|  |  |
| --- | --- |
| ***Are-u-Queryous?* A Web-Based Platform for Democratizing Open Geospatial Data Access**  *From Queries to Maps, A New Way to See the World!* | |
|  | |
| Shape, rectangle  Description automatically generated | **Nicolas D’Alessandro Calderon**  Bachelor's degree in Techniques for Software Application Development  Localization Based Systems and Intelligent Spaces  **Project supervisor**  Joaquín Torres Sospedra  **Coordinating professor**  Antoni Perez-Navarro  **Date of submission**  April 01, 2025 |

  
This work is distributed under a

Creative Commons [Attribution-NonCommercial-NoDerivs](https://creativecommons.org/licenses/by-nc-nd/3.0/) 3.0 license.

**SUMMARY OF THE FINAL PROJECT**

|  |  |
| --- | --- |
| **Title of the project:** | *Are-u-Queryous? A Web-Based Platform for Democratizing Open Geospatial Data Access* |
| **Author name:** | *Nicolas D’Alessandro Calderon* |
| **Project supervisor:** | *Joaquín Torres Sospedra* |
| **Coordinating professor:** | *Antoni Perez-Navarro* |
| **Date of submission (MM/YYYY):** | *04/2025* |
| **Name of the degree:** | *Bachelor's degree in Techniques for Software Application Development* |
| **Topic of the final project:** | *Localization Based Systems and Intelligent Spaces* |
| **Language:** | *English* |
| **Keywords:** | *Geospatial Data, Open Data Visualization, Urban Analytics* |
| **Abstract** | |
| This study explores the challenge of making open data more accessible to the general public, addressing the gap between the availability of geospatial information and its practical use. Are U Query-ous? is a web-based application designed to enable individuals, regardless of their technical background, to explore and interpret geographic and demographic data intuitively. By integrating interactive maps and user-friendly visualization tools, the platform allows users to analyze regions based on economic activity, population distribution, and local trends.  Through a simplified interface, users can explore and filter publicly available data, identifying patterns relevant to their interests. The application is intended for individuals who are curious about urban development, seeking optimal locations for personal or professional activities, or analyzing demographic trends for research or decision-making. Additionally, the project examines the potential integration of artificial intelligence to facilitate data retrieval through natural language queries, further enhancing accessibility.  The development and results of the analyses in this data scientific report are intended to address all the concepts targeted above, but the author also expects to describe some fundamental principles underlying web apps development and data science. | |

Index

[01 INTRODUCTION 1](#_Toc194014748)

[1.1 Context and motivation 1](#_Toc194014749)

[1.2 Goals 4](#_Toc194014750)

[1.3 Sustainability, diversity, and ethical/social challenges 5](#_Toc194014751)

[1.4 Approach and Methodology 6](#_Toc194014752)

[1.5 Schedule 7](#_Toc194014753)

[1.6 Summary of the outputs of the project 11](#_Toc194014754)

[1.7 Brief description of the remaining chapters of the report 14](#_Toc194014755)

[02 STATE OF THE ART 15](#_Toc194014756)

[2.1. Current Geospatial Data Platforms 15](#_Toc194014757)

[2.2. Geospatial Data Visualization Technologies 16](#_Toc194014758)

[2.3. AI Integration in Geospatial Analysis 16](#_Toc194014759)

[2.4. Challenges in Democratizing Geospatial Data 17](#_Toc194014760)

[2.5. Applications of Geospatial Data 17](#_Toc194014761)

[2.6. Similar Software & Related Technologies 18](#_Toc194014762)

[2.7. Recent Academic Research & Case Studies 18](#_Toc194014763)

[2.8. Future Trends in Democratizing Geospatial Data 20](#_Toc194014764)

[References: 21](#_Toc194014765)

[03 SYSTEM DESIGN 24](#_Toc194014766)

[3.1 User Stories 24](#_Toc194014767)

[3.2 Use Case Diagram 29](#_Toc194014768)

[3.3 User interface design 30](#_Toc194014769)

[3.4 Navigation Flow 35](#_Toc194014770)

[3.5 Architecture Diagram 36](#_Toc194014771)

[3.6 Database design 38](#_Toc194014772)

[3.7 Class Diagram 40](#_Toc194014773)

[04 RESULTS 51](#_Toc194014774)

[05 CONCLUSIONS AND FUTURE WORK 76](#_Toc194014775)

[06 GLOSSARY 77](#_Toc194014776)

[07 BIBLIOGRAPHY 78](#_Toc194014777)

[08 APPENDICES 79](#_Toc194014778)

**List of figures**

**01 INTRODUCTION**

[Figure 1 1: Conceptual Representation of Project Summary 1](#_Toc193743104)

[Figure 1 2: Key Motivations for the Project 3](#_Toc193743105)

[Figure 1.3: Goals Breakdown 4](#_Toc193743106)

[Figure 1 4: Process Diagram 7](#_Toc193743107)

[Figure 1 5: Expected Outcome Visual Representation 11](#_Toc193743108)

[Figure 1 6: Technology Stack Used 13](#_Toc193743109)

**03 SYSTEM DESIGN**

[Figure 3 1: Use Case Diagram 29](#_Toc194021203)

[Figure 3 2 - Home Map View 30](#_Toc194021204)

[Figure 3 3 - Compare Areas View 31](#_Toc194021205)

[Figure 3 4 - Visualize Data View 32](#_Toc194021206)

[Figure 3 5 - Admin Upload Datasets View 33](#_Toc194021207)

[Figure 3 6 - Admin Features & Indicators View 33](#_Toc194021208)

[Figure 3 7 - Monitor Platform Performance View 34](#_Toc194021209)

[Figure 3 8 - Navigation Flow between the different Views 35](#_Toc194021210)

[Figure 3 9 - Architectural Diagram 37](#_Toc194021211)

[Figure 3 10- Entity Relation Diagram 38](#_Toc194021212)

[Figure 3 11 - UML Class Diagram 40](#_Toc194021213)

**List of Tables**

**01 INTRODUCTION**

[Table 1.1: Project Planning Phase 7](#_Toc194014812)

[Table 1.2: Design Phase 8](#_Toc194014813)

[Table 1.3: Implementation Phase 8](#_Toc194014814)

[Table 1.4: final Product and Report 9](#_Toc194014815)

[Table 1.5: Presentation Preparation Phase 9](#_Toc194014816)

[Table 1.6: Defence Preparation Phase 10](#_Toc194014817)

[Table 1.7: Ongoing Tasks Throughout the Project 10](#_Toc194014818)

**03 SYSTEM DESIGN**

[Table 3 1 Entities, Field, Types and Foreign Keys 39](#_Toc194014787)

01  
 INTRODUCTION

1.1 Context and motivation

This final project focuses on developing an intuitive **geo-analytics platform** that simplifies the exploration of open data through **interactive maps**. Many open data sources provide valuable insights into **urban planning, demographics, and economic activity**, but non-experts often struggle to extract meaningful information from them.

***Are-u-Queryous****?* aims to solve this problem by creating a **user-friendly interface** that enables individuals to explore and analyze spatial data without requiring technical expertise.

Additionally, if time permits, the project will explore the integration of a natural language processing model, to enable users to interact with the data using natural language queries. **This feature is considered an enhancement rather than a core requirement and will be evaluated based on project timelines** and feasibility.

At the end of the project, the system will provide a **fully functional prototype** that allows users to:

* **Filter** and **visualize** location-based open data.
* Identify **regional patterns and trends** based on economic and demographic factors.
* Utilize **intelligent search capabilities** to access relevant insights more intuitively.

A diagram of a diagram

AI-generated content may be incorrect.

Figure 1 1: Conceptual Representation of Project Summary

*Scope*

This project focuses on developing a web-based platform that allows users to explore and analyze open geospatial data in an intuitive way. The platform will integrate publicly available datasets from Barcelona and Madrid, two cities with well-structured open data portals that provide reliable and detailed information.

The scope includes the following key aspects:

* **Cities Covered:** The platform will use **open data from Barcelona and Madrid**, ensuring access to urban mobility, economic activity, and demographic datasets for meaningful analysis.
* **Data Integration**: The system will collect and process geospatial, economic, and mobility-related data, allowing users to filter and visualize insights interactively.
* **User Interaction**: The web application will feature an interactive map where users can explore regional trends, compare different areas, and extract useful insights without needing technical expertise.
* **Core Functionalities:**
  + *Mapping & Visualization*: Users will view geospatial data overlaid on maps, with filtering options.
  + *Urban & Economic Insights*: The system will present mobility trends, population density, and economic indicators based on selected areas.
  + *Accessibility & Usability*: The interface will be simple and user-friendly, ensuring that both professionals and non-experts can use it effectively.

By **limiting the project scope to Barcelona and Madrid**, the system will leverage well-organized open datasets while **maintaining a manageable level of complexity within the project timeline**. The structured data availability from these cities will support the development, testing, and validation of the platform, ensuring that it meets its intended objectives.

*Rationale*

Access to open data has grown exponentially, yet many users struggle to transform this data into actionable insights. While businesses and government entities benefit from sophisticated **geo-analytics tools**, individuals and small organizations often lack the resources or expertise to use these datasets effectively. This project is relevant because it seeks to **bridge this gap**, making open data truly accessible and usable for the **general public, students, researchers, and small businesses**.

Moreover, the relevance of geospatial analysis has expanded in fields such as **smart cities, sustainable urban development, and socio-economic research**. By providing an easy-to-use tool, this project supports the broader goal of promoting **data-driven decision-making at all levels of society**.

*Motivation*

In the past eight years I have been working and developing my professional career in the field of **data analysis**, so this project aligns with both academic and professional aspirations. The motivation for this project is coming from:

* A personal interest in **making complex data more understandable** for non-experts.
* A desire to provide **access to urban and economic and geomarketing insights** through intuitive visualization for the general public, students, researchers, and small businesses.
* The opportunity **to apply geospatial analytics in a real-world application**.

Additionally, the skills developed through this project, including data processing, backend development, frontend visualization, and user interface design, will be valuable in both academic research and professional settings.

**Key Motivations**

**Make data understandable**

**Provide access to insights**

**Apply geospatial analytics and geomarketing**

**Develop new skills**

Figure 1 2: Key Motivations for the Project

1.2 Goals

*Main Goal*

* To **develop a web-based app that enables users to intuitively explore, filter, and analyze open geospatial data**, making location-based intelligence more accessible to a non-technical audience.

*Sub-Objectives*

* **Develop an interactive mapping system** that allows users to visualize open data in an intuitive and engaging way.
* **Implement filtering and querying functionalities** to help users refine their search and extract relevant insights.
* **Ensure usability and accessibility** by designing a simple and intuitive user interface.
* **If time permits, experiment with integrating an NLP model** to allow natural language queries for filtering and searching data. This is considered a potential enhancement and not a primary project requirement.
* **Validate the effectiveness of the platform** through user feedback and iterative improvements.

A diagram of a network

AI-generated content may be incorrect.

Figure 1.3: Goals Breakdown

1.3 Sustainability, diversity, and ethical/social challenges

*Sustainability*

This project promotes sustainability by supporting the efficient **use of open data to improve urban and social planning**. By making geospatial information more accessible, individuals and organizations can make informed decisions about resource allocation, mobility, and land use, reducing unnecessary waste and inefficiencies. Additionally, the project aligns with the United Nations Sustainable Development Goals (SDGs), particularly [Goal 11 (Sustainable Cities and Communities),](https://sdgs.un.org/goals/goal11) by providing insights that encourage responsible urban development. The project has a minimal ecological footprint as it relies on existing digital infrastructure, avoiding additional resource consumption.

*Ethical behaviour and social responsibility*

The project considers ethical principles by **ensuring data privacy and security**. Since the system processes **publicly available open data**, it does not involve personal or sensitive information. However, the potential risks of misinterpreting data or using insights unethically are acknowledged. To mitigate this, the platform will provide **transparent data sources and disclaimers to ensure users understand the limitations of the information**.

*Diversity, gender and human rights*

The project is designed to be inclusive and accessible, allowing all individuals, regardless of background, gender, or technical expertise, to explore geospatial data. By offering a simple and user-friendly interface, it ensures that users with limited data experience can engage with geographic insights. Furthermore, **accessibility features will be considered**. The project aligns with the principle of equal access to information, promoting diversity and reducing barriers to data-driven knowledge.

1.4 Approach and Methodology

This project follows a structured development process **based on agile principles**, allowing for **iterative testing and feedback** throughout the semester. The key steps include:

*Data Collection and Processing*

* **Identify and integrate open datasets** (demographic, economic, urban mobility, etc.).
* **Clean and preprocess data** to ensure usability in the application.

*Backend Development*

* Build a **RESTful API** to serve geospatial data.
* **Store data in a database** optimized for efficient queries.

*Frontend Development*

* **Design a responsive user interface** with map-based interaction.
* **Implement data visualization tools** for filtering and exploration.

*Testing and Refinement*

* **Gather feedback** to enhance usability and features.

*Project Management and Development Workflow*

To manage the development process efficiently, we will use the Project Excel Gantt chart, which will help us visualize the project timeline, track progress, and ensure all milestones for *Are-u-Queryous?* are met on time.

1.5 Schedule

A detailed **schedule with milestones** has been defined, ensuring that the development process is structured, manageable, and aligned with the semester timeline following the structure of the phases defined in each CAT:

A graph with black rectangles

AI-generated content may be incorrect.

Figure 1 4: Process Diagram

CAT1: Project Planning Phase (Feb 19 - Mar 04)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 1.1 | * **Define project scope, research open datasets, and identify visualization requirements.** * **Define technology stack, assess integration challenges, and create a detailed timeline.** | Feb 19 - Feb 23 | - | Done |
| 1.3 | **Document sustainability considerations, set up development environment, and compile CAT1 documentation.** | Feb 24 - Mar 02 | 1.1 | Done |
| 1.4 | **Review and finalize CAT1 documentation.** | Mar 02 - Mar 03 | 1.1, 1.2 | Done |

Table 1.1: Project Planning Phase

CAT2: Design Phase (Mar 05 - Apr 01)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 2.1 | **Define user stories, database schema, and UI wireframes for the mapping interface.** | Mar 03 - Mar 09 | 1.4 | Done |
| 2.2 | **Design system architecture, research filtering algorithms, and create frontend class diagrams.** | Mar 10 - Mar 16 | 2.1 | Done |
| 2.3 | **Develop database documentation, implement login screen, and document state of the art in geospatial data visualization.** | Mar 17 - Mar 23 | 2.1 | Done |
| 2.4 | **Compile and review CAT2 documentation.** | Mar 23 - Mar 31 | 2.1 - 2.3 | Done |

Table 1.2: Design Phase

CAT3: Implementation Phase (Apr 02 - May 06)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 3.1 | **Set up backend database and implement data ingestion pipelines for open data.** | Apr 01 - Apr 06 | - | Done |
| 3.2 | **Develop backend services, authentication, and frontend components for map visualization.** | Apr 07 - Apr 13 | 3.1 | Done |
| 3.3 | **Implement filtering, integrate Leaflet.js, and develop data visualization components.** | Apr 14 - Apr 20 | 3.2 | Done |
| 3.4 | **Conduct unit testing, perform integration testing, and deploy prototype.** | Apr 21 - Apr 27 | 3.3, 3.4 | Done |
| 3.5 | **Compile implementation documentation and finalize CAT3 documentation.** | Apr 27 - May 05 | 3.1-3.4 | Done |

Table 1.3: Implementation Phase

**Important note**: The development strategy was modified to follow a frontend-first approach, leveraging Supabase as a BaaS solution. This allowed for faster deployment and removed the dependency on mock data during the initial stages.

CAT4: Final Product & Report (May 07 - Jun 03)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 4.1 | **Refine UI based on testing feedback.** | May 07 - May 11 | 3.14 | Done |
| 4.2 | **Implement advanced filtering, economic activity visualization, and optimize database queries.** | May 12 - May 18 | 4.1 | Done |
| 4.3 | **Implement (optional) NLP capabilities and conduct comprehensive system testing.** | May 19 - May 25 | 4.1 | Done |
| 4.4 | **Fix bugs, optimize performance, and prepare final deployment package. Create user documentation, write results and analysis, draft conclusions, and compile bibliography.** | May 26 - Jun 01 | 4.1 | Done |
| 4.5 | **Finalize and format the project report, then submit CAT4.** | Jun 01 - Jun 03 | 4.1-4.4 | Done |

Table 1.4: final Product and Report

CAT5: Presentation Preparation (Jun 04 - Jun 10)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 5.1 | * **Create presentation outline, storyboard, and design slides.** * **Prepare demonstration script, record key platform features, and create a narrated presentation video.** | Jun 04 - Jun 08 | 4.14 | - |
| 5.2 | **Review, finalize, and refine presentation based on feedback.** | Jun 08 - Jun 09 | 5.1 | - |

Table 1.5: Presentation Preparation Phase

Defence Preparation (Jun 11 - Jun 17)

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 6.1 | * **Prepare defence presentation based on feedback.** * **Anticipate potential questions and prepare responses.** * **Practice presentation delivery** | Jun 11-Jun16 | 5.2 | - |
| 6.2 | **Final defence presentation** | Jun 17 | 6.1 | - |

Table 1.6: Defence Preparation Phase

Ongoing Tasks Throughout the Project

| **Task ID** | **Task Description** | **Week** | **Linked** | **Status** |
| --- | --- | --- | --- | --- |
| 7.1 | **Gant Planning updates** | Feb 19 - Jun 17 | - | - |
| 7.2 | **Weekly supervisor check-ins** | Feb 19 - Jun 17 | - | - |
| 7.3 | **Documentation updates** | Feb 19 - Jun 03 | - | - |

Table 1.7: Ongoing Tasks Throughout the Project

For a detailed breakdown of the project plan, tasks, and timeline, please refer to the attached **are-you-queryous-planning.xlsx file.**

1.6 Summary of the outputs of the project

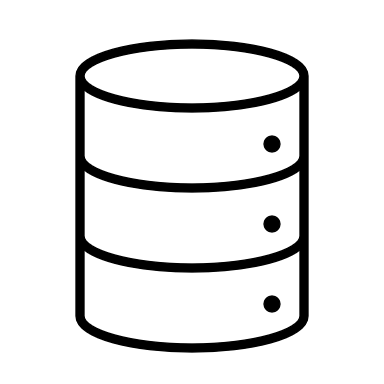
With its completion, the project will deliver:

* A functional web application where users can **explore and visualize geospatial data** interactively.
* An intuitive filtering system allowing users **to refine results based on key indicators**.
* A structured **ETL** serving **open datasets** with a focus on usability and efficiency.
* A research report detailing **the impact of accessible open data visualization**.

This project contributes to the broader goal of **making open data actionable and meaningful** for a **diverse audience**, reinforcing the importance of **geospatial intelligence in everyday decision-making**.

**Open Data**

**API**

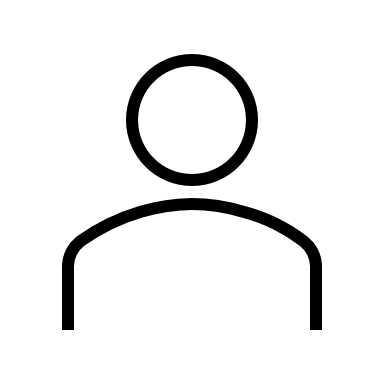


**Backend**

**Frontend**



**Are-u-Queryous? web App**



**ETL**

**User**

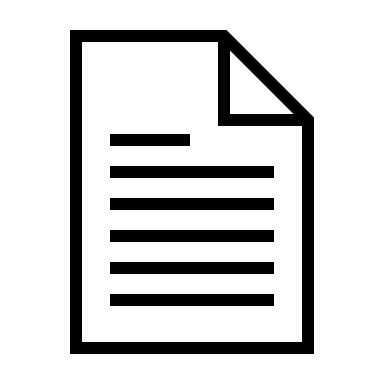


Figure 1 5: Expected Outcome Visual Representation

The project will leverage a modern web technology stack to ensure efficiency, scalability, and a seamless user experience. The core technologies include:

*Frontend:*

* **React.js** for building a dynamic and interactive user interface.
* **Leaflet.js** for mapping and geospatial visualization.

*Backend:*

* **Supabase** as a **BaaS** platform.
* **PostgreSQL/PostGIS** for storing and querying spatial data efficiently.

*Data Processing & Integration:*

* **Python** and **Pandas** for data preprocessing and transformation.
* **GeoJSON** format to represent geospatial data and serve it dynamically.

*Deployment & Hosting:*

* **GitHub Codespaces (Docker-integrated)** for cloud-based development
* **Vercel** for frontend deployment.
* **Supabase** for backend (DB, Auth, API, Storage).

*Potential AI Integration (Time-Permitting Feature):*

* The project may experiment **with integrating a pre-trained NLP model from** to process natural language queries.
* If implemented, a small widget will allow users to enter queries in plain text (*e.g., "Show me the most populated districts in Madrid").*
* This AI-based functionality is considered an exploratory addition, meaning it will only be developed if time and resources allow.

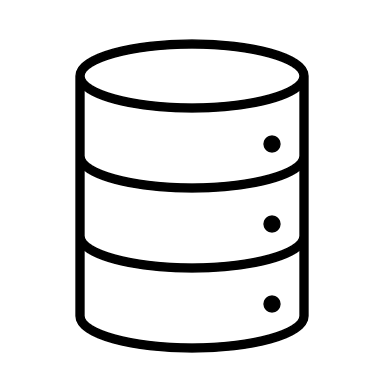
These technologies will enable the development of a robust and scalable application that can effectively serve users with varying levels of expertise in data analysis and geospatial exploration.

**Development Environment**

Docker + GitHub Codespaces

**Commit**

**Are-u-Queryous? web App**



**Frontend**

React.js + Leaflet.js

**API**

**PostgreSQL/PostGIS**

**Deploy**

**Production Environment**

Supabase + Vercel

Figure 1 6: Technology Stack Used

1.7 Brief description of the remaining chapters of the report

This section provides a short overview of the chapters that follow in this report:

**Chapter 2: State of the Art**

This chapter explains what technologies and tools are available already to work with maps and location information. It shows how people are trying to make this kind of information more accessible for any user, not just for professionals. It describes websites and programs that help users to explore and understand maps. We also explain how new technologies like AI are making it easier to ask questions and get responses in the form of maps. Finally, we discuss what problems there still exists and what people are doing to solve them.

**Chapter 3: System Design**

This chapter introduces the functional architecture of the *Are-u-Queryous?* platform, detailing how the system responds to the user's interactions and vice versa. It discusses the primary user stories, brings in the use case diagram, defines the user interface views, and describes the navigation, database, and technical architecture of the web application. This chapter is fundamental to recognizing how the project converts technical concepts into a product that can be used.

**Chapter 4: Results**

This chapter presents what we accomplished in our project. It shows how the final web application works and how users can explore maps and filter data without technical knowledge. It includes screenshots of the interface and examples of how it makes open data more accessible. This section demonstrates the practical outcomes achieved through our methodology.

**Chapter 5: Conclusions and Future Work**

The final chapter looks at what we learned from this project and how well we met our goals. It discusses the challenges we faced and what we could improve in the future. It also suggests new features we could add later, like the natural language processing option for easier searching. This chapter evaluates our progress against our original timeline and objectives, and addresses sustainability, diversity, and ethical considerations.

02  
 STATE OF THE ART

**Democratizing Geospatial Data Access: State of the Art**

Making geospatial data available to everyone, not just experts, is called "*democratizing geospatial data*." In recent years, new tools, platforms, and AI have made it easier for regular people to use maps and location data (Bhoda, 2023a).

This section looks at the current progress in this field as is intended to cover the state of the art across key areas, including current platforms, visualization technologies, AI integration, challenges, applications, related technologies, and recent research findings.

2.1. Current Geospatial Data Platforms

Modern geospatial platforms help make data available to everyone. Today, we have both free and paid platforms that help people use map data.

A well-known open-source example is **OpenStreetMap (OSM)**, which allows people to create and edit maps (Glasze & Perkins, 2015). This has made mapping more open to the public. Other free tools like **QGIS**, **GRASS GIS**, and **GeoServer** allow users to analyze and visualize spatial data without expensive software (Bhoda, 2023b). These tools support many data formats, making it easier to share information.

On the commercial paid platforms side, **Esri’s ArcGIS** is one of the most powerful platforms (Esri, 2023). It was originally a desktop software, but now it also offers cloud-based tools like **ArcGIS Online**. These allow organizations to create and share maps with the public. Another important paid platform is **Google Earth Engine (GEE)**. This popular tool provides satellite images and advanced analysis and also allows researchers to study changes in the environment, such as deforestation, using free satellite data (Gorelick et al., 2017).

There are newer cloud-based web platforms like **CARTO** and **Mapbox** that make it simpler to connect map data with other information and also help developers and businesses to integrate maps into their applications (CARTO, 2024). These platforms provide tools that make geospatial analysis more accessible to non-experts. Finally, **Government and open data APIs** also allow users to access updated maps, weather data, and other statistics (Government of India, 2021).

Thanks to all these platforms, *geospatial data is now available to more people than ever before*.

2.2. Geospatial Data Visualization Technologies

User-friendly tools are key to making map data accessible. Web mapping tools like **Leaflet.js** and **Mapbox GL JS** allow people to create interactive maps in a web browser without advanced skills on *Geographic Information System (GIS)* software (Marten, 2019).

Tools like **Kepler.gl** let users drag and drop data files to create maps instantly. Business tools like **Tableau**, **Microsoft Power BI** and **Amazon QuickSight** now include map features, so business and data analysts can make maps without learning GIS software (Foursquare, 2023). **CesiumJS** is another useful tool that specializes in 3D geospatial visualization, allowing users to create 3D maps of cities and landscapes (Chamberlain et al., 2024).

These visualization tools turn complex data into maps that anyone can understand, hiding the technical details that used to require expert knowledge. Users no longer need advanced training in GIS to work with maps and analyze spatial data.

2.3. AI Integration in Geospatial Analysis

Artificial intelligence (AI) is transforming how we use geospatial data. **GeoAI** refers to the combination of artificial intelligence and geographic data to automate complex tasks (Esri, 2023). AI can quickly analyze satellite images and detect objects like buildings, roads, and forests, reducing the need for human analysis.

One of the most advanced developments is the use of Natural Language Processing (NLP) to allow users to ask questions about maps using everyday language. For example, instead of learning GIS commands, using tools like **ChatGeoAI** or **Chat2Geo** allow users to type questions like "show me areas with high tree loss in 2023" and get maps as answers (Mansourian & Oucheikh, 2024).

Companies like Esri are also developing AI assistants for GIS software. These tools allow users to get quick answers about geospatial data, using Machine learning models that can also predict future trends, such as traffic patterns, climate changes, or urban growth (Lartey, 2024).

As it is happening in other fields, the biggest advantage of AI integration in geospatial analysis is that it reduces the skill barrier. In the past, only experts could perform complex spatial analyses, but now, AI tools make it easier for anyone to work with maps and location data.

2.4. Challenges in Democratizing Geospatial Data

Despite progress, several problems still make map data hard to use for everyone:

* **Cost**: Some advanced data and tools remain expensive (Plunkett, 2019).
* **Complexity**: Working with map data still requires some technical knowledge.
* **Internet access**: Cloud-based tools need good internet connections .
* **Data formats**: Different map data comes in many formats that don't always work together (Hallas et al., 2022).
* **Large file sizes**: Map data files can be very big and hard to share (Gorelick et al., 2017).
* **Awareness**: Many people don't know how useful map data could be for them.

These challenges are being addressed through better education, open data policies, and more user-friendly tools.

2.5. Applications of Geospatial Data

Geospatial data is used in many industries, including:

* **Urban Planning**: Cities use geospatial data to plan infrastructure, transportation, and housing. Maps help identify areas that need better roads, public transport, or green spaces.
* **Transportation & Logistics**: Delivery companies use geospatial data to find the fastest routes. Ride-sharing apps rely on maps to connect drivers and passengers.
* **Environmental Monitoring**: Scientists track deforestation, pollution, and climate change with satellite images and GIS analysis.
* **Disaster Management**: Maps help emergency services respond to floods, earthquakes, and wildfires. Open-source mapping tools like **Ushahidi** allow local communities to share real-time disaster information.
* **Retail & Business**: Companies use geospatial data to find the best store locations, track customer movement, and optimize marketing strategies.
* **Agriculture**: Farmers use satellite images and AI-powered tools to monitor crop health and optimize irrigation.

By making geospatial data available to more people, we can solve real-world problems more effectively.

2.6. Similar Software & Related Technologies

Many technologies help make geospatial data more accessible:

* **Cloud Platforms**: Services like Google Earth Engine and Microsoft Planetary Computer allow users to access and analyze large-scale geospatial data online.
* **Geospatial APIs**: Tools like Google Maps API and Mapbox API make it easy to add maps and location services to applications.
* **Open Data Portals**: Governments and organizations provide free access to maps and location-based data through online portals.
* **Drones & IoT Sensors**: Drones help collect high-resolution aerial imagery, while IoT sensors track environmental changes in real time.

These technologies work helps to bring geospatial analysis to more people, not just specialists.

2.7. Recent Academic Research & Case Studies

In the last five years (2019-2024), researchers have made important progress:

**Natural Language Tools (2024)**

Researchers created ChatGeoAI, a system that lets people use simple language to make maps and analyze data (Mansourian & Oucheikh, 2024). Users just type what they want, and the computer does the technical work.

**Easy Access to Big Data (2020-2022)**

**CyberGIS** projects made it simpler for non-experts to use powerful map tools:

* CyberGIS-Vis created easy-to-use web maps for COVID-19 data (Han et al., 2024).
* CyberGIS-Compute helped regular users run complex map calculations on powerful computers (Michels et al., 2024).

**Community Mapping (2019-2023)**

Studies on OpenStreetMap showed how regular people can create useful maps (Glasze & Perkins, 2015).:

* Communities developing countries built their own maps instead of relying on external ones
* When people can create maps themselves, it helps improve local government services

**Policy Changes in India (2021)**

India changed its laws to make map data free for everyone (Government of India, 2021):

* Removed strict rules about who could make and share maps
* Helped local businesses create new map services
* Increased the use of map data in business and farming

**Easier Data Sharing (2019-2022)**

Researchers worked on making map data easier to find and use:

* A Canadian project created a "Spotify for map data" where users only download what they need
* This helped Indigenous communities and park managers respond to emergencies

**AI for Community Help (2022-2024)**

Projects used AI to solve social problems:

* The RAMP project used AI and satellite images to help plan healthcare services
* Other studies used AI to spot patterns in citizen-reported issues to improve city planning

Democratizing geospatial data has made great progress, with open-source tools, cloud platforms, and AI making maps and spatial analysis accessible to more people (Bhoda, 2023b). Advances like **GeoAI**, web-based visualization, and natural language processing are lowering technical barriers, allowing non-experts to work with geospatial data more easily (Esri, 2023).

However, challenges remain. Many tools are still expensive, and some require technical skills (Plunkett, 2019). Data formats are not always compatible, making it difficult to share information across platforms (Gorelick et al., 2017). AI models improve geospatial analysis, but they need better transparency to ensure accuracy and trust (Lartey, 2024). In addition, limited internet access in some areas makes it hard to use cloud-based tools (Tula Foundation, 2024).

Future research should focus on standardizing data formats, improving user-friendly AI tools, and reducing dependency on high-cost platforms (Alamri, 2024). By addressing these gaps, geospatial data can become even more accessible, helping businesses, governments, and individuals solve real-world problems more effectively.

2.8. Future Trends in Democratizing Geospatial Data

Looking ahead, these are the trends that will make map data even more accessible to everyone:

* **AI Assistants**: New AI tools will let people talk to maps in normal language. Users will be able to ask questions and get answers without learning special commands.
* **Mobile Apps**: More map tools will work well on phones, helping people in areas with limited computer access.
* **Local Knowledge**: Communities will create their own maps that include cultural information and local knowledge not found in official maps.
* **Mixed Reality**: New technology will combine real-world views with digital map data, making it easier to understand location information.
* **Low-Code Tools**: New software will let people without programming skills create custom map applications.
* **Data Standards**: Better ways to share data between different systems will make it easier to combine information from many sources.

These trends show that geospatial data will become even more useful for everyday decisions. As technology improves and becomes simpler to use, more people will be able to benefit from location-based information in their daily lives, businesses, and communities.

2.9. State of the Art Conclusions

The review of current geospatial data platforms, visualization tools, and AI integration shows that significant progress has been made in democratizing access to spatial information. Tools like Leaflet.js, OpenStreetMap, and ChatGeoAI provide users to explore maps and datasets without technical expertise. AI is especially promising in reducing barriers through natural language interfaces and predictive modelling. These innovations are complemented by an expanding ecosystem of open data portals and visualization frameworks.

However, challenges remain. Data fragmentation, inconsistent formats, limited awareness, and the need for high internet connectivity are still a barrier for a democratized and general access. Many users also lack the skills or tools to interpret data effectively. Therefore, there is a growing need for platforms that combine simplicity, visual intuition, and smart querying to close this gap. This project contributes to that mission by proposing a low-barrier, map-centric application designed to empower everyday users with meaningful geospatial insights.

References:

* Alamri, S. (2024). *The geospatial crowd: Emerging trends and challenges in crowdsourced spatial analytics*. MDPI Open Access Journals. <https://www.mdpi.com/2072-4292/13/23/4745>
* BCG Digital Ventures. (2020). *How UP42 is helping institutions combat COVID-19 using geospatial data*. Medium. <https://medium.com/bcg-digital-ventures/how-up42-is-helping-institutions-combat-covid-19-using-geospatial-data-eec1b7f81151>
* Bhoda, S. K. (2023a). *Making geospatial technology accessible: Why it shouldn't be a privilege*. LinkedIn. <https://www.linkedin.com/pulse/making-geospatial-technology-accessible-why-shouldnt-privilege-bhoda-cxrgc/>
* Bhoda, S. K. (2023b). *Open geospatial data: Democratizing access to dynamic intelligence. LinkedIn*. <https://www.linkedin.com/pulse/open-geospatial-data-democratizing-access-dynamic-santosh-kumar-bhoda-0ktbc/>
* CARTO. (2024). *Democratizing spatial analysis with raster data on the cloud*. CARTO Blog. <https://carto.com/blog/democratizing-spatial-analysis-raster-data-on-the-cloud>
* Chamberlain, H. R., Darin, E., Adewole, W. A., Jochem, W. C., Lazar, A. N., & Tatem, A. J. (2024). *Building footprint data for countries in Africa: To what extent are existing data products comparable?* Computers, Environment and Urban Systems. <https://www.sciencedirect.com/science/article/pii/S0198971524000334>
* DroneDeploy. (2018). *Democratizing geospatial data with drones*. Geospatial World. <https://geospatialworld.net/article/democratizing-geospatial-data/>
* Esri. (2023). *The emergence of GeoAI in planning*. Esri Industry Blog. <https://www.esri.com/en-us/industries/blog/articles/the-emergence-of-geoai-in-planning/>
* Foursquare. (2023). *Foursquare brings enterprise-grade spatial analytics to your browser with Kepler.gl 3.1*. Location Foursquare Blog. <https://location.foursquare.com/resources/blog/products/foursquare-brings-enterprise-grade-spatial-analytics-to-your-browser-with-kepler-gl-3-1/>
* GeoRetina. (2023). *Chat2Geo Project*. GitHub. <https://github.com/GeoRetina/chat2geo>
* Glasze, G., & Perkins, C. (2015). *Social and political dimensions of the OpenStreetMap project*. In J. Arsanjani (Ed.), OpenStreetMap in GIScience. Springer. <https://www.geographie.nat.fau.de/files/2018/02/Glasze-Perkins-Social-and-Political-Dimensions.pdf>
* Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). *Google Earth Engine: Planetary-scale geospatial analysis for everyone*. Remote Sensing of Environment, 202, 18-27. <https://doi.org/10.1016/j.rse.2017.06.031>
* Government of India. (2021). *Guidelines for acquiring and producing geospatial data*. T-KARTOR. <https://www.t-kartor.com/blogs/democratizing-geospatial-data>
* Hallas, M., Price, R., & Haithcoat, J. (2022). *Replicable AI for Microplanning (ramp): Democratizing Geospatial Data Science for Global Health*. AGU Fall Meeting 2022, Chicago, IL. <https://ui.adsabs.harvard.edu/abs/2022AGUFMGC42E0755H/abstract>
* Han, S. Y., Kim, J. S., Jiang, Y., Kang, J. Y., Park, J., Han, C., Michels, A., & Wang, S. (2024). *CyberGIS-Vis for Democratizing Access to Scalable Spatiotemporal Geovisual Analytics: A Case Study of COVID-19*. Illinois Experts Research & Scholarship. <https://experts.illinois.edu/en/publications/cybergis-vis-for-democratizing-access-to-scalable-spatiotemporal->
* Hird, J. (2021). *Satellite time series and Google Earth Engine democratize the process of forest-recovery monitoring over large areas*. MDPI Open Access Journals. <https://www.mdpi.com/2072-4292/13/23/4745>
* Lartey, D. (2024). *GeoGPT+: Democratizing geospatial data analysis with AI. Data Storytelling Corner*. <https://medium.com/data-storytelling-corner/geogpt-revolutionizing-geospatial-data-analysis-with-ai-78e3f7e4c1e1>
* Mansourian, A., & Oucheikh, R. (2024). *Physical Geography aChatGeoAI: Enabling Geospatial Analysis for Public through Natural Language, with Large Language Models*. ISPRS International Journal of Geo-Information, 13(10), 348. <https://www.mdpi.com/2220-9964/13/10/348>
* Marten, K. (2019). *Geospatial analysis made easy with two new spatial functions: MakePoint and MakeLine*. Tableau Resources. <https://www.tableau.com/blog/geospatial-analysis-made-easy-two-new-spatial-functions-makepoint-and-makeline>
* Michels, A. C., Padmanabhan, A., Xiao, Z., Kotak, M., Baig, F., & Wang, S. (2024). *CyberGIS-Compute: Middleware for democratizing scalable geocomputation*. SoftwareX, 26, 101581. <https://doi.org/10.1016/j.softx.2024.101581>
* Plunkett, G. (2019). *The role of SDIs in the democratization of geospatial data and services*. Esri Canada. <https://resources.esri.ca/news-and-updates/the-role-of-sdis-in-the-democratization-of-geospatial-data-and-services>
* Tula Foundation. (2024). *Connecting the dots: Democratizing geospatial data*. Tula.org News. <https://tula.org/news/connecting-the-dots-geoconnections>

03   
SYSTEM DESIGN

This chapter explains how the app works. It shows *who uses it, what they do, and how the app answers their actions*. We also include Hand-drawn sketches of the different GUI elements and the navigation between them aiming at helping to understand how the app is built.

3.1 User Stories

**1. General User (Main Actor)**

|  |  |
| --- | --- |
| **Who?** | Any visitor of the web app such as general public, students, researchers, small business owners and other non-technical users. |
| **What they do?** | Explore the map.  Apply filters by indicators (age, wage, etc.)  Compare different areas.  Understand data in a simple way.  Possibly ask questions (if NLP is enabled). |

User Story 1: How can the user see and move around the map?

|  |  |
| --- | --- |
| **US01 - View Map and Explore Data** | |
| **As a** | General User |
| **I want** | to open the app and see the map. |
| **So that** | I can select an available city (Madrid or Barcelona) and explore it visually. |
| **Acceptance Criteria** | Zoom in and out, move around the map, see basic data like population economic information. |

User Story 2: How can the user choose what kind of data they want to see?

|  |  |
| --- | --- |
| **US02 - Apply Indicators Filters** | |
| **As a** | General User |
| **I want** | to apply filters like age, wage or education. |
| **So that** | I can see in the map specific information that I care about. |
| **Acceptance Criteria** | Filters easy to select. Map updates immediately to show filtered data. |

User Story 3: How can the user look at differences between two places?

|  |  |
| --- | --- |
| **US03 - Compare Two Areas** | |
| **As a** | General User |
| **I want** | to compare two zones on the map. |
| **So that** | I can see differences in things like population or income and make data driven decisions. |
| **Acceptance Criteria** | Areas or polygons well defined with clear numbers, colours and charts |

User Story 4: How can the user understand the data?

|  |  |
| --- | --- |
| **US04 - Understand Data Visually** | |
| **As a** | General User |
| **I want** | to see the data in a way that is easy to understand. |
| **So that** | I don’t need any special training or technical knowledge to use the map and understand the information. |
| **Acceptance Criteria** | The app uses simple icons and clearly displays the information. |

User Story 5: How can the user talk to the app in normal words?

|  |  |
| --- | --- |
| **US05 - Ask Natural Language Questions** | |
| **As a** | General User |
| **I want** | to ask questions in plain text like “where are more young people?” |
| **So that** | I can find and visualize answers without using menus or filters. |
| **Acceptance Criteria** | The app accepts a typed question and returns correct answers for visualize in the map. |

**2. Admin User (Secondary Actor)**

|  |  |
| --- | --- |
| **Who?** | The person in charge to maintaining and updating the application |
| **What they do?** | Upload or update open datasets.  Manage system configuration (e.g., which cities or indicators are available).  Monitor platform performance.  Enable or disable features. |

User Story 6: How can the admin add or change the open data?

|  |  |
| --- | --- |
| **US06 - Upload or Update Open Datasets** | |
| **As a** | Admin User |
| **I want** | to upload or update datasets in the app. |
| **So that** | the application always shows up-to-date and correct information. |
| **Acceptance Criteria** | Upload a CSV or GeoJSON file and check it is correctly added to the system. |

User Story 7: What kinds of data do we allow users to filter?

|  |  |
| --- | --- |
| **US07 - Manage Configuration** | |
| **As a** | Admin User |
| **I want** | to be able to choose which indicators are available for the general users. |
| **So that** | I can control what types of data (wage, age, population, education etc.) users can explore. |
| **Acceptance Criteria** | There’s and enable/disable indicators option in the admin or settings page. The changes should be visible for the general users. |

User Story 8: What tools do we allow the user to use?

|  |  |
| --- | --- |
| **US08 - Enable or Disable Features** | |
| **As a** | Admin User |
| **I want** | to disable or add new features. |
| **So that** | I can maintain the application balanced and easy to use and controls which app functions area available. |
| **Acceptance Criteria** | Every feature is listed in the settings page with a switch/option to turn it on/off. |

User Story 9: How can the admin check if the app is working well?

|  |  |
| --- | --- |
| **US09 - Monitor Platform Performance** | |
| **As a** | Admin User |
| **I want** | to check the system performance and the usage logs. |
| **So that** | I can detect if there is any problem and monitor peak usage hours. |
| **Acceptance Criteria** | Clearly see the logs errors and user activity data in the settings page. |

**3. System (Support Actor)**

|  |  |
| --- | --- |
| **Who?** | The application itself. |
| **What they do?** | Process and filter the requests.  Manage database operations.  Delivers data to the frontend and populate the map.  Handles search, filter and comparison. |

In this section, we showed the main people and parts that use or support the app. These are called actors, and each one has a special role:

* **General User**

This is any person who visits the app. They can be students, researchers, or people curious about the city. They use the map, add filters (like age or income), compare areas, and look at the information. If the AI tool is active, they can also ask questions in normal words.

* **Admin User**

This person controls the app. They upload new data, choose what filters and features are active, and check if everything works well. The admin helps keep the app updated and useful for others.

* **System**

The system is the app itself. It works in the background. It takes care of the data, processes the filters, and shows the correct information on the map. It also helps show answers when someone uses the search or comparison tools.

These three actors work together. The General User interacts with the map and data. The Admin User manages the content and features. The System makes sure everything works and connects correctly. Together, they create a smart and easy-to-use platform for geospatial data.

3.2 Use Case Diagram

**Are-u-Queryous? web App**

**Enable/Disable Features**

**Upload/Update Datasets**

**Manage Indicators**

**Monitor Performance**

**View Map and Explore Data**

**Visualize Data**

**Natural Language Queries**

**Compare Areas**

**Apply Filters**

**<<Include>>**

**<<Extend>>**

**<<Extend>>**

**General User**

**Admin User**

Figure 3 1: Use Case Diagram

This diagram shows how people use the *Are-u-Queryous?* app. As we already define, there are two types of users: **General User** and **Admin User***.*

* The **General User** can use the map, add filters (like age or income), compare areas, and see simple charts. They can also ask questions in simple words if allowed.
* The **Admin User** can upload new data, select what filters to show, switch features on or off, and check if the app is working well.

The lines show what each user can do. Some tasks need other tasks:

* **Compare Areas «include» Apply Filters**: comparing two areas always uses filters. We cannot compare two areas without first applying filters.
* **Apply Filters «extend» View Map**: applying filters adds more features to viewing the map but is not always required. We can view the map without applying filters, but applying filters makes it more powerful.

3.3 User interface design

In this part, we show how the app will look using wireframes. These are simple screen designs made with basic shapes. They are not the final design, but they help us plan where to put each element, like the map, filters, buttons, and charts.

We used low-fidelity wireframes to focus on the layout and the user experience. This method is fast and easy to change. It allows us to improve the design step by step and make sure the app will be easy for everyone to use. These wireframes also help explain the idea to others before we start building the real app.

*Home View*

This is the first screen of the app. The user can choose a city and see the map. The map has filters like age or income. The user can also ask questions in natural language. It is simple and easy to use.

Figure 3 2 - Home Map View

**View Map and Explore Data (US01)**

Select City

Logo

**Apply Filters (US02)**

**Ask Natural Language Questions (US05)**

*Ask a question (e.g. “Where are more young people?”)*

Age

Income

Education

Population … etc



Admin

*Compare Two Areas View*

This screen helps the user compare two areas on the map. The user selects one area and then chooses another area to compare. Both areas are shown side by side, with a simple shape for each one.

Below the areas, there is a table that shows the main information, like population and average income. This helps the user understand the differences between the two areas.

This view is useful for people who want to make decisions based on data, for example choosing the best place to live or start a business. It is easy to read and works without needing technical knowledge.

**Comparison Results**

**Population:**

**Avg. Income:**

45,600

38,200

23,600 €

28,246 €

**Compare Two Areas (US03)**

Barcelona

Area 1 (Selected)

Area 2 (Compare)

Logo

Figure 3 3 - Compare Areas View

*Visualize Data View*

This screen shows more information using charts and graphs. It helps users understand the data better. For example, they can see population or income for each area. It is good for users who want to learn more after using the map.

E

D

C

A

B

**Understand Data Visually (US04)**

Logo

Back to Map

**Example: Population by District**

Figure 3 4 - Visualize Data View

*Admin View*

This is the screen for the admin. The admin can upload new data, change settings, and check if the app is working well. The admin can also turn features on or off. This helps keep the app updated and easy to use.

Figure 3 5 - Admin Upload Datasets View

Figure 3 6 - Admin Features & Indicators View

City:

Barcelona

Demographics

Upload

Dataset Type:

**Upload or Update Open Datasets (US06)**

Logo

Drag & Drop Files Here

or

Browse Files

**Datasets**

Features & Indicators

Analytics

Logout

Figure 3 7 - Monitor Platform Performance View

Figure 3 6 - Admin Manage Configuration & Management View

Age Distribution

Income Levels

Educational Skills ... etc.

Natural Language Query ...etc.

**Manage Configuration & Feature Management (US07 / US08)**

Logo

Logout

Datasets

**Features & Indicators**

Analytics

**Available Indicators**

**Available Features**

2025-03-24 10:15:22 INFO: User login successful (admin)

2025-03-24 10:15:22 ERROR: Dataset upload failed - invalid format

2025-03-24 10:15:22 INFO: New dataset uploaded (Barelona-Demographics-2025)

**System Logs**

**Monitor Platform Performance (US09)**

Logo

Datasets

Features & Indicators

**Analytics**

**Daily Active Users**

**Query Distribution**

Logout

3.4 Navigation Flow

City Data Exploration

City Data Visualization

Admin Login

**Apply Filters**

**Admin Panel**

(Admin View)

**Home**

(Map View)

**Ask Natural Language Queries**

(Search Box + NLP Engine)

**Visualize Data**

(Charts/Graphs View)

**Compare Areas**

(Compare View)

Figure 3 8 - Navigation Flow between the different Views

This diagram shows how the user moves between different parts of the app. The user starts on the **Home** screen. From there, they can:

* **Explore the map** and see data for a city.
* **Apply filters** to choose what kind of data they want to see (like age or income).
* Go to the **Compare Areas** screen to look at two places side by side.
* Go to the **Visualize Data** screen to see charts and graphs.
* If the AI is active, they can also **ask questions** in normal language and see answers.

If the user is an **Admin**, they can log in and go to the **Admin Panel**. There, they can upload data, change settings, and check if the app is working well.

3.5 Architecture Diagram

To design the right architecture for this project, we followed these simple steps:

*Step 1: Understand the Project Context*

We looked at the business domain. The project is about maps and city data. Users can easily see information about cities like Barcelona and Madrid. Our technical tools will be React.js, FastAPI, Python, and Supabase database. We will use cloud services (Heroku and Vercel). We chose simple technology because we have a time constraint, and the project should be delivered on the expected time. We made a list of the important features where we can highlight easy-to-use app, load fast and cost little money.

*Step 2: Prioritize Requirements*

We made a list of features and put them into three groups:

* **Must-Haves**: Map view, data filtering, and city information.
* **Should-Haves**: Simple user interface and fast responses.
* **Nice-to-Haves**: Natural language questions (NLP), analytics.

*Step 3: Evaluate Trade-offs*

We asked some questions to choose the best options:

* **Cost vs. Quality:** We picked free or cheap tools to keep costs low.
* **Speed vs. Maintainability:** We want clean code but also quick results since the final project timeline Is tight. We chose simple code that’s easy to maintain.
* **Scalability vs. Simplicity:** Right now, simplicity is important. We can make the app bigger later.

*Step 4: Apply Simplification Principles*

We will follow three simple ideas:

* **YAGNI:** We will build only what we really need.
* **KISS:** Our solutions will be simple and easy to understand.
* **MVP:** We will create a small app first. Later we will add more features.

*Step 5: Select an Architectural Pattern*

After defined all the requirements, we finally chose a **layered architecture**. It’s simple to understand, easy to build, and a good fit for this project timeline. This pattern also makes adding new features easy in the future:

**Presentation Layer**

**Frontend**

React.js + Leaflet.js

*(Deployed on Vercel)*

**Static Assets**

(CDN)

**NLP Module**

Pre-trained Face Model

*(Optional Feature)*

**Application Layer**

**API Gateway**

Supabase REST & GraphQL APIs (no custom gateway required)

**Data Transformation Module**

Python + Pandas

**Geospatial Analysis Service**

GeoJSON, Spatial Join, PostGIS Ops

formation Module

Python + Pandas

**Data Layer**

SQL Queries & Responses

**External Data Sources**

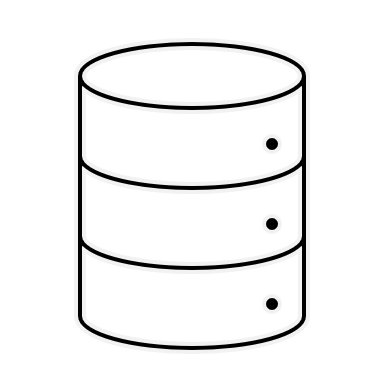
Barcelona, Madrid Open Data

**Database**

PostgreSQL/PostGIS

*(Deployed on Supabase)*

HTTP Request & Responses



E

L

T

E = Extract | T = Transform | L = Load

Figure 3 9 - Architectural Diagram

3.6 Database design

The database for the *Are-u-Queryous?* platform is designed to manage two primary types of geospatial data:

* **Aggregated data** (Indicators such as population or income)
* **Geolocated data** (Point Feature such as schools or bus stops)

To ensure **flexibility and simplicity**, the database uses a traditional relational model with normalized tables and a virtual abstraction layer that allows indicators to be linked to multiple geographic levels.

**Geographical Level**

**Indicator Definition**

**Indicator**

**defines**

**Point Feature**

**aggregated at**

**City**

**District**

**has**

**has**

**Neighbourhood**

neighborhood\_id

1

1

1

N1

N1

N1

1

1

1

**defines**

Is a

**Geographical Unit**

Figure 3 10- Entity Relation Diagram

In the **Entity-Relationship Diagram**, we introduce a virtual entity called **Geographical Unit**. This acts as a common structure that includes City, District, and Neighbourhood, using an “is-a” relationship.

The **Indicator** table stores values for different geographic levels and links to the Geographical Unit by using two fields: *geo\_level\_id* (which defines the type, like city = 1) and *geo\_id* (the ID of the specific city, district, or neighbourhood). This makes the database cleaner, faster, and easier to manage.

The **Point Feature** table uses latitude and longitude to show exact map locations. If needed for visualizations calculations, a point can also be linked to a *Geographical Unit*, so we can group or filter features by area, like showing all schools in one district.

|  |  |  |
| --- | --- | --- |
| **Entity** | **Description** | **Attributes** |
| **City** | Cities covered by the platform | city\_id (PK, Integer), name (Text) |
| **District** | Areas inside cities | district\_id (PK, Integer), name (Text), city\_id (FK → City), geom (Geometry) |
| **Neighbourhood** | Areas inside districts | neighborhood\_id (PK, Integer), name (Text), district\_id (FK → District), geom (Geometry) |
| **GeographicalLevel** | Combines all location types | geo\_level\_id (Integer), name (Text), |
| **IndicatorDefinition** | Indicator types | indicator\_def\_id (PK, Integer), indicator\_name (Text), unit (Text), description (Text) |
| **Indicator** | Stores value for one place and year | indicator\_id (PK, Integer), indicator\_def\_id (FK → IndicatorDefinition), geo\_level\_id (Integer), geo\_id (Integer), year (Integer), value (Decimal) |
| **PointFeature** | Individual location on map | point\_id (PK, Integer), feature\_type (Text), name (Text), latitude (Decimal), longitude (Decimal), geo\_level\_id (Integer), geo\_id (Integer), properties (JSONB) |

Table 3 1 Entities, Field, Types and Foreign Keys

3.7 Class Diagram

This diagram shows how the main entities of the platform are related, including indicators, geographic levels, and point features.

**City**

*+city\_id: Integer*

*+name: String*

**City**

*+district\_id: Integer*

*+name: String*

*+city\_id: Integer*

*+geom: Geometry*

**District**

*+neighborhood\_id: Integer*

*+name: String*

*+district\_id: Integer*

*+geom: Geometry*

**Neighbourhood**

*+indicator\_def\_id: Integer*

*+indicator\_name: String*

*+unit: String*

*+description: String*

**IndicatorDefinition**

*+indicator\_id: Integer*

*+indicator\_def\_id: Integer*

*+geo\_level\_id: Integer*

*+geo\_id: Integer*

*+year: Integer*

*+value: Decimal*

**Indicator**

*+point\_id: Integer*

*+feature\_type: String*

*+name: String*

*+latitude: Decimal*

*+longitude: Decimal*

*+geo\_level\_id: Integer*

*+geo\_id: Integer*

*+properties: JSONB*

**PointFeature**

**City**

*+geographical\_level\_id: Integer*

*+name: String*

**GeographicalLevel**

1

N

defines

has

has

1

1

N

N

**City**

*+geo\_level\_id: Integer*

*+geo\_id: Integer*

**<<view>> GeographicalUnit**

1

N

1

N

derives

derives

derives

uses

uses

aggregates at

aggregates at

Figure 3 11 - UML Class Diagram

This **UML Class Diagram explains how the system is organized**. It shows how indicator values are connected to different areas like cities, districts, or neighbourhoods. Instead of using names like “city” or “district” directly, we use a table called *GeographicalLevel* with IDs. This keeps the database clean and fast.

The ***GeographicalUni*t is a view** that combines all areas (cities, districts, neighbourhoods) into one, so we can use simple queries. Both *Indicator* and *PointFeature* use this view to link their data to a place on the map.

This design helps keep the database flexible, clear, and ready for new types of data in the future.

*How Is the Data Queried?*

To avoid using text like “city” or “district” in every row, we use a number called *geo\_level\_id*.

For example:

* 1 = city
* 2 = district
* 3 = neighbourhood

This keeps the data faster and cleaner.

However, since we store cities, districts, and neighbourhoods in different tables, we need a smart way to join them.

To do this, we create a called *GeographicalUnitView*. This view joins all areas into one table.

With this view, we can write simple queries like:

SELECT i.indicator\_id, i.year, i.value, g.name

FROM Indicator i

JOIN IndicatorDefinition d ON i.indicator\_def\_id = d.indicator\_def\_id

JOIN GeographicalUnitView g ON i.geo\_level\_id = g.geo\_level\_id AND i.geo\_id = g.geo\_id;

Now we can:

* Get the name of the area (like “Eixample”) automatically
* Filter by area type if we want (only districts, only neighbourhoods, etc.)
* Keep the logic simple for future developments.

*Example Record:* ***Indicator***

We want to store the **average income in Eixample district (Barcelona) in 2023**:

1. *City Table (parent)*

|  |  |
| --- | --- |
| city\_id | name |
| 1 | Barcelona |

1. *District Table (parent)*

|  |  |  |  |
| --- | --- | --- | --- |
| district\_id | name | city\_id | geom |
| 4 | Eixample | 1 | [polygon geom] |

1. *GeographicalLevel (parent)*

|  |  |
| --- | --- |
| geographical\_level\_id | name |
| 2 | district |

1. *IndicatorDefinition (parent)*

|  |  |  |  |
| --- | --- | --- | --- |
| indicator\_def\_id | display\_name | unit | description |
| 1 | Average Income | € | Avg. income per household |

1. Example record for the average income in Eixample district (Barcelona) in 2023:

|  |  |
| --- | --- |
| Field | Value |
| indicator\_id | 501 |
| indicator\_def\_id | 1 |
| geo\_level\_id | 2 |
| geo\_id | 4 |
| year | 2023 |
| value | 26800 |

This record means: “*The average income in district ID 4 (Eixample) in 2023 was €26,800.*”

*Example Record:* ***Point Feature***

We want to add **a public school in Gràcia neighbourhood**:

1. *City Table (parent)*

|  |  |
| --- | --- |
| city\_id | name |
| 1 | Barcelona |

1. *Neighbourhood Table (parent)*

|  |  |  |  |
| --- | --- | --- | --- |
| district\_id | name | district\_id | geom |
| 22 | Gràcia | 4 | [polygon geom] |

1. *GeographicalLevel (parent)*

|  |  |
| --- | --- |
| geographical\_level\_id | name |
| 3 | neighbourhood |

1. A public school in Gràcia neighbourhood.

|  |  |
| --- | --- |
| Field | Value |
| point\_id | 304 |
| name | "Escola Gràcia" |
| feature\_type | "school" |
| latitude | 41.403 |
| longitude | 2.160 |
| geo\_level\_id | 3 |
| geo\_id | 22 |
| properties | {"type": "public", "students": 420} |

This record says: “*There is a public school in Gràcia (neighbourhood ID 22) with 420 students, located at the given coordinates.*”

04   
METHODS AND RESOURCES

The full source code and documentation for the *Are-u-Queryous?* platform is available at: <https://github.com/nicodalessandro11/uoc-tfg-auq>

4.1. Development Strategy and Project Organization

This project was developed step by step, following a clear and modular structure. Each part of the system was placed in its own folder. Later, all parts were moved into a single repository (monorepo) to improve organization, make integration easier, and allow shared testing.

The development followed this sequence:

1. **Database design**: The database was the first component implemented. It was built using PostgreSQL + PostGIS with schema migrations and RLS policies to support geospatial data. The full schema, test data, and reset tools are included in the **auq\_database/** folder.
2. **ETL development**: The second step was building the data engine in the **auq\_data\_engine/** folder. It collects, cleans, and uploads open datasets into the database. The scripts are written in Python and follow a clear structure based on city and dataset type.
3. **Frontend-first approach using BaaS:** Instead of creating a backend API first, the frontend was developed early using Supabase. Supabase works as a **Backend-as-a-Service (BaaS)**, providing a ready-to-use API, authentication, and direct database access. This allowed the frontend to use real data and help improve the schema during development.
4. **Custom backend:** After the frontend was completed and well tested, a custom backend (**auq\_backend/**) placeholder was developed containing the exact information needed to develop an isolated backend that can offer full control over API logic and improve security and performance for production use in case needed.
5. **Deployment and Containerization:** For development, we used GitHub Codespaces, which already includes a container. This made it easier to start working without needing to set up Docker manually. On the other hand,

The frontend was deployed using Vercel, which connects directly to the GitHub repository. When we push changes to the main branch, the frontend updates automatically.

The backend uses Supabase, which gives us a hosted database (PostgreSQL with PostGIS), authentication, storage, and APIs, so we didn’t need to build or deploy a custom backend.

We also created an ETL pipeline using GitHub Actions. This pipeline runs automatically every day at midnight (UTC), when we push changes to important folders, or manually from the GitHub interface.

The pipeline installs all dependencies, runs the ETL code, and then runs tests to check everything works. If it fails, it sends a message. All sensitive data like the Supabase keys are stored safely using GitHub secrets.

1. The **monorepo** folder structure is:

* **auq\_database/:** SQL schema, seed data, reset, migrations.
* **auq\_data\_engine/:** Python ETL scripts organized by city and dataset.
* **auq\_frontend/:** Web app built with React, Next.js, Leaflet, and Tailwind CSS.
* **auq\_backend/:** Placeholder for a potential custom backend API. It includes documentation of the API endpoints and data structures currently managed by Supabase, serving as a reference in case a fully custom API needs to be developed in the future.
* **auq\_nlp/:** Reserved for optional future natural language features (not yet implemented).
* **shared/:** Common utilities and functions used across modules.

The project uses professional practices such as semantic versioning, a standardized Git commit template, and detailed changelogs.

This modular structure and step-by-step approach helped build a flexible system, where each part could be developed, tested, and improved separately.

4.2. Technologies and Tools Used

The project uses modern technologies, organized in four main areas: database, data processing, frontend, and infrastructure.

*4.2.1 Database*

* PostgreSQL (v15+) with PostGIS (v3+).
* Supabase as cloud DB with built-in REST API and auth.
* Row-Level Security (RLS) for role-based access (anon, authenticated, service\_role).
* Security-definer views were removed for Supabase compatibility.
* SQL scripts for schema definition, seeding, and reset logic.

*4.2.2 Data Engine (ETL)*

* Python 3.11 with Pandas, GeoPandas, Shapely.
* Folders: **data/raw/,** **data/processed/**, **scripts/, tests/.**
* Tests: geometry integrity (**test\_geometry\_integrity.py**).

The ETL process follows three main steps:

1. **Extract**: Download raw datasets into **data/raw/.**
2. **Transform**: Clean and validate data, save results in **data/processed/.**
3. **Load**: Upload clean data to Supabase if all tests pass.

*4.2.3 Frontend*

* **Next.js** (React-based framework): Used to build the web application.
* **React**: For creating dynamic user interfaces with components.
* **Leaflet**: JavaScript library for interactive maps.
* **Tailwind CSS**: For fast and responsive styling.
* **Supabase Client SDK**: Used to connect the frontend directly to the Supabase database.
* **React Context**: For managing shared state like selected city, filters, and map layers.

The frontend is organized by pages (map, compare, visualize, admin) and uses context providers for state and data loading. Data is always fetched from the real database, not mock files.

*4.2.4 Infrastructure and Dev Tools*

* **Git**: For version control and project history.
* **GitHub**: Hosting the repository and managing code.
* **GitHub Codespaces:** Used as the main development environment, with a built-in container to avoid manual setup.
* **GitHub Actions**: Used to automate tasks like running the ETL pipeline and testing.
* **Makefile:** For running tasks like database setup and tests.
* **Excel calendar**: Used for planning and tracking progress.
* **Markdown files**: All parts of the project include a **README.md** and there’s also a **CHANGELOG.md** file in the root of the repository with a summary of the git logs .

4.3 Development Methodology

The project followed a **step-by-step** and **modular** development method. Each part of the system (database, data engine, frontend, and backend) was developed separately, but always with the full system in mind. The development was not based on a formal agile framework, but it used short iterations with clear goals.

Each phase of the project had its own focus:

1. **Database First**:  
   The database was the foundation of the project. The first tasks were to design the data structure, define the tables, and insert test data. Geospatial features were added using PostGIS.
2. **ETL Development**:  
   After the database was ready, the data engine was created to collect open datasets, clean them, and upload them to the database. Each script was tested with real data. Only after passing all validation checks, the data was uploaded to Supabase.
3. **Frontend-Driven Integration:**  
   Instead of using mock data, the frontend was connected directly to Supabase. This allowed the application to use real data from the beginning. While building the frontend, some changes were made to the database schema and ETL process, depending on what was needed in the interface. This method made development faster and more realistic.

**Important Note**: This approach **modified the system design presented in Section 3.3**, which had planned to build the backend first. However, the original design is kept in this report for **project traceability**. All changes and the final version of the system will be explained in the *Results* chapter.

1. **Testing and Validation**:

**Data Engine**

* **Pytest** for data validation and integrity.
* **test\_geometry\_integrity.py** ensures geometries match expectations.
* Scripts include automatic logging and summary outputs.

**Frontend**

* Manual testing with real data.
* Debug components for **Supabase** connectivity.
* Logs and browser dev tools for layer validation.

4.4 Project Planning and Workflow Tools

The project was planned to use a **simple but effective workflow** based on personal task management and version control.

1. ***Planning***

The project Excel was used to organize the main phases of the project, track weekly goals, and set deadlines. This calendar helped follow a clear timeline and avoid delays.

1. ***Version Control****:*

Git was used for version control. Each change was saved in commits using a clear and professional shared commit template including the type of change, date, and a short explanation. This made the history easy to read and follow.

* Follows semantic versioning (v1.0.0, etc.).
* Uses standardized commit format (see **commit\_template.md**).
* Changelog updated automatically (**CHANGELOG.md**) using **generate\_changelog.py.**
* GitHub used for version control, backup, and review.

1. **Documentation**

Each part of the project includes its own **README.md** file, with instructions and explanations. A global **CHANGELOG.md** file was used to record all important changes during development. This documentation helps understand what was done and why.

1. **Makefile Automation**:

A **Makefile** was used to simplify common tasks like resetting the database, running tests, or loading seed data. This saved time and reduced the chance of errors. The **Makefile** provides standardized CLI commands, among others we can find:

* **make reset-and-migrate-db**: Drops and rebuilds DB schema.
* **make run-engine**: Runs data ingestion pipelines.
* **make test**: Runs Pytest validations.
* **make install-frontend**: Install frontend dependencies
* **make dev-frontend**: Run frontend development server

1. **Python Development Guidelines (ETL)**

* Based on **python\_scripts\_guidelines.md**, all ETL scripts follow these:
* Logging for traceability (own module using emojis developed for this project).
* Modular functions (extract, transform, load).
* Validation before upload.

1. **Data Privacy and Access Control**

The project uses only open public data, with no personal or sensitive fields.

1. **Supabase RLS policies** restrict access to ensure read-only for anon.
2. **service\_role** handles ETL uploads securely.
3. Admin access is protected via frontend login and Supabase rules.
4. Dataset sources and disclaimers are shown in the UI.
5. **Natural Language Queries (Additional feature not yet implemented)**

Although not implemented in the final version, the project includes a placeholder directory named **auq\_nlp/,** created as a foundation for future expansion.

1. Goal: Ask questions like “Where are more young people?” and get answers from the data.
2. It tests **LangChain** with **OpenAI GPT-3.5**, and models like **bert-bas**e and **distilGPT**.
3. The script loads a language agent, connects to the Supabase database, and returns a smart answer.
4. Not part of the final version, but ready to add later.

This simple setup allowed the project to move forward in a controlled and organized way, without needing complex tools.

4.5 Methods and Resources Conclusion

This chapter is short, but for me, it is **the heart of the whole project**.

I feel very proud of the system I built. Every part of it, the clean code, the logs, the tests, the automation, and the folder structure, was carefully made and well documented.

I learned how to build a project that is easy to understand and ready for the future.

In many companies today, the biggest challenge is to create good and clean data for Artificial Intelligence. This project is my way to show how to do it with quality, care, and best practices.

To really understand how it works, I recommend reading the code, README files, and all the documentation in the [GitHub repository](https://github.com/nicodalessandro11/uoc-tfg-auq/tree/main/auq_data_engine). **The importance of order and clear documentation is the most important thing I take with me after these many years of study.**

04   
RESULTS

In this chapter, we will show **the final results of the project step by step**. Each step will follow the data flow from the beginning to the end. We will explain how each part of the system was built and how we made it better during the development process.

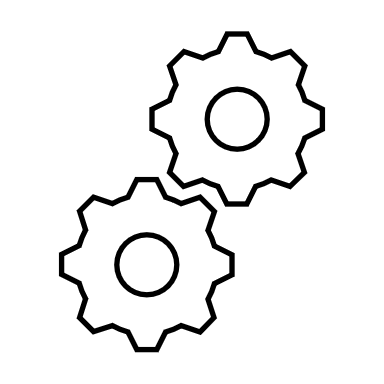
The goal is to show not only what works, but also **how we improved the design**: Some parts were changed from the original idea in Chapter 3 - System Design to make the system easier, faster, or safer. These changes were small, but important. This project shows how a real implementation can be different from the initial plan. That’s why we wanted to keep track of the original design (chapter 3), and its results and changes showed them clearly in this chapter.

**Core Modular Services**

Database and BaaS

Frontend

ETL Engine



Helper, logs, config

Automation tasks

**Shared Infrastructure**

*auq\_data\_engine/*

*auq\_database/*

*auq\_frontend/*

*shared/*

*Makefile, yml, etc …*

We will go through each main stage of the system:

* The **ETL engine** that collects and prepares the data
* The **database** where we store the cleaned information
* The **frontend** web app where users can see and explore the data
* The **automatic update system** that keeps the data system

For each stage, we will explain what was planned, what we changed, and what we finally implemented. This structure aims to help readers understand the full picture of the project, from data collection to the final user experience.

4.1 ETL - AUQ Data Engine

The Data Engine is the part of the system that gets the data, cleans it, and saves it in the database. This process is called **ETL**, which means:

* **Extract** → In our engine it gets data from city APIs
* **Transform** → In our engine it cleans and check the data
* **Load** → In our engine it saves it to Supabase (the cloud database)

This part runs before anything else. Without clean data, the rest of the *Are-u-Queryous?* platform would not work.

*4.1.1 Folder and Structure*

The entire code for this data engine implementation is inside the [**auq\_data\_engine/**](https://github.com/nicodalessandro11/uoc-tfg-auq/tree/main/auq_data_engine) folder of the repository. Each city has its own scripts grouped in a folder, and general functionalities that builds this workflow are in the root folder. **For a full scope of the implementation, a thorough read of the sub-repository README, code and detailed documentation is recommended**.

**auq\_data\_engine/**

**├── barcelona/** # Scripts for Barcelona

**├── madrid/** # Scripts for Madrid

**├── upload/** # Upload functions to Supabase

**├── tests/** # Data validation (pytest)

**├── data/** # Processed datasets

**├── main.py** # Main runner for all ETL steps

**└── pyproject.toml** # Config file

*4.1.2 What the Engine Does (ETL Steps)*

**Extract**

* It gets data from city open data portal.
* The system gets data from different sources.
* **There are two types of data:**
  + Static data (does not change often)

- Example: districts and neighbourhoods

- Comes from a public S3 bucket (Supabase Storage)

- The link is written in the **api-file-manifest.json** file

- No API is needed

* + Dynamic data (changes more often)

- Example: parks, libraries, and other

- Comes from city open data APIs (Each city has a custom API)

- **Barcelona** → uses SQL queries (CKAN API)

- **Madrid** → uses REST endpoints (JSON, CSV)

**Transform**

* Data is cleaned using **pandas** and **GeoPandas**
* Bad values are removed, and formats are fixed.
* Tests run using **pytest** checks among others that all areas have a geometry, district and neighbourhood matches, Data types are correct, coordinates are within a valid range, etc.

**Load**

* Only clean data is uploaded to Supabase (a **Postgres**/**PostGIS** database).
* Upload only happens if all tests pass. A centralized script was developed to manage this process: **/upload\_to\_supabase.py.**

*4.1.3 General Flow*

1. We run the script *main.py*.

2. It calls the correct ETL scripts in order:

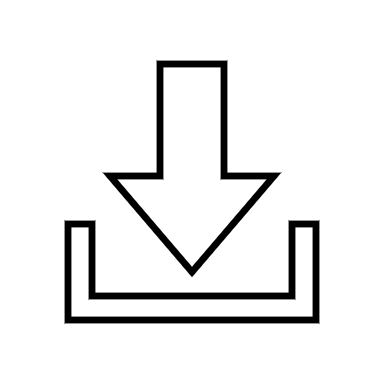
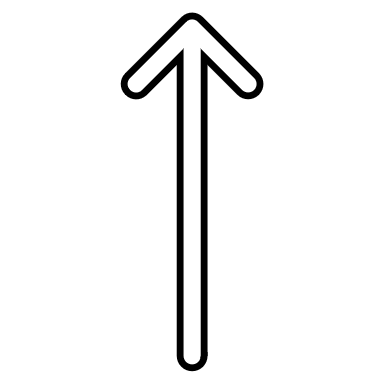
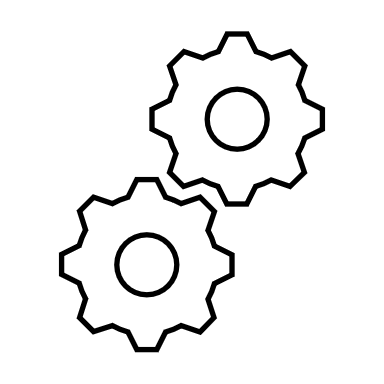
* First districts
* Then neighbourhoods
* Then indicators
* Then point features

1. After cleaning, we run tests to check the data.
2. If everything is OK, we upload the files to Supabase.

**Extract**

**Load**

**Transform**



*Database*

*Open Data*

*load\_districts.py, load\_neighbourhoods.py, load\_indicators.py, load\_point\_features.py*

*api\_client.py data\_manifest.json*

*upload\_to\_supabase.py*

*tests.py*



*data/processed/ insert\_ready\_[…]*

*main.py*

**Important note: The order of operations really matters here**. To build the tables correctly, we first need to upload the parent category data. This is because we need the IDs from the parent categories to maintain the relationships when we add data to the child tables.

*4.1.4 Scripts (what they do?)*

*main.py*

* This script starts the pipeline.
* It calls the other Python scripts to load and clean data for one city.
* It runs each part in order: districts, neighbourhoods, indicators, point features.

*api\_client.py*

* Each city has its own API client to get the data.
* Barcelona uses SQL-style queries and Madrid uses simple links REST.
* Both return CSV or JSON and work with pandas.

*load\_districts.py*

* Gets map of city districts
* Cleans and formats the geometry
* Saves ready JSON file

*load\_neighbourhoods.py*

* Gets map of neighbourhoods
* Links them to the right district and cleans geometry
* Saves ready JSON file

*load\_indicators.py*

* Reads CSV with numbers (population, surface, etc.)
* Links each number to the right neighbourhood
* Saves ready JSON file

*load\_point\_features.py*

* Connects to the city’s Open Data API
* Gets places like parks or libraries
* Adds coordinates and links to neighbourhoods
* Saves ready JSON file

*upload\_to\_supabase.py*

* Reads the cleaned data files.
* Uploads them to Supabase.
* Checks if each upload was OK.

*test\_\*.py*

Runs checks on the processed data

* Makes sure the geometry is correct
* Tests that files are not empty or broken
* Stops and skip uploads if something fails

*4.1.4 Scripts Running and Automation*

The AUQ Data Engine **was configured to be run in two ways**:

**1. Run with Python (manual):**

Run the full ETL process: extract, transform, test, and upload.

*PYTHONPATH=shared python -m auq\_data\_engine.main*

Testing and skip the upload step:

PYTHONPATH=shared python -m auq\_data\_engine.main --skip-upload

**2. Run with the Makefile (recommended)**

Run full engine:

*make run-engine*

Run without upload:

*make run-engine-dev*

*4.1.5 Automated deployment with GitHub Actions*

To make sure the data is always up to date, the system runs automatically every day.

This is done using **GitHub Actions**, a tool to automate tasks:

It runs:

* Every day at 00:00 UTC
* Every time code is pushed to the main branch

It does:

1. Runs the full ETL script
2. Runs all tests
3. Uploads the data only if all tests pass

It uses two secrets (stored in the GitHub repository):

* SUPABASE\_URL: the link to the database
* SUPABASE\_SERVICE\_KEY: the key to upload the data

The workflow is defined in [**.github/workflows/etl-pipeline.yml**](https://github.com/nicodalessandro11/uoc-tfg-auq/blob/main/.github/workflows/et-pipeline.yml)

*4.1.6 AUQ Data Engine Conclusion*

The described process represents the engine because is the base of the whole project. Without clean and safe data, the maps and charts of our application would not work. Even small improvements (like tests, folder structure, or upload rules) made the system better. This chapter shows the real steps needed to go from an idea to a working system.

***Are-u-Queryous? Formatted Data***

***Preprocessed Data***

***Open Data***

The **AUQ Data Engine** was probably the most time challenging part of the whole project. I thought it would be fast and simple because I have experience working with data, but I was wrong. This task needed a lot of time, organization, and thinking. First, it was very hard to find the right datasets in the open data portals. There are thousands of files, and only some of them were useful for this project. Then I had to clean the data, understand what each column meant, and organize it in a clear and useful way. Also, I had to work with different formats, some files were in JSON, others in CSV, and some even had strange coordinate systems that were very hard to understand.

I also had to make big decisions: should we use TopoJSON or GeoJSON? Should we simplify the data or keep all the details? Should we load everything or only some parts? Every small choice affected the whole system. Some data from Madrid, for example, came with compressed coordinates that made everything more difficult. I had to test, try again, and change my plan many times. It was not easy at all.

Still, after many days and nights of work, I finally reached a working version: version 1. This version is strong, clean, and flexible. It is ready to be used and also ready to grow. In the future, we can add new cities, new indicators, or new points of interest very easily. The structure is made for that. Even if the current data is small, the system can grow a lot. This is why I believe this ETL engine is one of the most powerful results of the project. It was hard work, but it taught me a lot, and I am very proud of it.

4.2 AUQ Database and Backend-as-a-Service (BaaS)

*4.2.1 Introduction: Why This Chapter Matters*

In this chapter, we explain how the database works and why it is important for the project. The database is not only used to store data, it is the centre of the whole system. It connects the ETL, the frontend, the users, the API, and even the logs. Everything is built around the database.

The order of this chapter may look strange at first. Normally, we would explain the database before the ETL. But in this project, the two parts are very connected. The database needs to exist before we can upload data. But to understand what the database does, it helps to first see how the data is collected and cleaned. For this reason, the chapter comes after the ETL section.

This chapter will show how the database was created, how it changed during development, and how it supports user accounts, permissions, configuration, and API access. It will also explain how email login works, how logs are saved using edge functions, and how we used a real domain and SMTP service for professional email support.

This is one of the most technical parts of the project. It shows how the backend was designed not only for now, but also for the future. The goal was always to make something clear, flexible, and ready to grow.

As proposed in the design, the tool used for the backend is Supabase, a BaaS (Backend as a Service). Supabase gives us a PostgreSQL database, authentication, REST APIs, edge functions, storage, and more, all in one place. This helped us build a professional and complete backend, even with the limited time that this project require. By using Supabase, we could focus more on the logic and design of the app, instead of building everything from scratch.

The entire code for this database implementation is inside the [**auq\_database/**](https://github.com/nicodalessandro11/uoc-tfg-auq/tree/main/auq_database) folder of the repository. **For a full scope of the implementation, a thorough read of the sub-repository README, code and detailed documentation is recommended.**

*4.2.2 Reverse Design: From Frontend to Schema*

When designing a system, the usual way is to start with the database. First, you define the tables. Then, you build the backend and the frontend on top of that. But in this project, we used a different method.

At the beginning of the project, we need to follow the proposed plan for each CAT delivery. So, our starting point was the design requested in CAT2 and described Chapter 3 – System Design, where we created an early version of the database schema and defined how each part of the platform would work. That original design helped us imagine the full system: the cities, the indicators, the map views, and the user experience. Even this was done to match the structure required by the Final Degree Project TFG, it gave us a good base to start working and also to define the logic of the platform.

But when it was time to build, we needed a strategic approach. The time assigned for development was really tight. That was when the decision of **Frontend First Methodology** seems the most suitable option. We deployed the simple schema of the initial design, then we create the initial ETL Data Engine that fit this structure, and then we move directly to the frontend.

The reason was simple: **we wanted to early see how the user would interact with the app**. So, we built real screens and components using Next.js and Leaflet. This gave us a very clear picture of what data was needed and in what format. For example, the "Compare View" shows charts with data from two different neighbourhoods. To support this, we needed a view that stores indicators and can be filtered by zone and year. In another case, the "Map View" shows points of interest like libraries or sports centres. For this, we needed a table that supports categories, coordinates, and so on.

By starting from the frontend, we were able to define a database schema that was simple but complete. We only created the fields and relationships that were actually needed. This made the system lighter and easier to maintain.

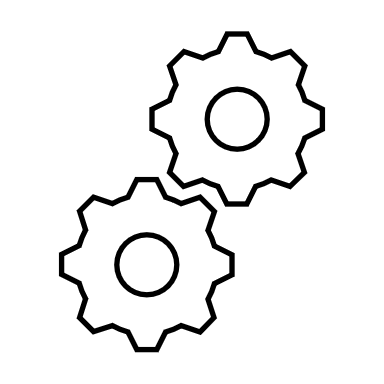
This way of working also helped improve and refine the ETL. After building a page and defining its data needs, we adjusted the pipeline to prepare and upload exactly that information. Each part of the system became more focused and consistent.

This method also made testing easier. Once a new frontend feature was added, we could directly test if the backend and database responded correctly. This **feedback loop** helped speed up development and avoid complex changes later.

**Frontend**

**Database and BaaS**

**ETL Engine**



Define Data Needs

Update Schema

Schema changed – adjust ETL

Fetch Data

To summarize, starting from the frontend and applied this feedback loop methodology gave us three big advantages:

* A clear idea of what the user needs
* A database that is exactly aligned with the UI
* A faster and more focused development process
* No need of mock data, since we have direct connection to Supabase BaaS even at early stages of frontend development.

Even if this is not the most traditional method, it worked very well for a solo project with limited time and changing ideas. It also shows how flexibility and adaptation are key parts of real software development.

*4.2.3 Design Evolution: Authenticated Users and Personalization*

One of the major changes from the initial design was the inclusion of a new actor . In the original design, we planned to build a simple web app for exploring open data. The focus was on cities, maps, and indicators. The system would work for any user, without a login. But during the development process, we realized that this design was not enough. We needed to change the approach.

As the app became more complex, we saw that different users might want to see different things. One user may only care about sports data. Another might only be interested in young population or housing indicators. Also, when more cities, more indicators, and more map features are added, the app could become too heavy. If we show everything to everyone, the system would be slower, and the user experience would be worse.

For this reason, we introduced a new actor: the **authenticated user**. Now, users can log in and adjust the app configuration and save their preferences. Each user can choose which indicators and features to see. They can have their own configuration, saved in the database. This helps reduce the load on the frontend and also gives a more personal experience.

Admins also became important. Admin users can control what is visible by default. For example, they can decide which indicators or layers are active when the app starts. This is useful in case the app is used for education, city planning, or any specific case where some information is more important than others.

We also added a few security decisions. To make a user an admin, someone must go into the database and set the value **is\_admin = true**. This is not something a normal user can do from the app. It is simple, but for a small team and test environment, it works very well.

To support all of this, we created new views in the database, added new routes in the frontend, and created the profile and **user\_events** tables. These store who the user is and how they interact with the app.

We also updated the navigation. Some parts of the app are now only visible to logged-in users. For example, the admin dashboard or the user configuration page. This helps keep the app clean and clear for each type of user. Adding authentication made the app more complex, but also much more powerful. It makes the platform scalable, ready for more data, more users, and more use cases. This change was not planned at the start, but it was a key improvement that came from real experience during development.

To illustrate this design evolution, we may add the following new user stories and a simple data navigation flow to Chapter 3 – System Design. These additions show how the introduction of authenticated users impacts the app’s behavior, allowing a more personalized and scalable experience.

**4. Authenticated User (Main Actor)**

|  |  |
| --- | --- |
| **Who?** | A registered user who logs in with their email and gains access to more advanced features and personalization |
| **What they do?** | All the functionalities of a general user.  Save their preferred filters or configurations  Choose what indicators and layers they want to see by default |

User Story 10: Personalize navigation and save configuration

|  |  |
| --- | --- |
| **US10 - Save Configuration** | |
| **As a** | Authenticated User |
| **I want** | to save my selected indicators and filters |
| **So that** | when I navigate and use the app, I see the data that is important to me |
| **Acceptance Criteria** | After logging in, the system remembers my filters and shows my personalized data view |

Authentication Login (Sign-in)

**Configuration Panel**

(Config View)

**Home**

(Map View)

**Sign Up**

(Sign-Up View)

*4.2.4 Schema Definition and Evolution*

In this section, we explain the logic behind its database structure (schema) and how it evolved over time to support the needs of the frontend and the ETL.

We started designing the schema using the Postgres SQL language in Supabase, with the PostGIS extension enabled. PostGIS is what allows us to work with maps and geometries, like polygons (areas) and points (features). It was enabled from the Supabase dashboard before running the schema.

The full schema is stored in the file [**schema.sql**](https://github.com/nicodalessandro11/uoc-tfg-auq/blob/main/auq_database/schema.sql), which contains all tables, views, functions, and security rules. This file follows a clear structure, with comments to help future developers understand each part. Also, the final database entity relation diagram has been uploaded to the repository and can be found at this [**link**](https://github.com/nicodalessandro11/uoc-tfg-auq/blob/main/auq_database/supabase-schema-diagram.svg).

**Core Tables**

* **cities**: stores the city name (e.g. Barcelona, Madrid)
* **districts and neighbourhoods**: store areas with geometries (polygons)
* **geographical\_levels**: links zones to their level (city, district, neighbourhood)
* **indicators** and **indicator\_definitions**: store stats like age, income, population
* **point\_features** and **feature\_definitions**: store libraries, parks, museums, etc.
* **profiles, user\_config,** and **user\_events**: support authentication, preferences, and logs

All **geometry fields use SRID 4326**, which is the standard for latitude/longitude coordinates.

**Naming Conventions**

* Tables use **snake\_case** (e.g. **point\_features**, **user\_config**)
* Foreign keys always reference id from other tables
* Views follow the format: \*\_view
* Columns like **created\_at** and **updated\_a**t are included in all tables for consistency
* Unique constraints are used to avoid duplicates, especially in spatial data

**Views and Business Logic**

* Several views were created to simplify data access:
  + **geographical\_unit\_view** combines cities, districts, and neighbourhoods
  + **district\_polygons\_view** and **neighborhood\_polygons\_view** return geometry as GeoJSON
  + **current\_indicators\_view** and **time\_series\_indicators\_view** calculate stats over time, using logic to fill missing data by computing averages or sums from other levels
  + These views help the frontend get exactly the data it needs, without extra logic in the app.

**Security and Permissions**

* The schema includes Row-Level Security (RLS) rules. This means that each user only sees or updates what they are allowed to:
* Anonymous users can read public data, Logged-in users can see their own config and event history, Admin-level users (set manually in the profiles table) can manage all data.
* All permissions were set using clear policies and grants, and the system uses Supabase’s **auth.users** table as a base for secure login.

**Functions and Triggers**

* Two functions were created:
* **handle\_new\_user()** adds a profile when a new user signs up
* **execute\_sql()** lets us safely run dynamic queries from the frontend
* A trigger ensures that every time a new user is created, their profile is generated automatically.

To make all changes clear and reproducible, every update to the database schema was tracked using SQL migration files. These files are stored in the [auq\_database/migrations](https://github.com/nicodalessandro11/uoc-tfg-auq/tree/main/auq_database/migrations) folder and are applied in order during development. Each file includes a number and a clear name, such as [014\_create\_user\_events\_table.sql](https://github.com/nicodalessandro11/uoc-tfg-auq/blob/main/auq_database/migrations/014_create_user_events_table.sql). This makes it easy to follow the evolution of the schema over time. It also ensures that anyone working on the project can see exactly what was changed, when, and why. This process is key for collaboration, debugging, and long-term maintenance of the system.

*4.2.5 Supabase Authentication System*

Authentication is the process of identifying who a user is. In this project, we used **Supabase** to manage authentication. Supabase provides an easy and secure way to let users sign in and use the app. It also gives us tools to store extra information about each user.

**Email-Only Login**

In this app, users log in with **email only**. They don’t need a password. Instead, they receive a link in their email. When they click the link, they are logged in automatically. This is called a **magic link**. It is simple for users and still safe.

To make this work, we use the built-in Supabase auth service. When a user signs up or logs in, Supabase sends the email with the link. Once they click it, the app receives a **JWT token** (a secure token that identifies the user), which is stored in the browser.

**auth/callback Route and Profile Trigger**

After the user clicks the link, Supabase redirects to the route **/auth/callback**. This is handled in the frontend. At that moment, the system checks if the user exists in the profiles table. If not, a **trigger function** in the database automatically creates a new profile.

This is done using the function **handle\_new\_user**, defined in the schema. It uses the user's email and metadata to create a record in the profiles table, including fields like **display\_name** and **is\_admin**. This way, every new user has a matching profile in the database, and we can use this data across the app.

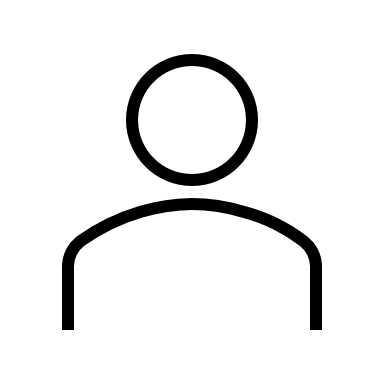
**Admin Users**

As we already explained, there are different types of users in the system:

* **Regular and Authenticated users**, who can log in, configure filters, and use the app.
* **Admin users**, who have special permissions.

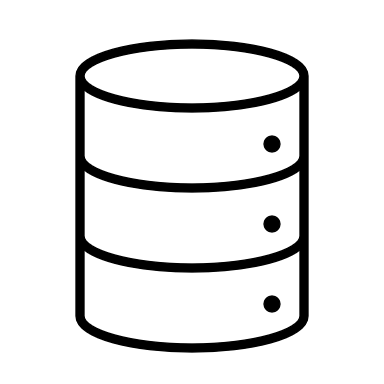
The difference is set with a simple flag in the profiles table: the field **is\_admin**.

By default, this is false. If we want to make a user an admin, we need to manually go to the database and change this field to true. There is no button or API to do this from the frontend. This is a very simple but **effective** way to manage admin access without needing a complex permission system. This choice also improves security, because only someone with direct database access can give admin rights.

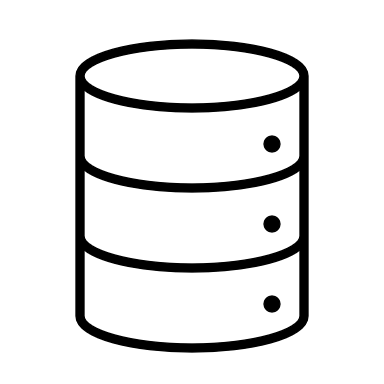


**User Signup**

**Redirect to /auth/callback**



**Supabase sends email with magic link**



**Supabase validates token and generates JWT**

**Database triggers handle\_new\_user function**

**JWT saved in browser (session)**

**Check/create user in profile DB**

**Home**

Authenticated User

*4.2.6 Supabase Authentication System*

As it was already explained, Supabase serves in this app not only a database, but also provides an auto-generated REST API for every table and view in the database. This means we don’t need to build a custom backend to access the data. Supabase gives us a ready-to-use API that follows the PostgREST standard.

This REST API was very useful during development. It allowed us to test data quickly using tools like Postman or the browser and also connect the frontend directly to the database using simple HTTP requests.

**Authorization: How It Works**

To use the API, we need to send the right authentication headers. There are two main types:

1. *Public requests (read-only)*

These use the apikey (anon/public) header. They can access tables that allow anonymous users (like cities or indicators). Example: apikey: your-anon-key

1. *Authenticated user requests*

These use a Bearer token, which is the user’s JWT. This token is stored after login and is sent with every request. Example: Authorization: Bearer eyJhbGciO…

This allows the API to know which user is making the request and return only their data (e.g. their config or events).

**RLS Protection**

All database access is protected by Row-Level Security (RLS). This is very important. It means that users only see or edit their own rows. You cannot access other users’ data, even if you try to change the API request. Admins can still use the service key, which bypasses RLS when needed (for uploading, testing, etc.). This gives us both flexibility and security.

**Example API Call**

To get all indicators for a district, a frontend component can make a request like:

GET https://project.supabase.co/rest/v1/current\_indicators\_view?geo\_level\_id=2&geo\_id=3

Authorization: Bearer <token>

This returns all available indicators for that area. The same logic is used for fetching points, cities, and so on.

*4.2.7 Logs and Analytics (Edge Function)*

In this project, we wanted the Admin to track what users do in the app. This can help us learn how people use the system and improve it in the future. To do this, we created a special backend script called an Edge Function.

Edge Functions are a feature of Supabase. They let us run custom server-side code close to the user, with low latency. We used this to create a function called **log\_user\_event.**

**What Does log\_user\_event Do?**

Every time a user does something important (like selecting an indicator, clicking a button, or switching cities), the frontend sends a POST request to this function. For this project, we defined 5 different event types. These include key actions that help us understand how users interact with the app.

The system is flexible. We can add more event types in the future if needed. It is easy to scale because the function accepts any valid event type and stores it in the same format. The only limit is performance. In a real production system, using this method may slow down the frontend if there are too many events or users. In that case, it is better to use a specialized analytics tool. But for this academic project, this solution is simple, direct, and works very well.

**The function:**

* Accepts only POST requests.
* Reads three fields from the request body:
  + **user\_id** (must be a valid UUID)
  + **event\_type** (example: "SELECT\_INDICATOR")
  + **event\_details** (extra info in JSON, optional)
* Checks that the **user\_id** and **event\_type** are present.
* Validates that the **user\_id** looks like a real UUID.
* Uses the Supabase service role key to connect to the database.
* Inserts the event in the **user\_events** table with the current timestamp.

If everything works, the function returns a success message. If something fails (like a missing field or bad data), it returns an error with details.

**Technical Tools Used**

* **Deno**: A modern runtime for JavaScript used by Supabase Edge Functions.
* **Service Role Key**: This allows the function to bypass Row-Level Security and write logs.
* **UUID Validation:** A small function checks that the user\_id is correctly formatted.

**Deployment**

This function was deployed using Supabase CLI and appears in the project as:

* **Name**: log\_user\_event
* **Type**: Edge Function
* **Deployments**: 1

It is live and ready to receive events from the frontend.

*4.2.8 Professional Email and Domain Setup*

**Domain**

To make this app more professional, I decided to buy a custom domain: **areyouqueryous.com**. This gives the project a serious and trusted image, and it allows us to use real emails and services linked to that domain.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Corporative e-mail**

I used [Zoho Mail](https://www.zoho.com/mail/) to create an email group called **admin@areyouqueryous.com**. This email is useful for admin tasks, but also for setting up services that require an admin contact. For example, this is the email used as the owner of the Supabase project. We wanted to give the app a professional character and avoid using GMAIL, etc.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

**SMTP Provider**

To send real emails (like sign-in links or system messages), I used [Mailgun](https://try.mailgun.com/) as an SMTP provider. Mailgun is a tool that lets us send emails from the app using our custom domain. I connected it to Supabase by adding the SMTP server, port, and credentials in the Supabase settings.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

To make everything work, I had to verify the domain. This included adding CNAME records, DKIM, and SPF values in the DNS settings of the domain provider. These steps are important to confirm that we are the real owners of the domain, and they help make sure that our emails don’t go to spam. This setup helps simulate a real production environment. While Mailgun is free for limited use, it’s enough for testing and development. It also helps show how a professional app should be deployed and managed. Using a real domain and email system makes the app look better, work better, and prepares it for future growth if needed.

**SMTP Configuration**

To make the app feel more real and professional, a custom SMTP email service was set up in Supabase using the current explained Mailgun configuration. Now, all confirmation and login emails are sent from **noreply@mg.areuqueryous.com**. This means users receive branded emails with the project name and style. It also improves trust and avoids spam folders. The email templates were fully customized, and the confirmation link now matches the app’s domain and looks clean. This setup helps the app feel like a real product and not just a test

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

*4.2.9 Summary and Scalability of this Project*

This chapter explained how the Are-u-Queryous? platform was built using Supabase, PostgreSQL/PostGIS, and other modern tools. The system includes authentication, automatic APIs, database views, and an edge function to track user events. All components work together to let users explore maps and data in an easy way. The solution is simple but also designed to grow. We followed best practices like RLS (Row-Level Security), modular architecture, and environment separation. If needed, it can support more cities, more event types, or even a full custom backend. In short, this system is both useful now and ready for future improvements.

4.3 AUQ Frontend

*4.2.1 TODO*

TODO

05   
CONCLUSIONS AND FUTURE WORK

*This section will be further developed in future Continuous Assessment Tests (CATs) and project iterations. Additional details, refinements, and validations will be incorporated as the project progresses.*

06  
 GLOSSARY

*This section will be further developed in future Continuous Assessment Tests (CATs) and project iterations. Additional details, refinements, and validations will be incorporated as the project progresses.*

07  
 BIBLIOGRAPHY

*This section will be further developed in future Continuous Assessment Tests (CATs) and project iterations. Additional details, refinements, and validations will be incorporated as the project progresses.*

08  
 APPENDICES

*This section will be further developed in future Continuous Assessment Tests (CATs) and project iterations. Additional details, refinements, and validations will be incorporated as the project progresses.*