the Point_Type enum. To make it possible to seamlessly move between database versions, migrations were designed to be reversible and (where possible) prevent data loss.

Unfortunately, PostgreSQL only allows the addition of enum values but not the deletion. Because of this, the down (reverse) migration for this change have to use a workaround. First, a transaction is started, then the existing enum is renamed and a new enum is created under the original name with the necessary values already removed. Afterwards, the Points table is modified to use the newly created enum. The old enum is then dropped and the transaction committed.

To satisfy the goal to have as little data loss as possible when moving through migrations, any point whose type is removed by a migration will have it added to a special value in the details field. Should the migration ever be re-applied, this field is then taken from the details object and translated into the correct enum again. Since all points need to have a point type, points whose type was removed will be given the type unknown. If down migrations are executed past the point where the unknown type was added, data loss is inevitable.

The second big change was the addition of the saved locations feature. This was originally called 'History', which is why the database tables are still called Location_History and History_Amenity_Counts. At first, only the former table was created, but as the feature evolved it became necessary to store a place name (called displayname in the database) and counts of amenities along side the saved location. This was also realised using migrations.

The History_Amenity_Counts table has a one-to-many relationship with the Location_History table, making it possible to include zero or more amenities in the saved location. There can only be zero or one entries for each point type per saved location. This is enforced by including both the point type and the id of the saved location entry in the primary key constraint.

The final database schema is shown in Figure 59.

The team recognises that neither the database nor the queries can be considered fully optimised. This was a conscious choice, trading some performance for quicker development time. Benchmarks of the point retrieval query performed by the team showed that the database took less than 100ms to retrieve 58,192 rows of points in a radius of 10 kilometres. The time it took to for the data to be transmitted to and be parsed by the frontend was in excess of 1.4 seconds. While there are performance bottlenecks in Magpie, the database's performance is currently not considered one of them.

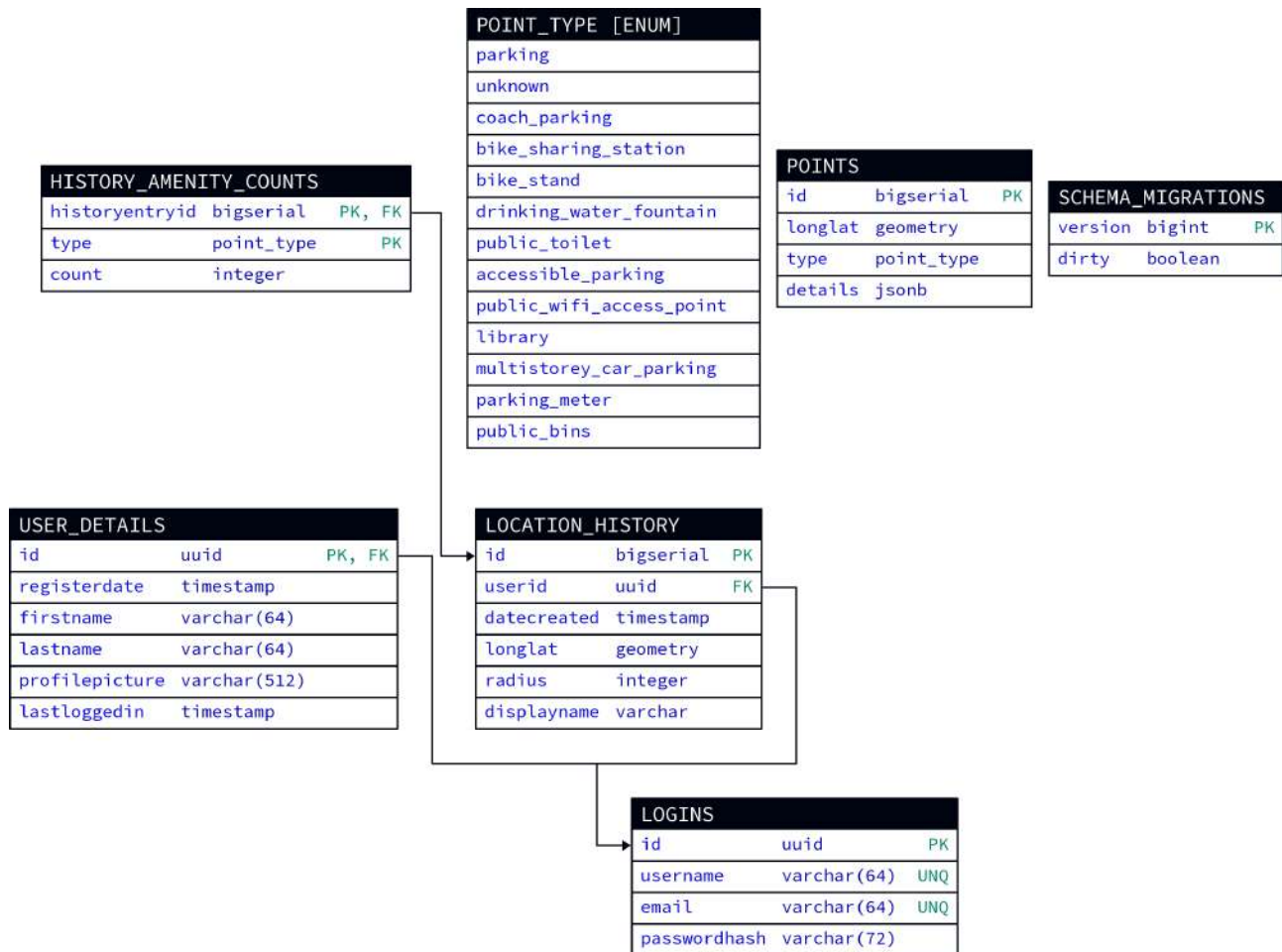


Figure 59: Final Database Schema

4.6.5 Routing

Routing the frontend's requests to the appropriate handlers was done using the **net/http** package, which is included in the Go standard library. The Go ecosystem provides a number of alternative solutions such as Gin, Gorilla or Fibre, which are mature and well documented frameworks for creating APIs. They provide many useful features, which the team acknowledged during their evaluation, but – in the team's opinion – they can also easily add a lot of complexity. Since the team was lacking in recent Go knowledge and didn't want to unnecessarily overcomplicate the project, it was decided to go with the built-in solution instead.

To realise the API functionality, the http server function provided by the net/http library is configured with a `http.ServeMux` object. This is where the server will send all incoming requests. These are then passed to nested routers based on prefix routing. For example, a request for the route `/v1/public/auth/User/login` would follow this path: HTTP Server → Public Router → Public Auth Router → Auth handler. A full overview of the routing in the backend can be found in Figure 60.

All requests are also routed through a stack of middlewares. Some middlewares are active on all requests (Logging, CORS) and some are configured based on route (Authorisation), see Section 4.6.7 (Middlewares).

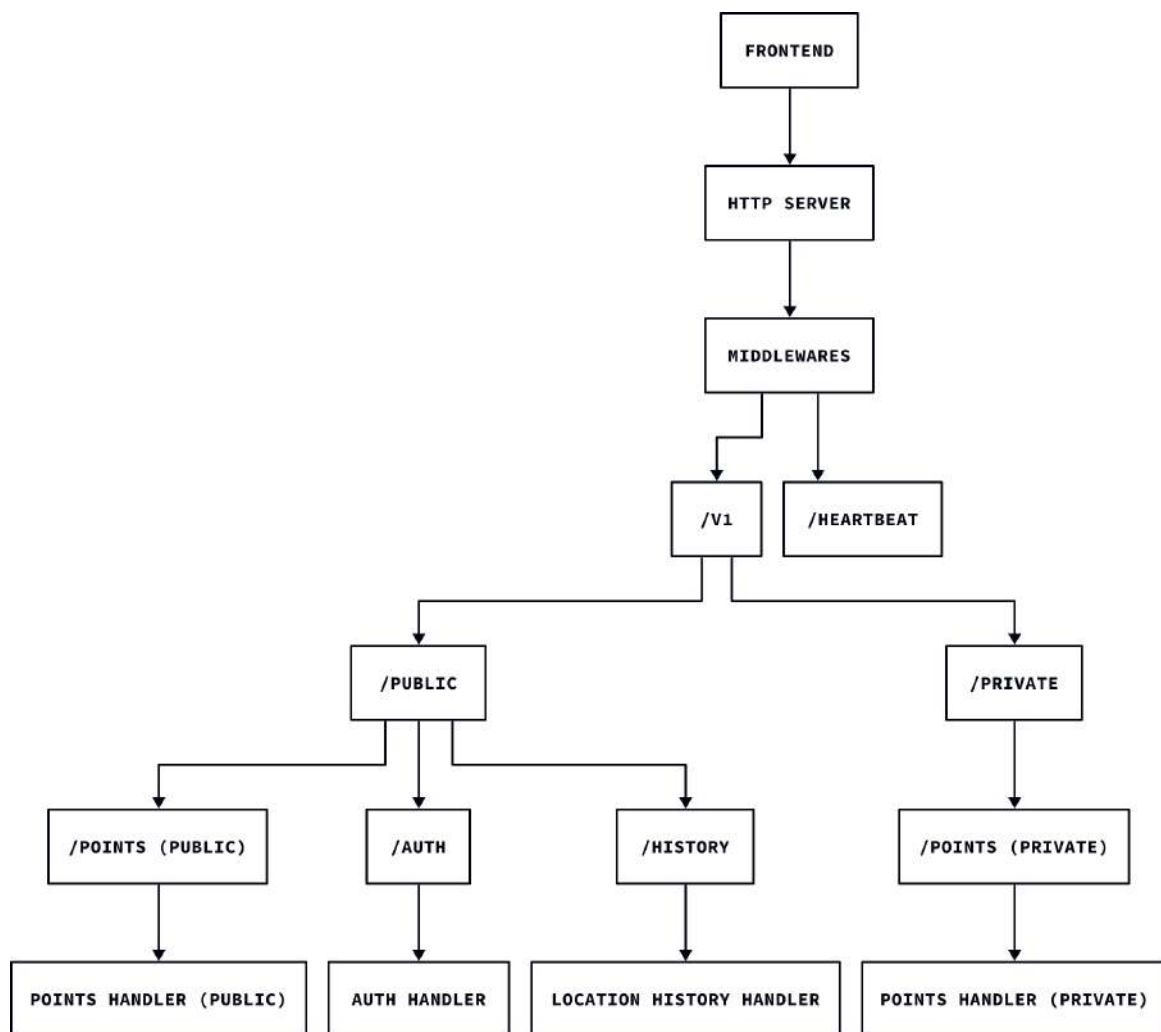


Figure 60: Routing in the Backend: from Request to Handler

```
func init() {
    AddRoute(route{"/public/", public()})
}

func public() *http.ServeMux {
    router := http.NewServeMux()
    router.Handle("/points/", http.StripPrefix("/points", pointsPublic()))
    router.Handle("/auth/", http.StripPrefix("/auth", auth()))
    router.Handle("/history/", http.StripPrefix("/history", locationHistory()))
    return router
}

func pointsPublic() *http.ServeMux {
    router := http.NewServeMux()
    handler := &handlers.PointsHandler{}

    // Authenticated access
    router.Handle("GET /inRadius", Authenticated(http.HandlerFunc(handler.HandleGetByRadius)))
    router.Handle("GET /{id}", Authenticated(http.HandlerFunc(handler.HandleGetPointDetails)))

    return router
}

func auth() *http.ServeMux {
    router := http.NewServeMux()
    handler := &handlers.AuthHandler{}

    // Public access
    router.Handle("POST /User/login", Public(http.HandlerFunc(handler.HandleLogin)))
    router.Handle("POST /User/", Public(http.HandlerFunc(handler.HandlePost)))

    // Protected access
    router.Handle("GET /User/{id}", Protected(http.HandlerFunc(handler.HandleGet)))
    router.Handle("PUT /User/{id}", Protected(http.HandlerFunc(handler.HandlePut)))
    router.Handle("DELETE /User/{id}", Protected(http.HandlerFunc(handler.HandleDelete)))

    return router
}

func locationHistory() *http.ServeMux {
    router := http.NewServeMux()
    handler := &handlers.LocationHistoryHandler{}

    // Protected access
    router.Handle("GET /{id}", Protected(http.HandlerFunc(handler.HandleGet)))
    router.Handle("DELETE /{id}", Protected(http.HandlerFunc(handler.HandleDelete)))
    router.Handle("POST /{id}", Protected(http.HandlerFunc(handler.HandlePost)))

    return router
}
```

Listing 6: *An example of how routing is configured in the backend*

4.6.6 APIs

All API routes that the backends expose are listed below. This information was available to the frontend and data stack developers throughout the project in the form of a collection of markdown files in the repository. This clearly communicated the capabilities of the API routes, making it easier to develop the application parts that interfaced with them. Providing strong documentation also eliminated a lot of ambiguity, resulting in more concise and efficient communication between the different areas of responsibility.

For a detailed explanation of the route access restriction modes, see Section **Middlewares** on page 91.

Public API

The public API is used to retrieve relevant data for display in the frontend and for authentication of the user.

- **Point Routes**

- **Name:** Point Details

- Method + Route:** GET /v1/public/points/{id}

- Description:** Provides additional details about a point. The structure and content of the details is different for every point type. The details are derived from the input dataset and added to each point on ingress.

- Access:** Authenticated

- Parameters:** Point UUID (path parameter)

- Returns:** JSON Object containing details about the point

- **Name:** Points in Radius

- Method + Route:**

- GET /v1/public/points/inRadius?long={}&lat={}&radius={}&types={}

- Description:** Retrieves all points in a circle. The circle is defined by its center coordinates and a radius in meters. The types parameter is optional. If a comma separated list of point types is provided, the returned data will be filtered to only contain points with the given types.

- Access:** Authenticated

- Parameters:** Longitude, Latitude, radius, types (all query parameters)

- Returns:** JSON object containing a list of points inside the given radius, filtered by the given point types

- **Saved Location Routes**

- **Name:** Retrieve saved locations

- Method + Route:** GET /v1/public/history/{id}

- Description:** Retrieves all saved locations for the account specified by the user id. In previous versions, this route included pagination functionality, but due to the requirements of a frontend library, the backend now returns all saved locations.

- Access:** Restricted

- Parameters:** User UUID (path parameter)

- Returns:** JSON object containing a list of all saved locations (including location, radius, number of amenities) for the specified user.

- **Name:** Delete saved locations
Method + Route: DELETE /v1/public/history/{id}
Description: This route allows the user to delete one or more of their saved locations. The route only deletes saved locations that are associated with the currently active account. Should this route receive ids of saved locations that are associated with another account or that don't exist, it simply ignores them.
Access: Restricted
Parameters: User UUID (path parameter), List of saved location ids (request body)
Returns: Status 202 if no errors occurred during deletion.
- **Name:** Save location
Method + Route: POST /v1/public/history/{id}
Description: Creates a new saved location for the currently active user.
Access: Restricted
Parameters: User UUID (path parameter), JSON Object describing the saved location (request body)
Returns: Status 201 if the location was saved successfully.

- **Authentication Routes**

- **Name:** Create new account
Method + Route: POST /v1/public/auth/User/
Description: Allows users to create a new account. Both the backend and database will perform uniqueness checks on the given email and username. First name and last name are optional. The given password will be hashed before being stored in the database.
Access: Public
Parameters: JSON object containing all information for the new account (request body)
Returns: The UUID of the new account and status 201 if the creation was successful.
- **Name:** Login
Method + Route: POST /v1/public/auth/User/login
Description: Allows users to retrieve an access token, given valid credentials.
Access: Public
Parameters: JSON object containing a password and a username or email.
Returns: A JWT bearer token and the user UUID if the login was successful.
- **Name:** Get user details
Method + Route: GET /v1/public/auth/User/{id}
Description: Retrieves the user details for the current user. This includes the users first name, last name, date of registration, date of last login as well as a link to a profile picture. Requests for the details of other users are denied.
Access: Restricted
Parameters: User UUID (path parameter)
Returns: JSON object containing additional details about the user.

- **Name:** Update user details

Method + Route: PUT /v1/public/auth/User/{id}

Description: Allows the user to update their own user details. With the exception of a new password, all parameters must be passed to this route, even if they aren't changed. New usernames and emails are checked for uniqueness by the backend and database. Keeping the same username or email does not violate the uniqueness constraint.

Access: Restricted

Parameters: JSON object specifying username, email, [password], first name, last name and profile picture link.

Returns: Status 202 if the data was successfully updated.

- **Name:** Delete account

Method + Route: DELETE /v1/public/auth/User/{id}

Description: Allows the user to delete their account. This also deletes all associated data like user details and saved locations. There is currently no way to undo this action.

Access: Restricted

Parameters: User UUID (path parameter)

Returns: Status 202 if the request was accepted.

- **Status Routes**

- **Name:** Heartbeat

Method + Route: GET /heartbeat

Description: This route is used by the automated deployment to check if the public backend is up and ready to receive connections. This helps alleviate issues with the starting order of services on a Kubernetes cluster.

Access: Public

Parameters: None

Returns: true if the service is ready to receive connections.

Private API

The private API is used by the machine learning and data stack to insert new datasets into the database. Currently, all private API routes have public access rights, as the private backend as a whole is guaranteed restricted access by the CoreDNS routing of the Kubernetes cluster. This allows the data stack to skip the authentication step, decreasing development overhead while maintaining high application security.

- **Point Routes**

- **Name:** Create new point

- Method + Route:** POST /v1/private/points

- Description:** Allows the data stack to create a new point in the database. Since the point details vary from point type to point type, they are stored in the database in the form of a JSON object. This also allows for less restrictions to the type of data that can be added here, making Magpie more flexible in development.

- Access:** Public

- Parameters:** JSON object containing the location of the point, the point type and a nested JSON object containing all the point details.

- Returns:** The integer id of the newly created point.

- **Name:** Update point

- Method + Route:** PUT /v1/private/points/{id}

- Description:** This route allows the data stack to update a point that is already stored in the database. This is not currently used in Magpie, but it will be useful in the future as datasets will become outdated over time.

- Access:** Public

- Parameters:** Point id (path parameter), JSON object containing the points new data

- Returns:** The point id and Status 202 if the update was successful.

- **Name:** Delete point

- Method + Route:** DELETE /v1/private/points/{id}

- Description:** Using this route, the data stack can delete a specific point from the database. This is currently not used in Magpie, but removing stale and outdated data will become necessary as time goes by.

- Access:** Public

- Parameters:** Point id (path parameter)

- Returns:** Status 202 if the request was accepted.

- **Status Routes**

- **Name:** Heartbeat

- Method + Route:** GET /heartbeat

- Description:** This route is used by the automated deployment to check if the public backend is up and ready to receive connections. This helps alleviate issues with the starting order of services on a Kubernetes cluster.

- Access:** Public

- Parameters:** None

- Returns:** true if the service is ready to receive connections.

4.6.7 Middlewares

Authorisation

The backend implements three middlewares that allow for route-based authorisation. Using this, some routes can be publicly accessible while others are only available to authenticated users in general or specific users only. This is accomplished by wrapping the handler function in the appropriate middleware when configuring the routing, see Listing 6.

The *public access middleware* simply forwards the request to the handler. This is used for routes that all users need access to, such as logging in or account creation. The *authenticated access middleware* extracts the JWT from the request and checks it for validity. Only if the JWT is valid is the request passed to the handler, if not, an error is returned.

The *protected access middleware* only allows requests to user specific resources that are made by that exact user. All routes with protected access are also checked by the authenticated access middleware. The user id is then extracted from the proven valid JWT. Comparing the extracted id with the user id given in as a path parameter, it can be determined if the user making the request is actually the owner of the resource. In that case, the request is forwarded to the handler function. If the ids do not match, the request returns an error.

Logging

The backend uses logs for monitoring purposes. The servers output informative messages, warnings and errors. The latter two are essential for figuring out any configuration issues with the server and are vital for debugging purposes. In addition, every request that is handled by the backend is logged including its status, route and precise execution time. This can be used to pinpoint requests that are failing with regularity or take significant time to complete. An example of the logs produced by the public backend server can be found in Figure 61.

```
2024/12/07 13:14:38 Warning: DATABASE_NAME not found in environment
2024/12/07 13:14:38 Attempting to connect to database
2024/12/07 13:14:39 Checking for new database migrations
2024/12/07 13:14:39 Database is up to date
2024/12/07 13:14:39 Successfully connected to database
2024/12/07 13:14:39 Listening on port 8080
2024/12/07 13:16:34 200 POST /v1/public/auth/User/login 399.438958ms
2024/12/07 13:16:35 200 GET /v1/public/points/inRadius 286.464792ms
2024/12/07 13:19:13 200 GET /v1/public/points/inRadius 21.044292ms
2024/12/07 13:19:15 200 GET /v1/public/points/inRadius 13.292083ms
2024/12/07 13:19:18 200 GET /v1/public/points/inRadius 8.44525ms
2024/12/07 13:19:19 200 GET /v1/public/points/inRadius 9.331166ms
2024/12/07 13:19:20 202 POST /v1/public/history/57ef90ed-6e4a-46d0-a4e1-3fdcb692461e 83.989833ms
2024/12/07 13:19:23 200 GET /v1/public/history/57ef90ed-6e4a-46d0-a4e1-3fdcb692461e 59.1255ms
```

Figure 61: Example of logs produced by the public backend server

Cross-origin Resource Sharing (CORS)

Since the frontend and the backend are not hosted at the same origin, it is necessary for the backend to enable cross-origin resource sharing. If this is not done, the user's browser will block any requests made to the backend. There was no need for custom CORS headers, so in the interest of simplicity an existing library (**Go CORS handler**) was chosen for this purpose.

4.6.8 Data Validation

Data that enters Magpie is validated on ingress. At the most basic level, there are checks for the presence of required parameters. But the data is also checked for logical errors, such as coordinates having a longitude greater than 180/less than -180 degrees or a latitude greater than 90/less than -90 degrees.

These checks are implemented in the data types that the incoming data is decoded into called **Data Transfer Objects (DTOs)**. Since Magpie uses JSON objects to transmit data, the decoding is handled by the built-in `encoding/json` library. Should any parameter not match up with the expected struct, the decoding fails. This is the first line of defence. In addition, all DTOs implement a `Validate` function, that can be used to specify any additional requirements for the incoming data. It is called automatically once the decoding process has finished. An example of a DTO definition can be seen in Listing 7.

While these checks cannot guarantee that the inserted data is actually correct, they do improve data integrity. During development, the sanity checks have proven to be instrumental in catching errors early when implementing a new function that sends data to the API.

```
type CreateUserDto struct {
    Username      string    `json:"username"`
    Email         string    `json:"email"`
    Password      string    `json:"password"`
    FirstName     string    `json:"firstname"`
    LastName      string    `json:"lastname"`
    ProfilePicture pgtype.Text `json:"profilepicture"`
}

func (self *CreateUserDto) Decode(r io.Reader) error {
    err := json.NewDecoder(r).Decode(&self)
    if err != nil {
        return customErrors.Payload.InvalidPayloadUserError
    }
    return self.Validate()
}

func (self *CreateUserDto) Validate() error {
    if len(self.Username) == 0 {
        err := customErrors.Parameter.
            RequiredParameterMissingError.
            WithCause(fmt.Errorf("Username is required"))
        return err
    }

    if len(self.Email) == 0 {
        err := customErrors.Parameter.
            RequiredParameterMissingError.
            WithCause(fmt.Errorf("Email is required"))
        return err
    }

    if len(self.Password) == 0 {
        err := customErrors.Parameter.
            RequiredParameterMissingError.
            WithCause(fmt.Errorf("Password is required"))
        return err
    }

    if len(self.Password) > 72 {
        err := customErrors.Payload.PasswordTooLongError
        return err
    }
    return nil
}
```

Listing 7: *An example of a DTO including data validation used by the backend*

4.6.9 Error Handling

At the start of the project, errors in the backend were communicated in the form of HTTP status codes. This is the easiest solution for passing errors to the frontend, so it was chosen in the interest of high development speed. But as the backend grew in complexity, many errors were returned using the codes 400, 401, 404 or 500. Since each of these could be caused by a multitude of different error conditions, it lead to a lot of manual searching for bugs involving both the frontend and backend team. It became obvious that errors in the backend needed to be communicated clearly and automatically.

To achieve this goal, all responses from the backend were wrapped in a response object. This structure holds either an error or the response content. Listing 8 shows a successful request, the error field is empty while the response field is populated. Listing 9 shows the response to a malformed request, the error field is populated with a custom error code and an error message for additional detail. The response field is empty. This allows the frontend to react appropriately to specific errors.

All custom error codes were thoroughly documented in a markdown file and available to the frontend developers throughout the project. An excerpt of this documentation can be seen in Figure 62. This prevented miscommunication and eliminated the need for joint bug hunting meetings, decoupling the development of frontend and backend further.

```
{
  "error": null,
  "response": {
    "content": {
      "id": "c251238f-aca3-4898-bd01-9674f5a88948",
      "registerdate": "2024-10-08T15:06:56.047718Z",
      "firstname": "Testy",
      "lastname": "McTesterson",
      "profilepicture": "http://example.com/profilepicture.jpg",
      "lastloggedin": "2024-10-16T09:54:01.79553Z"
    }
  }
}
```

Listing 8: *An example of a response DTO used for the successful retrieval of user details*

```
{
  "error": {
    "errorCode": 1202,
    "errorMsg": "Parameter invalid, expected type UUIDv4"
  },
  "response": null
}
```

Listing 9: *An example of a response DTO transmitting a custom error condition*

Errors

These are the errors that the backend may send as a response to a request by the frontend. If no error object is sent by the server, the server encountered an error while encoding the response and a HTTP Status Code of `500 Internal Server Error` is sent instead.

Custom Error Code	HTTP Response Status	Name	Description	Additional Information
1001	500 Internal Server Error	UnknownError	Unknown internal server error	This error is used when the reason for the error is not clear or no custom error has been implemented yet. If you can reproduce these errors, please create a new issue
1011	500 Internal Server Error	JwtSecretMissingError	Could not generate JWT, secret not set	JWT secret could not be read from environment variables
1012	500 Internal Server Error	HashingError	Could not hash password	An error occurred while hashing the password
1013	500 Internal Server Error	JwtParseError	Could not parse JWT	An error occurred while parsing the JWT
1021	500 Internal Server Error	JsonEncodingError	Could not encode json	An error occurred while encoding the json payload
1022	500 Internal Server Error	JsonDecodingError	Could not decode json	An error occurred while decoding the json payload

Figure 62: Excerpt from the Error Code Documentation for the Backend

4.6.10 Configuration

The backend servers are configured through environment variables. These can be provided in a `.env` file, which the backend will automatically load on startup.

The backend implements a system that allows for environment variables to be optional and for default values to be set. If the backend tries to access a variable that is only covered by a default value, a warning will be added to the logs to notify users of potentially unintended configurations (see Line 1 of Figure 61).

Making the backend fully configurable using environment variables makes deployment and configuration much easier – especially when compared to hard-coded values. This is also advantageous during development. Development environments can vary wildly between developers and this system allows each developer to have a configuration that fits their needs.

The environment variables were documented during development, no changes were made without updating the documentation. The readme file for the backend contained a list of all environment variables (see Figure 63).

The variable `MAGPIE_DB_URL` can be omitted if the connection information is instead given as separate environment variables (`LOGIN`, `PASSWORD`, `HOST` and `DATABASE_NAME`). This is used for deployments using Kubernetes. If both `MAGPIE_DB_URL` and the separate environment variables are set, only `MAGPIE_DB_URL` will be used to establish a connection.

Variable Name	Description	Default	Optional
MAGPIE_DB_URL	A valid PostgreSQL connection URL	-	No*
MAGPIE_PORT	The port the server will listen on	8080	Yes
MAGPIE_JWT_SECRET	The secret used to generate JWTs for authentication	-	No
MAGPIE_JWT_EXPIRY	The expiry time for JWTs (must be in hours or minutes)	24h	Yes
MAGPIE_CORS_ALLOWED_ORIGINS	Space separated list of allowed origins for CORS	-	Yes
MAGPIE_CORS_ALLOWED_METHODS	Space separated list of allowed methods for CORS	-	Yes
MAGPIE_DB_RETRIES	The number of times the backend tries to connect to the database on startup	6	Yes
MAGPIE_DB_RETRY_INTERVAL	The interval in seconds between database connection retries on startup	10	Yes
MAGPIE_SHUTDOWN_TIMEOUT	The amount of seconds the server has to handle existing connections before shutting down	10	Yes

Figure 63: *The environment variables available for configuring the backend*

4.6.11 Testing

During development, both backends were outfitted with automated unit testing. This allowed for bugs to be caught early and consistently. The tests covered all vital functions of the backends, namely the routing and – most importantly – the request handlers.

Implementing unit tests also allows for future regression testing, making it easier to spot issues like increased latency between versions of the backend.

The test implementation was done using the `testing` package from the Go standard library. While there are more advanced alternatives like `testify`, they were considered to be too complex for a project of this scope.

In addition to the standard library, `pgxmock` was used to simulate a database connection during test execution. This made tests significantly simpler to set up and implement, as no actual database connection needed to be maintained for the tests to be effective. Additionally, `pgxmock` provides a way to expect certain queries to be executed, which is a valuable tool for unit tests.

So called table-based tests were used to validate the backends functionality. For this, a list of inputs (i.e. a table) is defined, which will then be passed to the function to be tested one by one. Each definition also comes with success and failure conditions (see Listing 10). Using this, it is possible to efficiently create a multitude of testcases with different parameters which share the underlying test execution code.

```
{
  Name: "Valid input",
  Method: "POST",
  Route: "/points",
  InputJSON: `{
    "longlat": {
      "type": "Point",
      "coordinates": [11, 12]
    },
    "type": "parking",
    "details": {
      "test": 1234
    }
  }`,
  MockSetup: func(mock pgxmock.PgxPoolIface) {
    mock.ExpectQuery("INSERT INTO points").
      WithArgs(
        pgxmock.AnyArg(),
        db.PointType("parking"),
        []byte(`{"test":1234}`),
      ).
      WillReturnRows(pgxmock.NewRows([]string{"id"}).
        AddRow(int64(1)))
  },
  ExpectedStatus: http.StatusCreated,
}
```

Listing 10: *An example of a test case for a table-based unit test of the private points handler*

To discourage pushing or merging untested code, the decision was made to automatically run the defined test cases when changes to the backend are pushed to the repository. For this project, GitHub Actions was used as the CI/CD solution. At the time of writing this, there is unfortunately no built-in support for Go unit tests in GitHub Actions.

For this reason, a custom GitHub Action workflow was set up. It automatically runs the test cases and extracts the test results and code coverage. To make it easy to check the success of the test cases, the workflow then reports these metrics as comments on the relevant pull request (see Figures 64 and 65). The workflow is configured in a way where any failed tests or insufficient code coverage lead to the action failing. Since the repository expects all checks to pass before merging is permitted, this effectively prevents issues to enter the main code base and by extension the deployment.



Figure 64: Comments created by custom GitHub action for executing backend unit tests

github.com/2024-CMPU9010-GROUP-3/magpie/internal/util/testutil/handler.go:155	RunRawRequestTests	100.0%	✓
github.com/2024-CMPU9010-GROUP-3/magpie/internal/util/testutil/middleware.go:36	executeMiddlewareTest	81.8%	✓
github.com/2024-CMPU9010-GROUP-3/magpie/internal/util/testutil/middleware.go:79	executeLoggingTest	71.4%	⚠
github.com/2024-CMPU9010-GROUP-3/magpie/internal/util/testutil/middleware.go:106	RunLoggerTests	100.0%	✓

Figure 65: Excerpt from the hidden section on a coverage comment (see Figure 64), providing detailed, function level coverage

4.7 Conclusion Technical Solution

The **Magpie** platform demonstrates how emerging technologies can address complex urban planning challenges. By combining machine learning, a user-friendly interface, and a robust backend, it makes public amenity data more accessible and actionable. This integration reshapes how urban planners and other stakeholders interact with and interpret city data, providing a tool that empowers users with diverse technical backgrounds to gain valuable insights. As a result, **Magpie** helps inform better decision-making in urban development.

One of the platform's standout achievements is its ability to seamlessly integrate data from various sources into a unified, easy-to-navigate interface. Through automation, it handles tasks such as data extraction, cleaning, and visualization, offering a clear, comprehensive view of public amenities. The combination of geospatial data, machine learning, and interactive maps creates a flexible system that can evolve alongside changing urban planning needs.

A critical component of the platform's success is its use of scalable machine learning models, such as YOLOv8 for object detection. These models enable the platform to efficiently identify public amenities like parking spaces, eliminating the need for time-consuming manual data collection. Additionally, the platform's use of oriented bounding boxes enhances detection accuracy, allowing it to adapt to diverse geographical contexts. This scalability ensures the platform can handle growing amounts of data as cities expand, positioning it as a forward-thinking tool for urban planning.

Supporting this functionality is the platform's robust backend infrastructure, built with Go and PostGIS for geospatial data. This foundation allows for efficient data processing and search capabilities, ensuring users always have access to accurate information. The backend is designed with future growth in mind, ensuring the platform can handle increasingly complex data sets as urban environments become more intricate.

The development process wasn't without its challenges, particularly in terms of data variability, computing limits, and resolution issues. The team addressed these hurdles with creative solutions, developing data processing systems that could handle variations in geographic precision, data formats, and update frequencies. By working with smaller datasets and adjusting settings, they balanced training and accuracy, and advanced techniques helped overcome the limitations of low-resolution satellite imagery, enhancing the platform's overall reliability.

Throughout the development, continuous improvement was a key theme. Testing and refining features like object detection and parking spot identification allowed the team to steadily enhance the platform. Input from users and experts played a crucial role in improving usability, ensuring the platform is accessible to both technical and non-technical users.

Looking ahead, the **Magpie** platform is poised for further growth. With planned advancements in machine learning models, data integration, and user features, the platform is ready to expand its capabilities. These improvements will strengthen its impact on urban planning, supporting more sustainable resource management and ensuring equitable access to amenities. As cities continue to grow, **Magpie** can help planners design smarter, more efficient urban environments, shaping the future of urban data accessibility and analysis.

5 Software Management

5.1 Why is Software Management important?

In Software Development, it is said that the speed of work is often dictated by the quality of the tools used to perform that work. Knowing this, *Magpie* makes use of a number of tools to ensure that the development process is as efficient as possible.

This chapter will discuss the tools we used to create *Magpie* and how they were used to manage the project. We will also discuss the importance of the tools used and how they helped aid development and maintain a solid pace.

5.2 What is DevOps?

“DevOps is about fast, flexible development and provisioning business processes. It efficiently integrates development, delivery, and operations, thus facilitating a lean, fluid connection of these traditionally separated silos” Ebert et al., 2016

DevOps is a set of principles and practices that combines software development (**Dev**) and IT operations (**Ops**). The primary means of reducing friction between development and operations is through automation, and the use of tools to multiply progress, and reduce effort. This can be broadly sorted into two categories:

- **Continuous Integration (CI)**: This is the practice of automatically building and testing code every time a developer commits changes to version control. This ensures that the code is always in a working state. Ebert et al., 2016.
- **Continuous Deployment (CD)**: This is the practice of automatically deploying code to production servers every time a change is made. This ensures that the application is always up-to-date and that new features can be released quickly. Ebert et al., 2016.

The next sections will detail the tools used to implement **Continuous Integration** and **Continuous Deployment**.

5.3 Continuous Integration

In *Magpie*, the CI workflow is focused around **GitHub** and it's suite of tools. We decided on using this approach, because it enables a powerful flow:

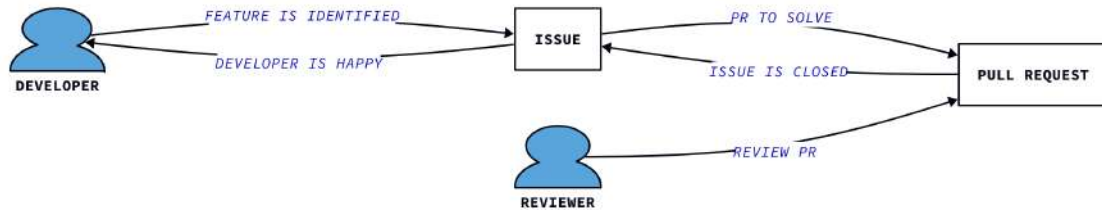


Figure 66: *The flow of work in Magpie*

5.3.1 Issues

GitHub Issues are used to track tasks, enhancements, and bugs to be worked on. In our experience, they are an effective way to keep track of what needs to be done, and what has been done. **GitHub Issues** can be assigned to team members, and can be linked to **Pull Requests**.

Issues represent the primary unit of work in projects using **GitHub**'s flow. In the below example, a checklist is presented- with each item indicating a task that needs to be completed. An issue's checkboxes, can also be assigned to **Pull Requests**.

On the right, the issue displays the **Assignees**, **Labels**, and **Projects** associated with the issue. This allows for easy filtering and sorting of issues, and helps to keep the project organized.

Under the description, the issue keeps track of edits, comments, and other interactions. This allows for communication to stay in context with the issue being discussed, and requirements- or additional information- to be added as needed.

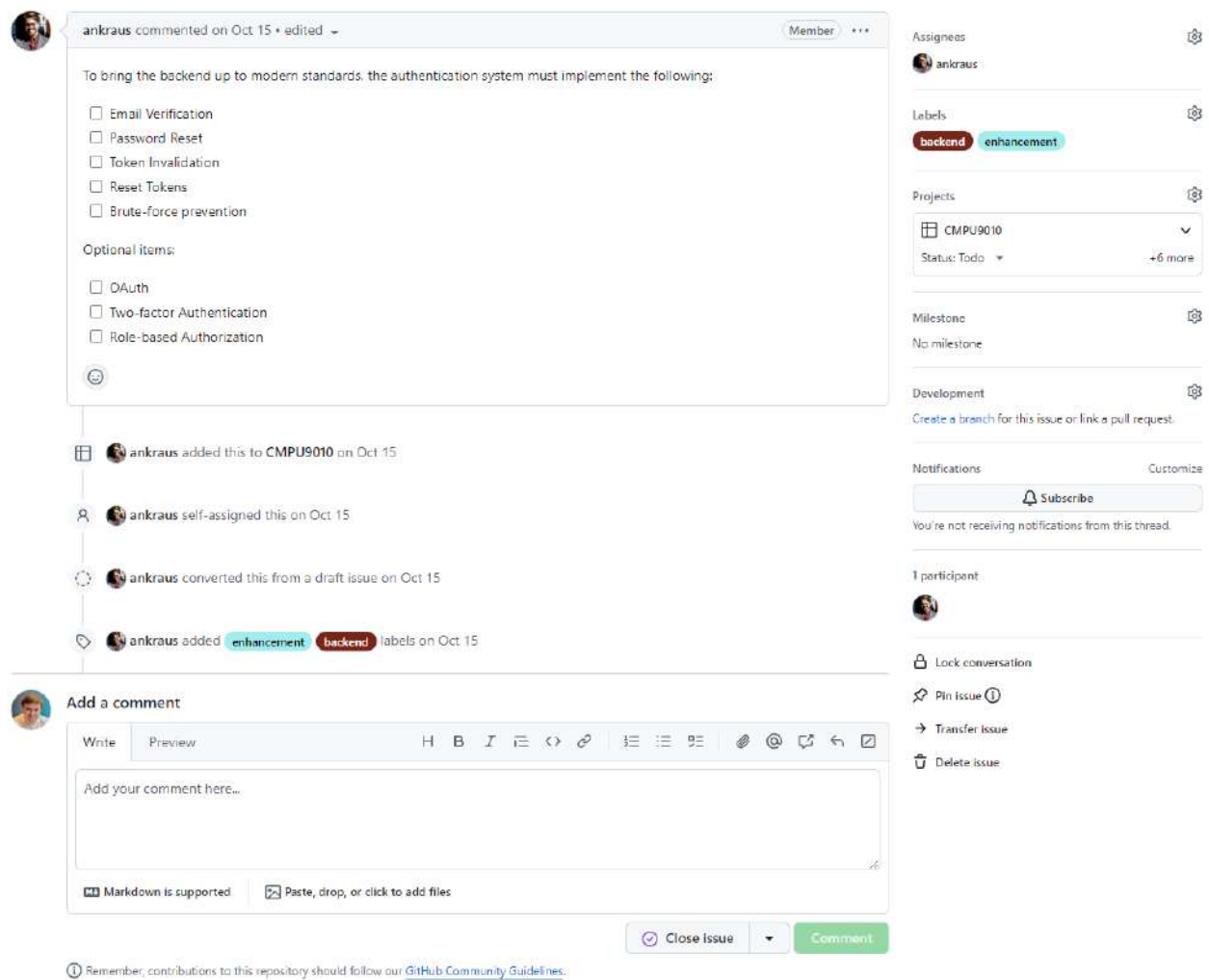


Figure 67: An example of a GitHub Issue

In general, issues are an efficient way to keep track of what needs to be done, however, in our experience, they can be tedious to write manually. To help with this, templates can be used.

To use a template, create a directory called **.github** in the root of the project, then create another directory called **ISSUE_TEMPLATE**, note that the capitalization is important. Inside this directory, any markdown file created in the prescribed format can be used as a template for issues, you can have as many templates as you like.

In *Magpie*, we used a template for **Bug Reports**, **Documentation Requests**, and **Feature Requests**. This reduced the friction of creating issues by minimising the amount of effort related to drafting an issue (using the template) and creating clear delineations between different categories of issues.

```
---
name: Bug Report
about: Create a report to help us improve
title: ''
labels: bug
assignees: ''
---

**Describe the bug**
A clear and concise description of what the bug is.

**To Reproduce**
Steps to reproduce the behavior:

1. Go to '...'
2. Click on '....'
3. Scroll down to '....'
4. See error

**Expected behaviour**
A clear and concise description of what you expected to happen.

**Screenshots**
If applicable, add screenshots to help explain your problem.

**Additional context**
Add any other context about the problem here.
```

Listing 11: *An example of an issue template used in Magpie*

5.3.2 Pull Requests

Pull Requests are used to merge code from one branch to another. In our experience, they are a great way to review code, and ensure that it is up to standard. They can be linked to **Issues**, and like issues, can be assigned to team members.

In the below example, the interface is largely similar to that of an issue, however there are a few key differences. Note that under projects, you can set a task that will be present in the Kanban board that will be discussed in the next section.

If this PR solves a milestone, or contributes to an issue containing a milestone, it will be displayed here. *Magpie* did not use milestones with the exception of the interim report, however we feel it can be valuable in projects with more clearly defined goals, such as major releases.

Note, that the pull requests we used also included automated CI jobs. These will be discussed in a later section.

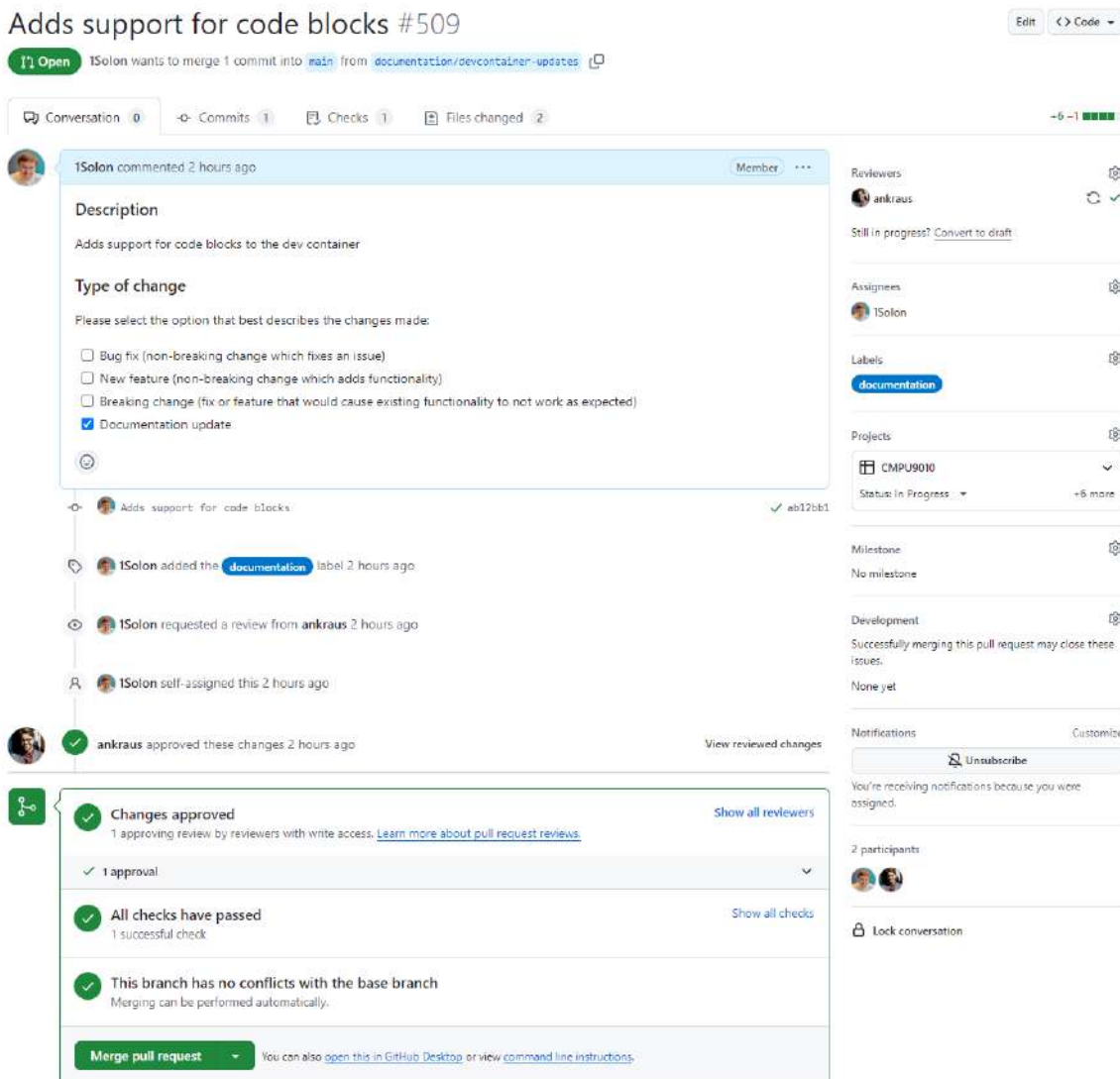


Figure 68: An example of a GitHub Pull Request

Like **Issues**, *Magpie* leverages **Pull Request Templates** to reduce friction. Templates reduce friction by minimising the amount of writing necessary when drafting a PR.

During the project, we at times ignored issue templates because because the benefits of writing fully-detailed issues, did not always return greater utility. However, we never ignored pull request templates.

PRs need a high level description of the changes made, and having a consistent format for this description was important to ease the burden on code reviewers, particularly working in a team environment.

Description

Replace this with a summary of the change. Please also include relevant motivation and context.

Fixes # (issue)

Type of change

Please select the option that best describes the changes made:

- ☐ Bug fix (non-breaking change which fixes an issue)
- ☐ New feature (non-breaking change which adds functionality)
- ☐ Breaking change (fix or feature that would break existing functionality)
- ☐ Documentation update

Changes

Replace this with a list of changes made in the pull request.

Listing 12: *An example of a pull request template used in Magpie*

5.3.3 Projects

GitHub Projects are used to track the progress of work in a project. In our experience, they are a great way to keep track of what needs to be done, and what has been done.

GitHub Projects can be used to create a **Kanban Board**. In our experience, this is an effective way to visualise the progress of work in a project. In the below example, the board is divided into columns, each representing a different stage of work.

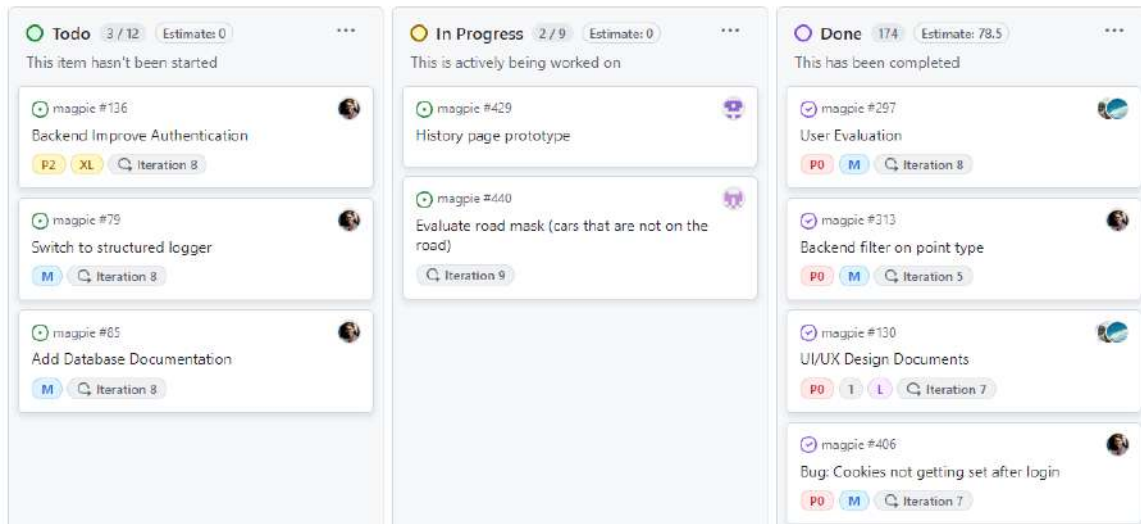


Figure 69: An example of a GitHub Kanban Board

In *Magpie*, we predominantly used the **Kanban Board** to provide a high-level overview of tasks within the project. In general, this allowed all team members to see what was being worked on, and what needed to be done.

As a card is just a representation of an issue, clicking on it will take you to the issue page. This closes the loop on the flow identified in the beginning of this section.

GitHub Projects includes a **Roadmap** tab. Unlike the **Kanban Board** tab, which gives an overview of task status, the **Roadmap** gives a temporal view of the timing associated with those tasks.

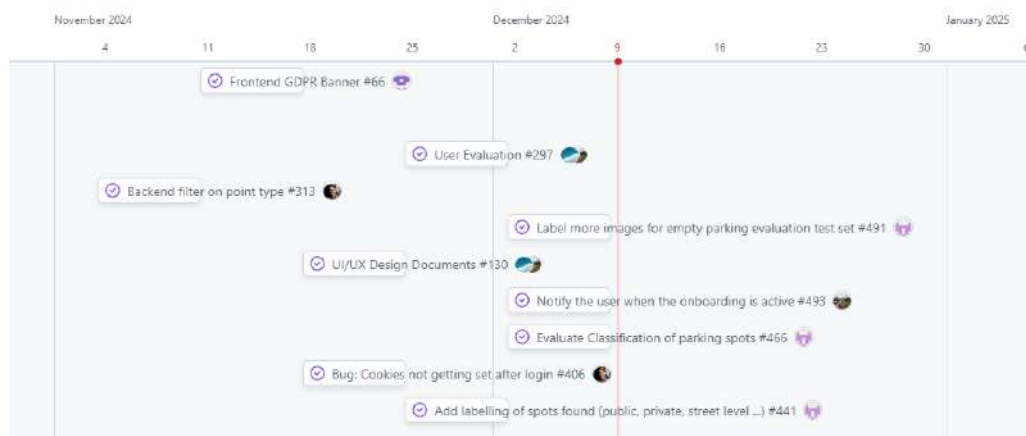


Figure 70: An example of a GitHub Roadmap

5.4 GitHub Actions

GitHub Actions are used to automate tasks in a project. In **Magpie**, we used **GitHub Actions** to automate tests, Housekeeping and building tasks. Housekeeping and tests fall comfortably within CI, however building is more closely aligned to CD- as such, it will be discussed in a later section.

5.4.1 Houskeeping

Magpie uses a branch naming scheme:

- **main**: The main branch of the project. This is the branch that is deployed to production.
- **backend/**: A branch containing backend features.
- **frontend/**: A branch containing frontend features.
- **documentation/**: A branch containing documentation.
- **python/**: A branch containing Machine Learning features.
- **update/**: A branch containing updates to the project.
- **misc/**: A branch containing anything not in the above categories.

This was initially done to trigger specific actions based on the name of the branch. However, it became an orderly means of seeing what branch was dealing with what task, in our experience, in projects like this with upwards of 8+ branches at a time, being able to, at-a-glance, determine what the branch was dealing with was very useful.

We decided against using branch rules to trigger actions because, by default, they do not function as expected. For example:

```
on:
  push:
    branches:
      - 'feature/*'
```

Listing 13: An example of a GitHub Actions workflow that will not work

From the above, you may expect that the workflow will trigger on any branches starting with **feature/**. However, this will only trigger on any branches being *merged into* **feature/**. This is not the desired behaviour, and as such, we decided against using this particular method.

To solve this problem, we instead triggered actions based on what *directory* was being changed. This was done by using the **paths** key in the workflow.

```
on:
  push:
    paths:
      - 'Backend/**'
```

Listing 14: An example of a GitHub Actions workflow that will work

In *Magpie*, to maintain our branch naming scheme, it was important to have automation in place to ensure that branches were named correctly. This was done using a **GitHub Action** that would check the branch name, and if it did not match the scheme, would fail the action.

Note that, in a **Pull Request**, certain actions will come up as checks. Checks will prevent a PR from being merged if they fail.

```
name: Branch Checks

on:
  pull_request:

jobs:
  validate-name:
    runs-on: ubuntu-latest

    steps:
      - name: Check out repository
        uses: actions/checkout@v4

      - name: Get branch name
        id: get_branch_name
        run: echo "branch=${GITHUB_HEAD_REF}" >> $GITHUB_OUTPUT

      - name: Validate branch name
        run: |
          BRANCH_NAME="${{ steps.get_branch_name.outputs.branch }}"
          if [[ ! "$BRANCH_NAME" =~ ^(
            backend/ frontend/|
            documentation/ python/|
            distribution/ misc/|
            update/
          ).* ]]; then
            echo "Branch name '$BRANCH_NAME' is not valid."
            echo "Rename the branch to match one of the following patterns:"
            echo "'backend/', 'frontend/',"
            echo "'documentation/', 'distribution/',"
            echo "'update/', 'python/',"
            echo "or 'misc/'."
            exit 1
          fi
    shell: bash
```

Listing 15: A github action that checks the branch name (this was edited for brevity)

The above action runs on every pull request, and checks the branch name. If the branch name does not match the pattern, the action will fail, and the PR will not be able to be merged.

5.4.2 Checks

GitHub Actions can be used to run checks on code. In *Magpie*, we used checks to run automated testing in the frontend and backend. The frontend and backend checks are discussed in those sections.

5.5 Continuous Deployment

5.5.1 Kubernetes

Kubernetes, often abbreviated as *K8s*, is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications.

- **Container Orchestration:** Kubernetes manages the lifecycle of containers, ensuring that applications are running as intended and automatically handling failures or restarts.
- **Scalability:** It supports horizontal scaling of applications through commands, resource monitoring, or automated scaling policies.
- **Load Balancing and Service Discovery:** Kubernetes provides built-in load balancing for distributing traffic across containers and automates service discovery for communication between microservices.
- **Self-Healing:** Kubernetes continuously monitors the health of containers, automatically replacing failed pods and rescheduling workloads on healthy nodes.
- **Declarative Configuration:** Kubernetes uses YAML files to define the desired state of the system, ensuring infrastructure-as-code principles.
- **Resource Management:** It allocates CPU and memory resources to containers based on defined requests and limits.

The architecture of Kubernetes consists of several components:

- **Master Node:** The master node is responsible for managing the overall cluster state. It comprises the following components:
 - *API Server:* Exposes the Kubernetes API, serving as the entry point for managing the cluster.
 - *Controller Manager:* Ensures that the cluster's desired state matches the actual state.
 - *Scheduler:* Assigns workloads (pods) to appropriate nodes based on resource availability and constraints.
 - *etcd:* A key-value store that maintains the cluster's state and configuration data.
- **Worker Nodes:** Worker nodes execute containerized applications. They consist of:
 - *Kubelet:* An agent that ensures containers are running as expected on the node.
 - *Kube-proxy:* Handles networking and forwards traffic to the appropriate pods.
 - *Container Runtime:* The software responsible for running containers (e.g., Docker or containerd).
- **Pods:** The smallest deployable unit in Kubernetes, containing one or more containers.

Kubernetes provides benefits for application development:

- **Portability:** Kubernetes supports any container runtime and runs across cloud providers, on-premises, or hybrid environments.
- **Automation:** Automates deployment, scaling, and recovery processes, reducing manual interventions.
- **Resource Optimization:** Efficiently manages resources, ensuring applications get the required CPU and memory.
- **Resiliency:** Enhances application uptime with features like auto-healing, rolling updates, and self-recovery.

We use Kubernetes in *Magpie*, for the following reasons:

- **Scalability:** *Magpie*'s backend services can automatically scale to meet the demands of varying workloads.
- **High Availability:** Ensures the application's critical services remain operational even during node failures.
- **Efficient Resource Allocation:** Kubernetes optimizes resource usage, enabling cost-effective deployment in cloud environments.
- **Simplified Updates:** Enables rolling updates for new features or patches without downtime.

5.5.2 Flux

Flux is a declarative, continuous delivery tool designed for Kubernetes, enabling GitOps practices for automated software deployment and management. By treating Git repositories as the single source of truth for a system's desired state, Flux synchronizes infrastructure and application configurations with the actual state of a Kubernetes cluster. This approach enables version control for Kubernetes clusters and promotes consistency.

At the heart of Flux is its ability to automate the reconciliation of Kubernetes manifests stored in Git. Flux monitors changes to the specified repository and ensures that the cluster reflects the desired state defined in these manifests. It eliminates the need for manual intervention during deployments, as any approved changes committed to the repository are automatically applied to the cluster.

One of the most notable advantages of Flux is its alignment with the GitOps methodology. In traditional CI/CD pipelines, application configurations and environments can drift due to manual overrides or unchecked changes. Flux mitigates this risk by enforcing the Git repository as the sole authority for configurations.

In a software project like *Magpie*, where speed of work and reliability are paramount, Flux provides an excellent solution for managing deployments. By automating synchronization with Git repositories, Flux reduces manual effort and ensures that infrastructure changes are consistently applied. Its declarative nature allows for rapid recovery in case of failures, as clusters can be restored to a known good state defined in version-controlled repositories.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: public-backend-deployment
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
      type: RollingUpdate
  selector:
    matchLabels:
      app: public-backend
  template:
    metadata:
    labels:
      app: public-backend
    spec:
      containers:
        - name: public-backend
          image: ghcr.io/2024-cmpu9010-group-3/backend-public:0.12.0
          resources: {}
          ports:
            - containerPort: 8080
          livenessProbe:
            exec:
              command:
                - curl
                - --fail
                - http://localhost:8080/heartbeat
              failureThreshold: 1
              periodSeconds: 30
            startupProbe:
              exec:
                command:
                  - curl
                  - --fail
                  - http://localhost:8080/heartbeat
              failureThreshold: 30
              periodSeconds: 10
```

Listing 16: *Example of a Kubernetes Deployment for the public backend*

5.6 Devcontainers

Devcontainers are container-based development environments that enables developers to work consistently across different machines and operating systems. By making use of containerization technologies, such as Docker, devcontainers contain all dependencies, tools, and configurations required for a project within a portable and reproducible environment. This approach significantly reduces the common "works on my machine" problem, as developers can ensure that their local development environment mirrors the deployment environment or the environments used by their teammates.

For the *Magpie* project, a LaTeX-based devcontainer was implemented to streamline the production and maintenance of the project's academic documentation. The LaTeX devcontainer provided a pre-configured environment with all the necessary tools, including the TeX distribution, LaTeX compiler, and essential packages required for document rendering. By containing the LaTeX environment in a container, team members could contribute to the report without manually installing or configuring LaTeX on their local machines. This eliminated compatibility issues arising from different LaTeX distributions or versions and ensured that all team members could compile the report(s) identically.

Using the LaTeX devcontainer, the team was able to focus on writing and formatting the document rather than troubleshooting configuration problems. The devcontainer integrated directly with Visual Studio Code, providing an editing experience with features such as syntax highlighting, and real-time rendering previews.

Additionally, the use of a LaTeX devcontainer aligned with the broader goals of reproducibility within the project. All configuration files, such as the `Dockerfile` and `devcontainer.json`, were stored in the project's Git repository. This ensured that the environment remained versioned alongside the project.

6 User Evaluation

6.1 User experience design & testing

Usability is a ‘*quality attribute that assesses how easy interfaces are to use*’ (Nielsen, 2012). The best way to assess the usability of an interface is to conduct usability testing.

Also referred to as *user testing* is a popular UX research methodology used to uncover problems in the user interface design of an application and learn about the target user’s preferences (Moran, 2019).

There are 3 core elements to user testing: **the facilitator**, **the participant** and **the tasks**. The facilitator administers the tasks to the participant, observes their behaviour and listens to their feedback.

Our approach is as follows:

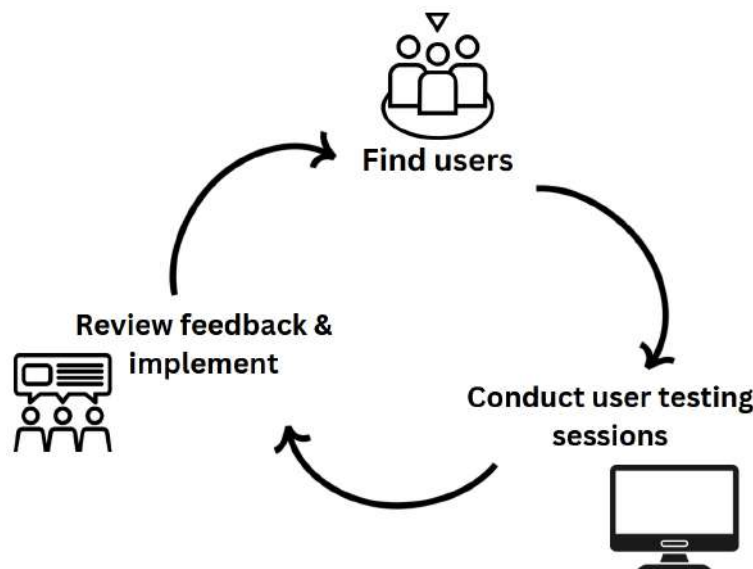


Figure 71: User Evaluation Process

Firstly, users from the research survey who left their contact were recruited to participate in user testing. Additionally, a call for participation was made through social media channels to seek out other users.

Next, online user testing sessions were conducted to discuss Magpie, explore the features and gather feedback. The notes taken during the sessions were then synthesized and reviewed to gather a list of new features to implement, items to change and content to remove. Lastly, the process started again in order to continue improving Magpie’s workflow and interface.

There were 11 in-depth interviews in total, that have been divided into the following categories: 6 Non-Expert Users, 3 Domain Expert, 1 UI/UX expert and 1 accessibility expert. *Non-Expert Users* are defined as those who use Magpie casually for personal interests. *Domain Experts* are defined as those who use Magpie as a tool for their work.

It is worth noting that some users have decided to remain anonymous, therefore have been labelled as *Anonymous n°*, while others will be referred to by their name.

Both controlled & uncontrolled approaches were used for the sessions. The controlled sessions were based around a strict list of tasks the user would complete and used metrics such as time taken, difficulty and task success rate. The uncontrolled sessions let the users freely roam the application while we observed their behaviour interacting with each element and initiate discussion to obtain feedback on features to improve.

A table with a list of general tasks is used to quantitatively evaluate each feature the user interacted with. Metrics measured are task difficulty and task success rate. The list of general tasks increased as the test sessions went on because new features were being added iteratively.

The difficulty of the task is related to its status and how much time a user spent on it. The status of a task can either be '*complete*', '*pass*', or '*fail*' where:

- *complete* is attributed when the user completes the task on their own. This is worth 1 point.
- *pass* is attributed when the user was able to complete the task but with our help. This is worth 0.5 points.
- *fail* is attributed when the user was not able to complete the task even with our help. This is worth 0 points.

Lastly, a short satisfaction survey in Figure 96 is administered at the end of each session to quantify user experience and provide a baseline for improvements. User behaviour is also observed during the session to complement these quantitative metrics.

The evaluation of Magpie has been divided into the following sections:

1. Non-Expert Users test sessions
2. Domain Experts test sessions
3. UX/UI expert review
4. Accessibility expert review

6.2 Non-Expert User evaluation

6.2.1 Non-expert User 1 - Brendan

Brendan is a professor with technological background. They are considered a non-expert user. An uncontrolled test session was conducted where they tested each feature of Magpie to find faults, system failures and bugs.

Main takeaways from Brendan's session: Magpie has potential for use by certain types of users but needs a lot more functionality. Here is a breakdown of the feedback:

- **Log in/Sign up:** email verification is important, especially for a service advertised towards professionals. Also, Brendan did not understand the need for username to be created, the username should automatically be the email address. As a group, we decided to keep the username field as a preference and for future work as the user will be greeted by the username chosen when the log in,, or it will display in the profile menu.
- **Tutorial:** the content of step 2 needs to be reworked and use more descriptive language. Also, the positioning of certain elements is off in Firefox browser. For step 4, the word 'dozen' should be changed to 'several' or similar especially if we are planning on adding more amenity data. Lastly, step 5 should point to the map, so that is a bug we need to investigate bug.
- **Dashboard:** we should implement a button to clear the marker and all the points from the map, right now the user is forced to reload the page to do that. In addition, it could be beneficial to certain users to leave the count of toggled off amenities. Also, compress the list of amenities to avoid scrolling. We found Firefox to cause a window size issue that we have been unable to address.
- **Map:** plus and minus buttons should be added to zoom in and out of the map, especially for users who are not familiar with mouse technology. Also, clicking on amenity icons should provide more information. Right now, it feels a bit empty. Lastly, some of the selected icons are not visually striking, perhaps find a way to emphasize them.
- **Technical:** when clicking on map, there is a slight offset between where the marker appears and where the user clicked. There is cause for investigation there. Also, maybe think of implementing something to avoid mis-clicking and loosing the original marker position.
- **Miscellaneous:** Brendan asked why location was being requested and what it was used for. Accepting Magpie to use the user's location only puts down their own location marker on the map indicating where they are located. The location information is not processed by the Magpie server, it is processed on the user's device. Additionally, a landing page is necessary to present Magpie and put forward the machine learning aspect of the project.

Overall: the interface is nice, but you might want to focus on producing a much more data-driven approach for the interface if you want to attract those kinds of users.

Average score from survey: 3.6 out of 5

Score breakdown: Appendix 97

6.2.2 Non-expert User 2 - Anonymous 1

This user has a background in technology at the doctorate level. This session was mostly uncontrolled, they browsed the application, tested the features and discussed their thoughts with us.

Main takeaways from the session: this user really enjoyed the presentation of information on the application, they found the amenities easy to understand and recognizable in the radius of the map. They tried to interact with the locked elements of the tutorial, and really enjoyed the confetti at the end of it. Also, before placing a marker on the map because when they accepted location tracking, their own marker appeared. To make more improvements, they suggested the following:

- **More features:** They feel like a search bar would help in the quest for information in specific location visually unknown to the user. They also thought the points would appear automatically on the map but instead she had to place her own marker. Perhaps there is an issue with onboarding not being retained.
- **Dashboard & Map:** There is an icon, the water fountain one that is neon blue, therefore blends into the white background of the dashboard and in the map; probably needs to be changed. Also, more data for example on transportation would be a big plus.
- **Miscellaneous:** If you want to appeal to more Non-Expert Users, adding more features like location sharing, integrating social interactions and make it mobile responsive would be the way to go. Also, a higher level of information.

Overall: it is a very good application, a very interesting idea to gather all this information in one place.

Average score from survey: 4.6 out of 5

Score breakdown: Appendix 98

6.2.3 Non-expert User 3 - Paul

Our next session was with Paul, a student in technological undergraduate degree. They gave their contact email in the research survey. A controlled approach was supposed to be used in this session, however they intuitively went on to explore the application on their own.

Table 6: Usability testing Tasks - Paul

Task	Status	Time taken	Difficulty	Errors
Load Magpie application	Complete	20s	1	N/A
Sign up	Complete	42s	1	N/A
Complete tutorial	Complete	60s	1	N/A
Place cursor on map and adjust radius to 250m	Fail	Skipped	Skipped	Skipped
Zoom in to road name level	Complete	5s	1	N/A
Place cursor on another area	Complete	5s	1	N/A
Zoom out to see full radius	Fail	Skipped	Skipped	Skipped
Filter to only view "Parking meter" data	Pass	120s	3	Required help
Filter to toggle off all amenities	Pass	37s	3	Required help
Go through tutorial and exit at Step 3	Pass	30s	3	Couldn't find icon
Log out	Complete	20s	2	N/A

Main takeaways from Paul's session: the map and the amenity data displayed is 'excellent', they would find it useful for local areas of the city. One aspect they advise we improve on is to make the choice of amenities more intuitive. This is further supported by their behaviour trying to click on the icon and amenity title on the dashboard to toggle it on and off, as well as the difficulties they encountered as shown in the general task table. Another point to improve on is to make the profile and tutorial icons more visible, demonstrated by the time they took to find them.

Average score from survey: 4.8 out of 5

Score breakdown: Appendix 99

6.2.4 Non-expert User 4 - Livia

Our next session was with Livia, another student in a technological undergraduate degree who left their contact in the research survey. This session also started out as a controlled test with a defined set of tasks, but just like Paul, Livia went on to explore the application skipping the tasks.

Table 7: *Usability testing Tasks - Livia*

Task	Status	Time taken	Difficulty	Errors
Load Magpie application	Complete	5s	1	N/A
Sign up	Complete	16s	1	N/A
Complete tutorial	Complete	44s	1	N/A
Place cursor on map and adjust radius to 250m	Complete	6s	1	N/A
Zoom in to road name level	Complete	8s	1	N/A
Place cursor on another area	Fail	Skipped	Skipped	Skipped
Zoom out to see full radius	Fail	Skipped	Skipped	Skipped
Filter to only view "Parking meter" data	Fail	Skipped	Skipped	Skipped
Filter to toggle off all amenities	Complete	18s	1	N/A
Go through tutorial and exit at Step 3	Complete	24s	2	N/A
Log out	Complete	20s	2	N/A

Main takeaways from Livia's session: very interesting project, useful and great; overall a very clear website. Biggest point of discontent for Livia was the tutorial, they let us know they have dyslexia and the tutorial could have been worded more effectively to cater to them and others with learning/visual impediments.

In addition, they tried to interact with the locked elements during the tutorial, suggesting intuition to put in practice what they are reading to validate the information absorbed. They also suggested adding more amenities such as public transports stops, scooter stands and student hubs. They liked how it was easier to understand the information visually compared to Google maps or Apple maps.

Average score from survey: 4.8 out of 5

Score breakdown: Appendix 100

6.2.5 Non-expert User 5 - Ben

Our next test session was with Ben, another student in a technological post-graduate degree who also left their contact in the research survey. This session took a more uncontrolled approach and let the users free roam the application without giving them specific tasks to complete. They were guided them in the beginning and certain discussions were initiated but overall, the user was free to explore and think aloud during the process.

Table 8: Usability testing Tasks - Ben

Task	Status	Difficulty	Errors
Load Magpie application	Complete	1	N/A
Sign up	Complete	1	N/A
Log in	Complete	1	N/A
Complete tutorial	Complete	1	N/A
Place cursor on map	Complete	1	N/A
Zoom in and out	Complete	1	N/A
Hold map and navigate	Complete	1	N/A
Adjust radius big/small	Complete	1	N/A
Clear marker & radius	Skipped	Skipped	Skipped
Deselect all amenities	Complete	1	N/A
Select one or more amenities	Complete	1	N/A
Find tutorial and exit midway	Skipped	Skipped	Skipped
Log out	Complete	1	N/A

Main takeaways from Ben’s session: the application does exactly what we described it to do- a GIS application to give a at a glance of amenities in Dublin. The overall impression is that it is a very helpful application, easy to use and effective. Points to improve are the loading times for the amenity points, perhaps directly being logged in after sign up to avoid repetitive steps, make the profile and tutorial icons more visible as they blend into the map, remove mac keyboard icons from the profile bubble, and if possible add more information on each amenity perhaps with tooltips, or add more amenity data like public transportation.

Average score from survey: 4.6 out of 5

Score breakdown: Appendix 101

6.2.6 Non-expert User 6 - Jakub

Jakub is a professional with a construction and technological background. They were recruited towards the end of product development to test Magpie. This was an uncontrolled test session where Jakub discovered the application on their own, discussing each feature, testing each feature, and were then given a small scenario ‘Put yourself in the shoes of an urban planner...’ to obtain a different kind of feedback from previous sessions.

Table 9: Usability testing Tasks - Jakub

Task	Status	Difficulty	Errors
Load Magpie application	Complete	1	N/A
Sign up	Complete	1	N/A
Log in	Complete	1	N/A
Complete tutorial	Complete	1	N/A
Place cursor on map	Complete	1	N/A
Zoom in and out	Complete	1	N/A
Hold map and navigate	Complete	1	N/A
Adjust radius big/small	Complete	1	N/A
Clear marker & radius	Pass	3	Did not see button
Deselect all amenities	Complete	1	N/A
Select one or more amenities	Complete	1	N/A
Find tutorial and exit midway	Skipped	Skipped	Skipped
Log out	Complete	1	N/A

Main takeaways from Jakub’s session: this tool simplifies the search for amenities however there are certain items that need to be considered to improve the application:

- **Log in/Sign up:** Email verification should be included, so that the user can confirm they have successfully signed up and also for security purposes.
- **Map:** some of the amenity data doesn’t have accurate locations, for example public toilets seem to be off by longitude, and multi-storey parking data seems incomplete. Also, water fountains are very hard to find on the map, we should consider changing its colour. Same with the profile and tutorial icon, they are hard to spot on the map. They would also like to double click on the map to clear it, more intuitive for them. And last thing, amenities with small count are hard to find in the radius, maybe make them more visible somehow.
- **History feature:** doesn’t see the use for a non-expert user, and again same for this tool, doesn’t see the use for them as a non-expert user but could be useful for a target user.
- **Extra features:** Search functionality would be very useful for those that are looking for a spot but don’t know where it is located visually. Also, an export feature would be useful for the scenario of urban planning, if I’m to put a report together, a visual from this tool would be helpful for illustration.

A notable behaviour indicator from them was that they tried to interact with the elements during onboarding, as have previous users. Due to technical limitations, we have been unable to make that happen. Future work. Overall, the tooltips for the icons is very interesting, a suggestion would be to add the type of parking, zoning information and tariff to the car parking amenity.

Average score from survey: 4.6 out of 5

Score breakdown: Appendix 102

6.2.7 Non-Expert User Summary

Non-Expert Users provided very valuable feedback throughout the usability testing phase of the project. New features have been implemented, reviewed and removed thanks to these sessions. For example, zoom buttons were added, tooltips were implemented and icons were changed. Most notable points raised during these sessions have been the want for more high level information on the amenities, more amenity data, the visual feedback from the system when it is loading the points and the onboarding.

The average success rate and average difficulty of a task has been computed in the table below. The average success rate has been calculated based on the values of task success, *complete* = 1, *pass* = 0.5 or *fail* = 0. The average difficulty has been calculated based on the values of task difficulty from 1 = Easy to 5 = Difficult.

Table 10: Usability testing Tasks - Non-Expert Users Summary

Task	Task Success Rate	Average Difficulty
Load Magpie application	100%	1
Sign up	100%	1
Log in	100%	1
Complete tutorial	100%	1
Place cursor on map	100%	1
Zoom in and out	100%	1
Hold map and navigate	100%	1
Adjust radius big/small	100%	1
Clear marker & radius	50%	3
Deselect all amenities	87.5%	1.5
Select one or more amenities	87.5%	1.5
Find tutorial and exit midway	75%	2.5
Log out	100%	1.5

Users struggled to find the icons in the beginning to log out or go through the tutorial again, as well as the button to clear the radius and points from the map. Almost all Non-Expert Users skipped the "Clear marker and radius task", showing that they either did not notice it or did not have a use for it.

To conclude, Non-Expert Users gave Magpie’s user interface a score of 4.5 out of 5 overall, radius slider scoring the highest in average.


Magpie UI Survey – General user Summary	
Smoothness of Log in/Sign up	4.67
Clarity of tutorial	4.33
Ease of use of amenity filters	4.50
Radius slider intuitiveness	4.83
Overall experience	4.17
Average score	4.5/5 

Figure 72: *User Evaluation - UI General Score Average*

There was no notable increase in score as the user evaluation progressed between the users, except in the beginning from the session with Brendan where we scored 3.6 out of 5 to the session with Anonymous 1 where we scored 4.6 out 5. Both users experienced the same version of Magpie, so the difference in score could come down to one personal preference, or a preconception.

6.3 Domain Expert evaluation

Domain Experts are users who would use Magpie as a tool for their work. We interviewed 3 individuals who fit this criteria. The Domain Expert test sessions took the following format:

1. **Getting to know:** this first step is designed to introduce ourselves to the participants, inform ourselves of their background and occupation and set the tone for the rest of the session.
2. **Introduction of Magpie:** this second step serves to give an overview of Magpie's purpose, features, data displayed and functionalities.
3. **Exploring Magpie + Discussion:** this third step will allow the participants to explore the application, ask question and reflect on how Magpie can be used to meet their needs for their work.
4. **Satisfaction survey + End of session:** this last step serves to conclude the session by asking the participants to fill in a satisfaction survey to quantify their experience using Magpie.

6.3.1 User 7 - Bryan Boyle

Bryan Boyle is a lecturer at the University College Cork in Occupational Sciences and Therapy. With a doctorate in Computer Science and Occupational Therapy, they have published several papers related to inclusivity in public spaces (Alice Moore, 2023), and the role of technology in the lives of individuals with disabilities (Bryan Boyle, 2022).

They gave their contact information during the research survey. They are categorized as a *Domain Expert* because they utilise amenity data in their work, and Magpie could potentially be a tool they use. Firstly, we got to know Bryan Boyle, his professional experience and the amenity data he uses for his research. His area of research focuses on topics of child development and inclusivity.

What do you use amenity data for in your professional life?	<ul style="list-style-type: none"> ▪ Find out about school geography ▪ Explore amenities and facilities around schools ▪ Research into inclusivity and child development
What amenity data are you usually accessing?	<ul style="list-style-type: none"> ▪ Location of schools ▪ Facilities around schools ▪ Transportation
How are you currently accessing this data?	<ul style="list-style-type: none"> ▪ Data.gov mostly
What issues have you run into accessing and/or using this data?	<ul style="list-style-type: none"> ▪ Accessing data that is not publicly available

Figure 73: User Evaluation - Bryan Boyle information

Next, we introduced Magpie and started the uncontrolled session. The approach was to guide Bryan Boyle through the homepage whilst introducing Magpie, and then let them explore on their own while discussing their thoughts. They were able to complete most of the general tasks, except for zooming in and out. This is because they were not familiar with mouse scrolling, the onboarding was not clear enough and zoom buttons were not present on the map.

Table 11: *Usability testing Tasks - Bryan*

Task	Status	Difficulty	Errors
Load Magpie application	Complete	2	N/A
Sign up	Complete	1	N/A
Log in	Complete	1	N/A
Complete tutorial	Complete	1	N/A
Place cursor on map	Pass	3	Required help
Zoom in and out	Fail	5	Required help
Hold map and navigate	Complete	1	N/A
Adjust radius big/small	Complete	1	N/A
Clear marker & radius	Complete	2	N/A
Deselect all amenities	Complete	2	N/A
Select one or more amenities	Complete	1	N/A
Find tutorial and exit midway	Complete	2	N/A
Log out	Complete	1	N/A

Here were the main takeaways from Bryan's session:

- **Data:** they mainly looked at amenities related to accessibility such as accessible parking, public toilets, water fountains, etc... They would like to see more data such as public transportation, facilities like schools, hospitals and stations. What really interests them is the type and quality of an amenity surrounding the specific location they are searching for, necessitating higher-level information to add to the tooltips.
- **General:** they really enjoyed using the search bar, they found it intuitive and quicker to use than placing a marker on the map. They are very interested in the project and look forward to seeing the final product.
- **Behaviour:** Bryan seems to have less than average proficiency with technology. Throughout the session, we noticed that first they used Edge as a browser, they got lost looking for the tab among the many windows they had open, their browser window was not full screen and additionally struggled zooming into the map.

Overall, Bryan found Magpie to be simple and very straightforward, demonstrated by his 5 out of 5 score for the user interface. He would use it as a tool for his PhD research, mainly on desktop which further solidifies our user persona and why we prioritized desktop compatibility before mobile.

Score breakdown: Appendix 103

6.3.2 User 8 - Dr. Sarah Rock

Professional user Dr. Sarah Rock is a transport and urban planning specialist with over 15 years of experience working in both the private and public sector. As chair of the MSc in Urban Regeneration at the TU Dublin School of Architecture, she pushes for sustainable and innovative solutions to land use and urbanisation, specialising in accessibility planning and network design.

We recruited them through social media channels to take part in a user testing session. They mostly use amenity data to understand an existing area that has plans for upgrade or demolitions, as well as studying transportation routes to plan new stops and service new areas.

What do you use amenity data for in your professional life?	<ul style="list-style-type: none"> ▪ Explore amenities/facilities along planned transport routes ▪ Understand components of existing area for planning
What amenity data are you usually accessing?	<ul style="list-style-type: none"> ▪ Bus routes ▪ Schools ▪ Zoning information ▪ Public spaces ▪ Public transportation
How are you currently accessing this data?	<ul style="list-style-type: none"> ▪ Myplan.ie ▪ Local development plans online ▪ Google maps ▪ County council datasets
What issues have you run into accessing and/or using this data?	<ul style="list-style-type: none"> ▪ Little information on the true level (quality, quantity, capacity) of amenities

Figure 74: User Evaluation - Dr. Sarah Rock information

Next, Dr. Rock loaded the Magpie application and started exploring, whilst discussing each and every feature. They were able to complete all general tasks while skipping a few. One question they asked was very interesting: *how do we define amenities?* Different professions within urban/transport/city planning define amenities in different ways, and they found our way of defining it interesting and slightly different from the way they would've defined it: *'Amenities are any sort of public facilities that you can use.'*

Table 12: Usability testing Tasks - Dr. Sarah Rock

Task	Status	Difficulty	Errors
Load Magpie application	Complete	2	N/A
Sign up	Pass	2	Tried to sign up on log in page
Log in	Complete	1	N/A
Complete tutorial	Complete	2	N/A
Place cursor on map	Complete	1	N/A
Zoom in and out	Complete	1	N/A
Hold map and navigate	Complete	1	N/A
Adjust radius big/small	Complete	1	N/A
Clear marker & radius	Skipped	N/A	N/A
Deselect all amenities	Skipped	N/A	N/A
Select one or more amenities	Complete	1	N/A
Find tutorial and exit mid-way	Skipped	N/A	N/A
Log out	Skipped	N/A	N/A

Here were the main takeaways from Dr. Rock's session:

- **Data:** the tooltips on each icon are very interesting. Perhaps cross-check the information there and add/remove details. For example, the bike sharing amenity tooltip has name, number, and address, but the number seems to be an ID and not the number of bikes at the bike sharing station.
- **Additional features:** for their work, Dr. Rock likes to have a satellite view of the area they are inspecting. They found that when they are zoomed in at the street level, they find it hard to understand the map. Perhaps adding a satellite layer for enhanced visualisation. Also, an export feature would be amazing, with a scale and table of the amenities found on the location selected on the map.
- **Miscellaneous:** the car parking amenity has lots of potential because it relates to the use of public space which is very contested and a big issue in urban planning. Magpie displays that information in an easy manner and there is so much potential to expand the parking detection and help fuel research on how much cars take up public space in cities and inefficiencies using public space for parking especially with limited land.

Overall, they stated that Magpie is a fascinating and really useful project in its own right. The radius slider is great, the amenity count is very useful, Dr. Rock can see Magpie being used for planning and research not so much non-expert user. Points to improve would be more amenity data and revise the information on the amenity tooltips. They scored Magpie's UI a 4.6 out of 5, the tutorial and the filters losing a star for lack of clarity and intuitiveness.

Score breakdown: Appendix 104

6.3.3 User 9 - Odran Reid

Odran Reid has over 30 years of experience in the public, private and NGO sector in Europe. With degrees in economics from Trinity and spatial planning from TU Dublin, they specialise in retail planning, socio-economic analysis and community consultation for local development projects. They are currently a lecturer in spatial planning at the TU Dublin School of Architecture. We recruited them through social media channels to take part in a user testing session. They mostly use amenity data for socio-economic research and consulting for community project developments.

What do you use amenity data for in your professional life?	<ul style="list-style-type: none"> ▪ Consulting for community engagement ▪ Retail planning ▪ Rural planning ▪ Socio-economic research
What amenity data are you usually accessing?	<ul style="list-style-type: none"> ▪ Socio-economic data ▪ Planning application information ▪ Development maps
How are you currently accessing this data?	<ul style="list-style-type: none"> ▪ Google maps ▪ County council datasets ▪ Publicly available datasets ▪ On the ground
What issues have you run into accessing and/or using this data?	<ul style="list-style-type: none"> ▪ Data doesn't exist ▪ Data not publicly available

Figure 75: User Evaluation - Odran Reid information

Before the session started, Odran Reid told us they qualified their technological proficiency as 'very low' and to be patient with him. This was confirmed when he struggled to share his screen at the start of the session, however he was very comfortable navigating the map, zooming in and out and using the features of the dashboard. Here were the main takeaways:

- **Amenities:** they suggested some minor content changes related to the labelling of amenities, for example 'library' should be 'public library', 'public wifi' should be 'free public wifi' etc. . . Also, it would be beneficial to differentiate the types of bike sharing system (Dublin Bikes, Bleeper, Moby Bikes). More data should be added like schools, public transportation, zoning information and vacant buildings.
- **Features:** giving the user the chance to draw their own radius could be very interesting. Also, adding a '300 meters' preset for the radius slider would cater to planners more because of the 15 minute walkable rule. This rule states that 300m is a walkable distance for abled individuals and is used a lot in retail/town planning.
- **Miscellaneous:** they suggested to look at *POBAL* for more datasets relevant to our project.

Overall, Odran Reid believes Magpie has commercial potential but needs more datasets and more detail to turn this into the go-to tool for planners in any sector. They expressed how they would've enjoyed having this tool earlier on for their consulting work. They scored Magpie's UI a 4.6 out of 5, the log in and sign up losing a star for lack of email verification and smoothness.

Score breakdown: Appendix 105

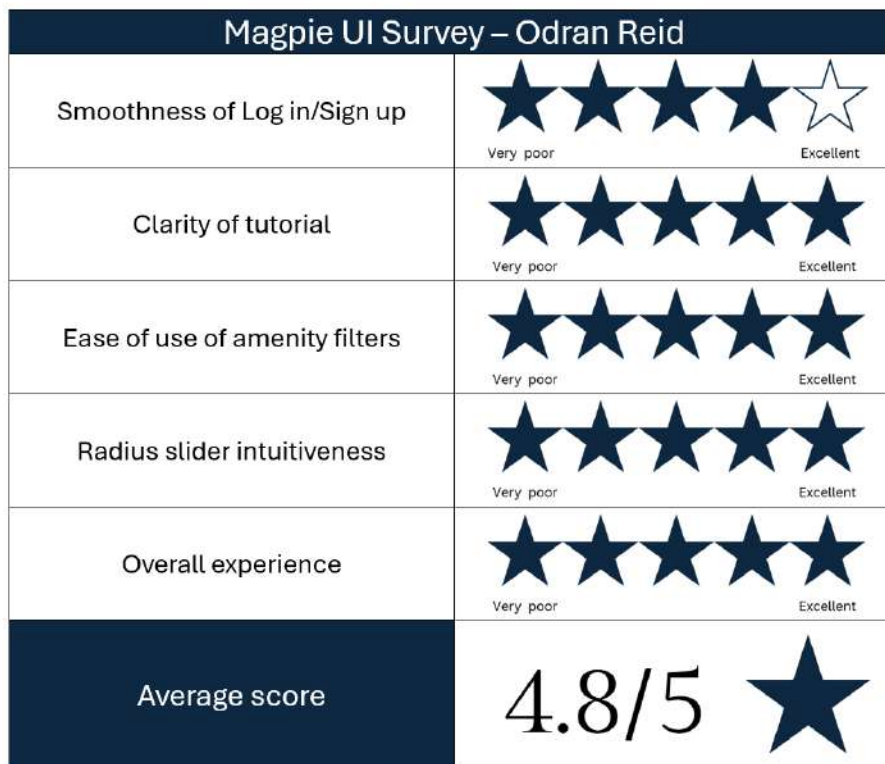


Figure 76: User Evaluation - UI Score Odran Reid

6.3.4 Domain Experts overview

Interviewing these three users was a really amazing experience. It's easy to put all planners into one basket however, they all have different needs, experiences and wants and these sessions really opened our eyes on that.

The main changes these users would like to see are **more data**, more **precise information** on the amenity tooltips, an **export feature** and improvements to the onboarding.

The scores they gave on Magpie's UI averaged to 4.8 out of 5, which is incredible. This score validates Magpie's project vision and end goal: to create a easy to use tool to give a glance view of amenities in an area.

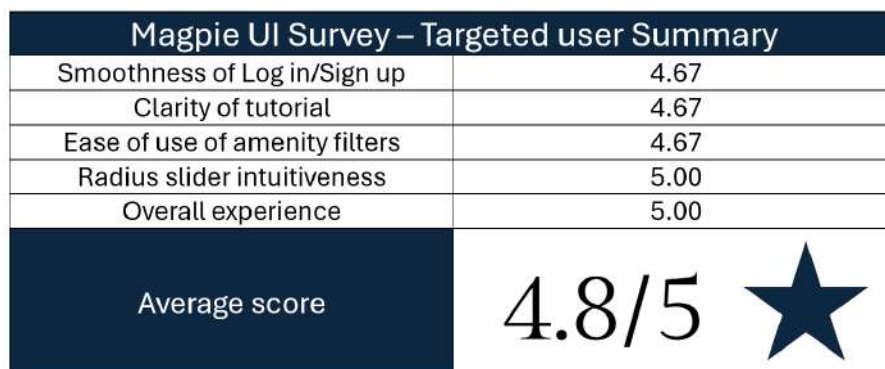


Figure 77: User Evaluation - UI Target Score Average

6.4 UI/UX Expert Review

The goal of this review is to evaluate the user interface of Magpie at different stages of development with regards to key UI/UX general guidelines.

UI design guidelines differ from UX ones because of their fundamental differences: one is user based and the other isn't. **UI design** aims to create a visually appealing, intuitive and userfriendly interface that allows for easy navigation and interaction with the application.

On the other hand, **UX design** aims to create a meaningful user experience through designing an entire user journey. Together, they work to shape the path to a successful and enjoyable digital product (Hamidli, 2023).

10 major guidelines were devised in the 1990's by Jakob Nielsen, the 'father' of web design. These guidelines have shaped several UI and UX design creations in the last decade (IxDF, 2016). Below are the ones outlined as most relevant to Magpie's core:

- **User control & freedom:** the users should be able to retrace their steps such as undoing and redoing previous actions, exiting processes midway etc. . .
- **Recognition over recall:** ensure the system is not heavy on user's cognitive load and prioritizes easily recognizable design items over one's that rely on memory.
- **Aesthetic & minimalist design:** keep the visual information to a minimum, display only necessary components and clearly visible unambiguous means of navigating through the application.
- **Help & documentation:** provide documentation that is easily accessible, relevant and worded in a way that will guide users to the desired outcome

We requested a review of our system from UI/UX professional Andrea Curley. They are a professor at a top technological university, specialised in web development, web design and user experience.

Two sessions were conducted: one on November 13 after the publication of our first minimum viable product, and the other on December 9 at the end of the development timeline. Both sessions were conducted online through a videoconference meeting on Teams, where we presented Magpie, explored the features, discussed them, and administered a questionnaire at the end.

The questionnaire is divided into 3 sections pertaining to *visual design*, *information architecture*, and *compliance*.

Different types of questions were included, such as 'Closed' 'Scale' and 'Open' to allow for both the quantitative and qualitative measure of Magpie UI.

Open-ended questions tend to require more cognitive efforts, thus answers can vary in quality and quantity depending on the individual. Close-ended questions on the other hand offer standardized responses which can be more easily quantified and require less cognitive effort (Ai-Hong Chen, 2020).

Table 13: Expert Review Questionnaire

	Question	Question type
Q1	How user friendly is the log-in/sign up page?	Closed
Q2	How user-friendly is the on-boarding process	Closed
Q3	How effective is the visual hierarchy of the information on the dash-board?	Closed
Q4	Rate the clarity of the map visualization	Closed
Q5	How intuitive is the organization of amenity data categories?	Closed
Q6	Rate the following features from Worst (1) to Best (5)	Scale
Q7	How comprehensive is the amenity data coverage for Dublin city?	Closed
Q8	How valuable do you think this tool would be for the following use cases - 1: Not valuable at all, 5: Extremely valuable	Scale
Q9	Any additional comments on why this tool would useful/impractical for the above use cases?	Open
Q10	Evaluate the following technical aspects from Worst (1) to Best (5)	Scale
Q11	Rate the application's compliance with the items below from Worst (1) to Best (5)	Scale
Q12	Any additional comments regarding our application?	Open

6.4.1 UX Session 1

The first session provided very valuable insights on Magpie's workflow, user interface and technical components, as well as helped us to uncover critical bugs. These were the main takeaways:

Landing Page: Upon loading Magpie, Andrea Curley was directly taken to the mapview, which was not supposed to happen. After the review, we investigated the cause of this event and uncovered a bug in the authentication which we have been working on. Following this event, she suggested creating a landing page or some sort of introduction to ease the user into discovering Magpie.

Onboarding: Due to the bug explained above, the onboarding did not automatically start as it should have upon login. Nevertheless, Andrea Curley said that the user may want to intuitively press on the elements being highlighted during the tutorial, as she tried to do. This adds to the feedback received during casual testing for the implementation of this feature. Unfortunately, due to a technical limitations we are not able to solve it, only provide make certain changes to dissuade the user of doing so.

In addition, she suggested there should be an option to exit the tutorial at any time for users who don't want to sit through it. Lastly, the tutorial should be more visually striking and engaging in order to leave a lasting impression on the user.

Dashboard & Map: Currently, the hierarchy of items on the dashboard does not make sense to the average user, and is not intuitive to use. All the amenities are displayed when only one is selected (as shown in the figure below) and their count displays zero, which the user might interpret as there are zero other amenities in the area in addition to the one I selected.

The icons on the map are not visible enough, and zooming in & out on the map may not be

intuitive to the range of users and devices. Adding zoom buttons could help bridge that gap. At the moment, Andrea Curley noted that there is a disconnect between the map and the dashboard whereas they should be looked as one. She suggested adding amenity icons to the dashboard to help bridge that gap.

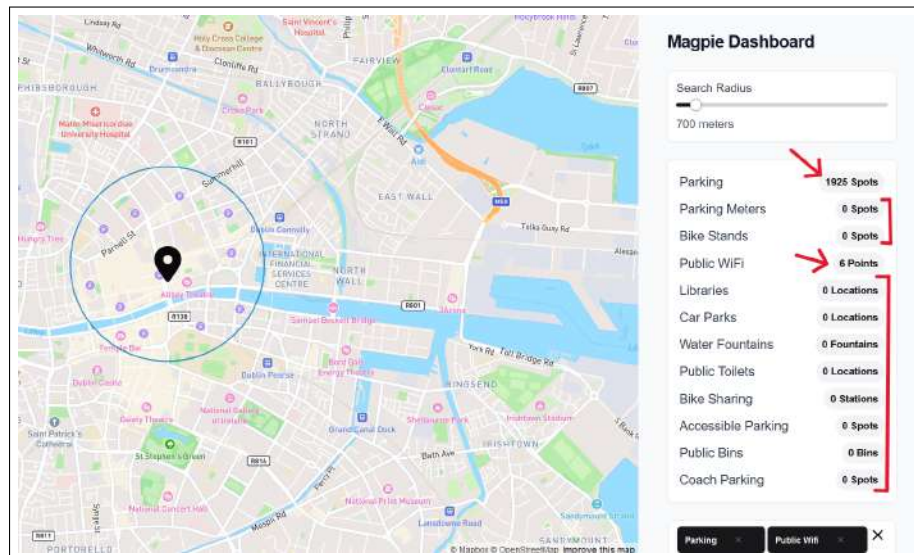


Figure 78: V.0.1 Magpie Dashboard & Map when 2 amenities are selected

Filters: If there are no amenities found in the radius of search, a message should pop up to tell the user so. Currently, it is not very clear if there are amenities present in the chosen area especially due to the small size & faded color of certain icons.

Log in/sign up: When trying to log in with credentials that don't exist, the system should return a proper error such as 'username doesn't exist'. Log out and account sign up went smoothly. Andrea Curley questioned the benefits of logging in for Magpie, to which we stated: Magpie was conceived with the idea of providing a service to working professionals; therefore logging in will allow the implementation of further features such as safeguarding their previous searches, storing exported reports, connecting with other members of your organization and much more.

The screenshot shows a 'Login' form with the following elements: a title 'Login', a prompt 'Enter your username or email below to login to your account', a 'Username/Email' input field containing 'az@adf.ie', a 'Password' input field with masked characters, a 'Forgot your password?' link, a 'Login' button, and a red error message: 'Login failed: {"error": {"errorCode": 1402, "errorMsg": "Wrong username or password"}, "response": null}'. At the bottom is a link: 'Don't have an account? Sign up'.

Figure 79: Error message when using inexistant username and password

Survey response: Below are the answers to the expert review survey. The response helps complement the oral feedback received during the session and provide some quantitative data as a baseline for the next evaluation. A score has been attributed to each section of the survey based on the responses from Andrea.

The first section covers usability of the main items of Magpie’s user interface which scored 3.5 out of 5. The items which brought the score down are the onboarding and the clarity of the map visualization, further supported by the vocal feedback Andrea provided on the un-intuitive flow and disconnect between the map and the dashboard.

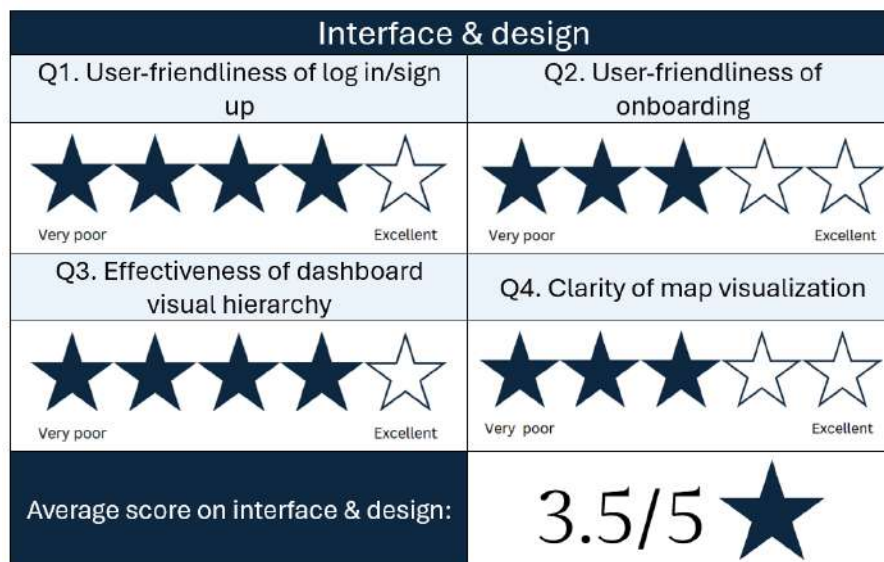


Figure 80: UX review 1 - UI score

The second section covers the information architecture and data quality of the amenities. It looks at the features displaying the information and the score reflects the problems aforementioned with the dashboard as well as the incomplete profile menu. Andrea Curley scored this section a 3 out of 5, onboarding and profile menu needing improvement.

Information Architecture Score breakdown: Appendix 106

Furthermore, Q8 asks Andrea to assume the value of Magpie as a tool for different use cases where our target users (Urban planners and Event planners) were rated as ‘Valuable’. This rating is useful but it remains an assumption.

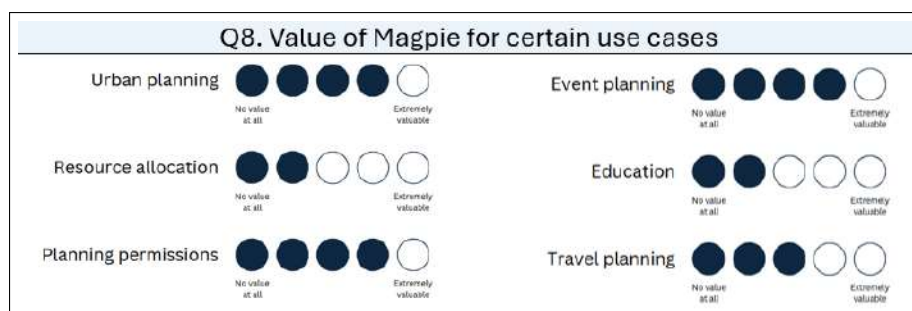


Figure 81: UX review 1 - Q8 from Information Architecture

Lastly, the third section covers the technical aspects of Magpie. The low score of 2.88 out of 5 reflects a major authentication bug encountered during the testing session, as well as severe lagging of the points on the map due to technical difficulties.

Technical Score breakdown: Appendix 107

Summary of 1st UX session:

Overall, Magpie scored 3.13 out of 5 for this first UX review session. This is the baseline, and the objective for the next session is to score above 3.5 out of 5 overall, and significantly improve the scores for the onboarding, the dashboard flow and the system responsiveness.

UX review – Session 1 scores	
Part 1 - UI	3.50
Part 2 – Information architecture	3.00
Part 3 – Technical performance	2.88
Total	3.13/5 ★

Figure 82: *UX review 1 - Overall score*

Andrea Curley found our interface sleek and minimalistic. However, she suggested that if we want to remain with this style, we need to ensure there is as little room as possible for ambiguity and confusion. The user needs to find it easy to move from one feature to another and understand the triggers. "Currently, Magpie looks so sleek that the user may not be able to see what they want."

6.4.2 UX Session 2

The second session informed us of the points we were able to improve on from the first session as well as features to be looked at in the future. These were the main takeaways:

Landing page: The new landing page provides a proper introduction of Magpie, however some adjustments can be made to the navigation bar, some visual feedback to let the user know the button they have clicked works; such as a bold overlay on the navigation item or different colour highlight. In addition, if you really want to push Magpie as a GIS application for professional urban planner users, it needs to be put at the forefront. Lastly, a log in button should also be present on the main page alongside the ‘sign up’ button.

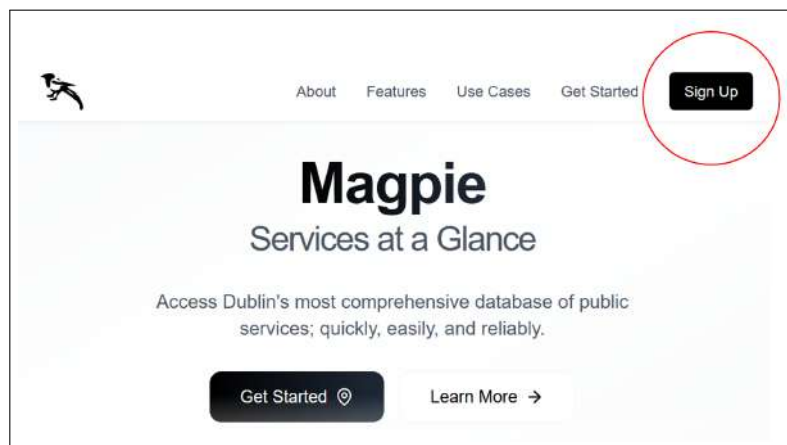


Figure 83: Magpie V.0.12 - Landing page sign up button

Dashboard: The rearranged dashboard has immensely improved the flow of selecting and deselecting amenities, adjusting the radius and clearing the map entirely of the points. The disconnect between the map and the dashboard has been addressed with custom icons and colours and labels.

Some suggestions to improve user experience would be to make the amenity row clickable so as to make it easier for the user to select and deselect it, as opposed to having to click the eye directly.

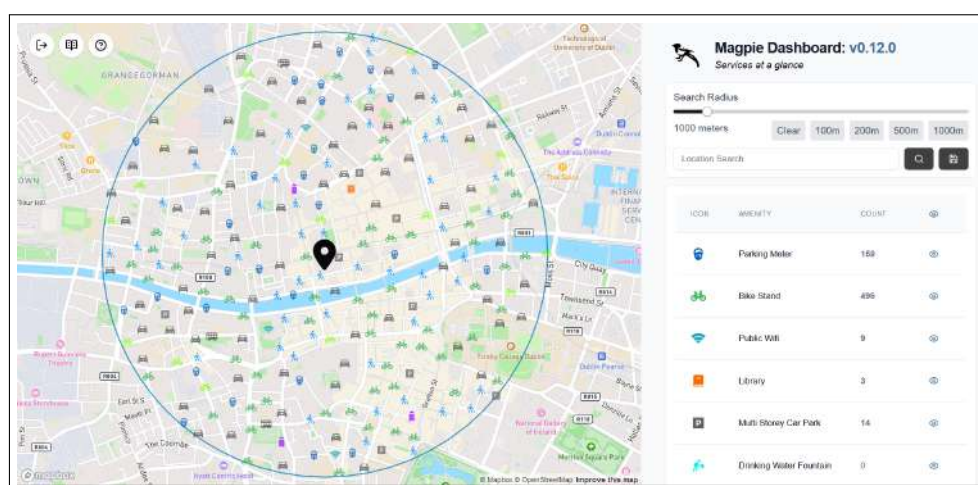


Figure 84: Magpie V.0.12 - New Dashboard

Map: The map looks very clean and the points are loading much better. However, one small issue to mention is the amount of icons being loaded within the search radius. At the moment, the icons group together under one when the view is very zoomed out, and expand and disperse the more you zoom in. However, certain icons like the ‘Public Bins’ are more in numbers visually than any other.

This is due to the way we have rendered the points on the map, which initially did not support it so we had to find a way around it through the layers. The items in the top most layer appear the most and ‘Public Bin’ is at the top right now. This is also something that will be detailed in future work.

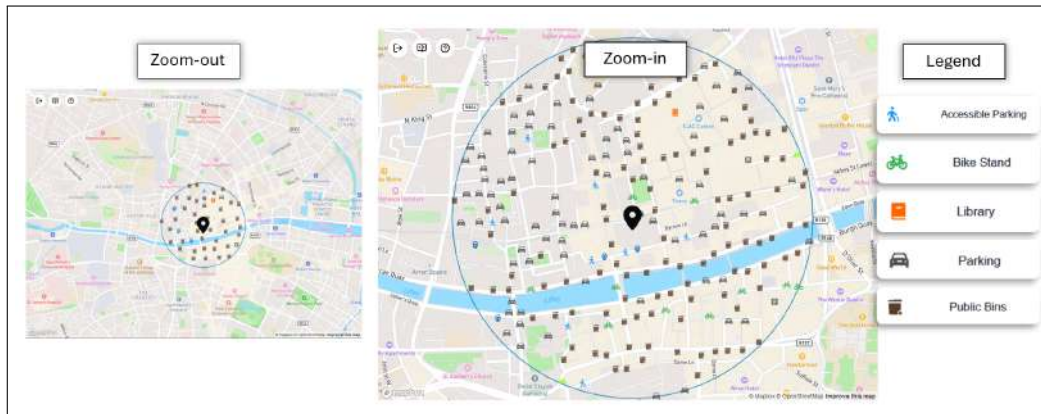


Figure 85: Magpie V.0.12 - Amenity Icons grouping relative to zoom level

Search & Save functionality: Good addition however it gives inconsistent results. This is due the the API we are using which does a keyword search through our database to retrieve results instead of a semantic search. Therefore street names with special characters or with many words may not always return accurate or any results unfortunately, or why if the search is saved it may not carry the search term.

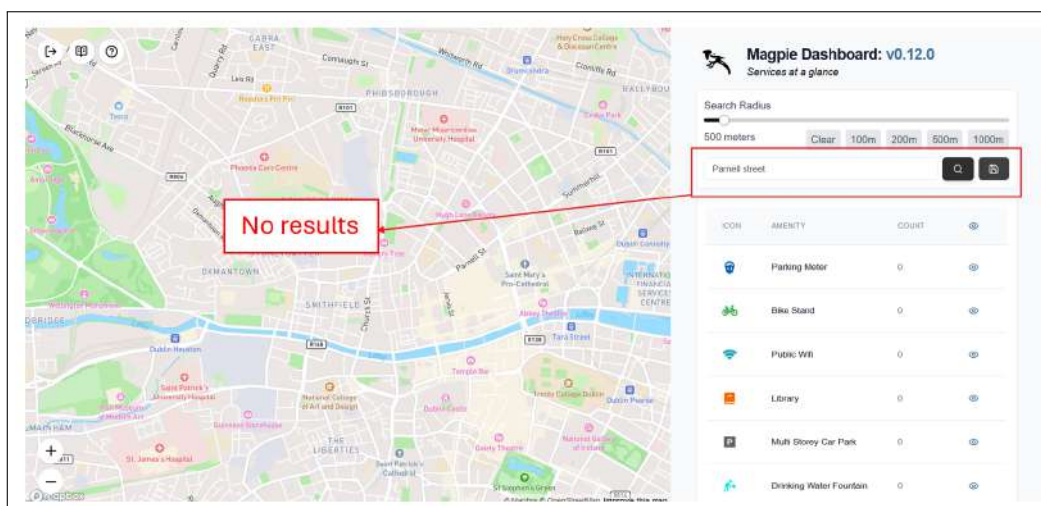


Figure 86: V.0.12 Magpie Search Failing

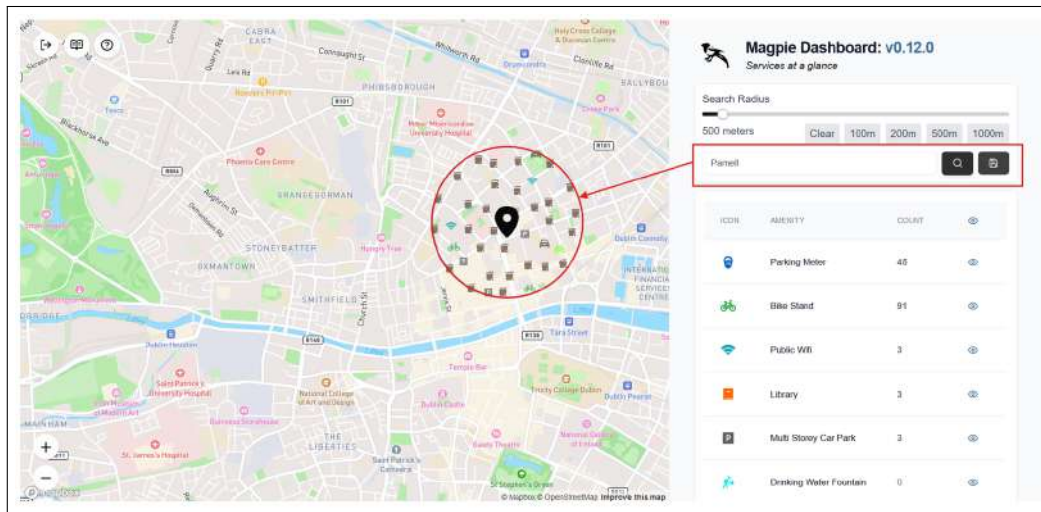


Figure 87: V.0.12 Magpie Search Working

Additionally, a successful search is saved with a different name than the query. This is again due to the nature of the search being keyword instead of semantic.

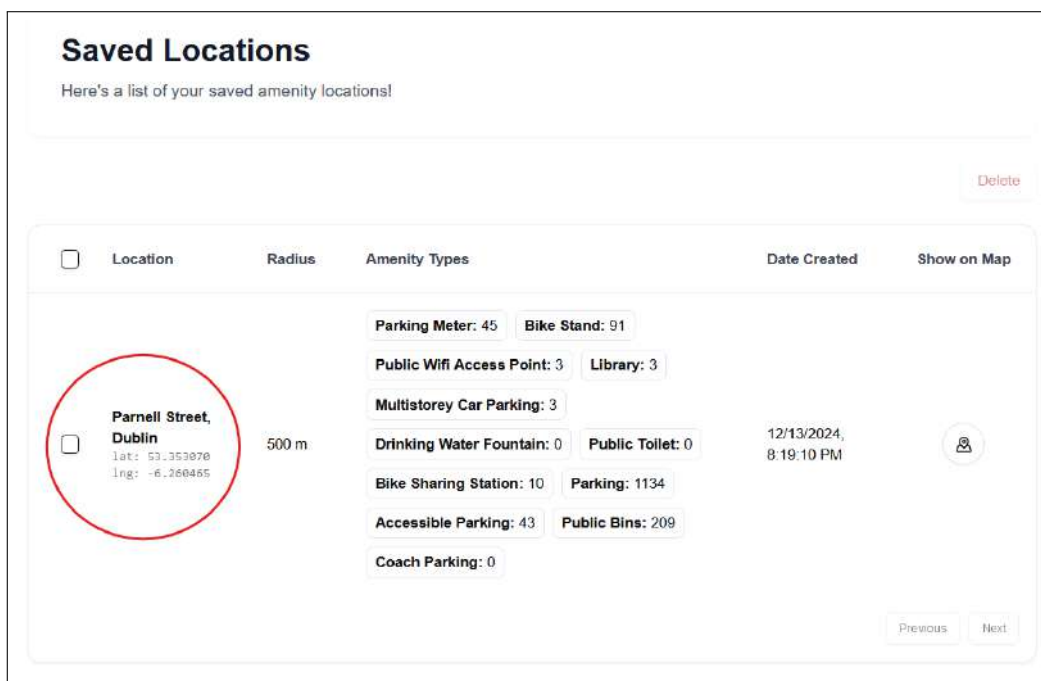


Figure 88: V.0.12 Magpie Saved Location

Last suggestion on this point is to be able to press enter to start the search, instead of being confined to the search button. The search feature will be further addressed in the future work section of the report on how to switch it to a semantic.

Survey responses: Andrea Curley scored the UI of Magpie a 4 out of 5, which is a net increase of 0.5 points from the first session.

Changes made contributing to this increase are the fixing of the overlapping onboarding elements and the updated icons with tooltips on the map.

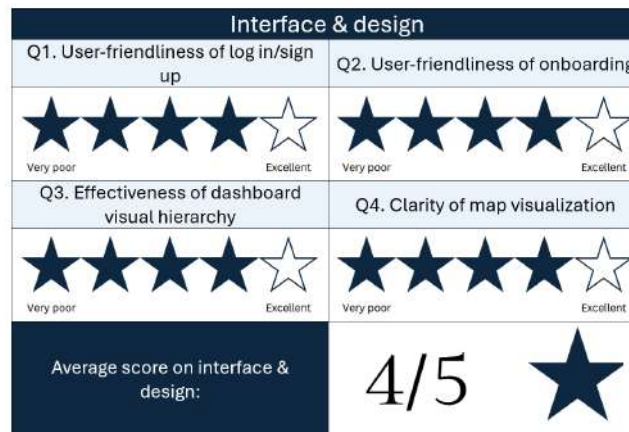


Figure 89: UX review 2 - UI score

The second section related to the information architecture scored a 3.63 out of 5, a net increase of 0.63 points from the first session. This increase is due to changes made to map with brand new amenity icons and fixed overlapped onboarding elements.

Information architecture Score breakdown: Appendix 108

In addition, Andrea Curley increased the value rating for resource allocation and planning permissions, due to the discussion we had during the second session regarding the interview sessions with the domain experts.

Lastly, the third section scored 3.1 out of 5, a good increase from the first session. This is reflected by the solve of the authentication bug and the feedback applied from the first session to improve the UI and accessibility.

Technical Score breakdown: Appendix 109

Overall: The feedback from the previous review has been applied diligently, the application looks clean and is easy to use. The goal of surpassing 3.5 out of 5 has been achieved with a total score of 3.58 out of 5.

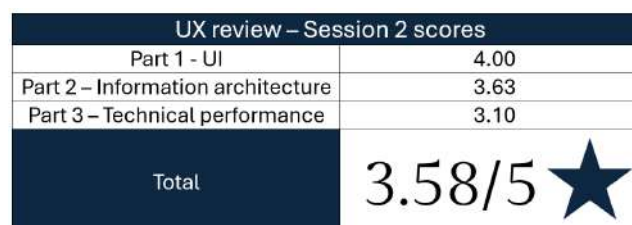


Figure 90: UX review 2 - Overall score

However, more visual feedback is needed for features like the search functionality and the selection/deselection of amenities, as well as the accuracy of the points displayed.

6.5 Accessibility Expert Review

In the context of this project, accessibility refers to considerations for users with visual, auditory, cognitive and/or physical impairments; inclusiveness refers to considerations for users for whom English isn't their first language, older users and users with gender or affective issues; and universal design refers to compatibility of Magpie with different devices, browsers and networks.

The EU Web Accessibility Directive provides a general set of guidelines which bound public bodies in Europe to make their websites and online content accessible. This directive came into effect at the end of 2016 and aims to *make public sector websites and mobile applications more accessible, and to harmonise varying standards within the European Union* (...) (Union, 2016).

Scope of directive	Public sector bodies website and mobile applications across EU member states
Compliance requirements	Web Accessibility Content Guidelines (WCAG) 2.1 Level AA <ul style="list-style-type: none"> • <u>Perceivable</u>: content with text alternatives, subtitles, captions • <u>Adaptable</u>: meaningful design across different devices and screen sizes • <u>Distinguishable</u>: contrasting elements and colours, sizable text, content on hover • <u>Operable</u>: interface with support for keyboard navigation, assistive devices • <u>Navigable</u>: consistent & predictable user interface, easy to understand language
Exemptions	<ul style="list-style-type: none"> • Limited functionality websites (legacy) • Websites not for essential public services • Websites with content that cannot reasonably be made accessible • Disproportionate burden

Figure 91: WCAG 2.1 Scope, Guidelines & Exemptions

However, there is an inherent limitation of accessibility for mapbased systems due to their visual nature. Visual impairment is categorized as having a type of eye disorder and/or a degree of vision loss; an estimate of 2.2 billion people globally have vision impairment (Organisation, 2019) and 297,000 (5.6%) of the population in Ireland (Ireland, 2023).

Magpie falls under the exemptions of the WCAG, as it has content which cannot be made reasonably accessible. However, efforts must still be made to make it as *accessible as possible*.

The paper by Hennig S., 2017 which relied on data from two projects aimed at developing web-map applications for visually impaired individuals has presented several recommendations, the following which guided Magpie's UI development and the subsequent accessibility evaluation:

1. **UI components:** Creating a UI that is simple, understandable and follows a clear predictable layout. Implement only necessary control elements, group them by similar focus and place them in areas that are familiar (similar in other programs). The UI should be flat while avoiding dropdowns, scrolling and nested/overlapping control elements.
2. **UI design:** Size and colour of UI components should be chosen to provide high contrast between different elements; if symbols are used, they should be easily recognizable. Complex backgrounds should be avoided.
3. **UI language:** Familiar and easy recognizable terms should be used on the interface so as to not scare off users with unknown technical terms and make the application accessible to all.
4. **UI interaction:** A user should be able to interact with the interface using both a mouse and keyboard on desktop; keyboard accessibility is very important for visually impaired users.

We requested a review from Accessibility expert Damian Gordon. He served on the board of the National Disability Authority, who advise the Irish Government on all matters related to disability. He has worked with a wide range of disability organisations, include the Centre Remedial Clinic, the National Council for the Blind, Arthritis Ireland, and the AgingWell Network. He has contributed to the development of hardware, software, legislation and training related to disability awareness and accessibility.

The goal of this review is to evaluate the accessibility, inclusivity & universal design of Magpie with regards to the guidelines and recommendations detailed above.

The review was conducted on November 21st during the usability testing phase of Magpie. The session was conducted online through videoconference meeting on Teams where we presented Magpie, gave Damian Gordon a scenario to follow to explore the application, discuss the scenario and the accessibility guidelines and then a survey.

The scenario given can be seen in the table below. A scenario was given so as to simulate the use of Magpie by one of our target users through the lens of an accessibility expert.

Table 14: *Scenario for the Accessibility review*

Scenario:
You are an architect contracted by Dublin City Council to expand the Dominick Street Recreation Centre, located on Dominick Street Lower, Dublin 1. As part of your assignment, you need to plan the expansion in a way that integrates effectively with the surrounding community and existing amenities.
Scenario tasks:
<ol style="list-style-type: none">1. Locate the Community Centre: Use the GIS application to locate the Dominick Street Recreation Centre on the map.2. Identify nearby amenities: Select the public amenities you think are relevant within a 500-meter radius of the recreation centre. These can include but are not limited to: bicycle stands, parking spaces, public wi-fi spots, public toilets.3. Analyse amenity density: Based on your findings, determine which types of amenities are abundant and which are lacking around the centre.4. Assess accesibility: Check how accessible the recreation centre is by identifying nearby transportation options. Note any gaps in accessibility that might need addressing.5. Plan for additional amenities: Suggest which new amenities should be added as part of the recreation centre's expansion to better serve the community. For example: if car parking is insufficient, recommend additional parking spaces.

The questionnaire covers UI design, interaction and language and aims to complement the vocal feedback of the session. It is made up of 3 sections with a mixture of closed questions rating different parts of the system, and open questions to provide more qualitative feedback.

Damian Gordon's behaviour will also be observed in addition to his success rate with general tasks related to the main features of Magpie. The success rate is based on the completion of a task the the difficulty encountered by the user during it.

The difficulty of the task is related to if they were able to complete it and how much they struggled during it. The status of a task can either be 'complete', 'pass' or 'fail', and the difficulty is score from 1 = 'very easy' to 5 = 'very difficult'.

Table 15: *General tasks score for the Accessibility review*

General task:	Status	Difficulty	Errors
Load Magpie application	Complete	1	
Sign up new account	Complete	1	
Log in	Complete	1	
Go through onboarding	Pass	2	Tried to click on locked elements
Place cursor on map	Complete	1	
Zoom in and out	Fail	5	Did not know how to use the mouse
Hold map and navigate	Complete	2	
Adjust radius big/small	Complete	1	
Clear marker and radius from map	N/A	N/A	Skipped
Deselect all amenities	Complete	1	
Choose certain amenities	Complete	1	
Find onboarding and exit midway	Pass	4	Could not find button
Logout	Pass	3	Could not find button

The review provided some useful insights with regards to missing features to complete the scenario and areas of Magpie that comply with accessibility standards. Below were the main takeaways:

Dashboard: Damian Gordon had some trouble locating the recreation centre during the scenario, and advise that have a search bar would remove the difficulty and give options for those who may be visually impaired to find a location on the map, and also for those who may not know where the area they're looking for is located.

Onboarding: Damian Gordon found the steps in the onboarding very wordy and too long, which may cause some users to skip through or not retain the information present there. In addition, some of the wording could be improved related to navigating the map as shown in the figure below.

This issue is further illustrated by the difficulty Damian Gordon encountered trying to zoom in and out on the map using his mousepad, having not understood how to do it from the onboarding step 5. Addressing the choice of words, the content and flow of the onboarding is important to ensure it is understood by all kinds of users.

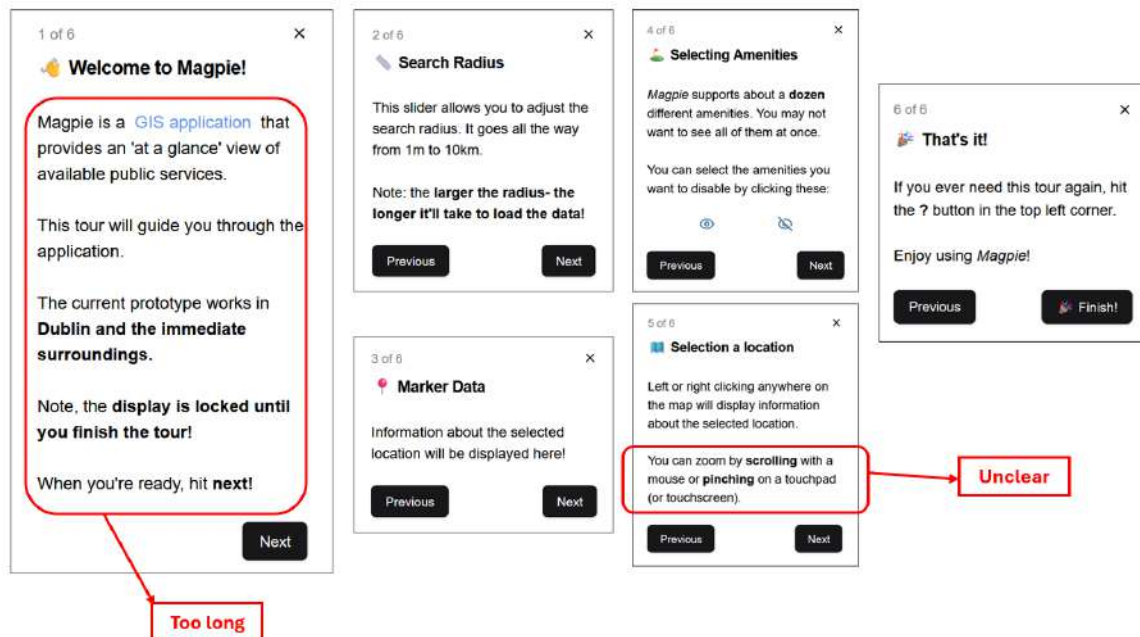


Figure 92: V.O.1 Magpie Onboarding Steps

Map: Going back to the zooming feature Damian Gordon struggled with, adding zoom in and zoom out buttons directly on the map would offer an alternative to those not able to use the mouse for the action, further improving Magpie’s accessibility. In addition, the profile and onboarding icons blended into the map which made it difficult for Damian Gordon to go back to the onboarding or log out as shown in the figure below.

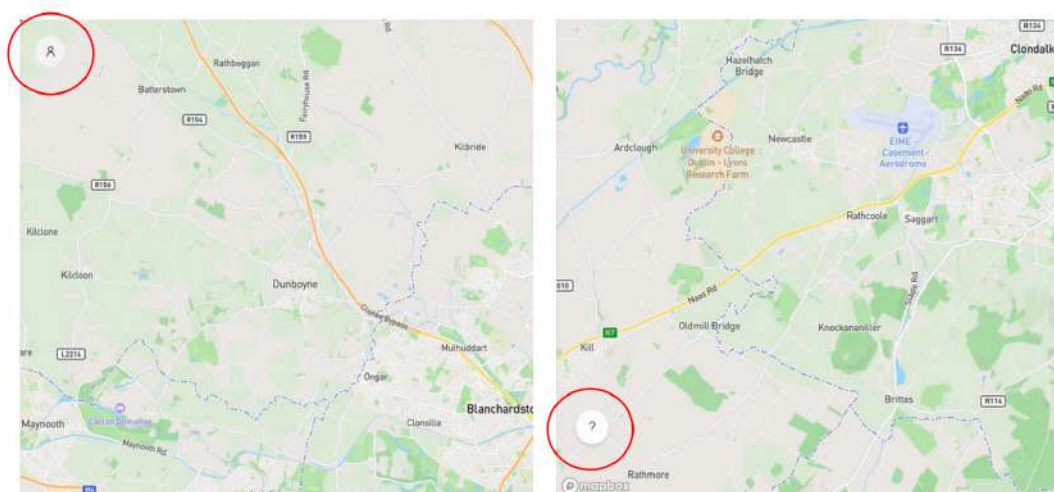


Figure 93: V.O.1 Magpie Profile & Onboarding icons

Lastly, his survey responses confirmed that Magpie adheres to the WCAG guidelines detailed above as well as fulfils the major goal of ease of use. The survey covered 3 major sections: UI components, UI Design & Language and UI interaction.

UI Components evaluate the position, quantity and accessibility of control elements such as the navigation buttons, the map controls, the dashboard toggles, etc...

Damian Gordon rated this section a 4.75 out of 5, the placement of control elements loosing one point because some of them were nested and therefore hard to notice or navigate to, like the profile and tutorial icons on the map mentioned above.

UI Components Score breakdown: Appendix 110

UI Design & Language covers the style of UI components as well as the textual language used on the interface. Damian Gordon rated this aspect of Magpie at 5 out of 5, noting that there is a clear and effective use of language and icons which both are easily recognizable and clear.

UI Design Score breakdown: Appendix 111

The last section, **UI Interaction** covers the range of interaction possible with the Magpie UI using different hardware and software tools. Damian Gordon also rated this section a 5 out of 5, citing good compatibility with industry standard screen readers and ease of navigation with both keyboard and mouse.

UI Interaction Score breakdown: Appendix 112

Overall, Magpie scores a 4.92 out of 5 score in accessibility, with perfect scores in both UI design, language and interaction. Points to improve relate to increasing accessibility by reducing the number of nested elements, and making adjustments to the tutorial as per vocal recommendation during the testing session.

Accessibility review scores	
Part 1	4.75
Part 2	5.00
Part 3	5.00
Total	4.92/5 ★

Figure 94: *Accessibility review Overall Score*

6.6 User evaluation summary

The user evaluation cycle of Magpie was key in solidifying our target user base, their wants & needs as well as ensuring our application was user-friendly, accessible and comprehensive.

The main takeaways from these in-depth sessions were:

- **More data:** Majority of users, both domain and non-expert expressed their want for more amenity data on Magpie. This ranged from transportation data for route planning to a water grid overlay for town planners.
- **More features:** From the start of the user evaluation cycle with non-domain experts to the end, users have requested more features. Clear buttons were added, tooltips and a search functionality was implemented to the satisfaction of the users.
- **Improved visual feedback:** A key issue raised by users and also noted from their behaviour is more visual feedback when interacting with Magpie's control elements. Users want to see a reaction when pressing on the search, whether it was successful or not, and to interact with the tutorial.

Below is the summary of the UI scores from all the users interviewed.

User group	User	Avg. UX Score
Non-expert	Brendan	3.6
	Anonymous 1	4.6
	Paul	4.8
	Livia	4.8
	Ben	4.6
	Jakub	4.6
Domain expert	Bryan Boyle	5.0
	Dr. Sarah Rock	4.6
	Odran Reid	4.8
UI/UX expert	Andrea Curley	3.58
Accessibility expert	Damian Gordon	4.92

Figure 95: UX score - Summary all users

There is still work to be done to improve the user flow of Magpie's interface, as shown by the results from the UI/UX expert. However, the results from the domain expert are extremely encouraging and validating to Magpie's core user goal of providing a useful tool to planners.

7 Future Work

7.1 Introduction

This section synthesizes the findings and outcomes of the project, summarizing how the core components—Machine Learning, Front-End, Back-End, and Deployment—contributed to achieving the project’s goals. Each sub-section reflects on the approaches adopted, challenges encountered, and the insights gained throughout the development lifecycle. The integration of these components played a pivotal role in delivering a functional and cohesive system, emphasizing both technical and user-centric considerations.

7.2 Machine Learning

7.2.1 Additional Data Sources

For future work, the most pressing task would be to add additional data sources, in particular transportation data such as bus and luas routes, bus stops and luas stations. Information on the water supply and the electrical infrastructure would equally be helpful to add, in particular information pertaining to the municipal water supply, the power grid, off the grid solar panels and backup generators. Furthermore, the current amenities should be extended to include more counties, close to Dublin such as Fingal, South Dublin and Meath, as well as extended to larger cities in Ireland like Cork, Galway, Limerick and Sligo.

7.2.2 YOLO Model Improvements

Retraining the YOLO model on a dataset containing more manually annotated satellite images should be explored in order to improve the model’s performance and reduce the number of false positives.

Experimenting with additional fine-tuning of the YOLO model and exploring lighter versions of YOLO, such as YOLOv8-nano should be explored, to see if it yields better results.

7.2.3 General Parking Detection Improvements

Improving the road mask’s accuracy in classifying cars as on the road and parked correctly is especially crucial, as the current misclassifications lead to issues afterwards particularly in the unoccupied parking detection section. It could be interesting to test out enlarging the roads by a certain ratio, depending on the road width to begin with in order to find a way of identifying only larger roads with multiple lanes.

The unoccupied parking detection could be further improved by adding additional heuristics to find the empty spots more efficiently. Moreover, the 2 cases that were removed as too many additional spots were being detected could be refined and implemented differently.

It would be interesting to explore methods to extend the detection to be able to identify partially occluded vehicles, as cars are often hidden behind trees, in shadows or near the edge of the images.

7.3 Frontend

In the future, one of the major focuses on the frontend development of Magpie will be the implementation of user feedback and the improvement of the look and feel of the platform. The current interface provides a great skeleton, but user interaction throughout the platform and accuracy of data displayed need a little more work.

7.3.1 Enhance Visual feedback

One of the main areas of improvement is in providing better **visual feedback** for certain features. Specifically, users have asked for more dynamic responses while interacting with the **search feature** and the **amenity selection/deselection**. Implementation of animations or progress indicators will make the process smoother and more intuitive, especially when users are filtering or navigating through large data. This could be followed by increasing the **precision** of the **points shown on** the map. The increased accuracy will give users a better place-marking experience and may also go on to increase the reliability of the information provided.

7.3.2 Integrate User Feedback

The feedback from these **target users** has been gold. In the user interviews, each urban planner brings varied experiences and a need for different kinds of requirements; hence, every single approach became relevant.

7.3.3 More Data

This was a clear indication of the need for feature customization to suit these diverging expectations. Hence, further work will involve the inclusion of **more data** in the tool in order to give a richer and more detailed view of amenities. Moreover, the **tooltip information** of each amenity will be enhanced with more specific and actionable data that will make the tool useful for more detailed urban planning tasks.

7.3.4 Export Data

Data export functionality- There has also been interest expressed in an **export feature** to take the data off-line and further analyze them. That will be also prioritized, so that planners who do need to make presentations of their findings or pass on some of their research can really use this tool much more effectively.

7.3.5 Improved Onboarding

This will also **improve** the onboarding experience for better acquaintance with the tool. Adding clearer instructions, tooltips, and maybe a small intro tutorial would let first-time users feel confident navigating through the interface. Nevertheless, considering those points that can be improved, the rating of the platform was brilliant: **4.8 out of 5** in terms of user satisfaction, what really gives an anchor to the vision and end this project had in mind, such as creating an intuitive and user-friendly tool to deal with a comprehensive overview of urban amenities in a given region. This will further substantiate the importance of Magpie to consolidate into one of the key sources at whatever stage on the subject matter of urbanism.

7.4 Backend

7.4.1 Application Security

Fix Issues with JWTs

As mentioned in the discussion of JSON Web Tokens on page 80, the use of JWTs comes with its own set of caveats.

The most pressing of these issues is the missing token revocation functionality. One solution for these issues would be implementing a list of revoked tokens on the server and offering the user a way to add a stolen token to that list. If an attacker then uses one of these revoked tokens, their requests can be easily denied.

Implement Email-based Features

Right at the end of the project, the team experimented with email-based features like email verification and password reset. Unfortunately, emails sent by our system were always either classified as spam or didn't get delivered at all as they were considered untrustworthy. Since the deadline was fast approaching, the team decided to abandon this idea and focus on other issues.

Nonetheless, these features are essential components for any public facing web application. This is why one of the next steps would be to implement email verification and password reset using a service like SendGrid. This way, the issue of emails being blocked automatically should be mitigated.

Replacing bcrypt

Switching out bcrypt for Argon2id is not as pressing as other issues on this list as it is still functional and secure, but sometime in the future it would be a good idea to switch to a newer password encryption solution. This would bring Magpie in line with the current recommendations for secure password storage by the Open Worldwide Application Security Project (OWASP) (2024b).

Replacing the bcrypt library would make it necessary to reset the passwords of all users, as it is not possible to convert a bcrypt hash to a Argon2id hash. This should therefore be done before the number of users grows to affect as few people as possible.

The issue of improving application security was kept in mind during the whole development process, but other issues always took precedence. Given more time, improving the application security would be one of the top priorities in terms of further backend development.

7.4.2 Performance

Improve Data Transmission Speed

Currently, when a user selects a large area of the map it can take a couple of seconds from when they place the circle until the points appear on the map. This is due to the data being sent in one big chunk which then needs to be processed completely before the points can be shown. This is not ideal and better responsiveness could improve the user experience.

To achieve this, the data would need to be transmitted in smaller chunks. This would allow the frontend to start processing and showing the points even while not all of them are loaded. While it would still take at least the same time to transmit all of the data, the user would be presented with some data earlier, making the application feel much more responsive.

One technical solution to this problem would be streaming the data between backend and frontend. This could be implemented using Server-Sent Events (SSEs), which is a one-way data transfer solution that allows for progressive delivery of large amounts of data. Alternatively, a WebSocket connection could be used to achieve a similar result.

Database Optimisation

As discussed in the section on database development on page 82, the database is not considered to be Magpie's biggest bottleneck. As such and due to limited time, other issues took precedence, so there is still room for optimisations in the database.

A straightforward optimisation would be to add a geospatial index to the points table. Using geospatial indexing would help the database eliminate many rows early in the querying process, leading to a much more efficient search. Some sources achieved a six times speed increase without any other optimisations (Ramsey P., 2023). The team recognises that such an index should have been added early in development, as it would reduce retrieval time and could make the application as a whole more responsive.

The query used to retrieve the points in a radius could also benefit from some optimisation. Currently, it uses the PostGIS `ST_DWithin` function to find points in a radius. To reduce the number of rows that the query needs to compare, a bounding box filter could be used before calling `ST_DWithin`. The query currently casts the `LongLat` column of type `geometry` to the `geography` type for every row on each request. Converting the point to `geography` is necessary for calculating the distance in meters, but it's possible to pre-calculate and store both types in the table to eliminate unnecessary computation.

7.4.3 Fully Automated Data Ingress

At the moment, new data is added to the system by manually running a Python script. This works for the current scope of the project, but should Magpie's service area be expanded in the future this would not be sustainable. Therefore, a fully automated solution should be put in place. This would take the form of a container that has facilities to add new data to the database, either when new datasets become available or upon user request.

For example, if a user queries an area for parking spots that is not currently covered by Magpie, the backend could send a request to the data container which would then fetch, classify and insert the data. While this approach would be too slow to provide the data in a timely manner to the user who requested it, it could still be stored in the database for when the next user requests it. There could even be a function where users get a notification via email when an area they queried has data added to it.

One challenge with this feature would be figuring out where new datasets overlap with old datasets. Since the points for parking are generated by a machine learning model, it is possible that point coordinates for the same parking spot might differ slightly between runs. It's possible to just discard points with the exact same coordinates, but there is no guarantee that no parking spot will be counted multiple times.

A different problem arises with external datasets (such as from `gov.ie`). While the coordinates of amenities such as public bins will not change much, it is not safe to assume that the format of the dataset will not change between releases. Designing an automated data ingress solution to handle arbitrary data formats correctly would probably be a project in itself.

Solving these problems would enable Magpie to serve a larger area with less reliance on manual input.

8 Conclusion

The increasing demands of urbanization necessitate innovative solutions to streamline city planning and promote sustainability. **Magpie** represents a novel geographical information system that automates the identification, aggregation, and visualization of public amenities. By addressing the fragmentation of traditional data sources and reducing reliance on manual methods, Magpie provides urban planners with actionable insights, enhancing decision-making processes and infrastructure analysis.

8.1 Project Management Strategy

The project was developed following an agile methodology, which enabled iterative design, development, and evaluation. Weekly sprint cycles and continuous integration practices ensured consistent progress and adaptive responses to challenges. Task management through GitHub, combined with pull requests and structured code reviews, promoted team collaboration and accountability. This methodology facilitated the alignment of individual contributions with the overarching project goals.

8.2 Challenges and Solutions

The team encountered several key challenges during the project lifecycle:

- **Labelling Software:** Finding an image labelling software was an initial hurdle. Ultimately, Label Studio, a popular labelling software was chosen for its easy to use interface and easily exported labels.
- **Data Quality and Extraction:** Low-resolution satellite imagery, seasonal variations, and occlusion presented obstacles to accurate object detection. To address this, the YOLOv8-OBb model was employed, offering high accuracy in detecting and classifying public amenities.
- **Scalability and Performance:** Processing large-scale datasets required optimization of machine learning pipelines and backend systems. By fine-tuning thresholds and leveraging efficient clustering algorithms such as DBSCAN, performance bottlenecks were mitigated.
- **Evaluation of the ML submodules:** The careful selection and accurate labelling of each test set was key in correctly evaluating our model's performance.
- **Team Coordination:** The interdisciplinary nature of the project demanded effective communication and knowledge sharing across team members. Regular meetings and clearly defined roles fostered synergy and alignment toward common objectives.

8.3 Key Contributions

The primary contribution of this project lies in the development of a robust system capable of automating the detection and analysis of public infrastructure. Unlike existing GIS tools that often require significant manual effort or advanced expertise, Magpie combines machine learning, data fusion, and interactive visualizations to deliver a user-friendly platform.

Future work will focus on extending the capabilities of Magpie:

- **Enhanced Data Integration:** Incorporating real-time data streams from IoT sensors, traffic systems, and environmental indicators to improve the accuracy and relevance of insights.
- **Object Detection Expansion:** Refining models to include additional infrastructure, such as bus stops, pedestrian zones, and green spaces, thereby broadening the scope of analysis.
- **Customization Features:** Introducing user-defined parameters for region-specific queries and enabling multi-layered data overlays for advanced analysis.

8.4 Lessons Learned

The development of Magpie provided significant insights into the complexities of urban infrastructure analysis and machine learning applications:

- The quality and reliability of data, particularly from satellite imagery, pose inherent challenges that require advanced image processing and model optimization techniques.
- Automation of data extraction and visualization can dramatically streamline the planning process, highlighting the potential of AI-driven tools in real-world scenarios.
- Effective project management, collaboration, and adaptability are essential when addressing multifaceted technical and organizational challenges.
- Teamwork encourages learning from each other and sharing knowledge, allowing members to rely on each other's strengths and overcome challenges as a team.
- Clear communication as a team and iterative development were key to implementing user feedback and meeting our project requirements.

8.5 Strengths and Weaknesses of Final System

Magpie's strengths and weaknesses provide both an overview of the milestones achieved and the limitations encountered during project development.

Our system's strengths relate to the core completed objectives set out at project inception:

- **Ease of use** - the system features an intuitive interface that has gone through several iterations thanks to the valuable user feedback. Both domain experts, non-experts and UI/X & accessibility professionals cited the simplicity of Magpie's interface, placement of control elements and visual appeal.

- **Accessibility** - the user interface design prioritised accessibility to ensure users with varying levels of technological proficiency, English language and accessibility concerns. Damian Gordon evaluated our system and gave it an incredibly high score, allowing us to complete this objective.
- **High-level visualisation** - the system visualises more than a dozen, complex, fragmented data types into a clear, sleek map visualisation that has been complimented all throughout the user evaluation cycle.

The limitations encountered were a result of the novel techniques undertaken for the car parking detection as well as technical components misbehaving:

- **Time & Computational intensity** - all the steps of the machine learning section of the project were both extremely time & computational intensive. The YOLO model used for the car detection required significant amount of training time, limiting the speed with which the development could progress. The road mask required several adjustments and testing sessions to improve model metrics, and the classification of spots revealed core issues in the model's limitations.
- **System response time** - the time it takes for amenity data to load on the map is not as fast as we would've like. This is due to the method implemented for data retrieval which is not suited for the large amount of data being requested.

The system excels in several key areas including the ease of use and accessibility of the user interface which were key to the project's goals. However, some limitations arose due to the novel machine learning techniques and the method implemented for data retrieval to the frontend.

8.6 Final Remarks

This report has outlined the key technical, managerial, and analytical components of Magpie, emphasizing its contribution to urban planning through automated detection and data visualization. While significant progress has been achieved, the proposed extensions highlight avenues for further improvement. By bridging the gap between fragmented data systems and practical decision-making tools, Magpie represents a step toward accessible, data-driven, and sustainable urban development.

9 Appendix

9.1 Data Sources Appendix

Parking meter = <https://data.gov.ie/dataset/parking-meters-location-tariffs-and-zones-in-dublin-city>

Bike stand = <https://data.gov.ie/dataset/dublinbikes-api>

Public Wifi = <https://data.smartdublin.ie/dataset/wifi4eu-access-points-dcc>

Library = <https://data.smartdublin.ie/dataset/dublin-city-libraries>

Multi-storey Car park = <https://data.gov.ie/dataset/multi-story-car-parking-space-availability>

Drinking water fountain = <https://data.gov.ie/dataset/drinking-water-fountains-dublin-region/resource/e71cf-4475-8d48-666d66cdc2c3>

Public toilet = <https://data.gov.ie/dataset/public-toilets-dcc/resource/8796f653-4517-495a-b8b5-cb47157deaac>

Bike sharing station (Bleeper) = <https://data.gov.ie/dataset/bleeperbike>


Bike sharing station (Moby Bikes) = <https://data.gov.ie/dataset/moby-bikes>

Accessible parking = <https://data.gov.ie/dataset/accessible-parking-spaces-dcc>

Public bins = <https://data.smartdublin.ie/dataset/dcc-public-bin-locations>

Coach parking = <https://data.gov.ie/dataset/coach-parking-dcc>

9.2 User Evaluation Appendix



User Experience - Maggie

How was your experience using Maggie? Please let us know below with this short questionnaire

* Required

- What device did you use to browse through Maggie? *
- How smooth was the sign up/log in process? *
- How clear was the tutorial? *
- How did you find the filters? *
- How was your experience with the interface? *
- What was the best thing about the application and why? *
- What was the worst thing about the application and why? *
- What would change about the sign up/log in page? *
- What would you change about the dashboard? *
- What was your overall impression of Maggie? *
- Do you have any additional feedback/ comments on the application or your experience using it? *

Figure 96: *User Evaluation - Satisfaction survey questions*

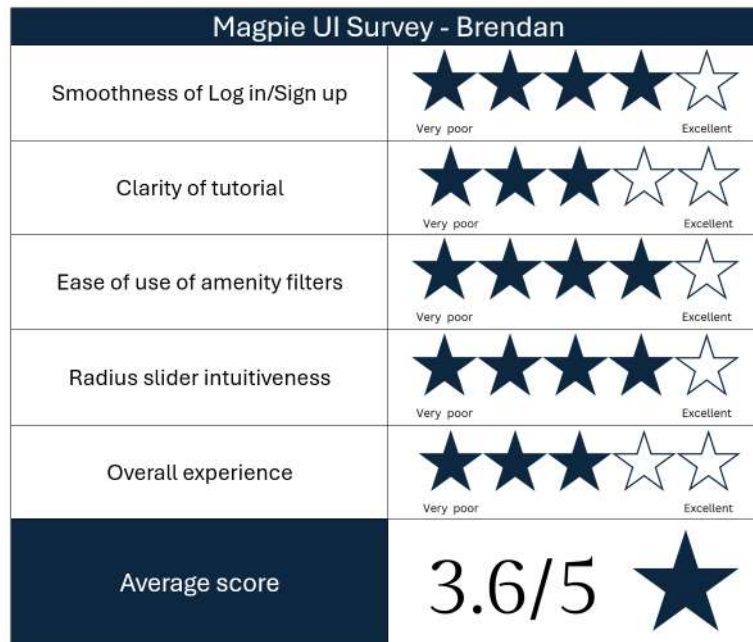


Figure 97: User Evaluation - UI Score Brendan

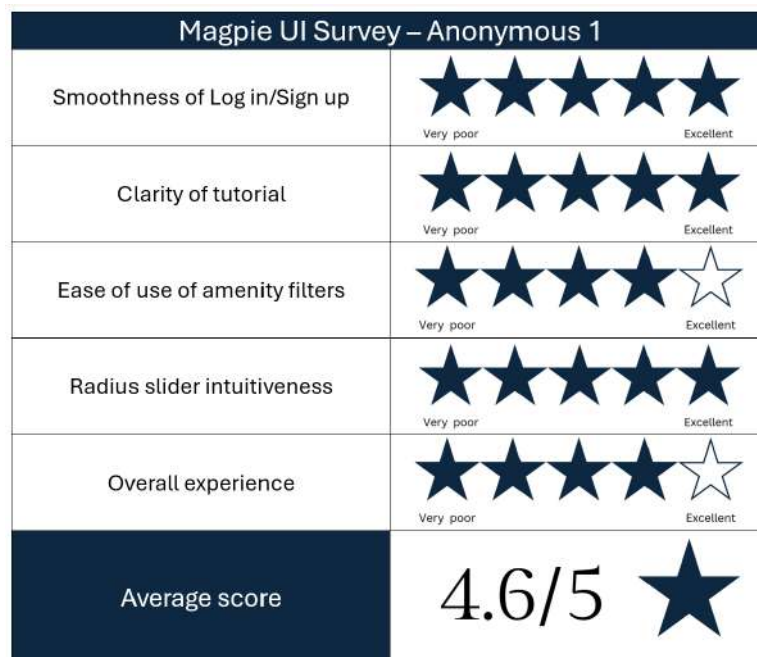


Figure 98: User Evaluation - UI Score Anonymous 1

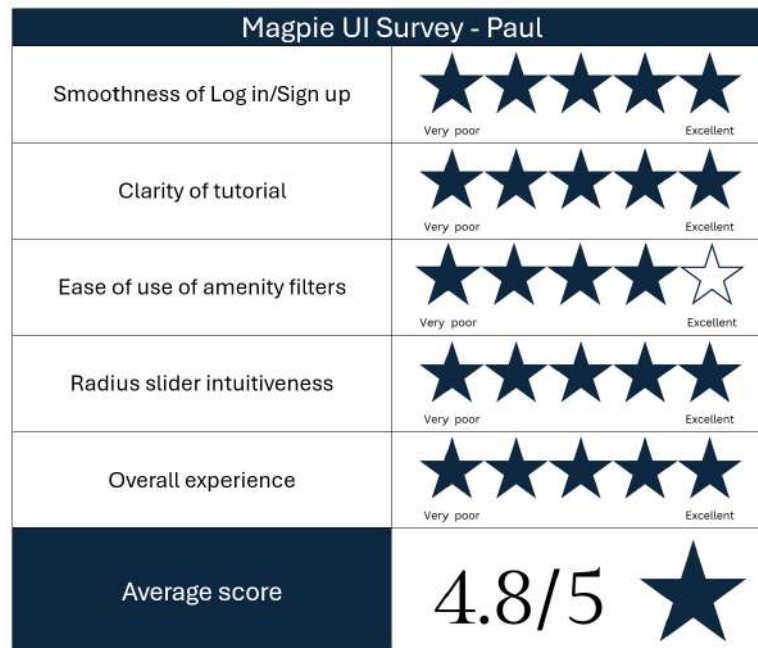


Figure 99: User Evaluation - UI Score Paul



Figure 100: User Evaluation - UI Score Livia



Figure 101: User Evaluation - UI Score Ben

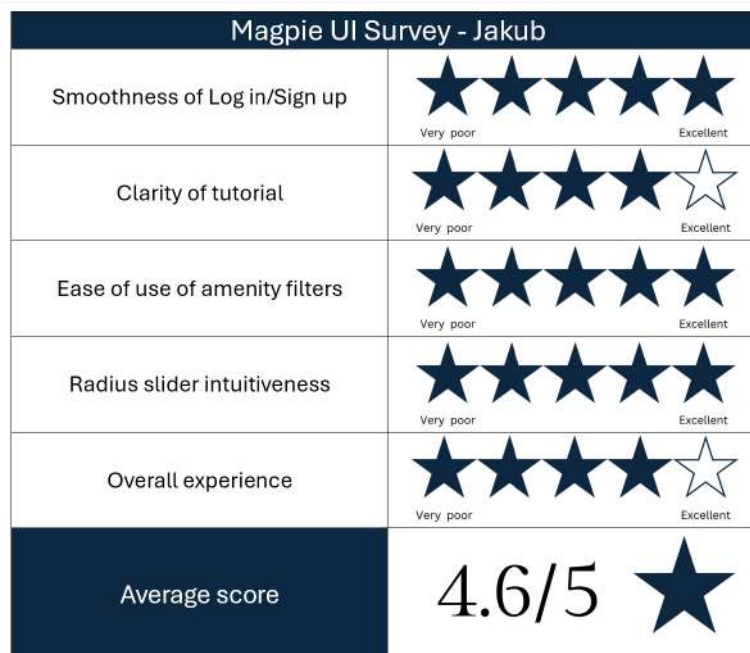


Figure 102: User Evaluation - UI Score Jakub



Figure 103: User Evaluation - UI Score Bryan Boyle

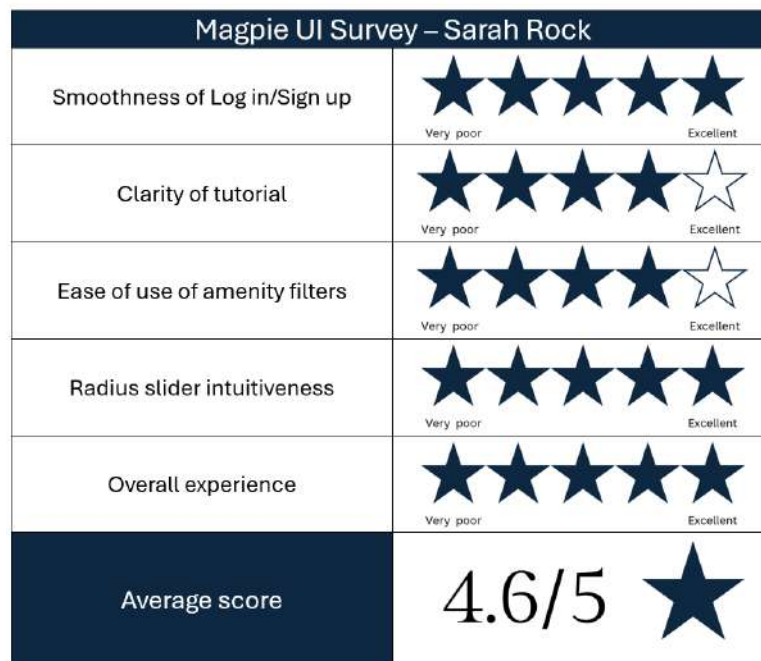


Figure 104: User Evaluation - UI Score Dr. Sarah Rock

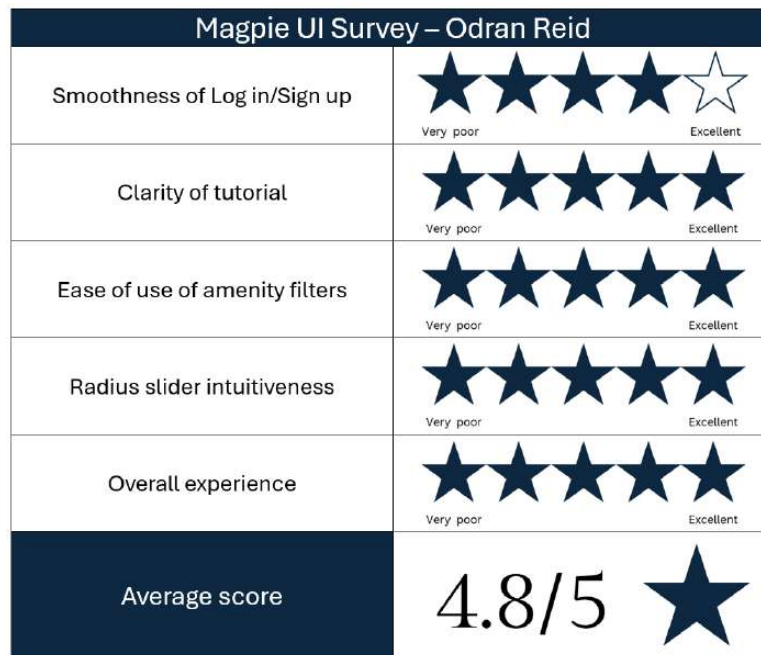


Figure 105: User Evaluation - UI Score Odran Reid

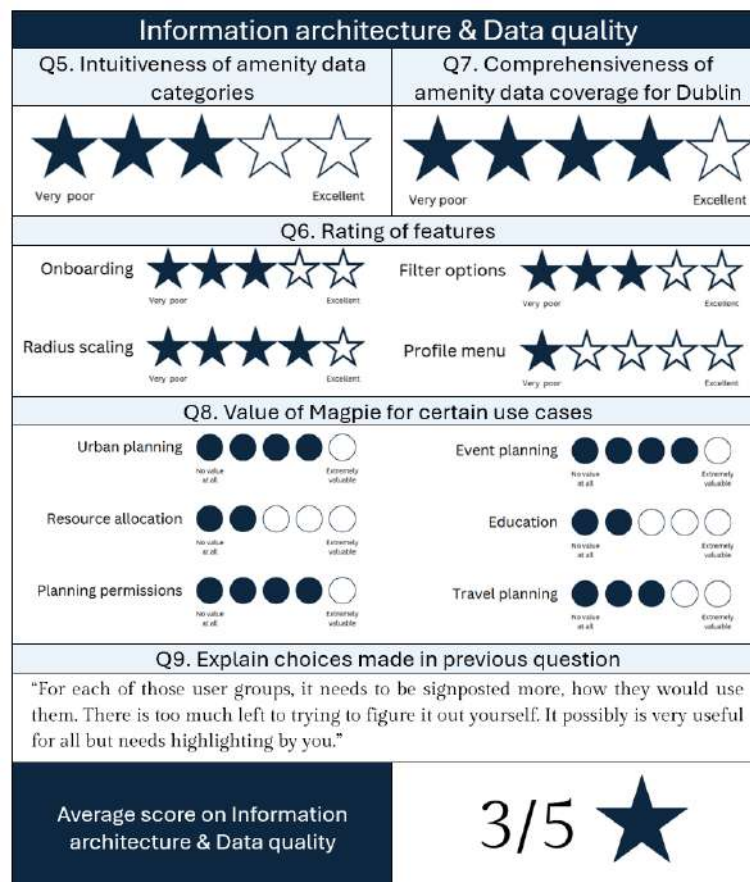


Figure 106: UX review 1 - Information Architecture score



Figure 107: UX review 1 - Technical Performance score

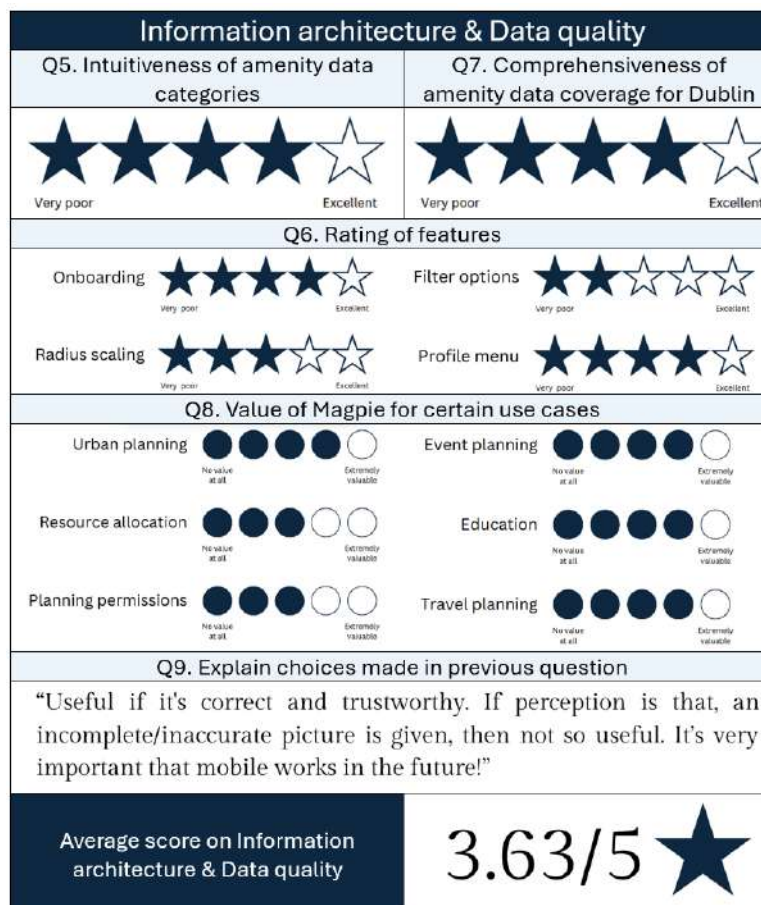


Figure 108: UX review 2 - Information Architecture score

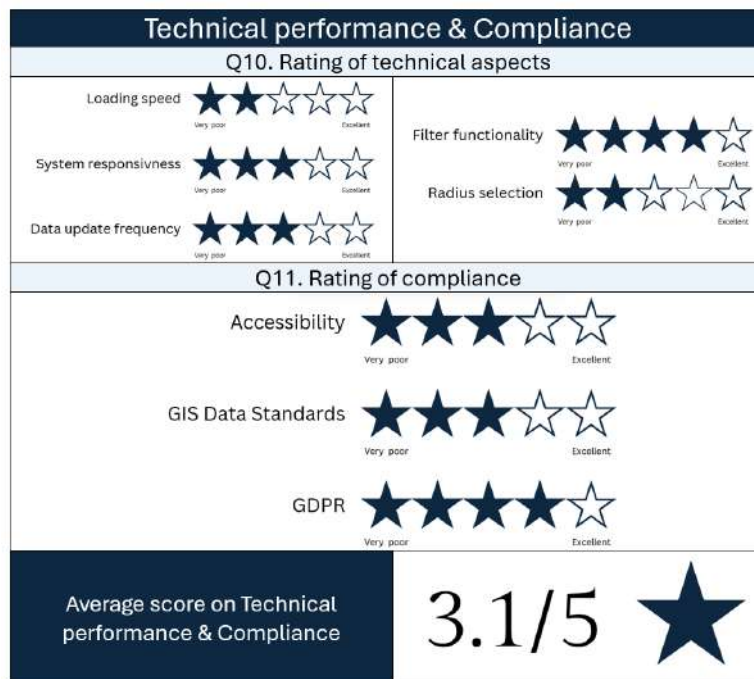


Figure 109: UX review 2 - Technical Performance score

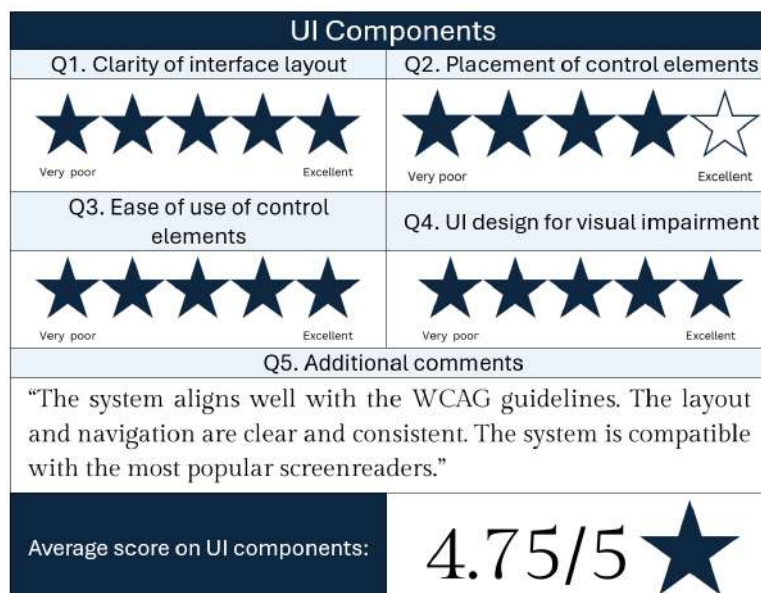


Figure 110: Accessibility review - UI Components Score

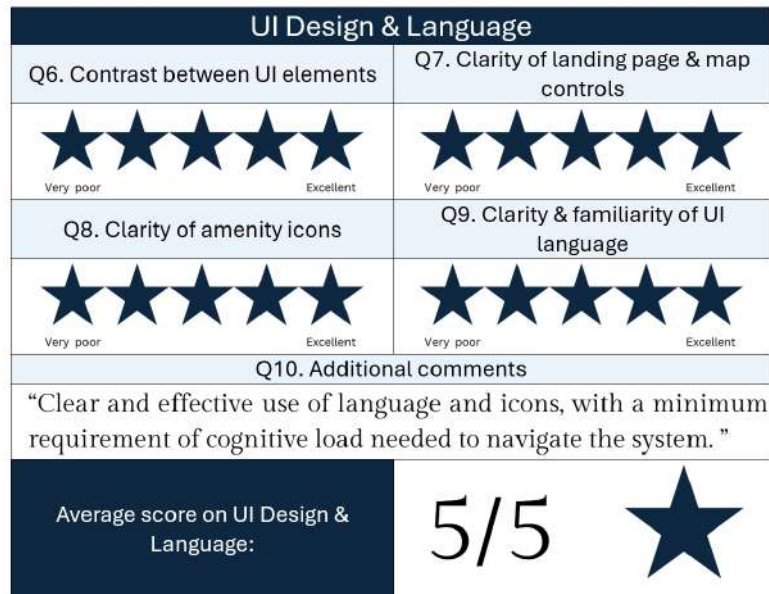


Figure 111: Accessibility review *UI Design & Language Score*

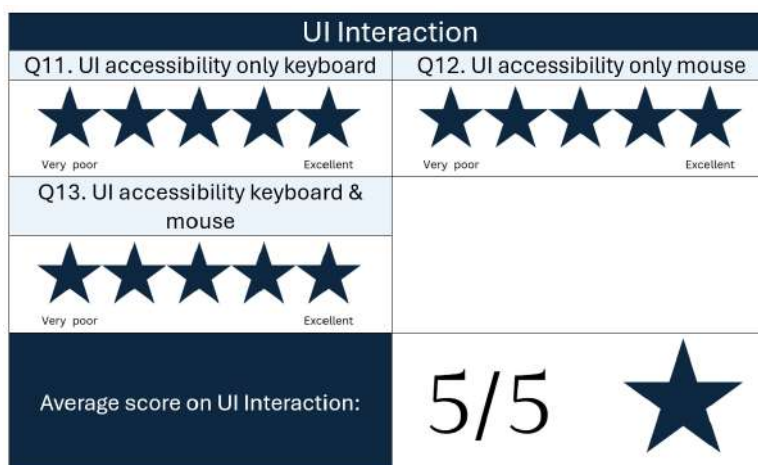


Figure 112: Accessibility review *UI Interaction Score*

References

- Ahmed, Maaz (2023). *The Power of UI and UX: How It Drives User Satisfaction*. Psycray. URL: <https://psycray.com/the-power-of-ui-and-ux-how-it-drives-user-satisfaction/>.
- Ai-Hong Chen Nurul-Farhana Abu Bakar, Carly Siu-Yin Lam (2020). *Comparison of open-ended and close-ended questions to determine signs and symptoms of eye problems among children*. URL: <https://pdf.sciencedirectassets.com/...>
- Alice Moore Bryan Boyle, Helen Lynch (2023). *Designing public playgrounds for inclusion: a scoping review of grey literature guidelines for Universal Design*. URL: <https://doi.org/10.1080/14733285.2022.2073197>.
- Allen, Natalie (2015). “Understanding the importance of urban amenities: A case study from Auckland”. In: *Buildings* 5.1, pp. 85–99.
- Bryan Boyle, Inmaculada Arnedillo-Sanchez (2022). *The Inclusion of Children on the Autism Spectrum in the Design of Learning Technologies: A Small-Scale Exploration of Adults’ Perspective*. URL: <https://doi.org/10.3389/feduc.2022.867964>.
- Clark, Terry Nichols et al. (2002). “Amenities drive urban growth”. In: *Journal of urban affairs* 24.5, pp. 493–515.
- Duivenvoorden, Eelco et al. (2021). “Managing public space—A blind spot of urban planning and design”. In: *Cities* 109, p. 103032. DOI: 10.1016/j.cities.2020.103032.
- Ebert, Christof et al. (2016). “DevOps”. In: *IEEE Software* 33.3, pp. 94–100. DOI: 10.1109/MS.2016.68.
- Fatma M. Talaat, Hanaa ZainEldin (2023). “An improved fire detection approach based on YOLO-v8 for smart cities”. In: *Neural Computing Applications*. DOI: 10.1007/s00521-023-08809-1.
- Fischler, Raphael (2012). “Fifty theses on urban planning and urban planners”. In: *Journal of Planning Education and Research* 32.1, pp. 107–114. DOI: 10.1177/0739456X11420441. URL: <https://journals.sagepub.com/doi/pdf/10.1177/0739456X11420441>.
- Gray, Kyle (2019). *Introducing sqlc - Compile SQL queries to type-safe Go*. Accessed: 2024-12-10. URL: <https://conroy.org/introducing-sqlc>.
- (2024a). *sqlc GitHub Repository Q & A*. Accessed: 2024-12-10. URL: <https://github.com/sqlc-dev/sqlc/discussions/3577>.
 - (2024b). *sqlc Language and Database Support*. Accessed: 2024-12-10. URL: <https://docs.sqlc.dev/en/latest/reference/language-support.html>.
 - (2024c). *sqlc Modifying the database schema*. Accessed: 2024-12-13. URL: <https://docs.sqlc.dev/en/latest/howto/ddl.html>.
- Hamidli, Nasrullah (2023). *Introduction to UI/UX Design: Key Concept and Principles*. URL: <https://www.studocu.com/in/document/anna-university/ui-and-ux-design/introduction-to-ui-ux-design-key-concept/79011825>.
- Hennig S. Zobl F., Wasserburger W.W. (2017). *Accessible Web Maps for Visually Impaired Users: Recommendations and Example Solutions*. URL: <https://cartographicperspectives.org/index.php/journal/article/view/cp88-pr/1574>.
- Hussain, Muhammad (2024). “YOLOv1 to v8: Unveiling Each Variant- A Comprehensive Review of YOLO”. In: *IEEE Access* 12. DOI: 10.1109/ACCESS.2024.3378568.
- Ireland, Vision (2023). *Facts about Sight Loss*. URL: <https://vi.ie/policy-advocacy/facts-about-sight-loss/>.
- IxDF (2016). *What are design guidelines?* URL: <https://www.interaction-design.org/literature/topics/design-guidelines>.

- Janpavle, Irena and Una Īle (2022). “The importance of active leisure areas in the context of urban planning”. In: *Rigas Tehniskas Universitates Zinatniskie Raksti* 18.1, pp. 120–130.
- Jha, A.K. et al. (2021). “A review of AI for urban planning: Towards building sustainable smart cities”. In: *2021 6th International Conference on Inventive Computation Technologies (ICICT)*. IEEE, pp. 937–944. DOI: 10.1109/ICICT50816.2021.9358596.
- Kapoor, Vipul (2024). *Oriented bounding box (OBB): All you need to know*. URL: <https://mindkosh.com/blog/oriented-bounding-box-annotation-all-you-need-to-know/>.
- Kaufmann, Talia et al. (2022). “Scaling of urban amenities: generative statistics and implications for urban planning”. In: *EPJ Data Science* 11.1, p. 50.
- Lei, Yulong, Johannes Flacke, and Nina Schwarz (2021). “Does Urban planning affect urban growth pattern? A case study of Shenzhen, China”. In: *Land Use Policy* 101, p. 105100. DOI: 10.1016/j.landusepol.2020.105100.
- Lynn, Theodore et al. (Apr. 2023). “Web Accessibility of Irish Local Government Websites”. In: *ICDS 2023 : The Seventeenth International Conference on Digital Society*.
- McGuirk, Pauline M and Andrew MacLaran (2001). “Changing approaches to urban planning in an ‘entrepreneurial city’: the case of Dublin”. In: *European Planning Studies* 9.4, pp. 437–457.
- Mehrshad Lalinia, Ali Sahafi (2023). “Colorectal polyp detection in colonoscopy images using YOLO-V8 network”. In: *Singla, Image and Video Processing*. DOI: 10.1007/s11760-023-02835-1.
- Moran, Kate (2019). *Usability (User) Testing 101*. URL: <https://www.nngroup.com/articles/usability-testing-101/>.
- Mugion, Roberta Guglielmetti et al. (2018). “Does the service quality of urban public transport enhance sustainable mobility?” In: *Journal of Cleaner Production* 174, pp. 1566–1587. ISSN: 0959-6526. DOI: <https://doi.org/10.1016/j.jclepro.2017.11.052>. URL: <https://www.sciencedirect.com/science/article/pii/S0959652617327105>.
- Nielsen, Jakob (2012). *Usability 101: Introduction to Usability*. URL: <https://www.nngroup.com/articles/usability-101-introduction-to-usability/>.
- Open Worldwide Application Security Project (OWASP) (2024a). *OWASP JSON Web Tokens Cheat Sheet*. Accessed: 2024-12-15. URL: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web-Token_for_Java_Cheat_Sheet.html.
- (2024b). *OWASP Password Storage Cheat Sheet*. Accessed: 2024-12-14. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.
- Organisation, World Health (2019). *World Report on Vision*. URL: <https://iris.who.int/bitstream/handle/10665/328717/9789241516570-eng.pdf?sequence=18>.
- Poirson, Patrick et al. (Sept. 2016). “Fast Single Shot Detection and Pose Estimation”. In: DOI: 10.48550/arXiv.1609.05590.
- Ramsey P., Leslie M. (2023). *PostGIS Workshop: Spatial Indexing*. Accessed: 2024-12-18. URL: <https://postgis.net/workshops/postgis-intro/indexing.html>.
- Takur, Namrata (2023). *A detailed introduction to Two Stage Object Detectors*. URL: <https://namrata-thakur893.medium.com/a-detailed-introduction-to-two-stage-object-detectors-d4ba0c06b14e>.
- Tanadechopon, Teerapong and Boontariga Kasemsontitum (Nov. 2023). “Performance Evaluation of Programming Languages as API Services for Cloud Environments: A Comparative Study of PHP, Python, Node.js and Golang”. In: *2023 7th International Conference on Information Technology (InCIT)*. IEEE, 293–297. DOI: 10.1109/incit60207.2023.10413079. URL: <http://dx.doi.org/10.1109/InCIT60207.2023.10413079>.
- Ultralytics (2024). *COCO Dataset*. URL: <https://docs.ultralytics.com/datasets/detect/coco/>.

Union, European (2016). *Directive (EU) 2016/2102 of the European Parliament and of the Council on the accessibility of the websites and mobile applications of public sector bodies*. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016L2102&from=EN>.

Zambanini, S. et al. (2020). “Detection of Parking Cars in Stereo Satellite Images”. In: *Remote Sensing* 12.13, p. 2170. DOI: 10.3390/rs12132170.