POLITECNICO DI TORINO

Master's Degree in Communications and Computer Networks Engineering

Master's Degree Thesis

Analysis, Design and Implementation of Proof-of-Concept Prototypes and Applications to support Applied Innovation of Enterprises

Supervisor

Prof. Fabio DOVIS

Company Supervisor

Vincent JOSET

Candidate

Valentin PEREIRA

May 2020

Abstract

As a Software Engineer in the New York City's *Capgemini Applied Innovation Exchange*, I have worked on the development of prototypes and applications meant to facilitate the adoption of innovative products and services.

The first focus has been the creation of an interactive dashboard for battery management in fleets of electric vehicles called *Panasonic-Cy* in which I have taken an active role from the application's design to its development and data modeling. A proof-of-concept prototype for optical character recognition of tire stickers has then been built, aiming to infer the characteristics from tires in real-time by scanning stickers, and redirecting users to the corresponding web page.

To foster opportunities and leverage the AIE ecosystem, a cross-platform application for the *National Retail Federation*, called *Capgemini Expo Guide*, has been developed, meant to provide a list of emerging startups for businesses, thanks to filtering criteria and relevance. Finally, a platform aimed to assess the readiness of Capgemini's client to welcome and sustain innovation was made, allowing users to get insights into their organization's innovation performances and relevant advice from the AIE team.

The following document will present my work on these different projects, for which I have been the only developer, therefore building the products from scratch to production.

Acknowledgements

I wish to express my deepest gratitude to Vincent Joset, my supervisor and Technical Lead at the New York Capgemini Applied Innovation Exchange. His knowledge, guidance, and trust have highly contributed to my success during this internship.

I would also like to express my special regards to the AIE Team. In particular to Peter Maloof, director of the New York AIE, for being an inspiring leader; and Charmaine Pek, Conrad Wade, and Samantha Burden, for their advice and kindness.

My heartfelt appreciation also goes to Fabio Dovis, my Academic Tutor at the Politecnico Di Torino, who gave me support and constructive comments on my Thesis.

Finally, I thank my parents who have undoubtedly contributed to my success and encouraged me to become a better version of myself since my childhood.

Table of Contents

List of Tables VI

List of Figures VII

Acronyms X

1 Introduction 1
- 1.1 About Capgemini . . . 2
- 1.2 About the Applied Innovation Exchange . . . 3

2 Panasonic - Cy 5
- 2.1 Project Context . . . 5
  - 2.1.1 Design Thinking . . . 6
  - 2.1.2 UX requirements . . . 7
  - 2.1.3 Functional requirements . . . 9
- 2.2 UI Design . . . 10
  - 2.2.1 Sketching . . . 10
  - 2.2.2 Wireframe/mockup . . . 11
  - 2.2.3 Visual Design . . . 12
- 2.3 Technical Development . . . 12
  - 2.3.1 Global architecture . . . 14
  - 2.3.2 Back-End: Server . . . 14
  - 2.3.3 Back-End: Database . . . 18
  - 2.3.4 Front-End . . . 21
  - 2.3.5 Production environment . . . 24
- 2.4 Outcomes . . . 24

3 Optical Character Recognition for tire stickers 26
- 3.1 Project Context . . . 26
- 3.2 Technical Development . . . 27
  - 3.2.1 Architecture . . . 27

3.2.2 OCR on tire stickers . . . 28
3.2.3 Mapping with website . . . 32
3.3 Outcomes . . . 32

4 National Retail Federation - Expo Guide 34
4.1 Project Context . . . 35
4.1.1 Functional requirements . . . 35
4.2 UX/UI Design . . . 36
4.2.1 Sketching . . . 36
4.2.2 Wireframing . . . 36
4.2.3 Refinements and Visual Design . . . 38
4.3 Technical Development . . . 39
4.3.1 Global architecture . . . 40
4.3.2 Back-End . . . 40
4.3.3 Front-End . . . 44
4.3.4 Builds . . . 46
4.4 Outcomes . . . 47

5 Applied Innovation Enterprise Readiness Assessment 49
5.1 Project Context . . . 49
5.1.1 Principle . . . 50
5.1.2 Functional Requirements . . . 51
5.2 UX/UI Design . . . 51
5.3 Technical Development . . . 53
5.3.1 Back-End . . . 53
5.3.2 Front-End . . . 58
5.4 Outcomes . . . 61

6 Conclusion 63

Bibliography 65

List of Tables

2.1 Functional and technical requirements for the Panasonic-Cy prototype 9

List of Figures

2.1 Example of fleet manager persona analysis 8
2.2 Example of User Journey for a fleet driver 8
2.3 First sketch for the fleet driver dashboard 11
2.4 First sketch for the fleet manager dashboard 11
2.5 Part of the wireframe of the fleet manager dashboard 12
2.6 Final visual mockup of the Panasonic - Cy dashboard 13
2.7 Architecture diagram of Panasonic - Cy 14
2.8 Architecture of Panasonic - Cy REST API 15
2.9 Suggestion of interventions on vehicles in Panasonic - Cy dashboard 17
2.10 Architecture of the Panasonic - Cy Database 19
2.11 Architecture of the Panasonic - Cy Front-End 22
2.12 Representation of State of Health for entire fleet - cluster - vehicle . 23
2.13 Map representation of entire fleet - cluster - vehicle 24
2.14 Final dashboard of the Panasonic - Cy prototype 25

3.1 Architecture diagram of the OCR application 28
3.2 Example of tire stickers to analyze 29
3.3 Sticker Recognition Pipeline 29
3.4 Screenshots from the tire sticker recognizer application 33

4.1 First sketch of Capgemini Expo Guide 37
4.2 Parts of the wireframe of Capgemini Expo Guide 38
4.3 Parts of Capgemini Expo Guide's final design 39
4.4 Architecture diagram of Capgemini Expo Guide 41
4.5 Architecture of the Capgemini Expo Guide Front-End 45
4.6 Screenshots from Capgemini Expo Guide on IOS 47
4.7 Statistics on usage of Capgemini Expo Guide 48

5.1 Parts of the wireframe of the Applied Innovation Enterprise Readiness Assessment 52

5.2 Architecture of the Applied Innovation Enterprise Readiness Assessment's Back-End 54
5.3 Architecture of the Applied Innovation Enterprise Readiness Assessment's Front-End 58
5.4 Data Visualization in the Applied Innovation Enterprise Readiness Assessment 60
5.5 Screenshots from the Applied Innovation Enterprise Readiness Assessment 62

Acronyms

AIE
Applied Innovation Exchange

R&D
Research and Development

NRF
National Retail Federation

BU
Business Unit

SBU
Strategic Business Unit

IT
Information Technology

AI
Artificial Intelligence

EV
Electric Vehicle

UX
User Experience

UI
User Interface

MEAN

MongoDB, Express.js, Angular and Node.js

REST

Representational State Transfer

ODM

Object Data Modeling

AWS

Amazon Web Services

POC

Proof Of Concept

OCR

Optical Character Recognition

CDN

Content Delivery Network

CORS

Cross-Origin Resource Sharing

CxO

C-level Executives

Chapter 1

Introduction

Innovation is everywhere. This popular word is often used when referring to technology but has also been the central element of a plethora of political and socioeconomic speeches for the past decades. While innovation is considered as the essence of advancement in modern societies, the term is generally uttered indiscriminately regardless of its meaning and origin [1]. In this document, we'll refer to innovation as the process of bringing to market new or more-effective products, services, technologies, or business models.

In today's competitive business environment, companies have no other choice but to remain agile to cope with ever-changing customer expectations. Those past years have seen a major shift in the way large corporates approach innovation, with an increased amount of resources, in particular financial and structural, dedicated to the development of environments that foster creativity and opportunities. Companies aim to work hand in hand with their clients, supplies, and stakeholders through open innovation programs [2], unleashing the potential of their networks. Another emerging approach consists of outsourcing innovation: by working with early-stage startups, large businesses manage to lower their R&D costs and reduce risks while increasing the speed to market and improving the quality and quantity of new products and services they develop.

Most companies are willing to follow some of those steps to stimulate an innovation culture across their organization. Provoking such a cultural change is, however, not an easy task. Internal teams may struggle to explore areas they don't know, network with new actors, and design new approaches. For this reason, innovation consulting can be used as a catalyst for expanding innovation ecosystems and innovating faster, by bringing external experience and knowledge of broader industries and sectors. This is where firms like Capgemini and in particular the Applied Innovation Exchange, their global innovation platform, come into play.

As a Software Engineer in the Technology team of New York's Applied Innovation Exchange, my role was to facilitate the adoption of innovative products and services

for Capgemini's clients by building proof-of-concept prototypes. From the research, benchmark, and analysis of emerging technologies and startups to the development and production of pilots, I have been working closely with the rest of the team to discover some innovation frameworks and develop my knowledge of the ecosystem. This document will present the major technical projects I have been working on during my 6-months internship. I will start by detailing the development of an interactive dashboard for battery management in fleets of electric vehicles for Panasonic. Then, after having presented the prototyping of an optical character recognition system for tire stickers, I will showcase my work on a cross-platform application developed for the National Retail Federation. Finally, I will introduce the creation of an innovation assessment tool used internally to evaluate our clients' readiness to apply innovation in their organizations. I would like to underline that, as the only developer in the New York office, I have been responsible for all the developments and productions of the aforementioned products. The part of my work related to the creation of documents and presentations on emerging technologies will not be detailed in this Thesis.

1.1 About Capgemini

Capgemini is one of the world's largest information technology consulting and outsourcing companies. Previously known as Sogeti, the French firm currently employs around 270,000 employees in over 50 countries around the world for €17 billion of revenues in 2019 [3].

The group is supporting its clients on a large range of industries, technologies, and solutions by helping them create new business and technology solutions. It is organized into four major Business Unit (BUs):

- Application Services: with a specific focus on the maintenance and development of complex IT applications, this BU takes care of major transformation projects.
- Consulting Services: by analyzing and developing its clients' services and processes, Capgemini allows them to improve their performances.
- Technology and Engineering Services: taken care of by Sogeti (one of Capgemini's sub-brands), this BU is meant to provide assistance and support to internal IT teams within client companies.
- Other Managed Services: oriented towards a broader range of services, its objective is to support clients on their business activities, on-demand and transaction services, as well as their IT infrastructure systems.

On the international level, Capgemini is regrouped in Strategic Business Unit (SBUs), which correspond to different geographical locations. Each of these SBUs are divided into Market Units focused on specific sectors:

- Financial Services
- Consumer Goods
- Industry, Automotive & Life Sciences
- Telecommunications, Media & Entertainment
- Public Sector
- Energy, Utilities & Chemistry

I did my internship at the Capgemini's Applied Innovation Exchange (AIE), the group's global innovation platform. The AIE operates cross Market Units in order to support and provide the tools to foster innovation for all Capgemini's clients.

1.2 About the Applied Innovation Exchange

The Applied Innovation Exchange is a global network of innovation labs created by Capgemini. Its goal is to bring an environment that fosters innovation within the company and for its clients, by researching and analyzing emerging technologies across all industries and sectors [4].

To enable enterprises to discover relevant innovations, the AIE works closely with startups and other internal Market Units to have the capability to quickly scale ideas into prototypes. It leverages a specific framework to facilitate the development and adoption of innovative ideas. This framework relies on four pillars:

- Discover: through ideation sessions and demonstrations, the AIE allows its clients to explore new technologies and find ways to apply them to transform their industry.
- Devise: to prove the value of those ideas, they are turned into prototypes, minimum valuable products or business model designs and tested upfront.
- Deploy: working in an Agile way allows the AIE to quickly iterate on the pilot launched by measuring its impact and improving it along the way.
- Sustain: the AIE supports the organizational and structural changes needed to create a culture of innovation and sustain its business value for clients.

As a Software Engineer at the AIE, I have mainly worked on the *Discover* and *Devise* aspects. I made deep analyzes of emerging technologies and trends in specific industries, developed proof-of-concept prototypes for some of our clients, installed demonstrations in the innovation lab space, and facilitated sessions both with startups and partners. This document will focus on the technical work realized while creating minimum valuable products.

Chapter 2

Panasonic - Cy

At the Applied Innovation Exchange, we have been working with several clients and partners to propose innovative solutions and showcase their viability.

The *Panasonic - Cy* project was launched in collaboration with *Fahrenheit 212* [5], one of Capgemini's sub-brands focused on innovation consulting. Our client, *Panasonic* [6], is a leader in battery manufacturing for Electric Vehicles (EVs) and is willing to offer a set of services for battery management of fleets of EVs. Based on our analysis of the market and the solutions that could be implemented, we decided to aim for a platform facilitating the management of the EVs through battery data visualization and recommendations based on predictions inferred from the battery's state.

After working with the team to identify the relevant features and create the UX/UI requirements, I have worked by myself on the development of the prototype. The time constraint to realize the functional proof-of-concept was high as we only had three weeks ahead of us. The team and I therefore worked in an Agile way to quickly iterate on the development and adapt the features implemented along the way.

2.1 Project Context

Those past years have seen the demand in Electric Vehicles increase a lot, mainly thanks to the progress in battery optimization and management. Most challenges in EVs are around the battery. It needs to charge efficiently, store energy in an optimal way, and operate safely. It is therefore crucial to properly understand and manage battery usage to reduce costs and optimize vehicle performances over time. Panasonic, as a leader in battery manufacturing, is willing to provide large fleet operators, such as *FedEx* or *UPS*, with battery management systems to help them manage their fleets in a more efficient manner. With thousands of vehicles

in activity daily in the United States, a lot of time and money can be saved if batteries are properly used.

The objectives of this platform are multiple and depend on the persona. Hereafter are the main ones that may have an influence on, or be influenced by, the battery management systems.

- Fleet drivers: they can have a huge influence on the battery state of their vehicle. The optimal charging time, the position of the grid charging stations, their driving behavior, the type of road they are taking, etc. can be taken into account to make them gain time while reducing the friction with the battery.
- Fleet managers: they can have a broader overview of the battery usage in multiple areas and use the environment, the type and age of the battery, as well as other elements, to reassign vehicles to specific paths and optimize batteries on a larger scale.
- Utilities and second-hand market: the energy stored in the batteries after working hours can be used to power the buildings or given back to the grid. Older batteries can also have second lives with some services more suitable to their reduced performances.

For this prototype, we decided to focus mainly on the elaboration of a dashboard for the management of the battery from the fleet drivers and fleet managers perspectives.

2.1.1 Design Thinking

Supporting innovation is not just bringing a solution to a very specific problem. The goal is to bring long-term value to our clients by giving them a competitive advantage. This can be achieved by trying to see the broader picture, to maximize the impact and benefits of the solution. At the AIE, for most of our projects, we have been using a methodology called Design Thinking. This human-centered process is aimed to help find creative and innovative solutions to our problems, by better understanding the people for whom the products are designed [7].

There are five main phases at the heart of the Design Thinking work process, each of them is meant to foster 'out of the box' thinking. We have used this process for the *Panasonic - Cy* project.

- Empathize: the team has not only engaged with our direct client Panasonic but also with the companies towards which the tool was directed, such as FedEx and UPS. This allowed to better understand which personas could benefit from the platform and the challenges at the different levels.

- Define: Thanks to the previous step, we have been able to identify the users' needs and the problems that arise for each of them.

- Ideate: For our ideation phase, we have brainstormed by creating *User Journeys* for each persona. We will detail this phase a bit more in the next subsection.

- Prototype: Once the requirements have been identified, I have been able to start developing the solution. The process will be explained throughout this chapter.

- Test: To iterate quickly and achieve a better product, we have tested the prototype along the way and redefined some of the featured. The final test is of course with our client.

Since this methodology has been used in most of the projects that will be detailed in this document, you can refer to this section when a reference to Design Thinking is made.

2.1.2 UX requirements

User Experience (UX) design is meant to ensure users have an enjoyable and instinctive experience when using our products, while *User Interface* (UI) design focuses on the visual aspect of the product to stimulate user engagement. As part of our ideation phase and to define the list of needs and general structure of our prototype, we have been defining the User Journeys for our different personas, to build the UX and UI around them [8].

Of course, prior to this step, the team had already identified the different personas and conducted analysis and interviews with people corresponding to each of them. This way, we could better understand the type of product we needed to build in order to fulfill their needs and overcome their pain points. Figure 2.2 gives an example of the persona identification and analysis for the fleet manager.

Defining the User Journey for each persona allows us to identify the new experiences that can be created for the users. It is a way to see the product from the user's perspective in a visual way. In our case, the two personas we are focusing on are the *fleet driver* and the *fleet manager*. The User Journeys are then used for understanding their needs and pain points to determine which features to build and how to prioritize them.

You can see a typical User Journey for a fleet driver in figure 2.2. The different actions and touch points identified both for the fleet driver and fleet manager have then allowed us to work more precisely on the features required for both of them.

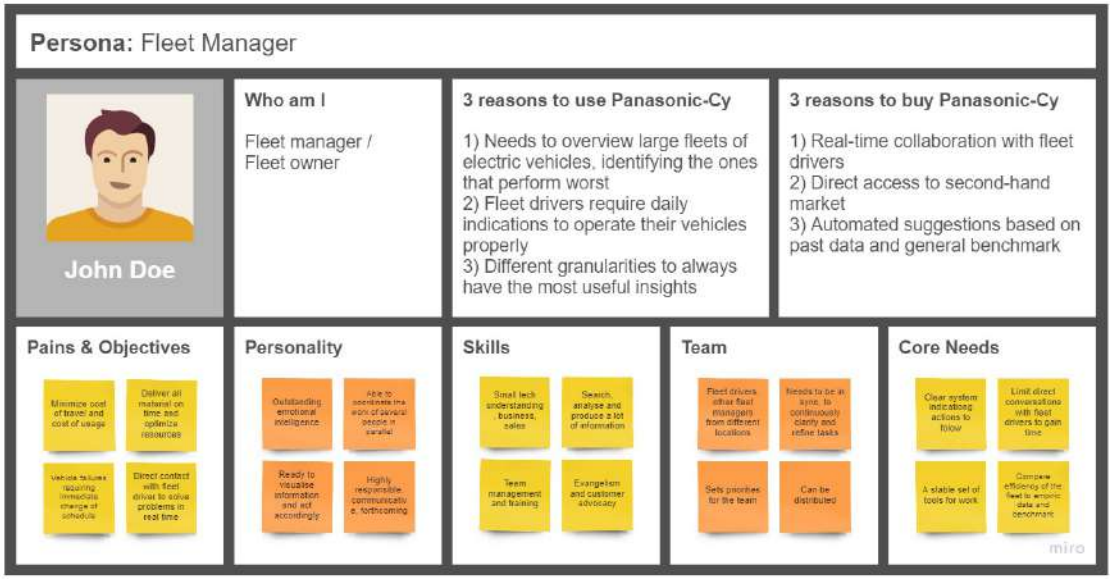

Figure 2.1: Example of fleet manager persona analysis

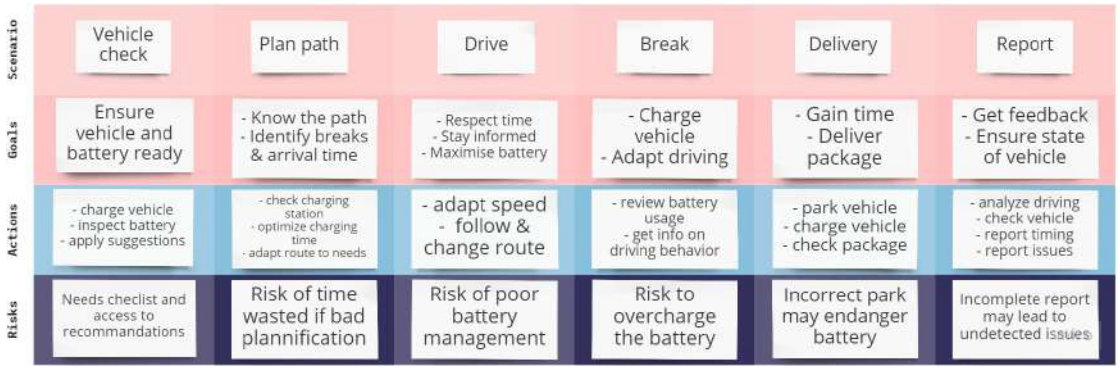

Figure 2.2: Example of User Journey for a fleet driver

Features for fleet drivers

The fleet driver will probably be on the road for most of the time. The application must then be accessible on mobile or tablet for him to ensure he can access it anywhere. When logging into the platform, he must be able to retrieve his currents/pending deliveries with all the delivery parameters (destination, expected arrival time, EV model, etc.).

When accessing the dashboard for a specific delivery, he should be able to see a checklist of elements to control on the vehicle, and related to the battery, before departing (ex: state of charge, current health of the battery, etc.). The dashboard will display relevant data such as the time before critical discharge, the optimal

charging moment or the suggested speed.

The dashboard will also feature a map with GPS tracker, indicating the nearby charging stations, the optimal path, information on the road, and parking spots. It will finally include some gamification features containing tips for a correct driving behavior and giving points to the drivers if they are respected.

Features for fleet managers

Fleet managers will mainly access the platform from a fixed location, such as a control center. Their dashboard is then meant to be used on a tablet or a computer. When planning a delivery, in addition of selecting the location, time and type, they will be able to pick the best EV model based on automated suggestions.

To overview the fleet activities, their dashboard will contain a map of the current vehicles on the road, with specific information for each of them (ex: battery health, charging speed, mileage, etc.). They will also have the possibility to visualize how all vehicles are performing based on the benchmark, to easily target the ones that need actions, with recommendations on how to maximise their performances.

A panel with all the alerts and critical battery statuses will also give general fleet performances (ex: Total cost of ownership, energy cost per mile, etc.) and and battery information. The dashboard will also give access to the battery market place, giving suggestions and allowing to sell or buy batteries.

2.1.3 Functional requirements

The functional requirements are inferred from the different features we aim to give to our platform. The elements we want to showcase to the fleet driver or fleet manager are functions of a set of parameters coming from the vehicle and the environment. The function requirements are shown in the transformation matrix presented in table 2.1.

	Design & Technical Requirements							
Functional Requirements	Battery size	Battery model	Grid capacities	State of Health	Energy consumption	Path taken	Type of drive	Energy Cost
Charging time	**X**	**X**	**X**	**X**				
Distance with current battery charge	**X**	**X**		**X**	**X**	**X**	**X**	
TCO Savings		**X**		**X**	**X**			**X**
Time to live		**X**		**X**		**X**	**X**	

Table 2.1: Functional and technical requirements for the Panasonic-Cy prototype

The goal of this step is to define a list of parameters and technical requirements

that we are willing to feature in our prototype. The correlation between the functional and technical requirements has been determined by analyzing different relevant papers [9]. Given the limited amount of time to develop the prototype, it may not always be possible to implement all these parameters. They serve as an indication for what the final product should contain.

The determination of the parameters, as well as the audit of clients for the UX requirements phase allow us to move forward with the first steps of the creation of the proof-of-concept, which are the sketching, wireframing and creation of the visual design. These elements will be detailed in the next section.

2.2 UI Design

Now that we have a better understanding of the needs of our users and our functional parameters have been determined, we can start building the User Interface. To ensure that the design will meet our client's requirements, we start with a brainstorming/sketching phase during which we simply try to validate the general flow of the product. Once this is completed, we develop a first iteration of the design under the form of a wireframing. The wireframing gives an overview of what the final product should look like, without putting too much emphasis on the design elements themselves. Finally, we establish a mockup of the visual design by adding the visual identity on top of the elements previously agreed upon. This visual design is then the starting point for the developer to build the front-end of the prototype.

Note that although I was mainly positioned on the development side of the projects, I have also been deeply involved in the elaboration of the UX/UI of every product. For *Panasonic - Cy*, I built the first sketches and wireframes and a designer helped with the final visual mockup.

2.2.1 Sketching

Sketching is a very important step in the development of the User Interface. It allows us to make modifications very quickly and easily validate the core concepts of our product. Figures 2.3 and 2.4 give an overview of the very first iterations of the UI. They are not detailed but allow us to validate the concepts we wanted to put into place. At this stage of the project, I had already started to develop the front-end in accordance with these layouts since the development time was very constrained. Ideally, the real build should not start before the first visual mockups are established.

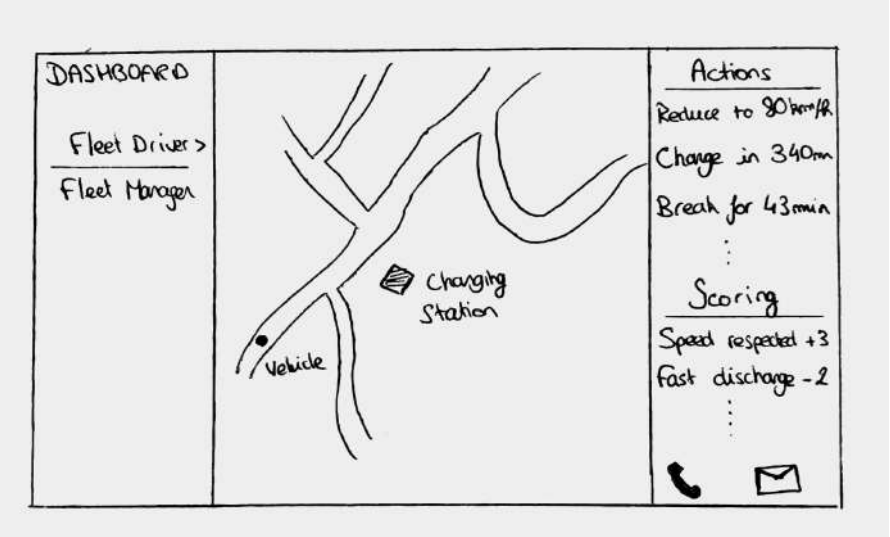

Figure 2.3: First sketch for the fleet driver dashboard

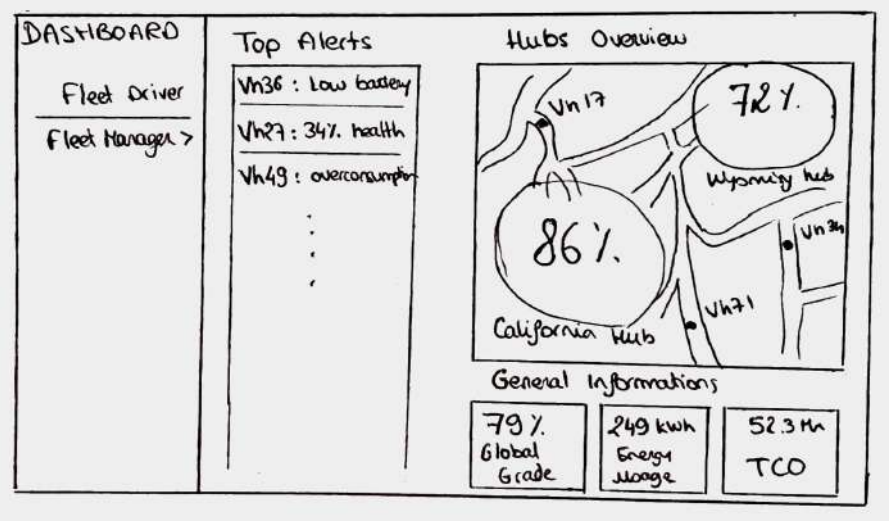

Figure 2.4: First sketch for the fleet manager dashboard

2.2.2 Wireframe/mockup

While the UI sketches and the general flow serve as a base for building our design, the wireframe is important to choose the style for our interface. We can usually build multiple wireframes with different styles and let the team vote for the one they prefer to ensure we are heading towards the right direction. At this stage, it is not necessary to include design elements or color schemes, although it can also give an idea of what the final product should look like. This medium detailed

representation of the design shows how the UI renders in static. It is also possible to make clickable prototypes at this stage, we didn't decide to do it that way for *Panasonic - Cy* but some of the other projects were built using this feature.

Figure 2.5 is a part of the wireframe that was used for the fleet manager's dashboard. It is slightly detailed but we will see that the final design is much more precise.

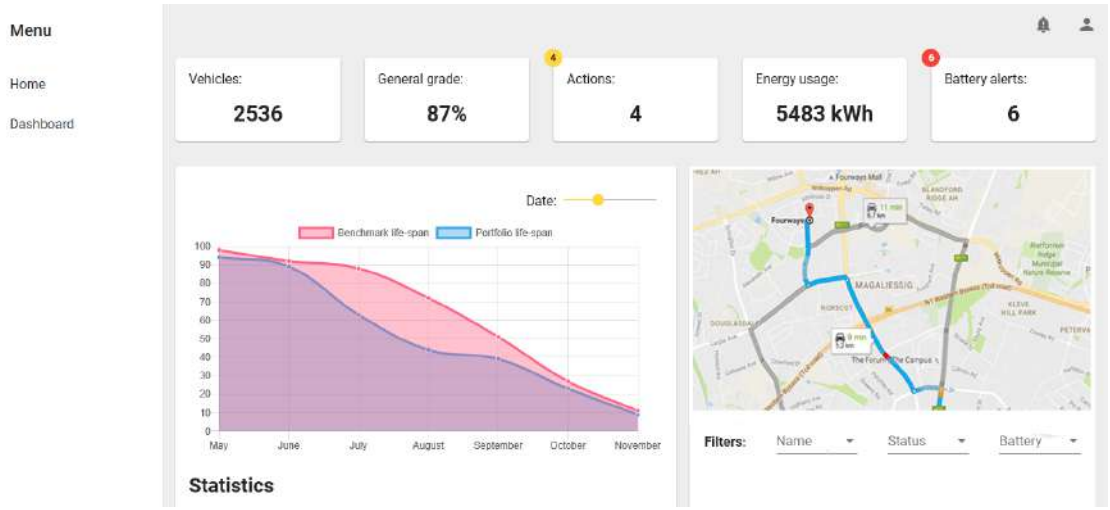

Figure 2.5: Part of the wireframe of the fleet manager dashboard

2.2.3 Visual Design

When the general style had been agreed upon, we have worked closely with *Farhenheit 212*, part of Capgemini, to define the final version of the design of our User Interface. A specific color scheme was chosen and the visual elements have been integrated to the design.

To make things easier for the development, the team worked with *Zeplin* [10] to translate design elements into code that can be directly integrated in the front-end. Figure 2.6 shows the final design of the dashboard for the fleet manager. We will see later on that the actual design of the prototype is almost identical to this mockup.

2.3 Technical Development

As soon as the first sketches were established, the build of the actual prototype started. Since the time constraint was high, we decided that it would be more

Figure 2.6: Final visual mockup of the Panasonic - Cy dashboard

efficient to iterate on the design and the development at the same time, rather than waiting for the design to be finished in order to start coding.

Knowing that a lot of elements of the User Interface would change along the way, I decided to focus the first phases of the development on implementing the main features and the back-end. I therefore started with my environment setup, for which I was totally free to use the technologies I would consider the most relevant to build the best possible prototype in the shortest amount of time.

The solution stack I chose was the MEAN stack, which stands for MongoDB, Express.js, Angular and Node.js. The advantage of this choice is that all components of this stack support programs that run in JavaScript, which then allows me to build both the client-side and server-side environments in the same language. Further details on each component of this solution will be detailed in their corresponding subsections.

Since the final product needed to be easily available for the client, we decided to run it into production. We decided to host in on an Amazon Web Services (AWS) instance. Note that the solution needed to be both available on tablets and smartphones. In order to make it easy to develop, we decided to only create a responsive web application. It could have been possible to use some cross-platform development environment. This type of environment has been used in other projects

explained later on.

2.3.1 Global architecture

As mentioned previously, we have used the MEAN stack to build this application. This not only allows us to rely mainly on JavaScript but also allows us not to reformat data during transfers from back-end to front-end. The front-end, built in Angular 8, communicates with the server through a REST API built with the framework Express. The server is directly connected to our MongoDB database thanks to the MongoDB driver running in Node.js.

Figure 2.7 gives an overview of the interactions between the different parts of the application.

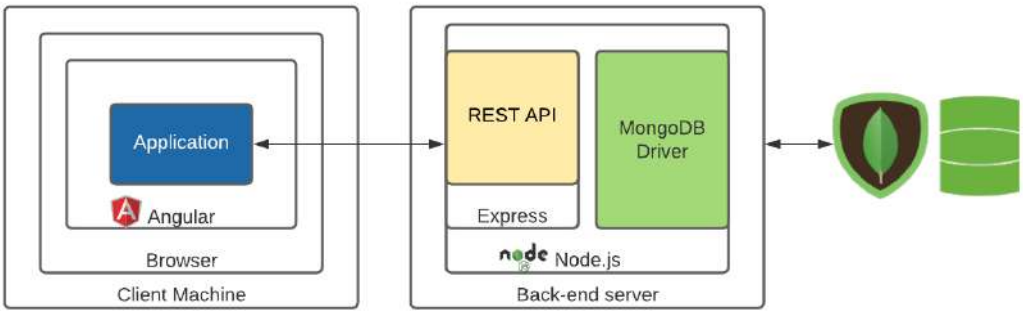

Figure 2.7: Architecture diagram of Panasonic - Cy

We could have made the choice to create a serverless application using some cloud functions for example. However, we decided to aim for a server with a REST API in case the prototype was scaled up and the server reused in the a project. We will later see that the server has been very useful, not only to ensure a lag-free experience for the user but also to allow automatic generation of our data.

2.3.2 Back-End: Server

The server is a RESTful API built with Node.js and the framework Express. It allows us to communicate directly with the database to preprocess the data that are then fetched from the front-end.

Our server's main goal is to fetch all data about the vehicles previously generated in the database (more about it in the section *Back-End: Database*), organize and provide this data to the front-end. It also allows us to run analyzes on the vehicles' parameters, defined in Table 2.1, to create forecasts and suggestions in the dashboard.

Architecture

The architecture of the server is based on a system of routes and controllers relying on the REST pattern. Figure 2.8 gives an overview of those different routes and their associated controllers.

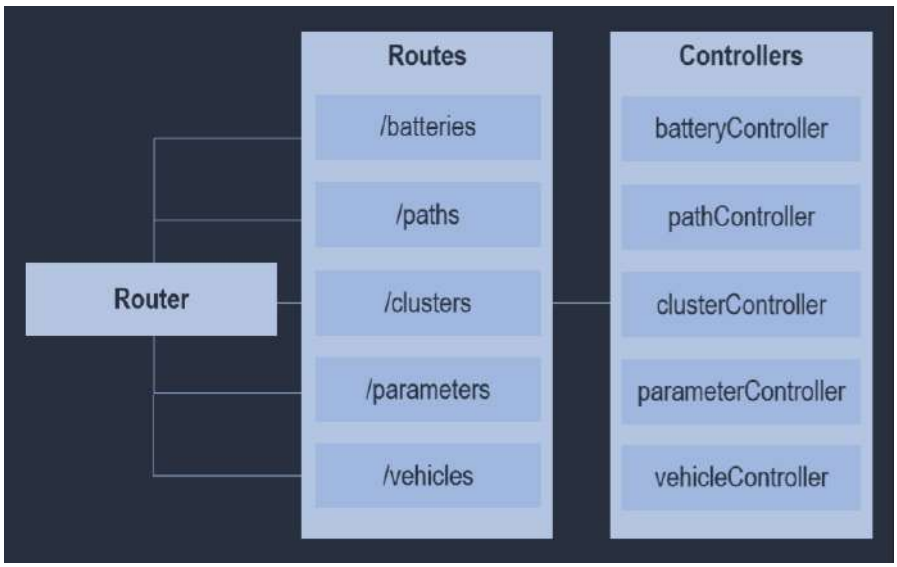

Figure 2.8: Architecture of Panasonic - Cy REST API

The router is used to redirect the requests to the corresponding page on the server. The routing is easily handled thanks to the Express framework. Each route is associated with its own controller. The routes are handling the requests to sub-routes to access specific requests for an element (ex: getting a vehicle with a specific id). The controllers are implementing the methods communicating with the database to render the elements corresponding to the request.

RESTful API

The API allows us to communicate with the server through simple HTTP requests. Given that this prototype was the very first version of the product presented to the client, it was very likely that a second prototype would be built later on. For this reason, building an API was a safer way to provide compatibility between the two prototypes by reusing the back-end environment.

The RESTful API, which stands for Representational State Transfer, is a particular approach aiming to optimize the communication between our different web services. Here, mainly between our back-end server and our customer-facing dashboard.

REST APIs function with the idea that every resource is accessible at a particular address. It is then possible to use resource identifiers to very simply access the elements we need to fetch from the front-end. In our case, as you can see in Figure 2.8, we created very explicit routes onto which we mapped our controllers. The front-end can then access the desired resource by sending an HTTP request to the corresponding route, the controller will execute the server-side code and return the data.

Examples: To get the list of all the vehicles in the database, send a GET request to:

```
/vehicles
```

To get all the paramaters of a vehicle at a certain date (say March 2020), send a GET request to:

```
/parameters/all/2020-03
```

Node.js and Express

Using Node.js as JavaScript runtime environment and Express as server-side framework makes it a lot easier to develop our REST API and back-end in general.

Node.js: It is an open-source, server-side platform that allows to easily build and scale real-time applications [11]. It also contains an extensive library of modules that make the development simpler.

Express: It is a web framework that works on top of Node.js and makes it much easier to built REST APIs with its structure allowing easy management of different HTTP requests thanks to its system of routes and controllers [12].

Intervention forecast

In order to create realistic intervention forecasts on our vehicles to identify what changes should be made to improve its performances, we ran statistical analysis on our datasets. Figure 2.9 shows what the intervention panel looks like in the dashboard.

One thing important to consider is that JavaScript is not an optimal language to run statistical analysis, usually not fast enough. For the development of this prototype, however, it was the simplest solution to have some of the required functionalities up and running quickly.

Figure 2.9: Suggestion of interventions on vehicles in Panasonic - Cy dashboard

For this, we used a library called *Jstat* [13]. For each vehicle in the fleet, we then calculated statistical analysis on its different parameters to identify which ones performed the lowest, and suggest interventions in accordance. A simple linear regression was then used to determine what TCO Savings such changes would lead to.

Example: To know the influence of the type of route, we used different score depending on the type of road. Ranging from 1 (extremely low duty route) to 10 (very high duty road). We then simply calculate the average score for each vehicle over their existence. Let TotP be the total number of rides the vehicle has made since its origin.

$$VehicleScore = \sum_{n=1}^{TotP} routescore \tag{2.1}$$

We can then very easily get the average vehicle score for the whole fleet. Let N be the total number of vehicles.

$$FleetScoreMean = \sum_{n=1}^{N} VehicleScore \tag{2.2}$$

And its standard deviation.

$$FleetScoreStd = \sqrt{1/N \sum_{n=1}^{N} (VehicleScore - FleetScore)^2} \tag{2.3}$$

By then using *Jstat*, we compare the value of a certain vehicle with the rest of the fleet to determine whether its score is normal or not.

```
P_VehicleScore = jStat.normal.pdf( VehicleScore, FleetScoreMean,
    FleetScoreStd );

if(P_VehicleScore < Threshold){
    diffScore = FleetScoreMean - VehicleScore;
    return diffScore;
}
```

The Threshold can be used to have a more or less selective algorithm. In our case, if a vehicle score was far from the average of the fleet's, we determined the *diffScore* to then quantify what type of routes should be prioritized.

It is important to note that this is not an optimal way to make these intervention forecasts, but is rather simple to implement considering the low amount of time we had to develop this prototype. If this was meant to scale, it would be better to use a back-end in *R* or *Python* and use some Machine Learning models to predict which parameters have a negative impact on the vehicle. It would also be performing much better in real-time.

2.3.3 Back-End: Database

The database we used for this project is *MongoDB*. It is a NoSQL database which uses documents instead of tables. We also used the Object Data Modeling (ODM) *mongoose* to simplify data management within the server.

Because of our limited amount of time and the difficulty to access large amount of data to use for the project, we decided to also model the data ourselves. To do so, I developed a script to auto-populate our database with relevant parameters following real-life patterns.

Architecture

The schema in Figure 2.10 presents the different database collections used in this project, as well as the relationships between them.

It is important to note that vehicles' parameters are changing over time. To model this phenomenon, we create a new *Parameter* object at each specific time interval (in our case, one per month). Therefore, each vehicle has an array of parameters representing its evolution through time. This is particularly useful for retrieving data at a specific date, plotting the history of the vehicle's data and making a prediction on future years.

MongoDB

MongoDB is a NoSQL database with a JSON document datastore. It means that the database isn't queried with SQL, and stores documents with a JSON-like format

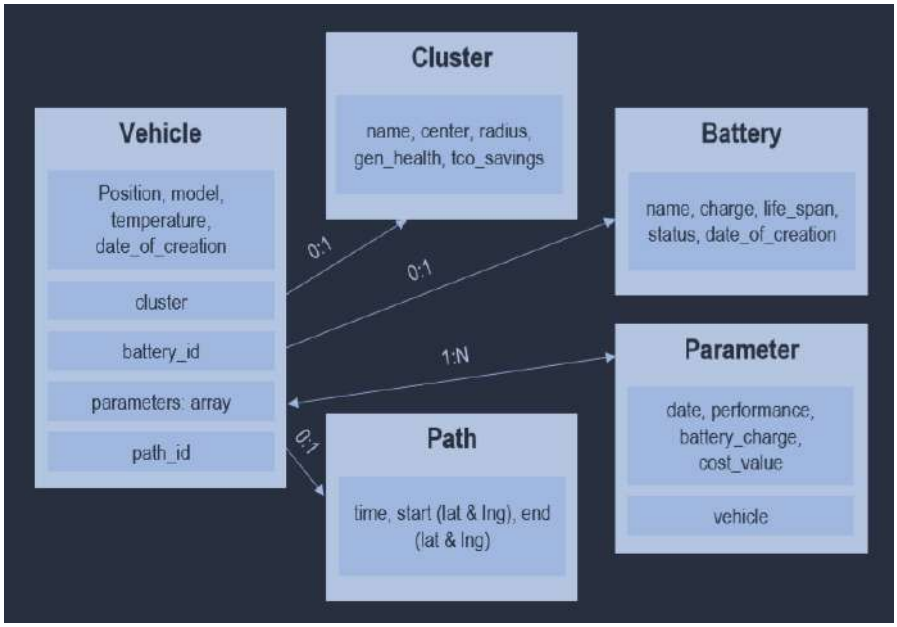

Figure 2.10: Architecture of the Panasonic - Cy Database

[14]. In our case, it is very useful because it is usually faster than other databases and scales very well.

Database population

Since we are modeling a huge number of elements, it is much more efficient to automatically populate the database with a generation algorithm. This auto-population script has to properly generate vehicles with relevant parameters and distributed among the different clusters.

The following script automatically erases the previous database and populates it with new data. It allows to generate the number of vehicles we want.

```
node populatedb {{dbAdress}} {{numVehicles}}
```

Cluster generation

To model a fleet manager overviewing different hubs through the platform, we created different Clusters. In our representation, each cluster is a state in the United States of America. This is not how real fleets are operating as they have hubs at much smaller scales, but was a way for us to simplify things while keeping the core concept.

Instead of representing each state like we had initially envisioned, we picked the five biggest ones that were also easily visible on a map, to make the visual representation cleaner.

To easily generate vehicles in each cluster, I created a JSON file collecting the boundaries in latitude and longitude of all the states I was interested in. You can see an example for California below.

```
1    "CA":
2    {
3      "name": "California",
4      "min_lat": 32.5121,
5      "max_lat": 42.0126,
6      "min_lng": -124.6509,
7      "max_lng": -114.1315
8    },
```

It was then easy to create the *Cluster* instances in the database by iterating on the elements of the JSON file. Each vehicle was finally assigned to a specific cluster by generating coordinates at random until they were falling into a specific cluster. This had the advantage to also distribute vehicles homogeneously among the clusters, with the biggest clusters having the biggest number of vehicles.

Vehicles' parameters

To generate data useful for our dashboard and modeling the behaviour of a real fleet, the most important part is the generation of the vehicles' parameters. Most of them can be found in the Table 2.1.

There are multiple challenges coming with this representation: we first need to ensure that the mathematical models used to characterize vehicles' behaviour correspond to possible data, but also that each vehicle individually ages following a specific pattern that makes sense based on his value at the time of observation.

In average, the battery loses 20% of its total State of Health in about 6 years, and has a polynomial degradation rate.

We therefore now that the shape of the function we need to use to model the battery life of our vehicles is of the form:

$$y = -x^a + b \tag{2.4}$$

Since at time 0, our battery is theoretically at a perfect state of health, we have that b = 100. Knowing that the decrease rate after 6 years is about 20%, we have that:

$$-6^a + 100 = 80 <=> a = \frac{ln(20)}{ln(6)} <=> a \approx 1.7 \tag{2.5}$$

Finally, since we are generating the parameters of our vehicle for each month instead of each year, we have the final baseline equation:

$$y = (\frac{x}{12})^{1.7} + 100 \tag{2.6}$$

Using the same method to find the extreme values for healthy and damaged vehicles, we could easily come up with the maximum and minimum values of the baselines to then randomly generate our vehicles' parameters from there, and have coherent representation.

Here is an overview of the code to generate the state of health and the decrease in cost. Note that for the cost, we made an approximation that the value of the vehicle decreases of 40% each year and is related to the overall state of health of the vehicle.

```
// Coef represents the amount of degradation of the vehicle
let coef = Math.random();
for (var date = new Date(date_creation); d <= projection; d.setMonth(
    d.getMonth() + 1)) {

    let baseline_value = -Math.pow(month / 12, 1.7 ) + 100;
    let baseline_value_max = -Math.pow(month / 12, 1.4 ) + 100;
    let baseline_value_min = -Math.pow(month / 12, 1.95) + 100;

    // Calculate state of health based on the random coef of
    degradation
    let newperformance = coef*(baseline_value_max -
    baseline_value_min) + baseline_value_min;

    // Calculate the new cost
    cost_value = cost_value - cost_value * (0.4 / 365) * (1 +
    performance/100);
}
```

This process has been followed to model the evolution of each parameter in our vehicles. This allowed us to very easily generate hundreds of vehicles with relevant data that we could then exploit in the front-end and for our estimations.

2.3.4 Front-End

The front-end is a simple single-page application and was built using the framework Angular 8. It consist of a main dashboard for the fleet manager and a simple tab for the fleet driver to view the map and receive messages from the fleet manager.

The dashboard itself is composed of a four main sections: The first one represents the overall performance of the fleet's vehicles in comparison to a general baseline.

This allows to easily detect which vehicles are performing worse than average and should be investigated. The second section is a map representing each cluster and vehicle, which enables and easy overview in a geographical manner. Under the map sits the third one, that offers a listing of all vehicles with a possibility to filter them based on some specific parameters. Finally, the last section displays all the parameters concerning the fleet or a specific vehicle.

Architecture

The architecture of the front-end is rather simple since most of the interactions are happening on the main module which hosts the dashboard.

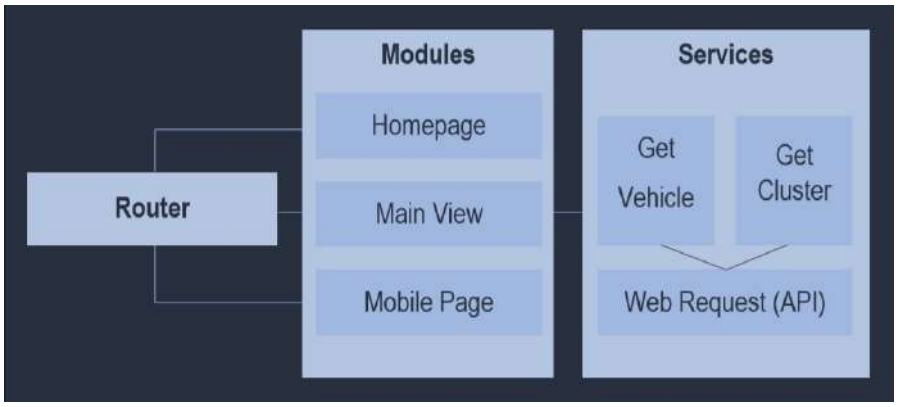

Figure 2.11: Architecture of the Panasonic - Cy Front-End

The router is taking care of rendering the correct page based on the request. We are using the @angular/router package which allows a simple handling of the routing as well as lazy loading, improving the app's efficiency.

The Homepage module is the first one the user accesses. It is displaying the landing page of the app and handles the login dialog. It then redirects the user either to the Main View (for fleet managers using tablet and desktop) or to the Mobile Page (for fleet drivers using mobile).

The Main View module is the core element of the app. It is the one interacting with the services to display real-time data changes on the main page. On the contrary, the Mobile Page module simply displays the map and chat features

The services are facilitating the communication with external APIs such as the server. They are all relying on the Web Request service which handles the requests coming from the other services

Angular

To build the front-end of *Panasonic - Cy*, we've used the popular framework Angular, and most particularly Angular 8 [15]. This framework allows to build efficient single page applications and benefits from a lot of libraries and pre-built material. The ability to create components and reuse them at different locations of the app also makes it very powerful to scale and build new features quickly.

Fleet status representation

We mentioned previously the importance of parameters generated for each vehicle to model the actual behaviour of the fleet. Figure 2.12 shows how the State of Health of the fleet performs compared to a baseline modeling the average behaviour of this type of vehicles.

There are three levels of visualization: The first one represents the status of the whole fleet. It is possible to evaluate some of the parameters such as the Total Cost of Ownership or the number of issues directly from the dashboard. The second level shows the behaviour at a cluster level. It allows to have a more precise overview in a specific hub. Finally, the last one is focused on a vehicle itself. In that cases, it compares the state of the vehicle to a benchmark relative to this particular vehicle. It also predicts how the State of Health of the vehicle will evolve in the future years if no changes are made to its overall use.

This visualization has been created using a JavaScript library called *Chart.js* [16]. It allowed me to build smooth charts with less burden than raw HTML and CSS, with the data fetched from the back-end.

Figure 2.12: Representation of State of Health for entire fleet - cluster - vehicle

Map representation

One of the requirements we got from our UX researches was to have the possibility for the fleet manager to overview the vehicles at different levels and very easily.

For this reason, we created a visual map representation of the vehicles, with a view of all the clusters, a view of a specific cluster and a single-vehicle view. Figure 2.13 shows how the map looks like at the different levels.

This interactive representation has been made possible through the use of the *Google Maps Platform* [17] which allowed us to use our data in the back-end to create the clusters and position the vehicles on the map.

Figure 2.13: Map representation of entire fleet - cluster - vehicle

2.3.5 Production environment

Since the prototype was aimed to be used in a different location from the one in which it had been developed, we decided that it would be more practical to actually launch the app into production to allow our team and clients to access it from anywhere. For this reason, we hosted the *Panasonic - Cy* platform on an AWS EC2 t2.micro instance.

The AWS EC2 provides us with the ability to easily scale the app if necessary. We used a t2.micro instance as it is the smallest one and the amount of storage needed to host our database, server and front-end was rather low.

2.4 Outcomes

The *Panasonic - Cy* prototype's development has been finished at the end of our three-weeks sprint. Figure 2.14 shows what the final dashboard of the prototype looks like. Note that the dashboard is fully functional and interactive at the general, cluster and vehicle level. All data used in the prototype are generated thanks to the mathematical models and hypothesis described in the previous sections. You can also compare the end product to the initial design that was proposed in Figure 2.6.

As the only developer on this project, it has been a real challenge for me to entirely build the back-end, front-end, and put the application into production in such a short amount of time. It allowed me to increase my development skills a lot, but also to learn about Design Thinking and work closely with UX/UI designers to determine all the characteristics the final product should have. Being part of the team as the software expert was also a way for me to raise compromises between what the team wanted and what was feasible from a developer standpoint.

This prototype has finally been shared with the whole team and been showcased to our client Panasonic along with all the business models and transformations that have been developed in parallel by the innovation strategy team.

Figure 2.14: Final dashboard of the Panasonic - Cy prototype

Chapter 3

Optical Character Recognition for tire stickers

To propose innovation to our clients, one of our main activities at the Applied Innovation Exchange was to build prototypes and *Proofs-Of-Concepts* (POCs).

Since now, we have been referring to those two terms indifferently. In reality, they are slightly different and are used in specific cases. A prototype (as we have seen in Section 2) is built to make decisions about the product's development. It is usually a way to test the design, usability and functionalities of the whole product we are aiming to create. A Proof Of Concept, on the contrary, is usually focusing on a specific idea or feature with the objective to test it feasibility and be able to assess whether it could be added to an existing product or project.

The project we will describe now has been built as a POC. The demand came from a team in *Fahrenheit 212*, part of Capgemini, working with some retailers selling vehicles' parts. It has required less time than the other projects that will be described in this document as it has not been needed to provide the same level of detail both in the ideation phase and in the design one.

3.1 Project Context

In the past years, automotive manufacturers, mechanics and retailers of vehicles' parts have been willing to make things easier for their clients. Most of the drivers do not have a deep knowledge or understanding of their vehicle and its different parts. It is then crucial to make things easy for them when it comes to controlling or changing some elements in their vehicle.

One of the solutions to tackle this problem is *predictive maintenance*. Internet Of Things solutions are used more often in recent vehicles to assess the behaviour and functioning of different parts of the vehicle, and directly alert the user in

case something is likely to happen. However, predictive maintenance in itself doesn't help the user with knowing what new part should be bought or if one is fundamentally better for their vehicle than another.

A good example of this is the tire industry. It is not always easy for people to understand the stickers on their car and know what tires to buy. One leader in this area, *SimpleTire* [18], is willing to offer flawless solutions for its customer, to find the right tire for their vehicle very easily.

For this reason, *Fahrenheit 212* and the AIE have been partnering to develop a small POC for SimpleTire to help its user scan tire stickers from their vehicles with a simple app, and be directly redirected to their website on the corresponding page featuring the tire they need.

3.2 Technical Development

As we mentioned earlier, this project being smaller, there was no need for an assessment of the customer or the elaboration of a design. We have then directly been working on the development of the POC. I have also been the only developer on this application, working with the team to analyze how to showcase this feature to our client.

Since this product was aimed to be used by SimpleTire's customers in any place and any given moment, we decided to only build it under the form of a phone application to demonstrate the feature. Specifically, we chose Android for the simplicity of development and compatibility reasons.

Most of the technical development has been linked to the elaboration of the OCR capabilities to recognize tire stickers and be able to properly read the text on them. The very first POC didn't have the possibility to understand the sticker in real-time. It required to take a picture first, crop it, and then only was able to properly understand the information from the sticker. While this was working properly, it was not user-friendly enough. We then decided to focus our efforts in making a real-time reader that would recognize the sticker directly while taking a video.

The following subsections will explain the strategy used to recognize multiple tire stickers with very different layouts and get relevant results.

3.2.1 Architecture

Given that the POC is aimed to be developed quickly and does not require to scale up nor be integrated to other elements, we have chosen to use a relatively simple architecture with an Android application communicating with a Firebase back-end to store the information on clients' searches and have a system of research history.

Figure 3.1 gives an overview of the interactions between the application and Firebase, facilitated by the Firebase SDK.

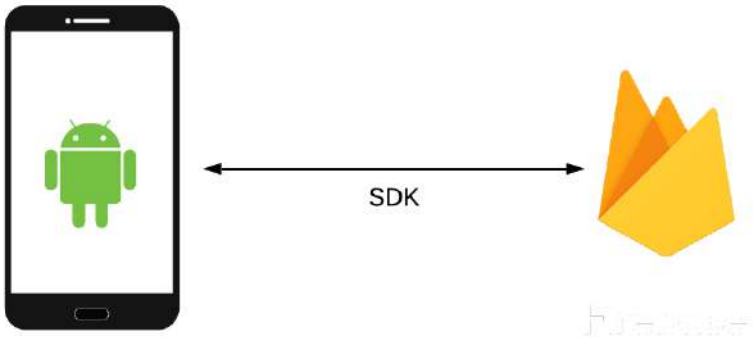

Figure 3.1: Architecture diagram of the OCR application

Android and Firebase

Firebase is a mobile and web development platform. It is used to provide cloud services very easily, without the need to develop a server. It works like an API accessed through the front-end or mobile app, which allows multiple interactions such as authentications, analytics, storage, etc. [19]

In our case, it gives us server-side capabilities, flexibility, and is relatively easy to implement, which makes us gain a lot of time. Most importantly, Firebase provides us with the Machine Learning capabilities required for our OCR on tire stickers.

3.2.2 OCR on tire stickers

Our main objective for this application is to be able to recognize the relevant information from a tire sticker, in real-time, in order to determine what are the characteristics of the tire and suggest relevant products from the SimpleTire's website.

Optical Character Recognition being a famous problem in Computer Vision, it is not very difficult to find algorithms allowing us to extract the text from the image. However, the real difficulty comes from the great diversity of stickers and the patterns they have. By looking at Figure 3.2 we can see that information are ordered in different ways depending on the manufacturers, which makes it harder to have an algorithm working for the majority of stickers.

In the following section, we'll see how we used Machine Learning to extract the relevant information from those stickers, even with different formats.

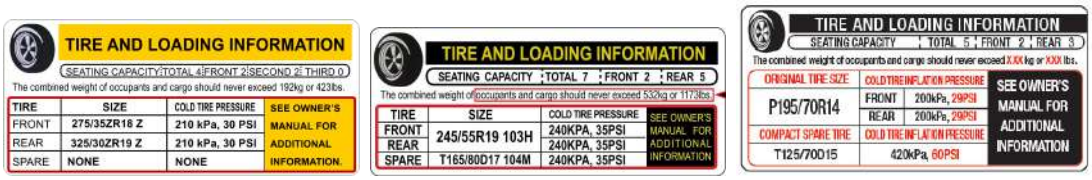

Figure 3.2: Example of tire stickers to analyze

Machine Learning Pipeline

To be able to identify patterns correctly among different type of stickers, I have decided to use a solution based on Machine Learning. The problem being quite complex and relying on existing sets of solutions (Optical Character Recognition for example), it was more efficient to use a Machine Learning Pipeline to achieve the end result.

In Machine Learning, a pipeline is a series of steps chained together for the model to execute and produce results successfully. In our case, the main problem relies on finding the correct information from the sticker and reading them to understand the parameters of the tire.

Figure 3.3 gives an overview of the Machine Learning Pipeline used for this project.

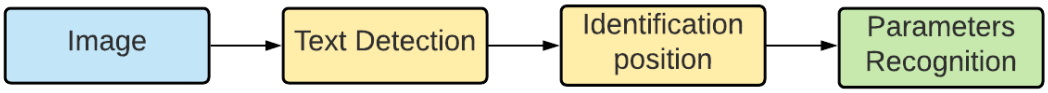

Figure 3.3: Sticker Recognition Pipeline

The first part consists in detecting the areas where text is present on the image. While we have identified all text areas, we run a quick search among them to see which ones contain the words *'Front'* and *'Rear'* as these are the only parameters that are interesting for us to characterize the tires. Once done, we need to find the parameter characterizing both the tires. These parameters can be in different zones, so using Machine Learning to recognize patterns help us know where to read them. Finally, when we have extracted the string characterizing the parameters for the front and rear tires, we simply need to infer the width, aspect ratio, and diameter of the tire out of them to be able to identify the exact model of the tires.

Text Detection

It is necessary to evaluate where the areas of text are to be able to identify which ones contain the elements we are looking for and their position. For this purpose, we use the Machine Learning kit from Firebase, which allows us to very easily implement pre-built methods and run them on our camera input.

```
for (FirebaseVisionDocumentText.Block block: result.getBlocks()) {
    String blockText = block.getText();
    Float blockConfidence = block.getConfidence();
    List<RecognizedLanguage> blockRecognizedLanguages = block.
    getRecognizedLanguages();
    Rect blockFrame = block.getBoundingBox();
}
```

With this step, we can have knowledge of all our block of texts, which can be provided as an input for the next part of the pipeline.

Identification of information position

Now that we have identified the different blocks of text in our image, we need to find the ones characterizing the front and rear tire. Indeed, these are the elements that we need to access SimpleTire's website and provide the customer with the tires he needs.

To do so, we so a simple research among our pre-identified blocks.

```
for (FirebaseVisionDocumentText.Paragraph paragraph: block.
    getParagraphs()) {
        String paragraphText = paragraph.getText();
        if (paragraphText.toLowerCase().includes('front')){
            Rect frontTireFrame = paragraph.getBoundingBox();
        }
       if (paragraphText.toLowerCase().includes('rear')){
            Rect rearTireFrame = paragraph.getBoundingBox();
        }
}
```

Once done, the position of the front and rear tire text boxes is fed into a multivariate linear regression Machine Learning algorithm, together with the position of the other text boxes identified. The model has been pre-trained with labeled data of stickers and allows to easily identify which boxes contain the parameters of the front and rear tires based on the location of the text on the image.

At the end of this step, we have identified which are the text areas containing the parameters of the front and rear tires. We can then feed the text into our parameter recognition algorithm to get the relevant parameters we need for the website.

Parameters recognition

The three parameters we need from each tire to properly identify them are:

- Width
- Aspect Ratio
- Diameter

As you can see in Figure 3.2, the information on tires are under the form:

$$PXXX/YYYRZZZ \tag{3.1}$$

In this example, XXX represents the Width, YYY the Aspect Ratio, and ZZZ the Diameter. However, the format of the information can change from one sticker to another. It is therefore important to do some post-processing on the string of characters once the text has been identified, to ensure the correct estimation of the parameters. The following piece of code shows an example of the post-processing executed on the rear tire string to extract the relevant parameters

```
if (rearString!=null) {
// Getting the Width
    if (rearString.charAt(0) == 'P') {
        rearWidth = rearString.substring(1, 4);
    } else {
        rearWidth = rearString.substring(0, 3);
    }

// Getting the Aspect Ratio
    rearRatioIndex = rearString.indexOf("/");
    rearRatio = rearString.substring(rearRatioIndex + 1,
   rearRatioIndex + 3);

// Getting the Diameter
    rearRimIndex = rearString.indexOf("R");
    rearRim = rearString.substring(rearRimIndex + 1, rearRimIndex +
   3);
}
```

3.2.3 Mapping with website

Once the parameters have been extracted following the method described previously, getting results from the website was a very easy task.

SimpleTire's website uses URL parameters for its queries. We then only needed to use the previously analyzed parameters in the query:

```
String url = "https://simpletire.com/catalog?width=" + frontWidth + "
    &ratio=" + frontRatio + "&rim=" + frontRim + "&rwidth=" +
    rearWidth + "&rratio=" + rearRatio + "&rrim=" + rearRim + "&
    maincategory=AUTOMOBILE";
```

3.3 Outcomes

The development of this POC has taken less time than the other projects described in this document. Indeed, the only focus has been the technical side of the application which has been easier thanks to the different Machine Learning functionalities available in Firebase.

Figure 3.4 shows images from the final application while detecting the tires' characteristics on a printed sticker. Note that it also works with colored stickers.

Given my deep interest for Machine Learning, this project has been extremely interesting as I had to put my theoretical knowledge into practice. It has also been an occasion to reinforce my knowledge in Android development.

The final POC was presented to the *Fahrenheit 212* team who shared it with SimpleTire. Further developments are also possible on top of this solution: Some sticker formats are not supported by the app, it could be interesting to improve the recognition algorithm to make it compatible with any type of sticker (this requires a benchmark of a greater number of sticker types). Other features could also be added, or the app could be integrated in a bigger project.

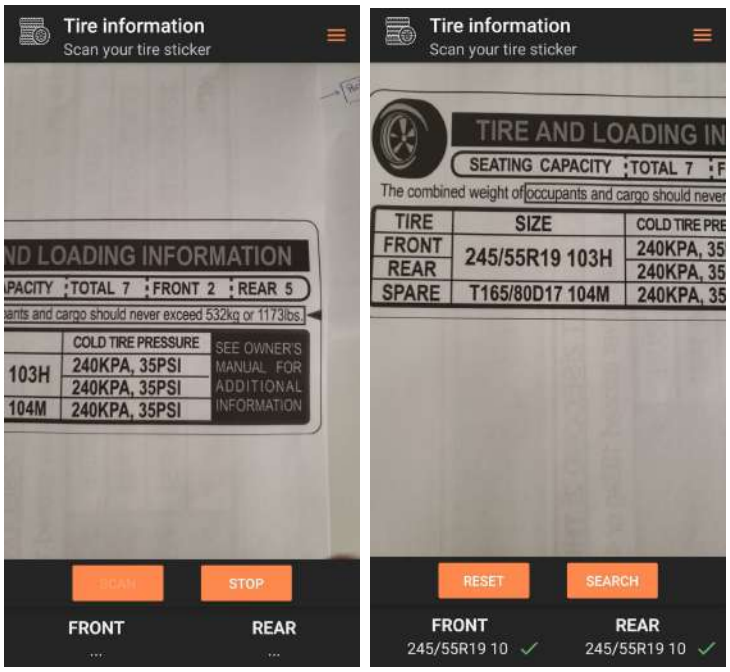

Figure 3.4: Screenshots from the tire sticker recognizer application

Chapter 4

National Retail Federation - Expo Guide

The Capgemini's Applied Innovation Exchange is an enabler of innovation. We have seen through the two previous projects that this can imply building prototypes, proofs-of-concepts, or minimum valuable products for our clients. However, assisting companies in their innovation journey is not limited to creating new products or services with them. It is also connecting them to a new ecosystem that can enhance their opportunities for innovating.

One of the strengths of the AIE is its ability to connect industry leaders together with startups and partners to broaden their ecosystem and foster creativity. With a strong knowledge of emerging technologies and companies in a plethora of industries, the AIE is a valuable bridge between what businesses know and what they could discover.

To create such network, it is necessary to actively participate and guide our clients through major events, as the *National Retail Federation Retail's Big Show* [20], the biggest event gathering tech companies, startups, and retailers, to imagine the future of retail together. It can, however, be difficult for companies to know what to look at given the high number of exhibitors in the event.

The goal of the *NRF - Capgemini Expo Guide* was then to simplify our clients' lives, by allowing them to discover the emerging vendors and solutions on the expo floor, based on their interests. The following section will explain in details how the project was elaborated from the UX/UI to the development and production. After working closely with the team to define the requirements, I came up with the first versions of the UX and design by myself. I have then been the only developer to create the application both for Android and IOS, and have worked on this project for about a month.

4.1 Project Context

The National Retail Federation is an association of retailers that aims to support and provide resources to retail leaders. Each year, the NRF hosts the *Retail's Big Show*, that regroups thousands of retailers and exhibitors to present them with the opportunities for the future of retail.

Capgemini has a high number of retail clients. In order to help them discover new horizons for their businesses, Capgemini organizes tours at the NRF to showcase exhibitors that have relevant solutions for them. Creating those tours, however, is time-consuming and requires someone dedicated to the client. This also limits the potential impact of Capgemini given that no other company than their already-existing clients can benefit from these tours.

To tackle this issue and create more client engagement, the Applied Innovation Exchange decided to create the *Capgemini Expo Guide* that would not only help our clients but also other retailers find the most relevant exhibitors for their business in a few clicks.

4.1.1 Functional requirements

The application is meant to allow retailers to easily navigate the NRF expo, by finding the exhibitors they need. Of course, the goal is also for Capgemini to raise awareness about their capacity to help retailers be more future-proof. Hereafter are listed the main features that the app should contain to guarantee a proper experience.

- Listing: a list of all the relevant companies previously curated by our ecosystem team.
- Topics: the possibility to explore companies related to a specific topic to ensure that retailers can find solutions for their main needs.
- Directions: the NRF is big and with thousands of exhibitors. It is necessary to make sure companies are easily reachable.
- Information on companies: a minimal set of information such as a description, the reason why we chose it, the contacts, etc.
- Search: the possibility to search for a specific company directly by name or by keyword.
- Favorites: a function allowing to save a specific company as favorite to easily retrieve it during or after the event.

- Contact form: to drive engagement to Capgemini directly through the application.

- In-app browser: to open relevant websites through the application and ensure the user stays as long as possible.

These functional requirements are used as general guidelines to know what to include in the app. The exact way with which they will be implemented is defined into more details during the UI/UX design phase.

It is important to note that the reflection process has not been explained in details in this section, as it is very similar to the one described in Section 2. We have also been using the Design Thinking methodology for this project.

4.2 UX/UI Design

Contrary to the other projects for which the User Interface was secondary, the *Capgemini Expo Guide* required a much deeper analyzis and wireframing as the objective was to make the application available to the public. For this reason, we followed a process very similar to the one described in Section 2 but with more iterations.

4.2.1 Sketching

The skething step has taken multiple phases, starting with paper sketches, like the one in Figure 4.1 but also digital ones, until converging towards an idea that satisfied most of us.

In this very first version, the idea was to guide the user through different tours based on his initial interests. He could firstly choose between *Retail* and *Consumer Goods Products* and would then have a set of categories to pick from in order to curate the list of corresponding companies. Out of this list, the user would then be able to click on the ones he had an interest for and find all the relevant information about the company.

We will see later that this flow has not been totally retained for the final solution because it was not the one providing the most optimal User Experience.

4.2.2 Wireframing

After validating the general elements we wanted the app to include thanks to the sketches, I built the first wireframe to have a better idea of the look and feel of the application.

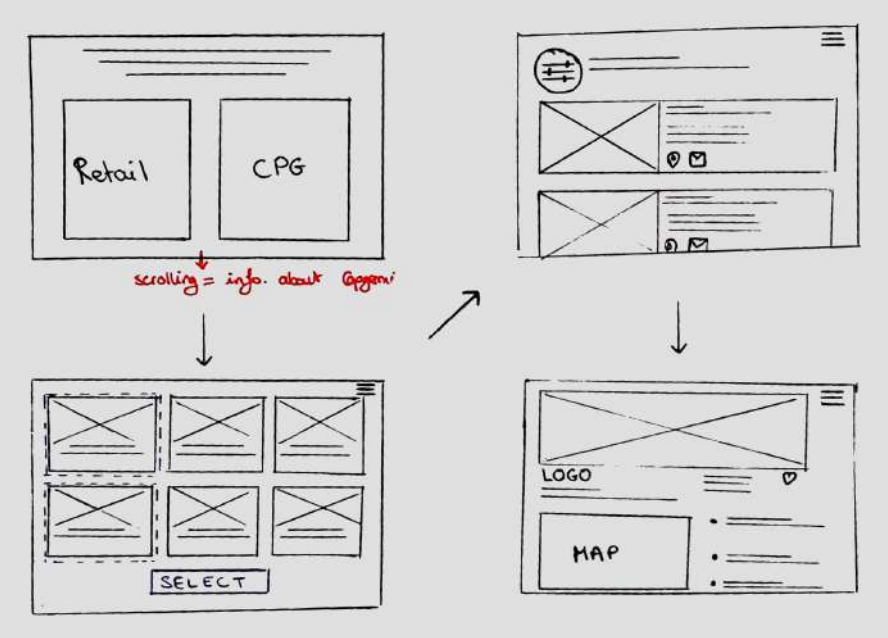

Figure 4.1: First sketch of Capgemini Expo Guide

For this project, we decided to use an interactive wireframe in order to test the flow with the team and ensure the User Experience was smooth. Interactive wireframes allow to have clickable elements and transitions within the wireframe, to transition to one screen to another smoothly, giving the feeling of using the real product.

Figure 4.1 shows some of the screens realized for the first wireframe of the *Capgemini Expo Guide*. Note that there have been several versions made with different colors and design in order to determine which one fit best our use case. This wireframe has been built using *InVision* [21].

InVision

InVision is used by designers and developers for prototyping. It allows to easily create interactive mock-ups and designs that can then be shared with the team. In addition, InVision contains a lot of useful features such as the possibility to create responsive designs and a library of pre-built elements that make it much faster to build a prototype and receive feedback from the team.

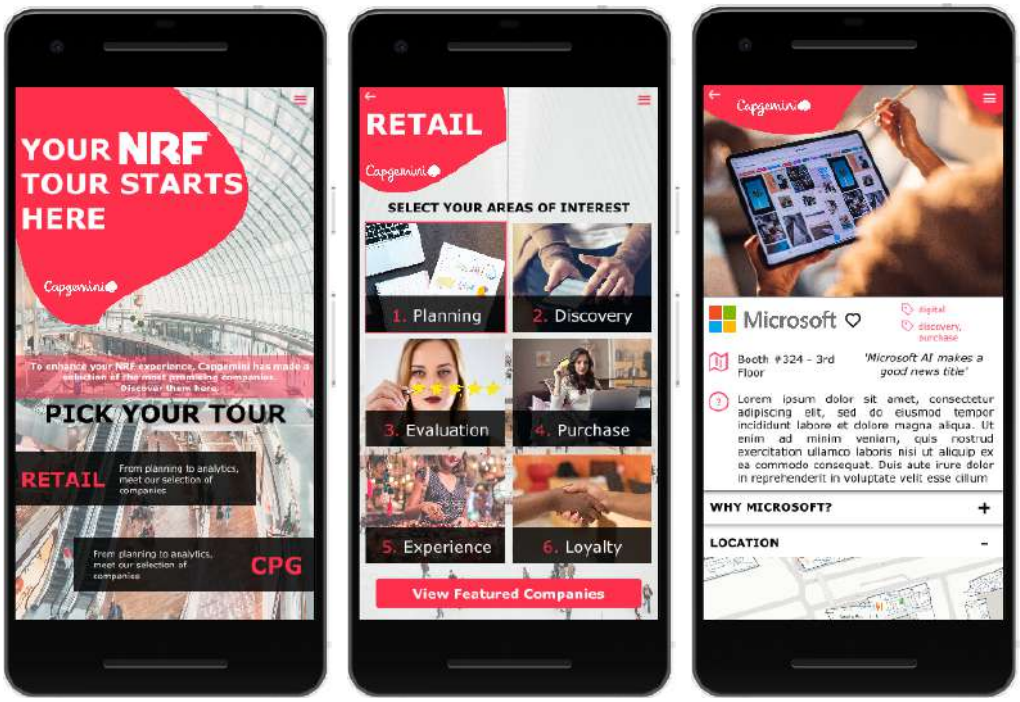

Figure 4.2: Parts of the wireframe of Capgemini Expo Guide

4.2.3 Refinements and Visual Design

The previous version of the wireframe initiated the development work of the application. This phase will be detailed in the next subsection. When the very first functional version of the app was functional, we put it to the test by providing it to some members of our team and collecting their feedback.

One element was pointed out: the flow in the app was sometimes unnatural or too slow. Indeed, the current refinement of companies works with a system a funnel. We are trying to explore the user's needs and interests to showcase the most relevant companies for him. However, it can happen that a user is interested in two totally different areas, or would benefit from some suggestions he may not have considered valuable for his business. Finally, the number of clicks to reach the desired list of companies was too big, which made the User Experience not optimal.

With this in mind, we decided to review the flow of the application in order to make it more intuitive and easy to use. To do so, we worked in collaboration with *Fahrenheit 212*, to gather ideas and feedback from design experts. After few iterations, we came up with a system of filters that would allow the user to curate the list of companies directly from the home screen and in a much more intuitive way. It would also make it easier to remove or add a filter and end up with different companies.

Figure 4.3 shows what the final visual design looked like for some elements of the application. This representation has been realized with *Zeplin* [10].

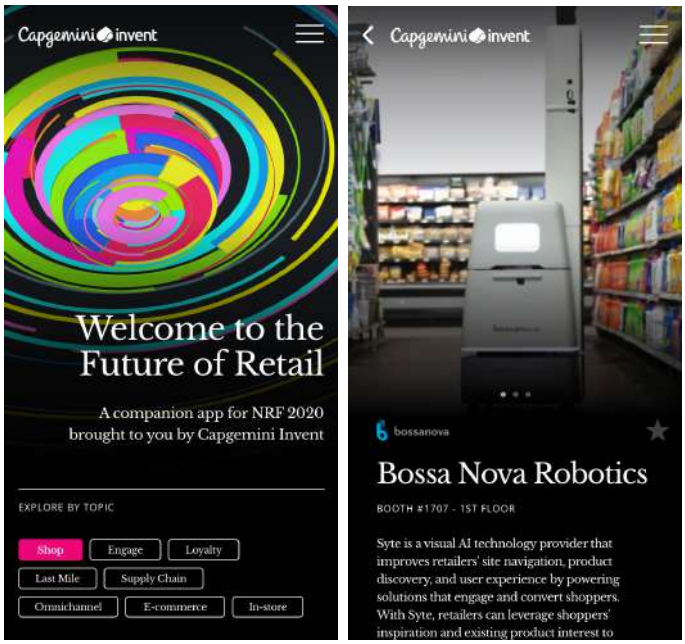

Figure 4.3: Parts of Capgemini Expo Guide's final design

Zeplin

Zeplin is a very popular tool used both by designers and front-end developers. Its multiple functionalities make it perfect for translating designs into code. It allows developers to directly access snippets of code out of the design imported by designers, as well as graphical elements, fonts and colors. In our case, this has been a useful tool to ensure the end application had the exact design which was provided by the designers.

4.3 Technical Development

The development phase of the *Capgemini Expo Guide* started with the first iterations of wireframing. Although the time constraint was not as sharp as for *Panasonic - Cy* (see Section 2), the app had to be ready before the event to make the necessary tests and ensure everything would run smoothly in production. For this reason, we worked again in an Agile fashion, making the development evolve along with the design and the features.

Contrary to the other projects so far, this application was required to be cross-platform. In order to make sure to reach as many people as possible, we had to make it available both on Android and IOS, and as a web-app. Given the complexity to handle the development on different platforms for a single developer, we explored possibilities to directly develop the application using a cross-platform development environment, to ensure compatibility with different devices and OS.

After exploring several possibilities, we decided to move forward with *Ionic*: an open-source SDK allowing to build cross-platform applications easily. To be more precise, the solution stack was composed of Ionic and Angular on the front-end, and Firebase on the back-end.

Such configuration allowed me to have a lot of flexibility as I could develop in JavaScript and have the application running on IOS and Android too. It also allowed me to have a serverless application, with cloud-functions running on Firebase. This made the development much faster and scalable to a wide number of potential users.

The following subsections will explain the development phase more in detail, as well as the technologies used both for the front and back-end.

4.3.1 Global architecture

To have an application available on multiple platforms in a short amount of time, we decided to use a cross-platform SDK for mobile development: Ionic. This SDK can be coupled with a front-end framework (Angular in our case) to allow to build the app more efficiently. From the developer's perspective, it is almost as if the product was developed as a single web-app.

We also decided to use Firebase in the back-end to avoid the development of our own server. This has been very useful as both the front-end and the Firebase cloud-functions could be developed in JavaScript, therefore increasing the development speed.

While Ionic allows to build content for mobile apps from a single code-base, it is not capable of compiling it to native devices by itself. For this reason, we are using Apache Cordova which is capable of rendering web content within native applications.

4.3.2 Back-End

Contrary to the other projects described in this document, the back-end of *Capgemini Expo Guide* isn't very complex. Its principal purpose is to store all the information for the companies listed in the app, as well as to provide statistical information on the app usage. It also contains the filtering algorithm running to optimize searches within the app, and the suggestion algorithm which provides

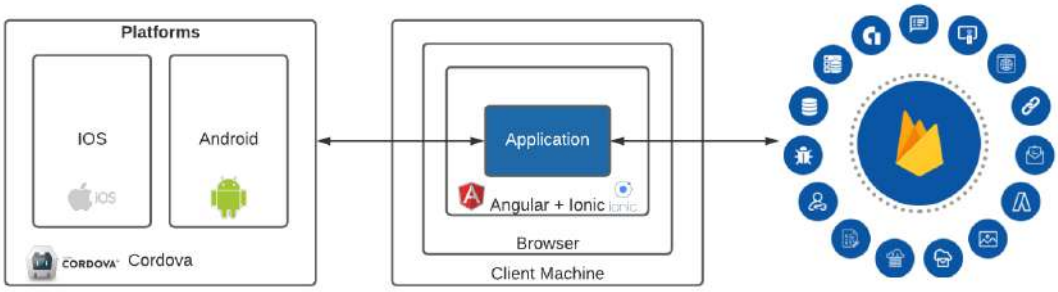

Figure 4.4: Architecture diagram of Capgemini Expo Guide

a list of relevant companies related to the one the user is looking at. Finally, it allows users to directly interact with the Capgemini team through a contact form that contains auto-mailing functionalities.

Database

The database we are using for this project is the Firebase *Cloud Firestore*. It is a NoSQL cloud database which allows real-time data synchronization with the front-end and scales very well. In our case, it is very useful as we can query the database directly from the front-end and may input larger numbers of companies in the future.

The elements which need to be stored in the database are the companies and the filters we want to use to allow our users to curate their research. Our models for *Capgemini Expo Guide* are straight forward: we are using one model for the filters and one for the companies. The code underneath gives an idea of what the model for companies looks like.

```
class Company {
    id?: string;
    name: string;
    category: string;
    tags: string[];
    booth: number;
    news_link: string;
    news_title: string;
    logo: string;
    why: string;
    headquarters: string;
    view_counter: number;
    like_counter: number;
    floor: number;
```

```
    mail: string;
    website: string;
    description: string;
    map_image: string;
    images: string[];
}
```

The advantage of using Firebase for such a project is that it contains an intuitive Graphical User Interface which allows content managers to very easily access and update data. That way, I could work on the database with members of the Team who didn't have a developer background, giving them the possibility to directly add information about a specific company through Firebase.

Cloud Functions

Given that we have not been developing a server for this project, we have used Firebase *Cloud Functions* to run back-end code in response to some particular events [22].

Cloud Functions are pieces of code stored on the cloud that can be executed to mimic server-side code in the application. They are triggered from client-side or database events and allow for an easy scale-up without the burden of creating a server.

In *Capgemini Expo Guide*, we have used Cloud Functions for two main purposes: managing filtering and suggestion algorithms (more about it in the next subsection), and creating auto-mailing capabilities for our in-app contact form.

To develop the auto-mailer, we have exploited the Firebase *Realtime Database* which allows to store data in synchronization with the client in real time. Each time a user submits the contact form, it is added to the database with the following code executed inside a Cloud Function:

```
exports.addMessage = functions.https.onRequest(async (req, res) => {
    // Grab the text parameter.
    const original = req.query.text;
    // Push the new message into the Realtime Database using the
    Firebase Admin SDK.
    const snapshot = await admin.database().ref('/messages').push({
    original: original});
});
```

Another Cloud Function is then triggered by the new entry in the Realtime Database and uses the data from this entry to send the email, thanks to a Node.js module called *Nodemailer* [23].

```
exports.sendContactMessage = functions.database.ref('/messages/{
    pushKey}').onWrite(event => {
  // Only send email for new messages.
    if (event.before === event.after) {
      return;
    }
    const val = event.after._data;
    const mailOptions = {
      to: 'xxxx', // Email address of the recipient
      subject: `[NRF2020][App] New Message from ${val.name} - ${val.
    organization}`,
      html: val.html
    };
    return mailTransport.sendMail(mailOptions).then(() => {
      return console.log('Mail sent to', mailOptions.to)
    });
});
```

Filtering & suggestion algorithms

Although the filtering algorithm seems trivial, its implementation is a crucial part of the application as it can totally change the User Experience and the purpose of the app.

Companies can be identified by multiple tags that are then used for the filtering. These tags can be location tags, such as the booth number or the floor, but also category tags like *'e-commerce'* or *'supply chain'*. How the algorithm reacts to the addition of multiple tags has a big impact on the type of results one may get.

We have first been thinking to use filters as mutually exclusive, so that, with two filters for example:

$$results = Filter1\ XOR\ Filter2 \tag{4.1}$$

This way, the user is refining results with each filter he is adding. It can, however, be misleading and the research often end up with no result if the two filters have low correlation. For this reason, we aimed to have a mutually inclusive filtering system, under the form:

$$results = Filter1\ OR\ Filter2 \tag{4.2}$$

Although providing more relevant results in most cases, we quickly realised that this filtering system wasn't optimal neither. Indeed, a user filtering for a specific floor could end up with results for another floor if a category filter containing

companies on another floor was selected. We then finally decided to use the following strategy:

$$results = Location\_Filter\ XOR\ (Category\_Filter1\ OR\ Category\_Filter2) \tag{4.3}$$

Where *Location_Filter* represents a filter on the booth or floor number and *Category_Filter* a filter on one of the categories. Note that we used only two filters in the examples above but the user can use any amount of filters in reality. Finally, in order to return the list of companies in the most logical order for the user, we used the following sorting algorithm:

```
// Sorting the returned list:
// First by amount of tags matching the filters, then by floor and
    alphabetical order
this.filteredCompanies.sort((a, b) => {
    if (a.nbActivateTags === b.nbActivateTags) {
        if (a.floor === b.floor) {
            return  a.company > b.company ? 1 : -1;
        }
        return (a.floor > b.floor ) ? 1 : -1;
    }
    return (a.nbActivateTags > b.nbActivateTags ) ? -1 : 1;
});
```

When it comes to the suggestion of other companies based on the one the user is looking at, we have been using a simple matching algorithm. The algorithm compares the feature vector of the current company with the one of other companies in our database. We start by taking 10 companies at random and look at how the matching performs. If feature vectors are similar up to a certain percentage, the company is added to the list of suggested companies. We repeat this process until we have at least 5 companies to suggest to the user.

4.3.3 Front-End

The front-end of *Capgemini Expo Guide* was built using the HTML framework Ionic on top of the web framework Angular 8. It consists of a Homepage from which the user can pick its filters and see the main list of curated companies, a Company page which gives all the detailed information on a company clicked by the user, a Favourite page listing all companies that have been added to favourite by the user, and a Contact page from which the user can contact the Capgemini team through a dedicated form.

Architecture

The architecture of the front-end is a bit more complex than for *Panasonic - Cy* since way more components are reused throughout the application.

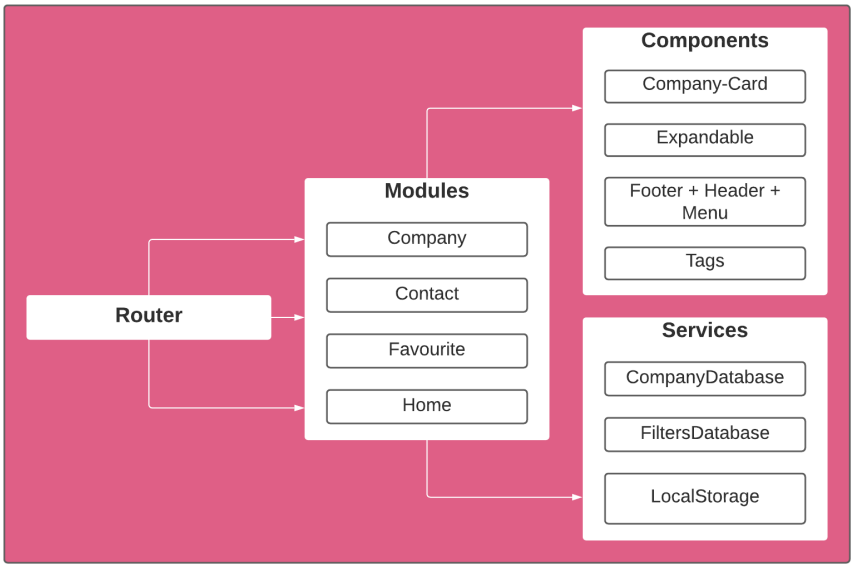

Figure 4.5: Architecture of the Capgemini Expo Guide Front-End

As for *Panasonic - Cy*, we are using the @angular/router package to manage the navigation between our different modules easily.

The user first accesses the Homepage module. This module displays the name of the application, the list of all the filters and the list companies corresponding to the selected filters (initially, the list contains all the companies). It then adjusts the companies' list in real-time based on the filters the user selects or deselects. It relies on all the Components and Services. Once a user clicks on a specific company, he is redirected to the Company module.

The Company module displays all the information about the company, taken from the database. It also features a list of suggested companies in relation to the one that was picked by the user. From this page, the user also has the possibility to add a company as favourite.

From the main menu, the user has several possibilities. He can first search for a company directly within the menu (thanks to a search-bar), which displays the results of the search directly in the menu, to make companies accessible from anywhere in the app. The user also has the possibility to access additional information about Capgemini and the AIE, and to go to his favourite list or to the contact form.

The Favourite module acts like a simplified version of the Homepage. It only displays the list of companies that were added as favourite without giving the possibility to search with filters.

Finally, the Contact module contains an interactive contact form which gives real-time feedback on the input (ex: if the email address is incorrect). It then triggers the cloud functions to send an automatic email once filled.

Ionic

As mentioned earlier, we are using the Ionic Framework for this project. Ionic is an HTML mobile application development framework that allows to build hybrid mobile apps [24].

Hybrid apps are very useful as they allow to have a single code-base for multiple platforms. However, performances are usually lower as the OS usage is not as optimized as for native apps. This is where Ionic comes into play, taking advantage of the native functionalities for each device while staying developing on top of powerful front-end frameworks (in our case, Angular 8).

Ionic runs on top of Apache Cordova [25] which wraps the HTML and JavaScript code into containers able to run on several platforms. It contains a lot of pre-built elements allowing to build UI components adapting to the platform for which they are built.

4.3.4 Builds

This application was meant to be used by clients on any platform during the National Retail Federation. For this reason, it had to be built mainly for mobile devices, but also needed to be accessible directly as a web app.

Web

We used *Firebase Hosting* to deploy our application as a web app, allowing us to very easily scale it to any number of users. Firebase Hosting also serves the app to a global Content Delivery Network (CDN). The CDN optimizes the server's response time based on the user's location. While this might not be necessary for the initial purpose of the app (all users being at the location of the NRF), it is very useful to ensure that users can still access their content in a very efficient manner after the event.

Native

To make our application available in the *Google Play Store*, we simply built it as an Android app thanks to Ionic and used the *apk*. The same process applied for

the *Apple Store*, for which we used *XCode* to generate the necessary packages and upload the application.

4.4 Outcomes

The development of *Capgemini Expo Guide* has taken around a month and a half. It has been made available on Google Play Store, Apple Store, and as a Web Application. Figure 4.6 shows the final application for IOS (note that the result is the same for Android and for the Web).

This project has been the occasion for me to work on cross-mobile development and learn about the underlying technologies. It was also my first application developed in a company that was made totally public. This has been a challenge given that the app had to be both pleasant visually and with an efficient UX. I also had to do a lot of A/B testing with the Team to ensure that all edge-cases were taken into account in the application.

The final version was published a few days before the NRF during which the app has been heavily promoted. It has been used both internally, for our account managers to have a better knowledge and overview of their tours with our clients, and externally, for external retailers to discover the ecosystem of companies curated by Capgemini.

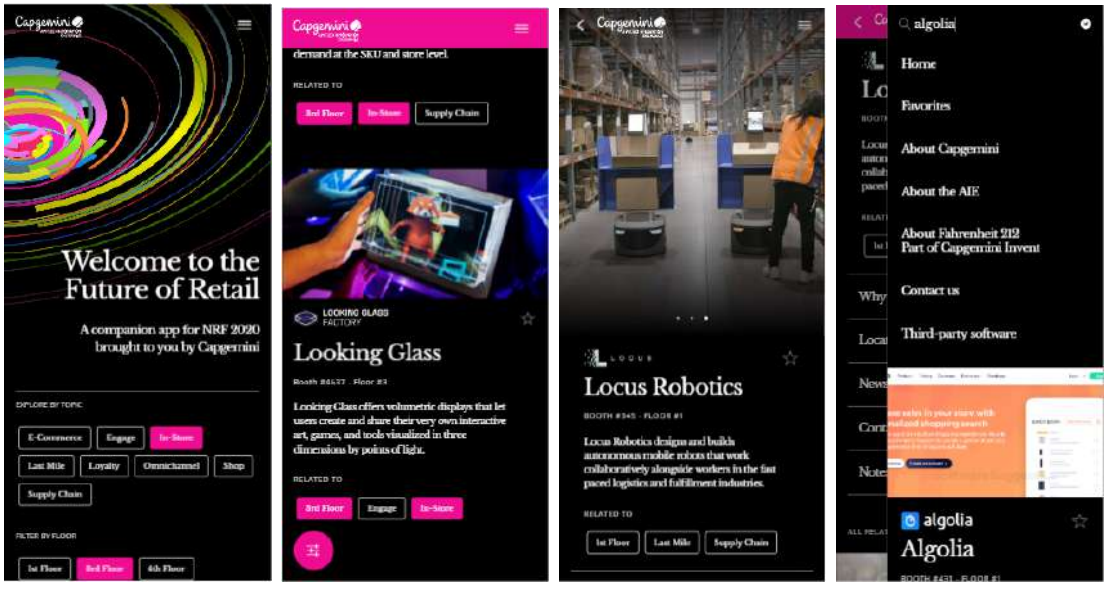

Figure 4.6: Screenshots from Capgemini Expo Guide on IOS

The outcomes of the applications were analyzed after the event. An app with the same purpose had been developed by the team a year earlier but ended up

with a very low number of users and not very positive feedback. On the contrary, this year's app has been more widely adopted and was very useful for the event itself. Figure 4.7 shows a slide presenting some of the results of the app.

Figure 4.7: Statistics on usage of Capgemini Expo Guide

After the event, the Team has been brainstorming on how the app could be scaled up to be more beneficial on the long-run. We came up with the idea to have an application presenting the most valuable startups, from Capgemini's perspective, in multiple areas. This app would be available to everyone and updates all year long, with special features for our clients.

Chapter 5

Applied Innovation Enterprise Readiness Assessment

Capgemini's Applied Innovation Exchange is not only designed to enable enterprises to discover relevant innovation, devise minimum valuable products, and deploy rapidly; one of the main pillars of its framework is also to sustain the innovation.

Rather than innovating, businesses need to create a culture of innovation to foster the emergence of new ideas from their teams and their ecosystem. This implies several changes in the governance, management, and structure of the company.

The AIE aims to accompany its clients all along their innovation journey, analyzing their potential to create innovation and helping them with shaping the changes needed to sustain it. Several discussions are usually needed to deeply understand the capabilities and needs of a client.

Capgemini's *Applied Innovation Enterprise Readiness Assessment* is here to allow Capgemini's clients, but also other businesses, to assess where their enterprise stands in their innovation journey and their maturity compared to the industry. Its main goal is to easily collect different points of view from the company, and analyze them to give a direction to the client. Gathering data from different businesses in different areas and industries also gives the potential to create a benchmark to give even greater insights in the future.

5.1 Project Context

To truly understand the potential of an enterprise to apply and sustain innovation, several bodies need to be assessed within the organization. Indeed, there may be a

discontinuity between what the leadership envisions and what other positions in the company experience. It is, however, not always easy or feasible to have these discussions with multiple bodies.

For this reason, Capgemini decided to build the *Applied Innovation Enterprise Readiness Assessment*, to provide an easy way to collect the necessary information to truly understand a company from the inside.

To achieve this, the leadership of Capgemini, as well as experts on innovation, have come up with a list of enablers which determine the readiness for innovation of a business. For each of these enablers, a series of statements allow the client to reflect upon their organization and determine how they are performing for this specific enabler. The analysis of the results for all the enablers then gives the possibility to assess the enterprise's proficiency for innovation.

I got the chance to be the technical lead of this project, working directly with the Chief Innovation Officer of Capgemini Group and the Global Innovation Program Manager to determine the functionalities and UX/UI of the tool. I have then been working on the development of the platform and participated to the internal launch with the leaders of all Applied Innovation Exchanges to present its technicalities.

5.1.1 Principle

The objective of the application is to assess the readiness of a business to sustain innovation. To do so, the user is assessed on different *Enablers* of the innovation.

Our model features 10 enablers of innovation. For example, *Processes*, *Technology*, or *Ecosystem*. To evaluate each of these enablers, the user goes through a list of statements related to the enabler and answers yes or no based on whether the statement applies to his organization or not.

Example of statement: *We have a process to allocate resources and assign decision-making authority for innovation initiatives*

The user can then provide a grade for each enabler on its current capabilities to execute it, thanks to a guided question. A score is then calculated based on the answers to the statements and the grade for each enabler, to calculate the *Priotity* and *Maturity* of the company in different areas of innovation.

Depending on the results, some suggestions can be made to the client on how to improve its innovation capabilities or what to focus on. Note that this assessment can be taken by multiple people at different level within the organization, which then allows for comparisons giving a better insight into the strengths and weaknesses of the enterprise.

5.1.2 Functional Requirements

This platform was initially meant to be used both internally and externally. It was later decided to focus only on the internal usage, for which clients would be able to use the app only if allowed by a Capgemini person. The main idea of the tool is to give the freedom to the client to take the assessment from different positions within its organization, and present a general overview of the results, which would then be discussed more in-depth with the Capgemini's team.

You can find hereafter the list of the main features that the platform should enable:

- Identification: a simple login form to ensure only clients allowed by the Capgemini team are able to take the assessment. The team should also have the possibility to easily edit the base of users allowed in the login.
- Explanation: have explanations regarding the platform and what the user is supposed to do.
- Interactive Survey: an online survey dedicated to the current client with the relevant questions for each enabler. The questions should also be easily editable by the team without prior knowledge of code.
- Storage: storing the results of all the surveys filled by the clients with relevant information, in compliance with GDPR laws.
- Analysis: exploit the results of the survey to give an overall score of the innovation readiness and infer the focuses the clients should take.
- Data Visualisation: a visual representation of the results for the client.
- Sending results: the possibility to send the page of the results to a specific email address.

As for the previous projects, these functional requirements come from a deeper analysis of the needs through the Design Thinking process, as described in Section 2.

5.2 UX/UI Design

As for the previous projects, we have also been through a phase of sketching and wireframing before creating the visual designs. These phases won't be detailed for this project, more information can be found in Section 2 and Section 4. Note that design has once again been a central element of the UI development, given that the application is customer-facing.

Although the platform doesn't have very complex features, we have been willing to provide an outstanding user experience while using the app. Our two versions of the wireframe have been realised with Invision [21] to easily show interactions between the different parts of our design.

Figure 5.1 shows some of the elements of the first wireframe that we created. In this wireframe, the user is first guided through the list of enablers before finding all the statements on a scrollable format. We realized that the number of clicks to access the assessment was too large, which made us reconsider the design as you can see in the next section.

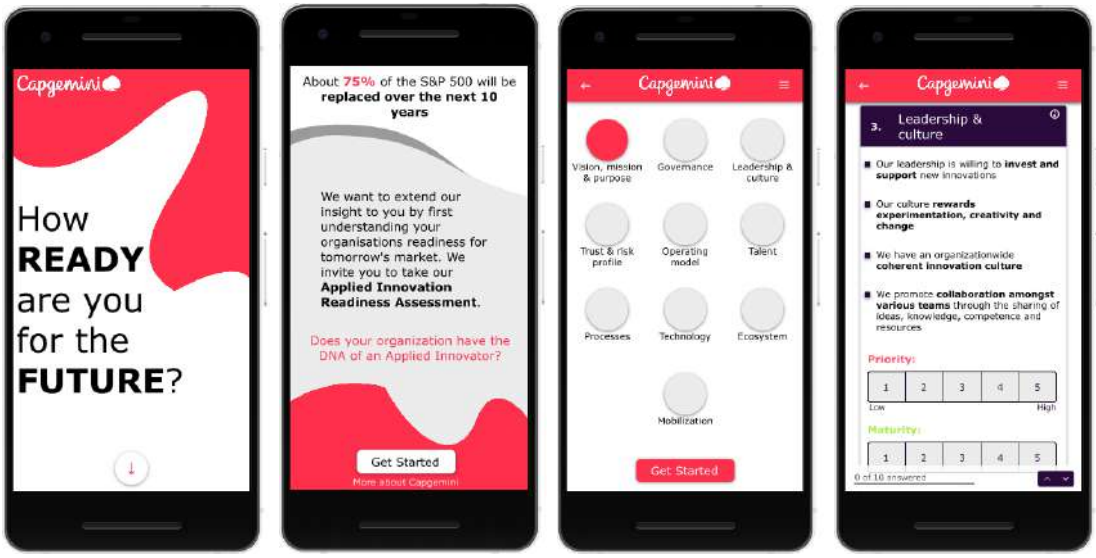

Figure 5.1: Parts of the wireframe of the Applied Innovation Enterprise Readiness Assessment

This version of the wireframe did not consider using any external application to develop the platform. However, due to the time constraint and the cost of development of the tool from scratch, the Team decided to use already-existing tools for the creation of the survey. This not only allowed to have a totally secure and visually-appealing survey, but also to have give the possibility to any member of the Team to edit the survey if needed.

Because of this, the design of the app was recreated with only a homepage and a results page with scrollable content. The rest of this section will explain the development of the application.

5.3 Technical Development

The development of Capgemini's *Applied Innovation Enterprise Readiness Assessment* has been done in an Agile manner. The project's initial time-constraint for the alpha testing was very short. A first version of the application was then developed in a few days only, containing the principal features and core concepts of the app.

After validating the general flow, several iterations have been needed to refine which data should be presented in the results page, and how. Multiple minor changes have then been made along the multiple weeks of beta testing.

The application was initially meant to be cross-platform. We wanted to provide the ability for any client or user to take the assessment anywhere. As we mentioned earlier, it was later on decided to reserve the app for an internal use only. Given that it didn't make sense to have the application available on the Google Play Store or Apple Store anymore, we only moved forward with the Web version.

For this reason, the technologies used to develop the platform are the same as the ones used for the *Capgemini Expo Guide*, described in Section 4. We used Ionic on top of Angular 8 to provide the cross-platform environment needed for the inital app, and Firebase in the back-end to process data without server. The global architecture diagram can be found in Figure 4.4.

The following subsections will give better insights into the technical development of the multiple features of the platform.

5.3.1 Back-End

As for *Capgemini Expo Guide*, the back-end of *Applied Innovation Enterprise Readiness Assessment* isn't very heavy. It has two main functionalities:

- Storing and Presenting data: After filling the survey, the data from the user is collected and stored in the Firebase Cloud Firestore. It is then reformatted by a cloud function to be presented on the results page. More about this will be detailed in the other sections underneath.

- Auto-Mailing: The user should be able to contact the Capgemini Team through a contact form in the app. The Team should also be notified each time a user completes a form, with the relevant information.

The back-end is entirely relying on Firebase, in which Cloud Functions have been developed to host the auto-mailing and data-presentation algorithms.

Architecture

The core element of the architecture are *Typeform* and *Zapier*, which are hosting the form and allows us to simply retrieve data and process them.

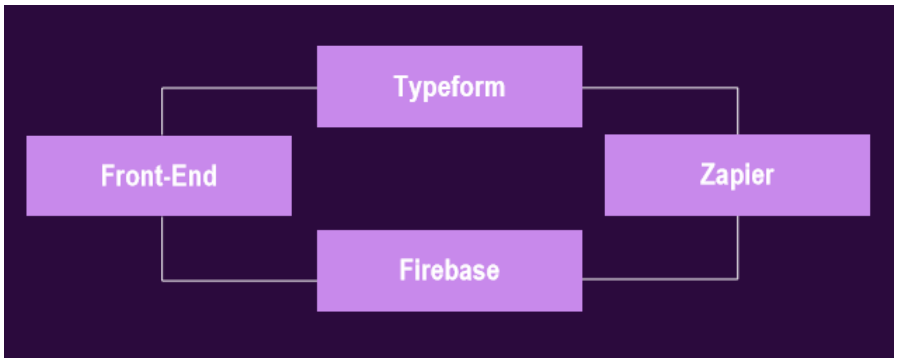

Figure 5.2: Architecture of the Applied Innovation Enterprise Readiness Assessment's Back-End

When the user lands on the Homepage of the front-end, he is able to access the survey after logging-in. This survey comes from Typeform and is embedded into our application thanks to Typeform's API.

After completing the Typeform, the results of the survey are directly sent to Firebase thanks to Zapier, which performs an action in response to the event triggered when a new response is added in Typeform.

Firebase then formats the results and returns them to the front-end, which in turn displays all the relevant data to the user.

Typeform

Our front-end is embedding a Typeform survey, which is at the core of the solution. It allows us to have a safe survey (protected from malicious usages), easily editable by the rest of the Team and independent from our app. This also limits the development time, as there is no need to develop the survey's functionalities by ourselves.

The drawback of such solution is the lack of flexibility given that the UI possibilities are limited. It also means that a third-party application has access to the data of our users. In our case, these drawbacks are minimal as the design of the form integrates well with our platform and that the data generated by the survey itself does not contain sensible information.

Typeform makes it easy to create forms and surveys online and collect the results. It has a well-supported API system which allows for embedding the surveys in web pages and retrieving the results easily [26].

Once the form has been submitted, it triggers an event called *Zap* within Zapier. The Zap takes the form that has been submitted and inserts it in our Firebase database.

Zapier

Capgemini uses proxies and firewalls which can sometimes make the communication with external APIs difficult or impossible. With a tool like Zapier, we avoid facing the Cross-Origin Resource Sharing (CORS) issues since we only interact with the database. CORS happen when we are trying to access a resource that is not hosted on the same domain as the request's origin domain [27]. In our case, it happens when requesting data from Typeform.

We are then using the Zapier online tool to facilitate the communication between Typeform and our Firebase database. Zapier allows to automate tasks by making it easy for end users to integrate the web applications they use. [28].

In Zapier, we created a Zap that is triggered each time data is received from the Typeform's response system. The Zap then transmits the new response received to our Firebase database, where it stores it raw.

Database

As we already explained in some of the previous projects, we are using Firebase , a mobile and web application development platform. It is a powerful tool for easily developing and scaling up apps. It includes several useful features such as analytics, real time databases, cloud functions, authentication systems and much more.

The Firebase *Cloud Firestore* is fed from the Zap triggered by the Typeform. It simply stores the responses received from each user. The code underneath represents the model used for the responses:

```
export class FormResponse {
    id?: string;
    formattedRes: ScoreData[];
    industry?: string;
    organization?: string;
    country?: string;
    businessProblem?: string;
    jobTitle?: string;
    genMaturity?: number;
    genPriority?: number;
    date: string;
    formID: string;
    tag:string;
    responses: string;
```

```
    submissionID: string;
}
```

The *formattedRes* attribute contains the data after being processed by the Firebase *Cloud Functions*. This reformatting makes the organization of data more intuitive and suitable for further operations.

Pre-treatment of the data

The responses from the Typeform are received as a long string containing all the questions and answers. We then need to parse that string to retrieve all the data we are interested in displaying, and calculate the scores based on them, this is a one time only process. All the treated data are then registered in the database under the right format, in the *formattedRes* attribute of our model, to make sure that the information can be directly displayed the next times.

This process is made possible thanks to a Firebase *Cloud Function* [22]. The function is triggered each time a new entry is added to our Firebase *Cloud Firestore*. It then uses a simple algorithm to parse the string and return the parsed elements organized in the *formattedRes* attribute.

Example of data before formatting:

```
"responses" : "### Which company are you from?
   Company_Name### Which country is your team based in?
   Country_Name### Vision, mission & purposetrue###
   Governancefalse###Leadership & culturetrue###Trust &
   risk profiletrue###Operating modelfalse###Tag (hidden
    field)xxxxx"
```

Example of data after pre-formatting:

```
"formattedRes" : {
    "question_1": {
        "label" : "Vision",
        "maturity" : 2.5,
        "priority" : 5
    },
    "question_2": {
        "label" : "Governance",
        "maturity" : 5,
        "priority" : 9
    },
```

```
    "question_3": {
        "label" : "Op Model",
        "maturity" : 10,
        "priority" : 4
    }
}
```

The *Priority* and *Maturity* scores are calculated during the pre-formatting, in the way explained in the Subsection 5.1.1. The pre-formatted data is then used to be displayed on the results page.

Note that pre-formatting the data is also useful for future usages that may include a benchmark per industry. Indeed, it will be much easier to make operations on formatted data than on the unorganised string received directly out of Typeform.

Cloud Functions

We have been using this powerful feature of Firebase, giving the possibility to run backend code in response to events triggered by Firebase features and HTTPS requests. More information can be found in Section 4. As for the *Capgemini Expo Guide*, we are using Cloud Functions for auto-mailing, both for sending the results and for the contact form.

Send Results: While the user sends the results to his email address, an email is sent both to the user's email address and to an AIE member to alert him about the new post. This is done by storing the message and relevant information in the 'submits' collection of our Realtime Database. When the new document is created, this triggers the function *onSubmitReceived()* which then uses Nodemailer to send the email both to the user and the AIE member. More information can be found in the Subsection 4.3.2.

Contact Form: When the user sends the contact form, an email is sent to an AIE member with all the information present in the contact form. As for Send Results, we store the message and relevant information in the 'messages' collection of our Realtime Database. When the new document is created, this triggers the function *onMessageReceived()* which also uses Nodemailer to send the email to the AIE member.

Note: onSubmitReceived() and onMessageReceived() have a similar structure as the sendContactMessage() function described in the Subsection 4.3.2.

5.3.2 Front-End

The front-end was built using the development platform Ionic v4, on top of Angular 8. Ionic allows to build cross platform mobile, web and desktop apps with one shared code base. It allows us to manage the responsiveness and usability of our app across different platforms easily. The Angular framework also allows us to benefit from a set of pre-built components and libraries to make the development easier. More information about Ionic and Angular can be found in the section 4.

Architecture

The architecture of the front-end can be found in Figure 5.3. It is rather simple and uses the angular router package which allows a simple handling of the routing and lazy loading to improve the app efficiency.

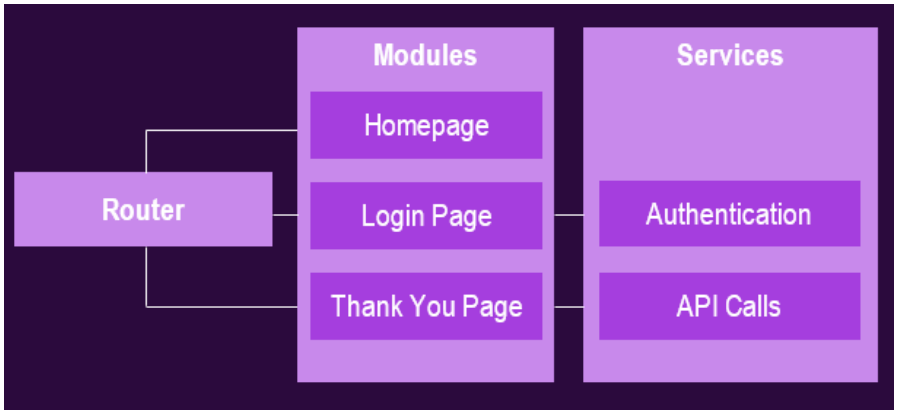

Figure 5.3: Architecture of the Applied Innovation Enterprise Readiness Assessment's Front-End

Homepage

The Homepage module has the main purpose of displaying a visually attractive landing page for the users. All the functionalities, such as the login, are handled in other modules.

The responsiveness of the design is mainly ensured by using the grid system in Ionic, with anchor points to properly adapt the layout to the screen size.

Header: The first section of the homepage is imported from the *Shared Components* of our application. We import the *Generic Module*, containing both the *Header Component* and *Footer Component*, to be able to reuse them in other parts of our app.

Footer: As for the Header, we import the Footer Component from the *Generic Module*.

The user then has the possibility to click on the Call to Action buttons, which will either open a login dialog if the user is not connected, or directly open the form if the user is connected.

Login: Clicking the Call to Action button will open a Modal controlled with the Login Module. This Module simply calls our Authentication Service which accesses the Firebase back-end to process the authentication. When the operation is finished, the Modal closes and the callback returns the answer from the database to the Homepage Module, which then allows the user to access the form or not.

Once logged in, the user has access to the form that was created using Typeform with a unique session ID allowing us to collect his form data. More information about Typeform can be found in the Subsection 5.3.1 of this document.

Generating the Survey: Our app uses an embedded Typeform to make the whole process easier. All information on embedding the Typeform can be found in the Typeform documentation. In order to generate a unique tag for each submission, we use Typeform hidden fields. We then generate a random ID that we use in the hidden field when the form is opened. This allows us to easily retrieve the answers corresponding to the specific form that was just submitted.

```
openSurvey() {
    // generateId() generates a unique random ID for the user
    this.tag = this.generateId();
    const popup = typeformEmbed.makePopup(
        this.FROM_URL, // URL of the typeform
        {
        mode: 'popup',
        autoOpen: true,
        hideHeaders: false,
        hideFooters: false,
        onSubmit: (eleme) => {
          this.navCtrl.navigateRoot('thank-you-screen?tag=' + this.
    tag.toString());
          popup.close();

        }
      }
    );
    popup.open();
}
```

Results Page

The Results Page is here to display the data submitted by the user in the form. All the treatments for the data are done through the Firebase Cloud Functions after retrieving the data from Zapier.

When the user lands on the Results Page, the module will automatically fetch the database, looking for the form that was just created using the tag generated in the Homepage. While searching for the document, the page displays a spinner.

If the information could not be fetched (because of an invalid tag, or no network connection for example), a toast is displayed informing the user to try again.

After the form results are collected from the database, the user can press the *'View Results'* button, which will format the data and display them into the graphs.

Data visualization: Data visualization is one of the central points of the platform. It is meant to present the user's results at a glance and give an overview of where his organization stands. Figure 5.4 shows some of the graphs used for the visualization. These graphs have been realized thanks to the front-end library *Chart.js*. Chart.js is an open-source library which makes data visualization easier [16].

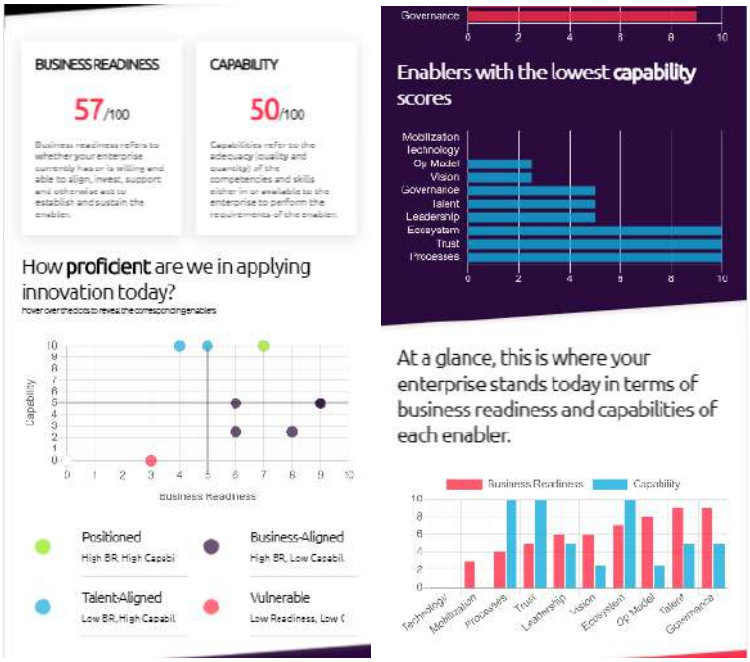

Figure 5.4: Data Visualization in the Applied Innovation Enterprise Readiness Assessment

While landing on the Results page, the user sees the overall results of his

organization in terms of *Business Readiness* and *Capability*. These scores give an understanding of the general performances of the organization in innovation.

A matrix chart is then used to determine more precisely how the company is performing with regards to each enablers. The enablers are categorized in one of the four following states: Positioned, Talent-Aligned, Business-Aligned, Vulnerable.

Finally, enablers with the lowest scores are put into perspective to underline the necessity for the organization to focus on them. Further advice are given by the Capgemini AIE based on the client's needs.

5.4 Outcomes

Capgemini's *Applied Innovation Enterprise Readiness Assessment* has been developed in about a month. The first functioning platform was available in a few days and the design, content, and data-visualization have been refined in an Agile manner in the following weeks. The application has been made available only as a Web app, thanks to Firebase Hosting.

During this project, I got the chance to work with the Chief Innovation Officer of Capgemini Group, as well as other leaders of innovation within the company. This has been the occasion for me to understand how innovation capabilities can be measured in enterprises, and how to build a reliable, and scalable application, taking into account all technical parameters such as GDPR, cybersecurity, latency, and availability in multiple countries and network environments.

The application has first been available for alpha-testing only within the Team, to spot the bugs and determine which elements could be improved. After this iteration, we started a beta-testing phase with all the Applied Innovation Exchanges of our worldwide network, during which we noticed some accessibility issues due to firewalls in some of the locations (mainly India and China). After working with the local networking teams to fix the issues, the final version of the application was open to be used by all Capgemini's innovation Teams. You can see some screenshots in Figure 5.5.

Once enough data will be collected, the platform is aimed to scale up by providing a benchmark based on a variety of filters. This way, it will be more efficient to assess the performances of a business in comparison to its competitors. Based on the feedback from clients, the app may also be publicly available as it was originally planned.

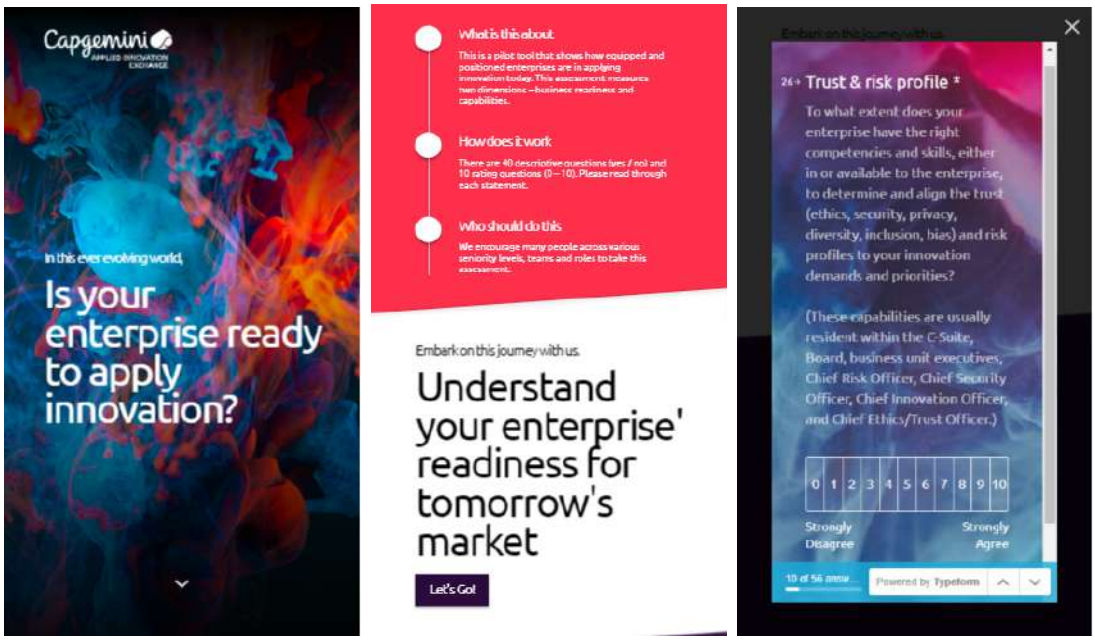

Figure 5.5: Screenshots from the Applied Innovation Enterprise Readiness Assessment

Chapter 6

Conclusion

This document has summarized my technical work as a Software Engineer in the Capgemini's Applied Innovation Exchange. In an environment of hyper-competitiveness, innovation is the key to stand out. By building prototypes and applications facilitating the adoption of innovation, I have actively participated in the AIE's mission to provide high value to its customers, fostering creativity and bringing ideas to life.

I had the chance to work in every aspect of the *Discover*, *Devise*, *Deploy*, *Sustain* framework through the multiple projects I took part in. With the development of the *Panasonic - Cy* prototype and the *OCR for tire stickers* POC, I enabled the ideation and analysis of new products and services for our clients, putting technical challenges into perspective to give better insights into the strengths and pain points of each project. My work on *Capgemini Expo Guide* and the *Applied Innovation Enterprise Readiness Assessment*, on the other hand, has been focused on enabling innovation by leveraging our ecosystems and identifying the needs of our customers. The resulting products, customer-facing, had to be carefully thought of in terms of design, user experience, and functionalities, to guarantee the least amount of friction with the users. It also brought new challenges such as the scale-up, security, and availability in multiple environments and locations. Of course, I have also written detailed documentations for each project I have developed.

My role at the AIE hasn't been solely technical. Innovation being an outcome of research, analysis, environment, technology, and more, it is crucial to take part in multiple phases of the process to understand it better. Some of my work has then been focused more on the research aspect, with the objective to identify new trends and understand the ecosystem in specific areas. I have conducted some research for our clients, on the future of the retail and the automotive industry, but also directly for Capgemini, on the new trends in omnichannel for example. I also had the chance to participate in the creation and facilitation of ideation workshops with our clients. To showcase new technologies to Capgemini's customers, I worked

on the installation of some technical demonstrations in the innovation lab space. Some of them have been exposed at some major events such as the *NYC Refashion Week*, where we presented a tool to automate returns for fashion brands [29].

This experience has been one of the richest in my student's life. Not only did I get the opportunity to develop my technical and analytical skills a lot, but I also discovered new technologies, industries, and methodologies that have widened my understanding of the world in general. I consider myself particularly lucky to have worked in such an environment, being in touch with CxOs from leading companies and surrounded by start-ups and experts.

Given the agility of the Team and with the objective to constantly provide better services to our clients, most of the projects I worked on are meant to be developed further. *Capgemini Expo Guide* should be revised to have a wider focus and allow enteprises from anywhere in the world to access a list of the most relevant startups in their area in a few clicks. The *Applied Innovation Enterprise Readiness Assessment* will use data collected from clients to perform benchmark analysis through a main dashboard. Finally, *Panasonic - Cy* and the *OCR for tire sticker* prototypes are meant to be used as baselines for further iterations.

Bibliography

[1] B. Godin. *The Idea of Technological Innovation. A Brief Alternative History*. Edward Elgar, 2020 (cit. on p. 1).

[2] H. Chesbrough, W. Vanhaverbeke, and J. West. *Open Innovation: Researching a New Paradigm*. OUP Oxford, 2006 (cit. on p. 1).

[3] Capgemini. *Capgemini - Company profile and key figures*. 2020. URL: `https://www.capgemini.com/company-profile-key-figures/` (visited on 05/18/2020) (cit. on p. 2).

[4] Capgemini. *Capgemini - Applied Innovation Exchange*. 2020. URL: `https://www.capgemini.com/service/applied-innovation-exchange/` (visited on 05/18/2020) (cit. on p. 3).

[5] Fahrenheit 212. *Fahrenheit 212*. 2020. URL: `https://www.fahrenheit-212.com/` (visited on 05/19/2020) (cit. on p. 5).

[6] Panasonic. *Panasonic*. 2020. URL: `https://www.panasonic.com/` (visited on 05/19/2020) (cit. on p. 5).

[7] T. Brown. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. HarperBusiness, 2009 (cit. on p. 6).

[8] Yale University. *Journey Maps Personas*. 2020. URL: `https://usability.yale.edu/understanding-your-user/journey-maps-personas` (visited on 05/20/2020) (cit. on p. 7).

[9] M. Mruzek, I. Gajdac, L. Kucera, and D. Barta. «Analysis of Parameters Influencing Electric Vehicle Range». In: (2016) (cit. on p. 10).

[10] Zeplin. *Zeplin*. 2020. URL: `https://zeplin.io/` (visited on 05/22/2020) (cit. on pp. 12, 39).

[11] Node.js. *Node.js*. 2020. URL: `https://nodejs.org/` (visited on 06/11/2020) (cit. on p. 16).

[12] Express. *Expressjs*. 2020. URL: `https://expressjs.com/` (visited on 06/11/2020) (cit. on p. 16).

[13] Jstat. *Jstat Github Repository*. 2020. URL: https://github.com/jstat/jstat (visited on 06/11/2020) (cit. on p. 17).

[14] mongoDB. *mongoDB*. 2020. URL: https://www.mongodb.com/ (visited on 06/11/2020) (cit. on p. 19).

[15] Angular. *Angular*. 2020. URL: https://angular.io/ (visited on 06/11/2020) (cit. on p. 23).

[16] Chart.js. *Chart.js*. 2020. URL: https://www.chartjs.org/ (visited on 06/22/2020) (cit. on pp. 23, 60).

[17] Google. *Google Maps Platform*. 2020. URL: https://developers.google.com/maps/documentation (visited on 06/11/2020) (cit. on p. 24).

[18] SimpleTire. *SimpleTire*. 2020. URL: https://simpletire.com/ (visited on 06/16/2020) (cit. on p. 27).

[19] Google. *Firebase*. 2020. URL: https://firebase.google.com/ (visited on 06/16/2020) (cit. on p. 28).

[20] NRF. *National Retail Federation*. 2020. URL: https://nrf.com/ (visited on 06/16/2020) (cit. on p. 34).

[21] InVision. *InVision*. 2020. URL: https://www.invisionapp.com/ (visited on 06/16/2020) (cit. on pp. 37, 52).

[22] Google. *Firebase Cloud Functions*. 2020. URL: https://firebase.google.com/docs/functions (visited on 06/19/2020) (cit. on pp. 42, 56).

[23] Nodemailer. *Nodemailer*. 2020. URL: https://nodemailer.com/ (visited on 06/19/2020) (cit. on p. 42).

[24] Ionic. *Ionic Framework*. 2020. URL: https://ionicframework.com/ (visited on 06/22/2020) (cit. on p. 46).

[25] Apache. *Apache Cordova*. 2020. URL: https://cordova.apache.org/ (visited on 06/22/2020) (cit. on p. 46).

[26] Typeform. *Typeform*. 2020. URL: https://www.typeform.com/ (visited on 06/22/2020) (cit. on p. 54).

[27] Mozilla. *Cross-Origin Resource Sharing*. 2020. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visited on 06/22/2020) (cit. on p. 55).

[28] Zapier. *Zapier*. 2020. URL: https://zapier.com/ (visited on 06/22/2020) (cit. on p. 55).

[29] *Refashion Week NYC*. 2020. URL: https://www.refashionnyc.org/ (visited on 06/29/2020) (cit. on p. 64).