EcoRoute - v1.0.1

Andrea Valentini

Last update: 15th April 2024

Contents

1 The project and project goals

The following is a description of the project problem and the goals to be achieved to complete the assignment. We have divided this section into three groups:

- The **preface** (or scenario) helps understand the environment to develop a sound software system.
- The **problem posed** section includes lists to emphasize the critical points.
- The **goals to achieve** by the assignment.

Note: We analyzed the **citizens' stakeholders** in this project.

Preface

Two urgent global concerns are environmental sustainability and climate change; because of air pollution and greenhouse gas emissions, transportation - especially urban commuting - contributes to worsening those issues.

Even today, urban areas are characterized by a heavy reliance on personal vehicles, which are seen as the most comfortable and efficient way of commuting, despite several studies showing that better alternatives exist in most cases.

Improving public transportation systems' efficiency can make them more appealing to daily commuters and is, therefore, a promising way to lessen environmental impact and, at the same time, to increase the overall quality of citizens' life (article).

Problem posed

The project "Eco-City Commute" (ECC) aims to create a comprehensive software system that makes public transportation within an urban area as easy and efficient as possible, promoting its adoption.

ECC receives data from sensors, deployed on public transport means, that provide:

- Information about their respective occupancy rates.
- Real-time information about public transit timetables.
- Information about bike and ride sharing, from specific services (think at ATM in Milano, BikeMi, Mobike, BlaBlaCar, ...).

Based on these pieces of information, ECC offers services to two types of stakeholders:

- **Citizens**: ECC offers a *mobile app* that allows citizens to input:
  - The origin (within the urban area);
  - The destination (within the urban area);
  - Eventually constraint, for example: they do not want to use a bike; they must arrive at destination within a certain timeframe.

The application takes the input and displays (output):

  - Environmentally friendly routes possibly combining different transportation means.

- **Urban area managers**: ECC offers to managers a dashboard through which they can visualize reports concerning the daily usage of the various available transportation means, their occupation rates and delays (if any).

Goals to achieve

We analyzed the **citizens' stakeholders**. So, the main goal was to develop a software system that offers citizens a mobile app to make public transportation within an urban area as easy and efficient as possible. Therefore, the document seeks to meet two objectives:

- Analyze the requirement aspects.
- Make a well-architecture design.

2 Requirement analysis

2.1 Relevant human and non-human actors

The only *human actor* is the citizen:

- **Citizen**. A user who uses the mobile application and enters the origin and destination of his journey. As the problem says, it can also add some constraints to the research.

Instead, there are several *non-human actors* that allow the user to search for some specific services or even a public transport timetable:

- **Sensor**. An electronic device installed on public transport vehicles that provides information about their occupancy.
- **PTT Server**. A Public Transit Timetable Server identifies the public transport company's server. The application makes some queries to this server to find out which public transport line is available at the time specified by the citizen actor.
- **BRS Server**. A Bike-Ride-Sharing Server identifies the server that the mobile application can query to get the information requested by the citizen actor. The BRS Server actor is as abstract as possible, because we want to be able to add as many services as we want.

  For example, a BRS server could be the BlaBlaCar site that the application queries to know if there is a car available for rent.

The Citizen is the primary actor because he is the stakeholder and the main user of the mobile application.

Instead, the non-human actors are all supporting actors because they provide a service to the application (e.g. the sensor offers the occupancy rate of public transport).

2.2 Use cases

The following use case diagram represents the EcoRoute application with a high level visualization. In the diagram it is possible to see the primary actor (citizen) that interacts (initiate) with the application and the supporting actors (Sensor, PTT Server, BRS Server) that support (participate) the application with the collection of the data from different services.

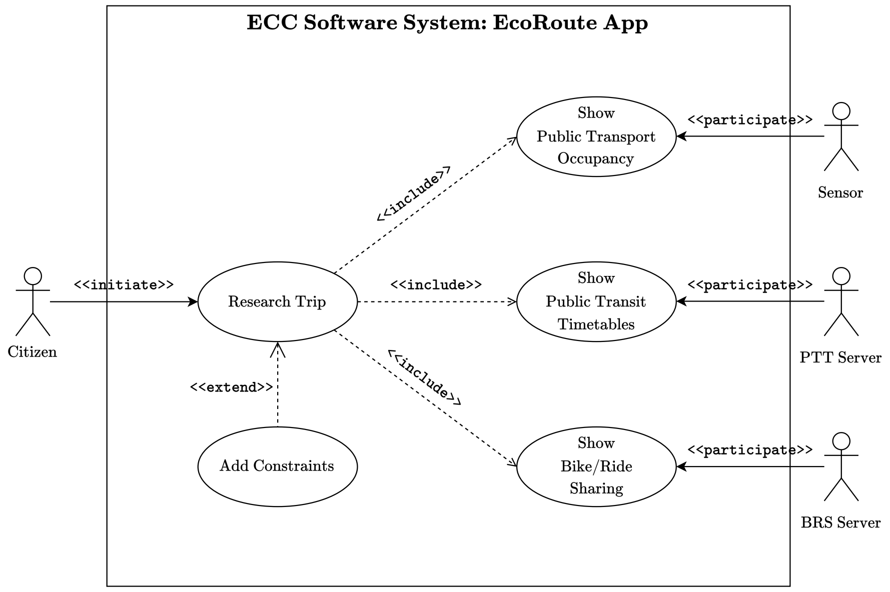

Figure 1: Use case diagram for the EcoRoute App.

To view/download the Use Case Diagram in high quality, click or scan the QR code below:

As we can see from the use case diagram (Figure 1), there are 5 use cases identified. On page 7, in Table 1, we give an exhaustive explanation of the Research Trip use case. However, in the following list, we talk about the other use cases:

- **Show Public Transport Occupancy**. The Public Transport Occupancy is a crucial resource gathered by the EcoRoute (app) from the sensors deployed on the Public Transport and used to avoid bottlenecks at peak times.

  We have included this use case in the Research Trip use case.

- **Show Public Transit Timetables**. The Public Transit Timetables are essential for finding the correct public transport route for the user. The app also uses them to find (and combine) the most environmentally friendly route.

  We have included this use case in the Research Trip use case.

- **Show Bike/Ride Sharing**. Bike/Ride Sharing is mainly used to suggest more ecological means of transport to the user, such as bicycles. The app also uses it to combine with public transport. We have included this use case in the Research Trip use case.

- **Add Constraints**. The user can add constraints, but this choice is not required by the system (as the project problem says). So the user can decide if to impose some limitations or not. We have extended the Research Trip use case with this use case.

Use Case: **Research Trip**	
Primary Actor	Citizen.
Supporting Actor	• Sensor; • PTT Server (Public Transit Timetable); • BRS Server (Bike-Ride-Sharing).
Description	A citizen, the application user, inserts the information about their trip. The information required is the origin and the destination. The constraints can be optional (Add Constraints use case extend Research Trip). Once the user does the research (e.g., by clicking the “research” button), three use cases extend the Research Trip: the Sensor, the PTT Server, and the BRS Server participate in the research made by the user, showing the different ecological choices. The user sees the final result, but the app makes the low-level requests, calculations, and others.
Data	Origin, destination, (optional) constraints, environmentally friendly routes.
Comments	The citizen must enter the origin and destination within the urban area.

Table 1: Detailed use case explanation - Research Trip.

2.3 Domain assumptions

The domain properties are descriptive assertions that are assumed to hold in the world (the part of the real world that is affected by the “machine”, in our case EcoRoute). In this project, the domain assumptions are as follows:

- *Sensors are correctly deployed on public transport means and can share occupancy rates with the ECC.*
- *Public transit timetables are available and can be acquire in real-time from the ECC.*
- *Services that offers bike and/or ride sharing can share their information with the ECC.*
- *The Citizens stakeholder interacts with the software system provided by the ECC through a mobile application (called EcoRoute).*
- *Citizens need to have an internet connection to use the mobile application. The device used should also have GPS.*

2.4 Requirements

2.4.1 Functional requirements

The functional requirements describe the interactions between the system and its environment. Then our application provides the following requirements:

- The mobile application will allow users (citizens):
  - To find an environmentally friendly route.
  - To add any constraints.
  - To view all or one of the Public Transport Occupancy, Public Transit Timetables, Bike/Ride Sharing information in the final result.
- The system should guarantee that the environmentally friendly route computed is the most eco-friendly and give preference to combinations of different modes of transport.

2.4.2 Non-functional requirements

The non-functional requirements specify or constrain characteristics of the system as a whole. Then our application provides the following requirements:

- The EcoRoute application must be available to all users (citizens) 24 hours a day, 7 days a week, including public holidays.
- Downtime during normal working hours should not exceed 1 minute, as this is when the application is most used by citizens.

  Holiday downtime should not exceed 5 minutes.
- The algorithm time to calculate the best environmentally friendly routes should not exceed 10 seconds.

3 Design

3.1 General description of the architecture

The architecture chosen for the EcoRoute project is Client-Server, but with the worker pooling (NGINX web server). The reason for this choice is discussed in section 3.3 on page 18.

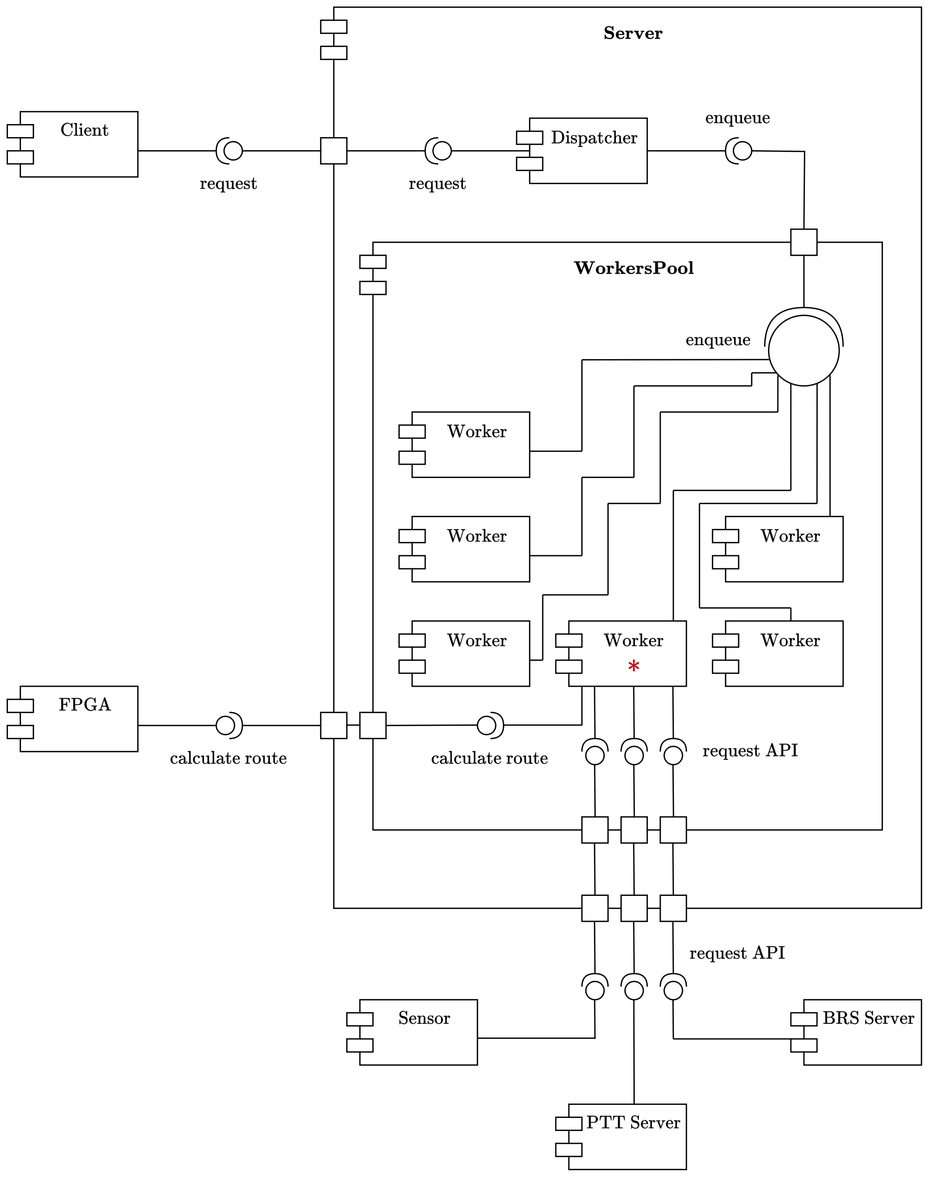

Figure 2: Component Diagram for the EcoRoute App.

To view/download the Component Diagram in high quality, click or scan the QR code below:

Before we go into details, let us assume that sensors are deployed on public transport means and that they publish updated data on occupancy rates at each stop. The data are saved into a HPC database, such as ScyllaDB. We have analyzed this assumption again in the section 3.3 on page 18.

(*) It is also important to note that only one module (Worker) is connected to the Sensor, PTT Server and BRS Server. This is a design choice to avoid a less readable component diagram. We can imagine that each Worker is connected to Sensor, PTT Server and BRS Server.

The following list examines each component of the component diagram (Figure 2):

- **Client**. The module represents the application installed on the stakeholder's (citizen's) device. The aim of the module is to send requests to the server to calculate the route requested by the citizen.
- **Server**. The server module is the NGINX web server (worker pooling technology). It is installed on the server side of the application and can be considered as the core of the architecture. In fact, its aim is to manage the high traffic of requested and instantiated resources. It also aims to send asynchronous requests to the different servers to obtain data about the services requested by the client. Finally, it can request the calculation of the route by interrogating the FPGA module.
- **Dispatcher**. This is the module that takes the request from the client and instantiates the resource. It can enqueue a request into a worker queue if a resource is unavailable. To maintain high performance and optimize scalability, the dispatcher can also drop incoming requests if the worker queues are full (we sacrifice availability in this case).
- **Workers and Workers Pool**. The worker pool is a container in which we can find 6 workers. Each worker can send (asynchronous) requests to the different servers to obtain the data necessary to calculate the best environmentally friendly routes. Finally, each worker can also delegate the calculation of the route to the FPGA module.

  Note that we follows the rule: one worker for each CPU core.[1]
- **Sensor (ScyllaDB)**. It is the module that interacts with the database that contains the data of the transport lines. In order to have a high performance, we assume that the database is a ScyllaDB and this module can query this resource.

[1]Source: Inside NGINX: How We Designed for Performance & Scale

- **PTT Server**. It is the module that interacts with the Public Transit Timetable Server. It can take the data from this resource. This module is an interface that allows to have a great generalization in order to implement as many services as we want. For example, we can easily implement ATM (Milan) or ATV (Verona).
- **BRS Server**. As a PTT server logic, this module can interact with the Bike/Ride Sharing server. This module is also a perfect generalization as a PTT server.
- **FPGA**. This is the module that interfaces with the FPGA hardware. This hardware may be physically located in the server room or in a service such as Amazon F1 EC2.

3.2 Sequence diagrams

The following section contains four sequence diagrams. The first three show how the architecture responds depending on the client's request (if there are constraints, etc.); the last one highlights a possible critical problem.

A simple request (no constraints)

This sequence diagram (Figure 3) is the most common and perhaps the most used. It shows a classic interaction between the client and the architecture.

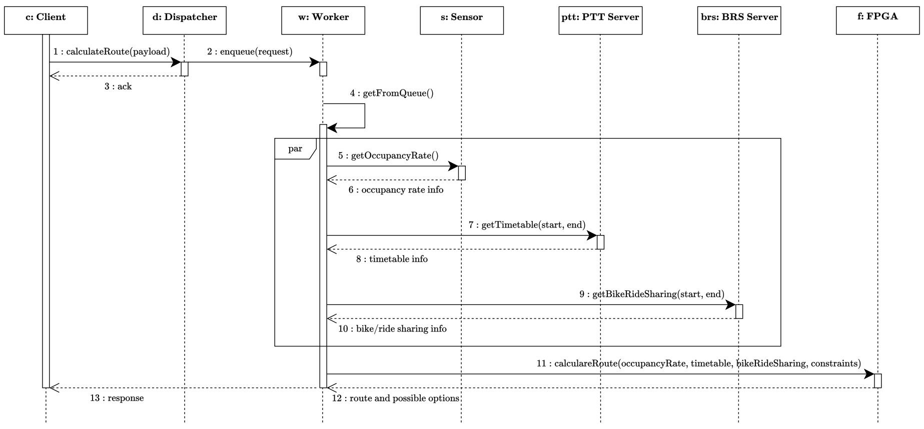
c: Client
d: Dispatcher
w: Worker
s: Sensor
ptt: PTT Server
brs: BRS Server
f: FPGA
1 : calculateRoute(payload)
2 : enqueue(request)
3 : ack
4 : getFromQueue()
par
5 : getOccupancyRate()
6 : occupancy rate info
7 : getTimetable(start, end)
8 : timetable info
9 : getBikeRideSharing(start, end)
10 : bike/ride sharing info
11 : calculareRoute(occupancyRate, timetable, bikeRideSharing, constraints)
12 : route and possible options
13 : response

Figure 3: Sequence Diagram: a simple request (no constraints).

To view/download the Sequence Diagram (Figure 3) in high quality, click or scan the QR code below:

1. A client requests the server to calculate the route. As a parameter, we pass the payload, a well-formatted package composed of the data inserted by the user.

2. The Dispatcher enqueued the request to one of the six workers available.

3. The Dispatcher communicates to the client with a simple ack.

4. The worker gets the request received from its queue.

5. In Parallel (par), the worker queries the database containing the sensors' data.

6. The worker obtains the data about the occupancy rate from the DB.

7. The worker makes a query to the PTT Server with a start (time) and an end (time) to obtain the Public Transit Timetable (already filtered thanks to start/end time).

8. The worker obtains the PTT data from the PTT server.
9. The worker queries the BRS Server with the parameter equal to point 7.
10. The worker obtains the data about BRS from the BRS Server.
11. Finally, the worker can request the FPGA to calculate the route and some possible options. The function's argument is the data obtained previously and constraints (None by default).
12. The FPGA send the response to the worker.
13. The worker ends the sequence, sending the response (route) to the client.

A request with constraints

This sequence diagram (Figure 4) shows the client's request for a trip where he/she would use bike sharing and a public bus service.

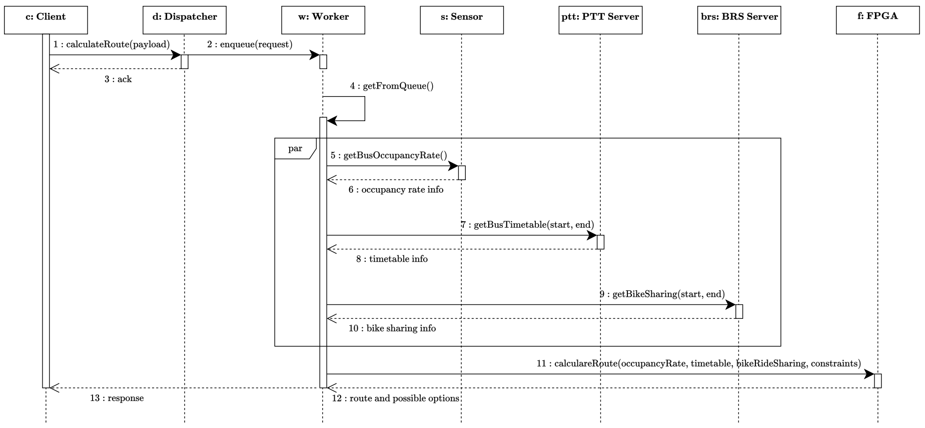
c: Client
d: Dispatcher
w: Worker
s: Sensor
ptt: PTT Server
brs: BRS Server
f: FPGA
1 : calculateRoute(payload)
2 : enqueue(request)
3 : ack
4 : getFromQueue()
par
5 : getBusOccupancyRate()
6 : occupancy rate info
7 : getBusTimetable(start, end)
8 : timetable info
9 : getBikeSharing(start, end)
10 : bike sharing info
11 : calculareRoute(occupancyRate, timetable, bikeRideSharing, constraints)
12 : route and possible options
13 : response

Figure 4: Sequence Diagram: a request with constraints.

To view/download the Sequence Diagram (Figure 4) in high quality, click or scan the QR code below:

The flow is the same as the sequence diagram on page 13 (a simple request (with no constraints)), but in this case, the methods are different:

5. The worker makes a query to the DB to obtain only the occupancy rate of the buses.
7. The worker queries the PTT Server to obtain only the bus timetables.
9. The worker queries the BRS Server to obtain only the Bike sharing service.

A request excluding a service

This sequence diagram (Figure 5) shows the client's request for a trip where he/she excludes a service (Bike/Ride sharing).

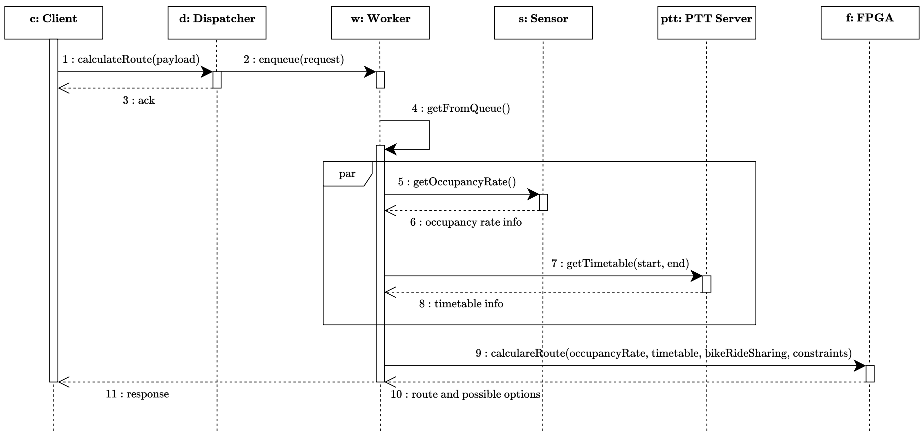

Figure 5: Sequence Diagram: a request excluding a service.

To view/download the Sequence Diagram (Figure 5) in high quality, click or scan the QR code below:

The flow is the same as the sequence diagram on page 13 (a simple request (with no constraints)), but in this case, there aren't two phases (query to the BRS Server).

Critical situation

The following sequence diagram (Figure 6) shows a critical situation that can occur when all available workers have filled the queue. This is a very rare case, but we will discuss it in the next chapter 3.3 on page 18.

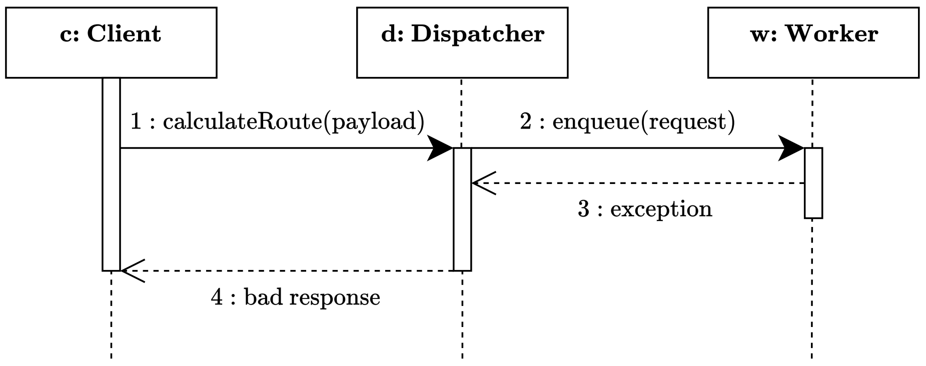

Figure 6: Sequence Diagram: critical situation.

To view/download the Sequence Diagram (Figure 6) in high quality, click or scan the QR code below:

3.3 Critical points and design decisions

We have divided this section into design decisions and critical points.

Design Decisions

☛ *What is the structure of this document?*

✍ This document aims to satisfy all requests asked by the problem, but at the same time, it wants to start from a most general diagram (Use Case) and finish with a very detailed schema (Sequence Diagram).

The initial part of the document should be understandable by any person because it would be very general. The public to which it is addressed is the stakeholders. So, it has a business form (we imagine presenting that schema to a stakeholder meeting).

Finally, we propose a component diagram and sequence diagram. These schemas give engineers a general view of the project's structure. We imagine presenting those diagrams to the CTO (Chief Technology Officer).

☛ *Consider the Figure 2 on page 10. Why does only one Worker component have a link to Sensor, BRS Server, PTT Server and FPGA?*

✍ As we say in that section, the reason concerns the legibility. If we had done a diagram with 24 arrows (6 workers per 4 arrows), it would have been impossible to read.

☛ *Why does the EcoRoute app have a Client-Server (worker pooling) architecture?*

✍ Client-Server architecture is one of the most famous models for creating an application. Furthermore, the worker pooling technique offers us many benefits, such as high concurrency (e.g., during working hours when stakeholders search for how to move into the city). From a benchmark of the NGINX web server[2], we can see that with 36 workers (cores), we can provide the service to almost one-third of Milan's population.

☛ *What is your assumption about the sensors? How do you get these data?*

✍ Each sensor is deployed by public transport. When public transport stops at the stop, the sensor sends the occupancy rate data to a database.

In this project, we assume that the Database is a ScyllaDB because we need a DB architecture that handles millions of operations with very low latencies (single-digit milliseconds in this case!).

We have considered other technologies, such as MongoDB, Cassandra, etc. However, we can see that ScyllaDB is unequal by studying the benchmarks.[3]

[2]Source: Testing the Performance of NGINX and NGINX Plus Web Servers
[3]Source: ScyllaDB Benchmarks

☛ *What are the constraints for you?*

✍ The user can't write any constraint he wants. However, the application allows us to select which service they want. Therefore, it is possible to choose only bus rather than metro transportation. It's possible to select if we want to make a part of the route with a bike or car (carpooling). We also have a timeframe label that allows us to choose the start time of the trip and the end time. Finally, it's possible to customize the route by inserting an additional position that says to the algorithm, "Calculate the route, but the journey must pass here".

☛ *Why do you choose an FPGA to calculate the route? And how can It do that?*

✍ First, an FPGA (Field Programmable Gate Array) is a configurable integrated circuit that can be programmed or reprogrammed after manufacturing. FPGAs are commonly used in applications requiring flexibility, speed, and parallel processing capabilities, such as telecommunications, automotive, aerospace, and industrial sectors.

We chose an FPGA because we want dedicated hardware that makes these calculations as fast as possible. The algorithm used by the FPGA is the classic Dijkstra.

For example, this is a possible architecture of the FPGA to run the Dijkstra's algorithm:

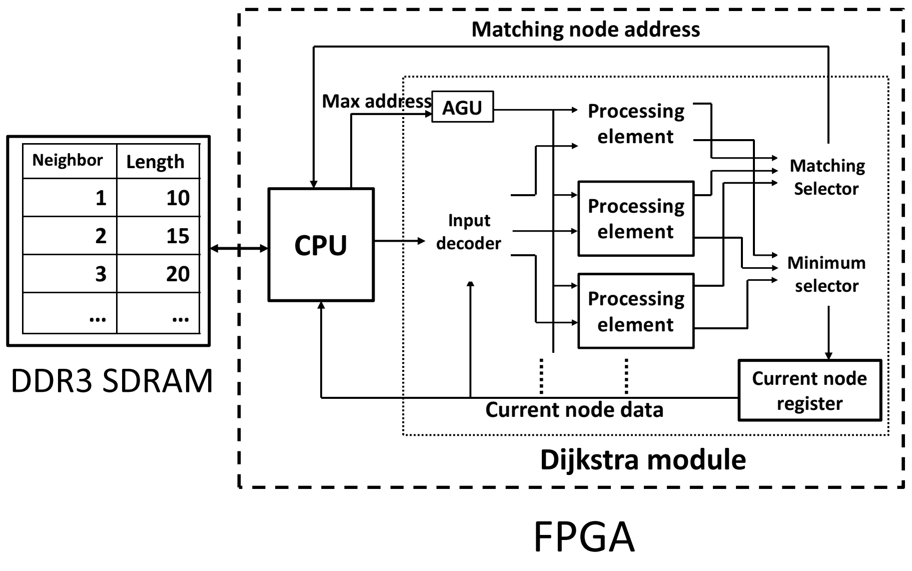

Figure 7: FPGA architecture. [1]

- ☛ *The app has no account management and no guidance once the user has obtained the route. The reason for this choice?*

  ✍ This app is not Google Maps. The main goal of this project is "a comprehensive software system [...] as easy and efficient as possible".

  An account is unnecessary, and this feature implies many consequences (how do we manage client's data? How do we protect the data?). Furthermore, registration could be tedious for the final user. And the performance side? You should save the research history into each account, so where should you put this data? This implies that we need another database. This choice doesn't make sense because the ratio of costs/benefits needs to be equal. Finally, again, the core goal of the project is another.

  Regarding the guidance, the app shows where you are with a GPS, so we can walk our path while watching the application.

Critical Points

- ☛ *What happens if any "Worker" has the queue full and cannot take any requests?*

  ✍ This critical point is infrequent. As the benchmarks show, the number of requests solved is very high. This means that the chance that this problem occurs is tough. We can overlook this aspect.

- ☛ *The algorithm is run in an FPGA. It makes sense, but what about the number of requests? If there are a lot, how can this hardware be managed?*

  ✍ The number of FPGAs depends on the number of active users. Also, it depends on the capital of the project. If it is very high, we can create a cluster of FPGA. Conversely, a helpful service such as the F1 instances of Amazon (EC2) could be convenient. This consideration must be taken with pliers because, in this case, the algorithm and the data will pass through the Amazon service (not so private).

  We may need another document to analyze if creating an FPGA cluster or using Amazon is convenient.

  The article "*Evaluation of an FPGA-Based Shortest-Path-Search Accelerator*" by Y. Takei, M. Hariyama and M. Kameyama shows an interesting performance report about the FPGA and Dijkstra's algorithm:

Input graph	FPGA (50MHz)	Core2 Quad (2.83GHz)
1024 Nodes, 3968 Edges	9.38	16.00
4096 Nodes, 16130 Edges	50.40	94.00

Table 2: Processing time (ms). [1]

- ☛ *The NGINX doesn't have high costs? Also, does each sensor push its data into a database, not increasing the costs?*

  ✍ Again, this depends on the capital of the project. If we need to reduce the costs, we can pass them on to FaaS (Function As A Service). It is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

  In this case, we pay only when the service will be used. We can several reduce the costs. For example, at night, no one should use the app (or fewer users).

- ☛ *What happens if all servers (e.g. BRS Server) (or one of them) are unavailable?*

  ✍ It doesn't matter to the application; it calculates the route with its data. If a service is unavailable, the client will be notified.

References

[1] Yasuhiro Takei, Masanori Hariyama, and Michitaka Kameyama. Evaluation of an fpga-based shortest-path-search accelerator. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 613. The Steering Committee of The World Congress in Computer Science, Computer ..., 2015.