



**University of
Zurich**^{UZH}

Analytics with Passive Wi-Fi Signals

Lucas Pelloni

Naglerwiesenstrasse 88, 8049 Zurich

Student ID: 13-722-038

Supervisor: Bruno Rodrigues, Christian Killer, Prof Dr. Burkhard
Stiller, Prof. Dr. Thomas Bocek

Date of Submission: December 6, 2019

Abstract

In the last years, the massive distribution of mobile devices like smartphones or tablets has radically changed our social life. Many individuals carry these smart devices all day. Therefore, being able to track smartphones can reveal interesting patterns of a person's life. In this regard, a large amount of data can be collected by monitoring signals emitted by Wi-Fi capable hardware. This is because, such devices are repeatedly emitting wireless signals with the aim of finding an access point within the Wi-Fi range. Subsequently, this captured data can be leveraged to extract knowledge which can be useful in different fields.

In a marketing context, this information could allow to obtain insights on how long visitors are engaged in an exhibition and how many visitors and observers are on site. Additional metrics, such as peak times, popular routes, and duration of certain visits, will become measurable.

This thesis work presents and extends a software solution which leverages passively gathered data from smartphones for revealing patterns for marketing agencies and brands. The proposed solution allows them to measure the effectiveness of their offline marketing investment with numerical values and a list of predefined Key Performance Indicators (KPIs).

In den letzten Jahren hat die massive Verbreitung von mobilen Geräten wie Smartphones oder Tablets unser soziales Leben radikal verändert. Viele Einzelpersonen tragen diese intelligenten Geräte den ganzen Tag über. Daher kann die Möglichkeit, Smartphones zu verfolgen, interessante Muster einer Person offenbaren. In diesem Zusammenhang kann eine große Datenmenge durch die Überwachung von Signalen gesammelt werden, die von Wi-Fi-fähiger Hardware ausgegeben werden. Denn solche Geräte senden immer wieder drahtlose Signale mit dem Ziel, einen Zugangspunkt im Wi-Fi-Bereich zu finden. Anschließend können diese erfassten Daten genutzt werden, um Wissen zu gewinnen, dass in verschiedenen Bereichen nützlich sein kann.

Im Marketing-Kontext könnten diese Informationen Einblicke darüber geben, wie lange Besucher an einer Ausstellung teilnehmen und wie viele Besucher und Beobachter vor Ort sind. Zusätzliche Kennzahlen wie Spitzenzeiten, beliebte Routen und die Dauer bestimmter Besuche werden messbar.

Diese Arbeit präsentiert und erweitert eine Softwarelösung, die passiv gesammelte Daten von Smartphones nutzt, um Muster für Marketingagenturen und Marken aufzudecken. Die vorgeschlagene Lösung ermöglicht es ihnen, die Effektivität ihrer Offline-Marketinginvestitionen mit Zahlenwerten und einer Liste von vordefinierten Key Performance Indicators (KPIs) zu messen.

Acknowledgments

My experience with the implementation and subsequently the writing of this master thesis has been one of the most rewarding academic experiences of my life. During these six months, I have learned many programming and technical details, appreciating its challenges and its beauties. However, this thesis would not have been possible without the help of some people. First, I would like to thank all those people who have contributed directly to my master thesis. After that, those who indirectly allowed me to reach such an important goal of my life: my master degree.

First of all, I would like to express my gratitude to my thesis advisors, Bruno, Thomas and Christian. I would like to thank them for their patience, guidance and support, and for their constructive criticism which always encouraged me to achieve my best.

Secondly, I would like to thank Axelra and Livealytics, for their support during my master thesis.

Finally, I would like to thank Prof Dr. Burkhard Stiller for giving me the opportunity to work together with the Communication Systems Group Lab at the University of Zurich, allowing me to use the most state-of-the-art infrastructures and technologies for my thesis.

I would like to spend some words for those people, who have supported me during the whole master degree. I would like to thank my parents for being there for me throughout all my master track and for supporting me. I could never have done it without you and I hope you will be proud of me.

Lucas Pelloni, 13-722-038

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
1.3 Context	2
1.4 Thesis Outline	3
2 Related Work	5
2.1 Tracking Human Mobility and In-Outdoor Positioning using Wi-Fi signals	5
2.2 Tracking Human Affluence using Wi-Fi signals	7
2.3 User profiling and Deanonymization using Wi-Fi signals	8
3 Approach	11
3.1 Stakeholders	11
3.2 Key Performance Indexes, Metrics and Dimensions	15
3.2.1 Dimensions	15
3.2.2 Parameter	17
3.2.3 Metrics	18
3.2.4 Key Performance Indicators (KPIs)	19
3.3 User Experience	21

3.4	Microservice-based Architecture	28
3.4.1	Monolithic vs. Microservices Architecture	28
3.4.2	Trade-Off: Shared Database and Microservices	29
3.5	Benchmark	31
3.5.1	Adjustment of values and KPIs	31
3.5.2	Normalization	35
3.5.3	Index	35
3.6	Predictions	36
3.6.1	Features Selection	36
4	Implementation	43
4.1	User Interface	43
4.1.1	React	43
4.1.2	Structure	44
4.2	AWS Serverless Architecture	50
4.2.1	AWS Core Components	50
4.2.2	Node.js and AWS Microservices	55
4.2.3	AWS CloudFormation Template	59
4.2.4	Predictions: AWS Time Series Forecasting	62
4.3	Internet of Things (IoT)	66
4.3.1	Information Gathering Process through IoT Sensors	66
4.3.2	Processing Data through the Processing Server	70
4.3.3	IoT Alerting Mechanism: How to track Sensors Status	71
4.3.4	Remote Sensor Configuration	72

<i>CONTENTS</i>	vii
5 Evaluation	73
5.1 Usability	73
5.1.1 SUS Results and Personal Feedback	74
5.1.2 Evaluated Screens	75
5.2 Predictions Accuracy	79
5.3 Costs	81
5.3.1 AWS Pricing	81
6 Final Considerations	85
6.1 Future Work	86
Bibliography	87
Abbreviations	97
List of Figures	98
List of Tables	100
A Installation Guidelines	103
A.1 Front-End	103
A.2 Back-End	104
A.2.1 Installation	104
A.2.2 Deployment to AWS	105
B Contents of the CD	107

Chapter 1

Introduction

1.1 Motivation

In the last decade, tracking individuals mobility using their smart devices has gained interest in the research community. In recent literature [7, 4, 28], many techniques for tracking have been investigated, such as video surveillance or GPS systems. However, these methods are often associated with large costs of installation and configuration. Indeed, they require either camera systems to be installed around the targeted surface or moving targets carrying appropriate devices in order to be able to be tracked. These techniques are usually invasive for targets and hardly practicable on a large scale. As described in [5, 9], less intrusive procedures for tracking people have also been investigated.

In this regard, a large amount of data can be collected by monitoring signals emitted by smart phones that have Wi-Fi enabled. This is because, such devices are repeatedly emitting wireless signals with the aim of finding an access point within the Wi-Fi range. By passively listening to these signals, it is possible to collect Wi-Fi traces containing information about the captured device, such its metadata or fabrication. The information gathering process does not require efforts from the device's owner, making therefore, the traceability process of a certain Wi-Fi-capable device unobtrusive. This allows the revelation of patterns and makes it possible to leverage the unobtrusive passive Wi-Fi measurements to extract value-added knowledge.

However, the collection of data can happen only if the target devices have the Wi-Fi enabled. Therefore, in order to collect piece of information with statistical significance, capable Wi-Fi capturing devices (*e.g.*, GL-iNetMiFi 4G Smart Routers [34]) are needed and have to be properly configured.

In the field of wireless networks, Wi-Fi scanning is a basic function of every wireless-based data transmission. This process can be viewed as a mechanism by which a certain client scanning device (*e.g.*, a smartphone) discovers the wireless networks in the range of its Wi-Fi adapter [2]. As part of this process, the client device, which wants to discover nearby networks is able to gather information about the channel, the signal strength, security configuration and capabilities. Usually, client devices are using the collected data

to exclude which networks they are not able to join and which one are capable to. In this sense, there are two techniques of scanning for Wi-Fi access points [1], *i.e.*, passive and active scanning. The latter consists of a client device actively looking for nearby wireless networks by transmitting signal probes and afterwards listening for callback answers from the access points in its range. The answer includes the configuration and characteristics of the network. The passive method consists of a client device passively listening for beacons sent from a nearby access point to announce their presence. Similarly to an active scan, transmitted probe responses contain the capabilities of the wireless network. In other words, the substantial difference between active and passive Wi-Fi scanning lies on the initiator. Indeed, an active scan is initiated by an access point, while a passive scan is started by a probe request of a certain access point.

In this regard, this thesis has as main goal the development of a software solution for extracting knowledge using a passive Wi-Fi tracking system.

1.2 Description of Work

The main use case of this thesis consists of quantifying the number of visitors in an open exhibition and to extract retrospectively a set of metrics and KPIs useful to interpret the performance of the fair. In order to achieve this, an analytic dashboard needs to be provided for measuring and benchmarking live marketing promotion activities, trade shows and retail spaces. Since for marketing agencies and brands is difficult to evaluate the effectiveness of their offline marketing investments, this added knowledge can help them to better plan their marketing strategy. Additionally, the system provides different benchmarks to compare the performances of marketing activities run by different stakeholders. Finally, the platform can be used to forecast how marketing activities would perform in similar events based on previous interactions.

All subsystems need to be orchestrated and well integrated into a web application. The software solution is developed from scratch, while part of the IoT hardware configuration is already provided. Finally, the thesis includes to investigate suitable hardware, reliability of collected data, scalability, distributed data collection, a user-friendly front-end, and derivable recommendations.

1.3 Context

In this regard, there are five main Stakeholders operating in the system (the detailed architecture's diagram and the corresponding relationship between these entities is described in Section 3.1).

1. **Users** have different roles and based on them can perform different actions. New users need an invitation to join the platform and can be created only by existing users with specific permissions.

2. **Agencies** are created and managed by users with specific permission. Agencies manage multiple clients and are responsible for the outcome of their marketing efforts.
3. **Clients** are created and managed by users with specific permission. Clients can manage a list of marketing campaigns.
4. **Campaigns** consists of a list of activities. They can be viewed as long-term promotional engines for their brands.
5. **Activities** can be viewed as single, short-term promotional events with the scope of advertising a product of a brand following a specific concept.

1.4 Thesis Outline

Chapter 2 describes different approaches presented in recent literature in the field of passive and active Wi-Fi tracking, while **Chapter 3** addresses the overall design architecture of the system presented in this thesis. **Chapter 4** continues with the implementation details of the approach presented in Chapter 3 and **Chapter 5** evaluates the system from different points of view and presents the conclusions. Finally, **Chapter 6** closes the thesis with the final considerations.

Chapter 2

Related Work

This chapter presents related works in the field of Wi-Fi tracking. All scientific works presented in the sections below tackle the goal of exploiting gathered data coming from Wi-Fi traces to provide different information patterns. Projects and papers presented below rely on the IEEE 802.11 standard [25] for their assumptions and calculations. This standard describes three types of packets [26, 11] that can be sent (and thus intercepted by apposite sensors) in WLAN networks:

1. *Probe Request*, used by Wi-Fi-capable devices to periodically scan for new AP.
2. *Association Request*, used by Wi-Fi-capable devices to connect to a specific AP.
3. *Reassociation Request*, used by Wi-Fi-capable devices to connect to another AP on the same network.

The presented works can be classified into three general classes according to the knowledge that want to be extracted using Wi-Fi tracking. Classes can be divided into three main categories:

- *Systems targeting human mobility and geographical localization of moving targets using Wi-Fi signals.*
- *Systems aiming at extracting patterns by counting individuals at mass events.*
- *Systems willing to correlate real individuals identities to a specific device (deanonymization) and perform user profiling with respect to certain social parameters.*

2.1 Tracking Human Mobility and In-Outdoor Positioning using Wi-Fi signals

In this category, academic literature investigate the correlation between human mobility and Wi-Fi AP by inferring the geographical position of a certain individual. Scientific

papers in this section compare with the work of this thesis in the approach for measuring individuals location using Wi-Fi signals. Indeed, different approaches, to that used in this work, for inferring human locations are presented.

In [4], a study has been conducted for six months and revealed the opportunity for exploiting ubiquitous GPS and Wi-Fi data for tracking the outdoor geo-location of a certain person. Indeed, location-based signals emitted at irregular intervals (*i.e.*, with high temporal resolution) by portable Wi-Fi-capable devices, makes possible to assign location to different Wi-Fi APs. Despite the fact that geographical coordinates are not attached to APs, it is sufficient to associate the first scan with geographical coordinates to that AP, since its placement tend to remain fixed. Thus, the fact that human mobility is habitual and has low entropy combined with the geo-location of Wi-Fi AP, makes possible to track high-resolution outdoor positioning.

Along the same lines, in [5] a system with low equipment and installation cost is described for passively tracking smartphones based on Wi-Fi signal detections. The approach consists of a trajectory and probabilistic estimation procedure, which relies on Viterbi's algorithm [16]. This algorithm is a dynamic programming algorithm which based on an input of observed events tries to guess the most likely sequence of hidden states. In the context of Wi-Fi tracking, the algorithm takes as input a set of observed detections of a moving smartphone and produces the most likely spatio-temporal route taken. Hence, instead of directly assigning single locations to the devices at each instance in time, the algorithm guesses the potential locations over time and computes the maximum probability trajectory.

In [6], a study has been conducted with a group of people walking in an area which have been tracked only using the magnitude of Wi-Fi signals from one single Wi-Fi access point. The study was based on a Wi-Fi transmitter and a Wi-Fi receiver (*e.g.*, a laptop), which have been located in different sides of the experiment area. Wi-Fi signals were reflected away of the people and the objects present in that area. While the signals were emitted, the moving devices could, consciously, interact with each other and with the objects on the path from the transmitted to the receiver. In the computational step, from these signals can be inferred with a Particle Filter the trajectories of the group in that area.

The approach presented in [7], consists of partitioning into a more abstract way Wi-Fi traces into individual trajectories perceived as periods of stops and moves. Afterwards, information patterns can be inferred about visited locations or even social behaviour. The study conducted presents three algorithms to determine stops and moves for GPS-based datasets. The algorithms are:

1. Cbsmot [17] which uses distance and time between consecutive detections (useful when the temporal resolution is low).
2. Dbsmot [18] that instead of taking a temporal factor into account, it takes the direction of movement. Because of this, it is able to state whether a device is moving in a straight line or is standing still and is going to greatly changing direction.
3. Stay Point detection [19, 20] which consists of the delimitation of different areas based on the accuracy of the measuring technique. When the device has a new position outside a given area, it entered a new area which is now taken as a pivot.

Similarly, [21, 8] present different algorithms for indoor location tracking using Wi-Fi signals. The first one, uses two methods to compute locations based on the signal information. It uses a signal propagation model, which estimates the fluctuation of the Wi-Fi signal in relation to the distance by applying a calibration process. Then, it guesses the distances from multiple AP based on the signal strengths. Finally, location can be linearly computed using triangulation [22]. The approach presented in [8] consists of the combination of two types of algorithms: deterministic and probabilistic. The deterministic one follows the triangulation principles, and thus uses the nearest neighbour algorithm based on signal strengths to estimate locations. The probabilistic one, uses the Bayesian algorithm and Bayesian law to guess the most likely location of the mobile device based on a certain probability. The final algorithm in [8] consists of a history-based tracking procedure. It makes use of either the probabilistic or deterministic algorithms to choose the most likely neighbours and then takes into account a series of consecutive signal strength (historical values) for location determination.

Finally in the field of Wi-Fi and location tracking, the quantitative analysis presented in [13], shows what rates of monitoring by smartphones devices, in a given metropolitan area, are necessary to achieve different levels of involuntary geolocation. The survey presented provides an evidence that a small percentage of sensors is able to track the majority of people of a dense metropolitan area. Afterwards, it shows that factors such detection rate, population density and broadcast radius might affect the tracking.

The system presented in [14] consists of a concept of device-free passive localization, where the people tracked does not even need to carry any type of device. The approach requires the installation of different Wi-Fi AP, which constantly emit signals in a given area. The idea consists of monitoring the emitted signals and computing values such as frequency ranges and signals strengths of the Wi-Fi AP. Then, the approach uses these values to detect changes in the environment, and thus correlate them with entities and their locations.

2.2 Tracking Human Affluence using Wi-Fi signals

Within this category, academic literature investigate potential insights which can be extracted by measuring influx of people at public events or wide open spaces using Wi-Fi signals. The scientific papers belonging to this section are highly related to the work of this thesis. For instance, Livealytics [3] has been taken as main use-case for the approach that is presented in Chapter 3.

Livealytics [3] is a Swiss-based company which provides an analytics solution for tracking moving entities with Wi-Fi signals in terms of influx of people in a mass event. The approach consists of geometrically placing different sensors, into the interested surface, that are able to passively listen to emitted Wi-Fi signals. Based on the predetermined sensors location and the perceived signal strength, the algorithm is able to compute to which dimension the walker-by belongs to. The dimensions can be viewed as surface circles which vary depending on different metrics and parameters, such as the estimated proximity to the stand, the dwell time range [23], the signal strength, etc.

In the same field, Probr [9] is an open source platform for passive Wi-Fi tracking which supports several use cases and consists of two separate subsystems called Probr-Core and Probr-Analysis. The first one supports the collection of Wi-Fi traces, the configuration of Wi-Fi interfaces and the graphical data aggregation through a Web application. The latter, analyses the data collected in Probr-Core. With these two systems, Probr is able to calculate how many moving people are in a room at any given time by counting how many unique MAC addresses are present.

In [10] a research-driven project was presented to gain insight into the Rhythm of the Campus of the Delft University of Technology using passive Wi-Fi monitoring. The patterns they found consisted of the duration of stay of the students per faculty and the moving patterns at and between different faculties. Similarly to [3], in order to achieve statistical relevance, 20 Wi-Fi monitors were spatially distributed and located into different zones of interests of the Campus and started collecting data during one week. Additionally, a dashboard was developed to visualize the recognized patterns.

The method presented in [11] consists of tracking people at mass events using passive Wi-Fi tracking. Indeed, the algorithm is able to scan and listen to emitted Wi-Fi signals and correlate the captured data to different locations. Similarly to [3, 9, 10], the visitor counter is computed using uniquely identified MAC addresses sent by Wi-Fi interfaces.

2.3 User profiling and Deanonymization using Wi-Fi signals

The academic works belonging to this section address the extraction of useful information and social patterns to correlate individuals and devices. The scientific literature in this section is related to the work of this thesis in the approach for linking MAC addresses to individuals.

In [12], passive Wi-Fi monitoring is used to exploit some vulnerabilities of the current WLAN protocol to perform malicious attacks.

- *Beacon replay attack* and consists of identifying a device through one or multiple *personally identifying wireless networks* (PIWN) to which in the past the device was connected to. Examples of PIWN can be home or work networks. Once a potential PIWN of one target is located, it can be impersonated and trigger a reaction from the device that has been linked to this particular network. By doing so, the device will try to connect and thus reveal information such as its MAC address.
- *Stalker attack* consists of physically following the target person and constantly collecting probe request frames in order to discover the MAC address behind that device.

Along the same lines, in [15] a white-paper is presented with the purpose of analysing the privacy impact of Wi-Fi analytics using MAC address as unique identifier. Furthermore,

guidelines are presented about how vendors should take security measures to protect the privacy of their devices. Finally, legal aspects are discussed about how in Europe a MAC address is considered to be personal information, also called pseudonymous data.

In [28], a system is presented which combines localization techniques for passive Wi-Fi probe requests with a system of cameras that captures the target entity through an image. The authors showcase how an accurate mapping between individuals and MAC addresses can happen by exploiting both systems.

A. Luzio et al. present in [29] a system which aims at deanonymizing people behind devices. Such solution, looks at the BSSID advertised in broadcast probe requests. Then, by querying a third-party service for Wi-Fi hotspots [30], the software is able to infer the geolocation of a certain individual.

Concerning user profiling, the works presented in [31, 32] investigate the correlation between social relationships by processing and analysing the affinity of their emitted probe request packets. The similarity computation is based on a set of fingerprints which have been predefined by the authors.

Chapter 3

Approach

The main idea of this thesis is to design a modular architecture that harmonically relates a group of five stakeholders with three different subsystems, such as a graphical UI, a microservice-based back-end and an IoT system. This chapter firstly presents a group of stakeholders which interact with the system. Secondly, it introduces a set of KPIs for measuring the outcome of a marketing event. Afterwards, it describes the design proposed for the back-end part. Subsequently, it presents the approach used for benchmarking different marketing activities, and finally it presents the solution for forecasting the outcome of marketing activities based on previous interactions.

3.1 Stakeholders

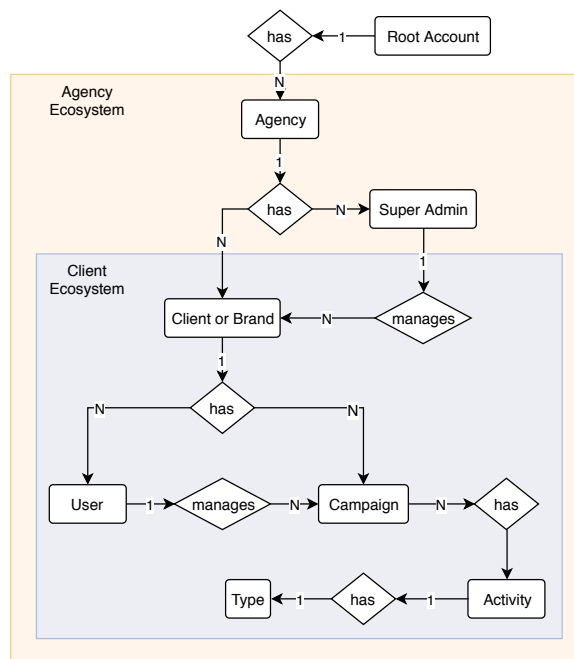


Figure 3.1: Stakeholders: Entity Relationship Diagram

In order to have a better overview of all stakeholders involved in the system, Figure 3.1 provides an Entity Relationship Diagram [33], *i.e.*, a graphical representation of all entities and their relationships to each other in the platform. Within the system, there coexist five main entities, *i.e.*, *users*, which have different roles, *agencies*, *clients*, *campaigns* and *activities*.

The platform supports different authentication and authorization layers, which implies users with different roles to be able to perform different actions. Additionally, new users need to be invited into the platform in order to be able to interact with other stakeholders. At the top of the roles hierarchy there are *root* accounts. These accounts are the most privileged in the system and have the strongest authorization and security level. Indeed, they are the only accounts which are consciously authorized to have a global overview of all other stakeholders and perform all possible facets of system administration. Since there are no security restrictions imposed upon to these accounts, they are able to perform all actions which can be performed by lower roles. Within this thesis, these permission-based layers are known as *cascade* effect. Finally, root accounts are the only ones which are able to create new agencies and can actively manage them.

In turn, *agencies* are per definition entities that contain and relate to multiple *clients* (brands). Generally, agencies are external marketing companies which conduct large marketing-projects and represent multiple brands simultaneously. In the proposed architecture, agencies are managed by *super admins*, which are responsible to assign new clients and invite new users to their specific agency's ecosystem. Similarly to root accounts, super admins can create new entities in their environment. However, they can not open new agencies or assign clients to an agency they are not responsible for.

Clients (or brands) can have multiple marketing campaigns and cannot exist without being assigned to an agency. For those clients which are not assigned to a third-party agency, a default agency will be assigned to them. In case of the existence of the same client in multiple agencies, this will be treated as two different entities.

While a super admin is responsible for a whole agency, similarly *admins* are in charge of managing a subset of clients that have been assigned to them by a super admin. Indeed, the latter defines the client's ecosystem for a client admin. Following the authorization cascade principle, client admins can create new entities in their environment, however, they cannot open new clients or assign new users to a client they are not responsible for.

Thereafter, *project managers* cannot invite new users to join the platform but are responsible to create and manage clients *campaigns*. In contrast, *analysts* can neither create campaigns nor invite new users and have almost everywhere within the platform read-only rights. Nevertheless, they are responsible for monitoring different marketing campaigns and ensure a successful outcome of all promotional activities.

Table 3.1 summarizes the overall permissions-based authorization concept, which states that different users with different roles perform different actions. It should be noted, that both super admins and admins can create new users with a set of conditions. In fact, both are able to invite new users within the platform only with an equal or lower role equal than theirs. For instance, a super admin is not allowed to create a new root account.

User role <i>Can create</i>	<i>Agencies</i>	<i>Clients</i>	<i>Campaigns</i>	<i>Activities</i>	<i>new Users</i>
Root	✓	✓	✓	✓	✓
Super Admin	×	✓	✓	✓	(✓)
Admin	×	×	✓	✓	(✓)
Project Manager	×	×	✓	✓	×
Analyst	×	×	×	✓	×

Table 3.1: User- and Role-based executable Actions within the Platform.

Subsequently, *campaigns* can be viewed as a series promotional activities which together with a marketing strategic plan aim at broadcasting a promotional message. Marketing campaigns have a start and end date and within this time frame can have a series of marketing *activities*.

A marketing activity can be viewed as a single, short-term and targeted promotional event with the scope of promoting the product of a brand. Each activity has a specific type, which may vary depending on the marketing approach. In total, there are eight different activity types:

1. Sampling or Giveaway

Distribution of samples to consumers such as commuters in a train station, passersby on a busy street, guests at an event, or shoppers in a mall. For example, handing out energy drinks at a concert. Generally, sampling can be mobile or have stationary sampling stands. Furthermore, it can have a promotional team actively distributing as many samples as possible. The contact with the consumer is very short and limited to distribution.

2. Tasting

Marketing activity where consumers taste a food or drink the product. For example, a wine tasting. Usually, this activity type offers a stand and promotional staff to prepare and offer the product. The contact with the consumer is generally brief.

3. Product or Service Demonstration

Marketing activity where a consumer can experience how a product performs. It shows how the product works and enables consumers to test it. Interested consumers use the opportunity to study and experience the product.

4. Sales Promotion

Essentially, the same as a product demonstration, but the consumer has the option to purchase the product or service.

5. Retail

Marketing activity involving the sale of products onsite, where the sales staff actively promotes the product.

6. Merchandising and Shelf

Floor plans and product or service displays designed to maximize sales, where no active promotional staff involved.

7. Display and Window

Marketing activity that promotes product exposure without interacting with the consumer, where no active promotional staff involved.

8. Event

Activity that directly engages consumers and invites and encourages them to participate in the evolution of a brand or a brand experience.

3.2 Key Performance Indexes, Metrics and Dimensions

In this chapter, the approach which has been used for measuring the performance of offline activities is presented. In order to be able to evaluate the effectiveness of a marketing offline investments for a specific brand, a set of *Dimensions*, *Parameters*, *Metrics* and *Performance Indicators* have been defined for each marketing activity. The idea behind these indexes is to quantify, using numerical and tangible values, the effort put into a marketing activity by a certain company.

3.2.1 Dimensions

Zones of Interest

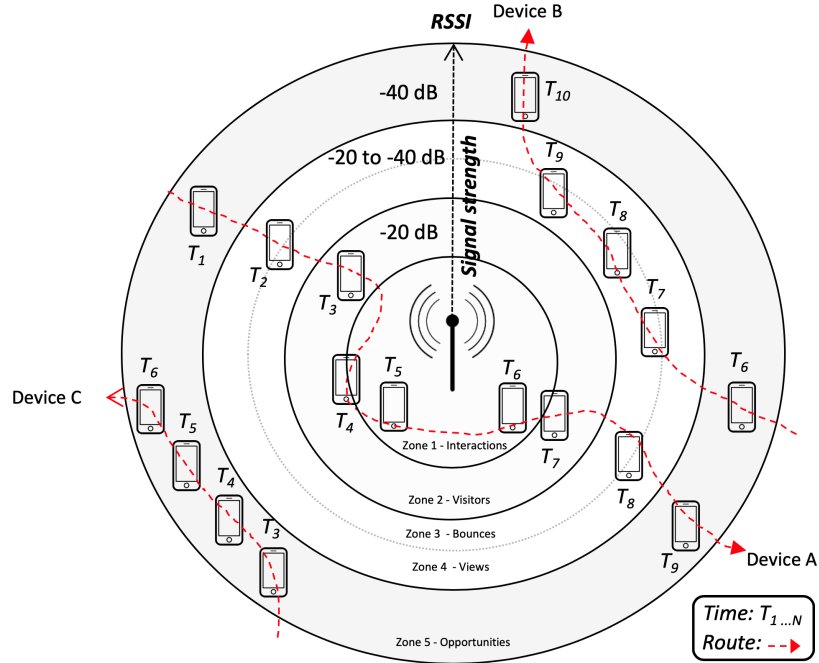


Figure 3.2: Dimensions: Zone of Interests

In this regard, the common pattern of all possible activity types presented in section 3.1 is a product that wants to be advertised in a certain location for a specific customer target. Therefore, an activity type is always associated to a physical place where the product gets demonstrated, distributed or tested. This always implies a group of people in motion who consciously or involuntarily come into contact with the product from different angles within the product area. Because of this, in order to state whether the interaction with the product has been voluntary or unintentional for a set of walkers, the surface is divided into five different circular surfaces as shown by Figure 3.2.

Each circular surface can be viewed as a zone of interest, whose radius is depending on the available area during the product demonstration. The approach consists of geometrically distributing a predetermined amount of sensors over the entire area. The more sensors have been placed into the surface, the more precise will be the measurements.

Indeed, the algorithm with which a certain person is assigned to a certain area of interest is based on the signal strength of the perceived Wi-Fi signal and dwell time algorithm [23]. The dwell time consists of a minimum and maximum time interval (in seconds) and can be configured independently of the zone. This interval, together with the signal strength, is then used to associate targets to specific areas of interest. Indeed, if the MAC address of a target is counted with a frequency which results to be within the dwell time interval, this target is assigned to the zone which corresponds to the perceived signal strength.

As shown in Figure 3.2, five different zone of interest have been categorized.

1. **Opportunities** (or Walkers-by, Passers-by, Footfall)
Count within the maximum range of the sensor(s). Describes the profile of a location in terms of total foot traffic. For the assignation, it takes into account the number of MAC addresses counted within the total pickup range of the sensors.
2. **Views** (or Impressions)
Count within the views zone of interest. Describes the number of times the visits zone was viewed. The calculation relies on the number of MAC addresses counted within a defined signal strength and minimum/maximum dwell time range for the views zone of interest.
3. **Visitors** (or Sessions)
Metrics for the visitors zone of interest. It describes the number of times the visits zone was visited. The calculation takes into account the number of MAC addresses counted within a defined signal strength and minimum/maximum dwell time range for the visitors zone of interest.
4. **Interactions** (or Goals, Targets, Leads)
Count of actions that indicates a positive lift for the brand or product, or an immediate impact on financial goals. The calculation takes into account the number of MAC addresses counted within a defined signal strength and minimum/maximum dwell time range for the interactions zone of interest.
5. **Bounces**
Count within the visitors zone of interest that leaves the zone before converting to a visitor. The calculation takes into account the number of MAC addresses counted within the defined signal strength of the visit zone with a dwell time *smaller* than the required minimum dwell time.

Period

A period is a dimension which defines a set of available data collection time frames for metrics and, by extension, performance indicators. At present, unique counts are available

in quarter hour, hourly, daily, and weekly periods. Counts can be aggregated to any period from hourly data. Frequency data is available for quarters and months. The hierarchy for time frames is defined as follows:

$$Quarter \rightarrow Month \rightarrow Week \rightarrow Day \rightarrow Hour \rightarrow Quarter Hour$$

Locations

A location is a dimension which defines a group of available data collection locations for metrics and, by extension, performance indicators. Currently, all metrics are available for individual activities and follow the hierarchy below:

$$Campaign \rightarrow Phase \rightarrow Place \rightarrow Location \rightarrow Activity (\rightarrow Zone of Interest)$$

Date-time

Date-time is a dimension which is defined as a date and time for collection of metrics and has the following hierarchy:

$$Year \rightarrow Quarter \rightarrow Month \rightarrow Week \rightarrow Day \rightarrow Hour \rightarrow Minute$$

3.2.2 Parameter

Parameters can be viewed as a set of numerical factors forming a set of boundaries which are at the base of all measurements.

Min / Max Signal Strength

As shown in Figure 3.3, for collecting metrics a set of customized GL-iNet MiFi 4G Smart Routers [34] have been configured. These devices are able to passively listen to emitted Wi-Fi signals and differentiate between different signal strengths. Detailed explanations about the sensors and their configuration and interaction with the platform can be found in chapter 4.3. The minimum and maximum signal strengths are a sets of boundaries for collecting metrics in a zone by defining an upper and lower limit for signal strength. Outside the range, the emitted Wi-Fi signals are ignored.



Figure 3.3: GL-iNet MiFi 4G Smart Router [34]

Min / Max Dwell Time

Minimum and Maximum dwell time are parameters which can be viewed as zone boundaries for collecting metrics in a zone by defining an upper and lower limit for visit duration (range in seconds). For instance, if the maximum dwell for the zone of interest "interactions" would have been configured to 10 minutes, a person who would have been stayed more than 10 minutes in that area would not be relevant for the analytics. In fact, the latter might be someone working for the promotional team. The Min/Max dwell time is a required parameter for the configuration of duration zones of interest.

Visit GAP

Visit GAP is a parameter which expresses the length of time in seconds after which a visit ends if a signal is not reacquired. For example, if the visit gap is 5 minutes and a visitor returns within 5 minutes after leaving, that counts as one visit. This parameter is required for the configuration of zones. Defaults is 5 minutes.

Duration Period

The duration period is a parameter which defines the minimum size in seconds for calculating the distribution of visitor duration. It is a required parameter for the configuration of duration and is correlated with the minimum dwell time for the visits zone.

3.2.3 Metrics

Count

Count (or counter) is a metric which expresses the amount of persons counted within a given time period. It is computed with the total amount of filtered MAC addresses

counted within a given time period. In the platform, count is always equal to or larger than unique count and is available for all zones and in quarter hour, hourly, daily, and weekly time frames. Furthermore, it can be aggregated to any time period from hourly data.

Unique Count

ss Per definition, unique count is a metric which can be viewed as the number of unique persons, expressed with the total number of unique MAC addresses within a time frame counted within a given time period. At present, unique count cannot be aggregated, *i.e.*, the sum of hourly unique counts per day is greater than the daily unique count. Furthermore, it is available for all zones and in quarter hour, hourly, daily, and weekly time frames.

Average Dwell Time

Average Dwell Time is a metric which expresses the average visit length distributed over a time period. This metric can be viewed as the average of last seen - first seen for count. Per definition, a visit ends when a MAC address has not been seen for a sufficiently long period of time, *i.e.*, a visit GAP. Average Dwell Time is available for all zones and in quarter hour, hourly, daily, and weekly time frames.

Frequency

Per definition, frequency is a metric which counts of how often visitors return. For instance, a frequency of one is the number of visitors that visit once while a frequency of two is the number of visitors that visit twice and so on. This metric is available on a weekly, monthly and quarterly basis.

Duration

This metric is a count of visits per duration, *i.e.*, the distribution of visitor durations. For instance, a count of 74 for a duration of 300 seconds indicates 74 visits for 5 minutes. It is based on dwell time per visit and duration setting and is available on a weekly, daily and hourly basis.

3.2.4 Key Performance Indicators (KPIs)

In the context of this thesis, KPIs [36] are numerical values that demonstrates how effectively a marketing campaign is performing and achieving its key objectives. Clients

(or brands) use KPIs at multiple levels to evaluate the outcome of their marketing investments in terms of reaching targets. High-level KPIs include overall factors, such as *Visibility*-, *Engagement*-, *Interaction*-, *Bounce*- and *Return-Rate*, *Loyalty* and *Engagement Reliability*.

Table 3.2 illustrates the equations expressing these KPIs.

KPI	Variable Name	Formula
Visibility (or Capture Rate)	V	$\frac{\text{Views Count}}{\text{Opportunities Count}}$
Engagement Rate	ERa	$\frac{\text{Visits Count}}{\text{Views Count}}$
Interaction Rate	IR	$\frac{\text{Interactions Count}}{\text{Visitors Count}}$
Bounce Rate	BR	$\frac{\text{Bounces Count}}{\text{Views Count}}$
Loyalty	L	$\frac{\text{Count}}{\text{Count Unique}}$
Return Rate	RR	$\frac{\text{Count} - \text{Count Unique}}{\text{Count}}$
Engagement Reliability	ERe	$\sigma(\text{daily ERa})$

Table 3.2: KPIs Formulas.

Visibility (or Capture Rate)

Visibility is a KPI which expresses the percentage of opportunities that saw the visit zone. This KPI describes the visibility of a an activity.

Engagement Rate

Engagement Rate is a KPI which is expressed as the percentage of views that converted to visits.

Interaction Rate

Interaction Rate is a KPI expressed as the percentage of visitors that stayed enough in the "interaction" zone to be considered interactors.

Bounce Rate

Bounce Rate is a KPI which can be viewed as the percentage of views that entered the visits zone but left before converting to a visit.

Loyalty

Loyalty is a KPi which is expressed as the average number of visits by a visitor. For instance, a loyalty of 1.9 indicates that, on average, people visited the activity about twice. This KPI will vary depending on the time period and is implemented for visits zone of interest. For example, weekly loyalty can usually be considered to be higher than hourly loyalty.

Return Rate

Return Rate is a KPI which is used to calculate the percentage of visitors that visit the activity more than once. Typically, return rate will vary depending on the time period, *i.e.*, weekly return rate is usually considered to be higher than hourly return rate.

Engagement Reliability

Engagement Reliability is a KPI which indicates the level of variability in daily performance and provides a measure for different levels of performance (*e.g.*, excellent or poor performances). This KPI is defined as the standard deviation of the daily engagement rate.

3.3 User Experience

In order to follow state of the art UX patterns, a design evaluation based on a clickable prototype has been performed before the implementation started. The mockups [37] aim at highlighting the core functionalities of the system and the interaction between the different UI components. It should be noted, however, that the clickable prototype does not reflect the final design in terms of UI elements, but shows a high-fidelity representation of the platform's appearance. In this regard, a list of core blocks and key elements have been identified during the design engineering phase:

1. Navigation

The platform needs to have an intuitive navigation concept. This implies, that the user must be able to navigate through different states and stakeholders by following a clear and unambiguous path. For this reason, a cascade structure has been adopted. This means, that all stakeholders presented in section 3.1 are connected to each other

through a navigable route. For instance, a client (or brand) might have a list of running campaigns, which might have in turn a series of promotional activities. As illustrated in Figure 3.4 clients, their campaigns and their activities are connected through different cards. Indeed, in this overview a client is immediately able to see its last running campaign and additionally all other campaigns. Similarly, its last running activity is shown together with all other activities. By clicking on a specific campaign or activity, the corresponding detail views are displayed. This high-level and presorted overview allows each client to see its last running items in their ecosystem. It is worth to highlight, that the last running activity must not necessarily be part of the last active campaign.

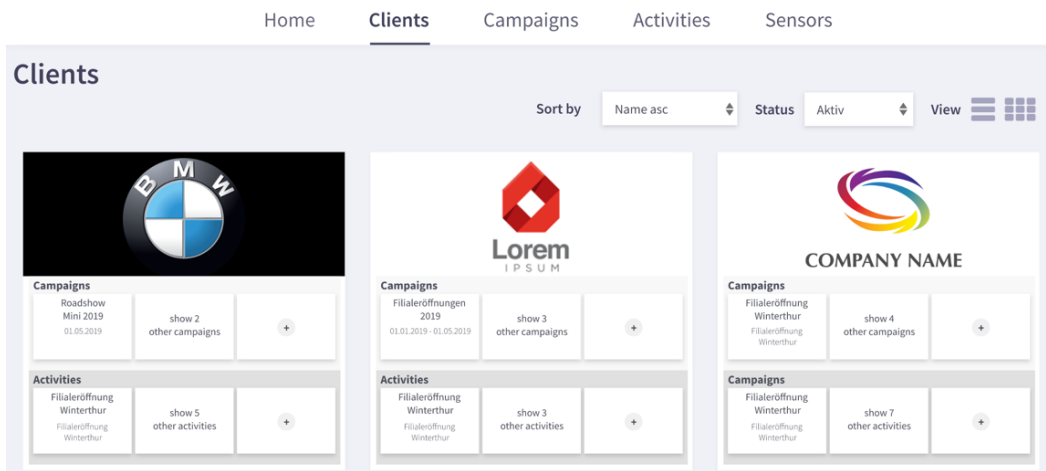


Figure 3.4: Design Evaluation: Clients Concept

On the other hand, as shown in Figure 3.5, the campaign level shows a list of running campaigns. Each card represents a campaign, it displays its overall score and contains two links with the next respectively last performed activity. Campaigns can be sorted by name, status, score and last modified. Consistently to the navigation concept, campaigns and activities are linked by a unambiguous path. Indeed, by click on a certain activity (either in the campaigns or clients level), the activity overview tab is displayed.

As illustrated in Figure 3.6, the activity tab contains the list of KPIs presented in section 3.2. These indicators are used to measure using numerical and tangible values the effectiveness of offline marketing investments and determine with a weighted average the overall score of the activity. The scoring system is explained in section 3.5.

The navigation concept is highlighted again on the top left corner of Figure 3.6. Indeed, it shows a nested navigation menu for navigating through the previous stakeholders by inverting the flow of the cascade effect.

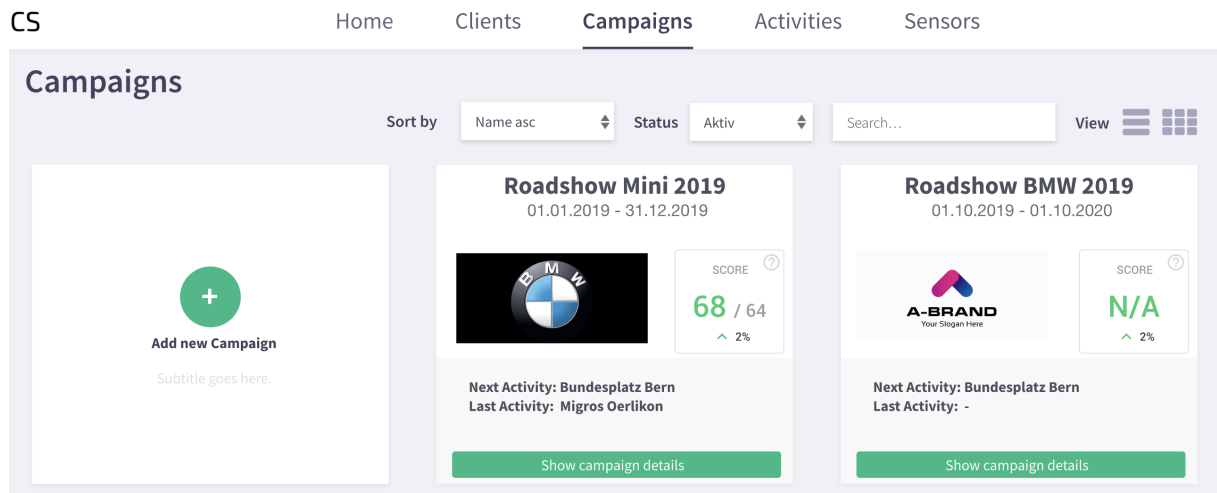


Figure 3.5: Design Evaluation: Campaigns Concept

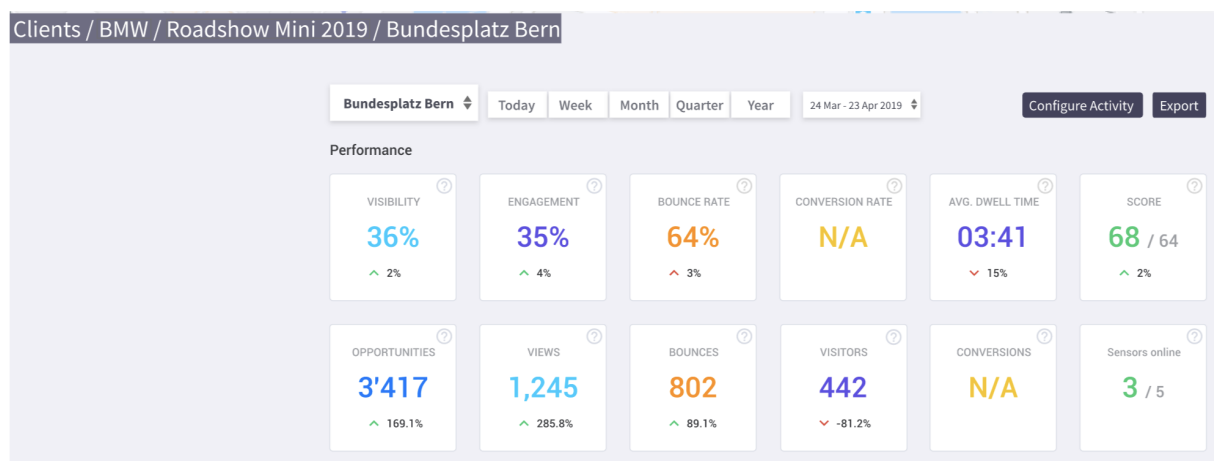


Figure 3.6: Design Evaluation: Activity Concept

2. Charts

Data visualization using multiple charts represent the key element of the dashboard. Indeed, the presentation of the collected data in a pictorial or graphical format is at the base of every analytical decision. Charts are necessary to identify patterns, give interactive visualization and visualize temporal and spatial dimensions in a single and compact format. In this regard, a list of relevant charts have been identified taking into account a set of metric, dimensions and activity-related factors.

For instance, as explained in section 3.1, there exist eighth different activity types. Each type can have different targets and characteristics which implies different analytics and data visualization for the activity itself. The main elements of the overall charts universe is described as follows:

- Funnel (3D) [38]

It is used to display the distribution of the collected data across the zone

of interests presented in section 3.1. Funnel Chart is also used to evaluate activities performances and see how effective a promotional team can turn interactions into closed deals. It can be used in 2D or 3D. Figure 3.7 illustrates the chart.

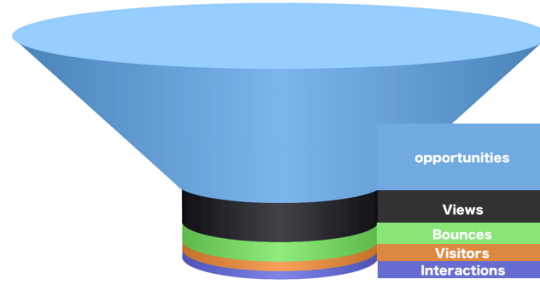


Figure 3.7: Charts universe: Funnel 3D

- Bar Chart [39]

It is used to visualize comparisons of data sets with discrete data. The focus is on values instead of categories since people read from left to right. Indeed, Bar charts can be useful to relate metrics such as frequency and average dwell time with the unique number of visitors for a certain activity. Bar Charts can be horizontal or vertical, depending on the use case. Figure 3.8 illustrates two different bar charts. The left one shows a comparison between frequency and unique count of visitors. The right one shows the average dwell time compared with the number of count by duration.

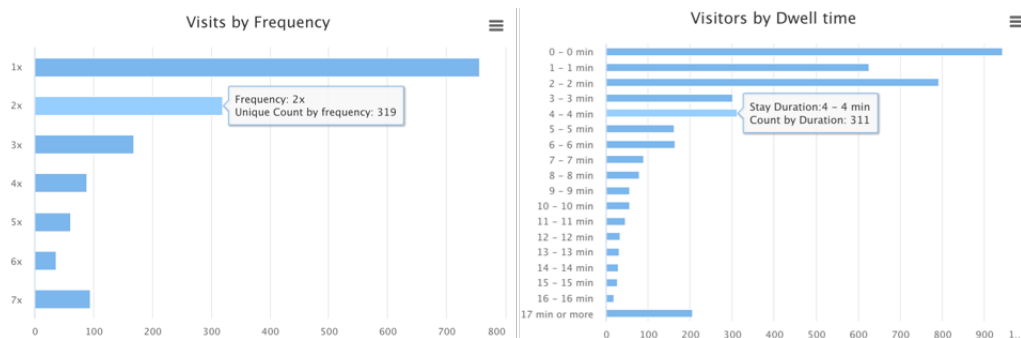


Figure 3.8: Charts universe: Bar Chart

- Heat Map [40]

Heat maps are graphical representation of data with individual values contained in a matrix. These values are represented with different colors shapes based on their intensity. They are used to relate metrics such as the average dwell time with different temporal dimensions (days and hours). Figure 3.9 illustrates a heat map with average dwell time sorted by hours and days.

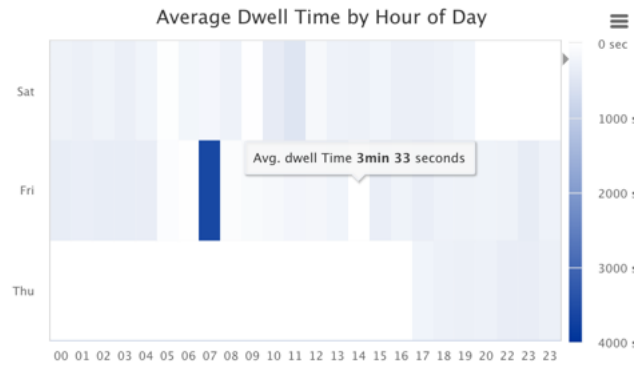


Figure 3.9: Charts universe: Heat Map

- Histogram Series [41]

Graphical representation of the data distribution. Histogram creates intervals (bins) and counts how many values fall into each bin. Histograms are useful to visualize and relate zone of interests with temporal dimensions and numerical metrics. Figure 3.10 shows the relationship between the metrics unique number of visitors, the temporal dimension days and the zone of interests wrapped into a single histogram.

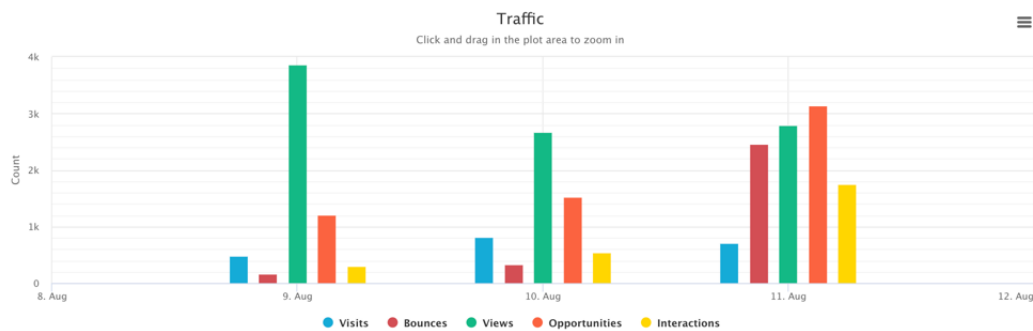


Figure 3.10: Charts universe: Histogram Series

- Line Chart [42]

The line chart is represented by a series of datapoints connected with a straight line. Line charts are most often used to visualize data that changes over time. Figure 3.11 illustrates a line chart which relates two metrics such as average dwell time and the visitor count with a temporal dimension expressed in hours.

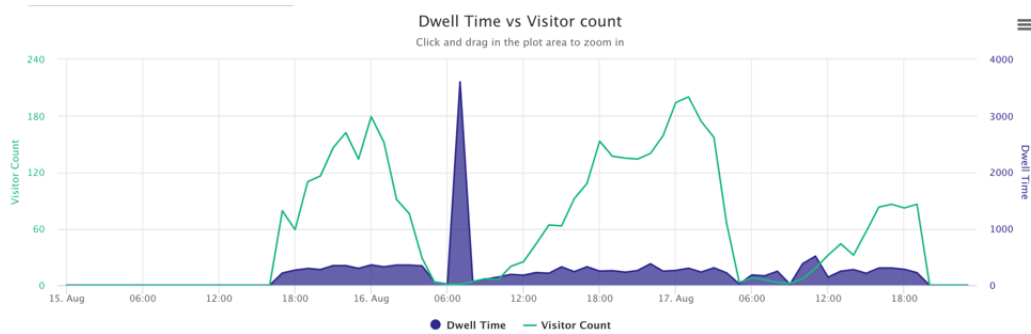


Figure 3.11: Charts universe: Line Chart

3. Cards

In recent literature, there has been a lot of interest towards creating modular UI/UX patterns that perform in a good manner across a variety of screens sizes. One of the most reused components consists of a *card*, *i.e.*, a snapshot-like display meant to stimulate users to click on it to discover more details of a certain element [43]. In a card-based design, the cards drive the inner navigation mechanism by connecting different application layers together. They can be very useful for grouping information together, inserting a variety of color shapes into an application and presenting a short summary of their content with links to additional details.

"A card is container for a few short, related pieces of information. It roughly resembles a playing card in size and shape, and is intended as a linked, short representation of a conceptual unit". [43]

In order to enhance the overall UX for the platform presented in this thesis, the strong hierarchy and dependencies among all stakeholders has been revisited by adopting a card-based design. Indeed, as already illustrated in Figures 3.4, 3.5, 3.5, stakeholders are represented by custom cards which present a short overview of the content and contain links to additional information. This allows the user to have a global overview with short pieces of information and to see more details just of the elements she/he is interested for.

Figure 3.12 illustrates the home tab of the dashboard. As explained before, these cards only have a limited amount of information wrapped into one digestible unit and are linked to further details about campaigns and activities. The header is showing the clients logos. Each card has a border around the grouped content, so that the visual-design elements inside the card gives a strong perception that they belong to the same entity.

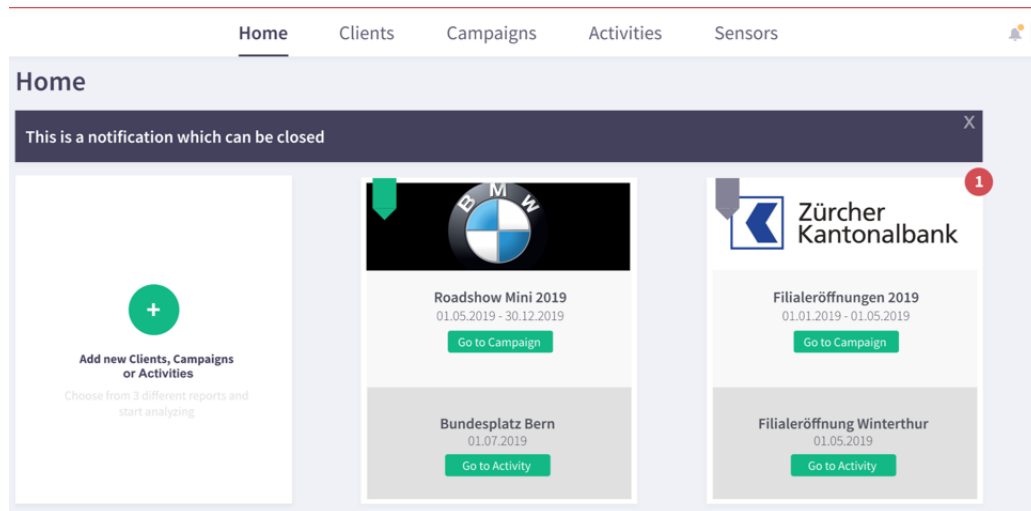


Figure 3.12: Card-based Design Principle

4. Tables

Despite to the fact that the default view of the platform follows a card-based principle, the dashboard offers the possibility to switch to a table-based format. Furthermore, there are certain elements (*e.g.*, sensors views) which need a well-structured representation via a tabular format due to the dynamism of the contained information. Indeed, as shown in Figure 3.13, sensors assigned to a certain activity emit continuously a variety of data which turns into different sensors states. These states are dynamically changing during the life-cycle of an activity and therefore need a more compact and extended representation which can be achieved with a tabular-based UI.

Sensor ID	Status	Next activity
Sensor #123	offline	01.08.2019, 10:00 - 14:00, Migros Oerlikon
Sensor #456	online	01.08.2019, 10:00 - 14:00, Migros Oerlikon
Sensor #789	online	01.08.2019, 10:00 - 14:00, Migros Oerlikon
Sensor #101	online	01.08.2019, 10:00 - 14:00, Migros Oerlikon
Sensor #111	not in use	01.08.2019, 10:00 - 14:00, Migros Oerlikon

Figure 3.13: Table-based Design Principle

3.4 Microservice-based Architecture

3.4.1 Monolithic vs. Microservices Architecture

Introduction

Traditional application models used to work with monolithic architectures, whose processes were tightly coupled, wrapped and exposed into a single API. This has as a consequence, that when one single API end point has a spike in demand, the entire instance has to be replicated in order for the application to be able to scale. This is not only costly but may lead to one single error being replicated multiple times. Furthermore, adding more functionalities into one single container might introduce more bugs as the code base grows.

This limitation makes it tedious to introduce new features, maintain the current code base and simultaneously be prepared to manage more incoming traffic in one single monolithic application. Since the high amount of tightly coupled processes, monolithic architectures are associated with the risk to have down-times and therefore have a leak in availability during their life cycles. Indeed, they are often characterized as applications with a *single point of failure*.

If on the one hand monolithic applications might result in application silos since they put all their core functionalities into a single container, on the other hand they are well-isolated, function-oriented and built to last rather than built to change.

With a microservices architecture (also known as *serverless*), the system can be built with independent components that run each application process as a service. Indeed, these services are loosely coupled, well-defined, encapsulated, independent from each other and exposed as a single container or API end point. Differently from a monolithic point of view, each service can be viewed as a reusable capability which needs to be as granular and abstract as possible.

The communication among these microservices is orchestrated via an API Gateway which is in charge of distributing the incoming traffic based on the demand. The API Gateway is also responsible for tasks such as application load balancing, caching of data, access control, API metering and monitoring [45]. Due to the fact that each microservice is isolated, it can be updated, scaled and deployed independently from each other.

If the application has a spike in demand in a specific service, only the relevant part can be scaled without having to replicate the whole application instance. The design principle followed by this architectural pattern is also known as *separation of concerns* [46] and aims at separating a orchestrated solution into distinct services, such that each service addresses a separate concern.

Modularity, and hence separation of concerns, is achieved by encapsulating information inside a microservice that has a well-defined interface. When concerns are well-separated, individual services can be reused, as well as developed and updated independently.

Database management

Despite the advantages aforementioned regarding microservices, a serverless architecture pattern significantly impacts the relationship between the database and the application layer [45]. In fact, the core principle of a microservice-based application consists of having for each service a different database schema to provide autonomy instead of sharing a global database structure across the whole application.

The main idea is to follow a decentralized design, which states that application data need to be persisted among multiple databases. Even though these databases can be hosted on different servers, a logical distinction needs to be made between their structures. Additionally, data need to be handled consistently in order to maintain a consistent application state.

Due to fact that a microservice-based architecture require multiple databases, transactions across services are not allowed making reporting and coordination more difficult within the application layer. This can have a huge impact on the data management once the application grows. Furthermore, this approach often results in code duplication by having redundant data stored in multiple independent databases owned by different services.

3.4.2 Trade-Off: Shared Database and Microservices

Although the ideal world scenario states that a microservice has to be responsible only for its own data and manage its own database, microservice-based architectures are often associated with some well-known challenges and need a couple of consideration to be made.

Indeed, stakeholders presented in section 3.1 are highly dependent on each other and rely on a set of resources that need to be shared with each other. Since the whole architecture is based on a *cascade* effect, once a specific resource is changed, other entities need to be updated in order to keep the data structure consistent.

For instance, if a marketing agency manages a group of five brands, the relationship can be viewed as one to n , where n is a undetermined number which ranges from zero to infinity. Let's image that one client of that agency has a running campaign. Similarly, the relationship between clients and campaigns is one to n , where n ranges again from zero to infinity.

Let's assume that the client needs to be deleted. This type of action would affect both relationships since the running campaign needs to be closed before the client can be deleted and then it has to be removed from the agency data structure.

In the current architecture, most of the database manipulation affect more than one of the entities presented in Figure 3.1. Because of this and due to the fact that a variety of services are highly related to each other, follow a complete microservice-based pattern might be tedious and over-engineered.

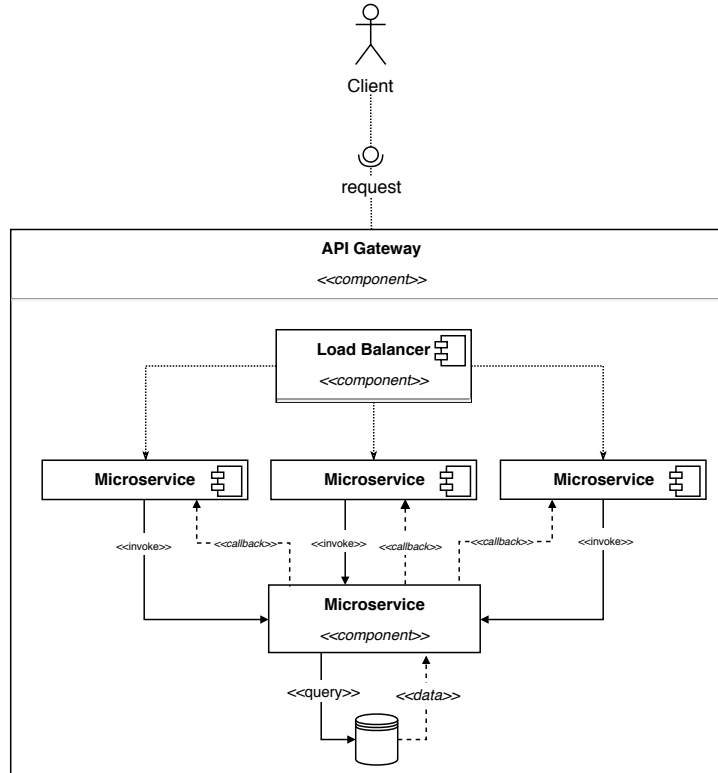


Figure 3.14: Serverless Architecture and Shared Database.

Therefore, the trade-off which has been taken consists of having a shared relational database for multiple microservices. However, despite the fact that there is a unique database shared by several services, for each micro application it is created and released a different database instance. This results in a still distributed approach. Hence, data integrity can be ensured and the database is capable of scaling by expanding its capacity based on the number of instances it has.

Figure 3.14 illustrates the design for the serverless architecture adopted in this thesis. The overall idea consists of having one microservice which is in charge of opening SQL connections, releasing database instances and sending queries to the database itself. This means, that if other micro functions need to perform data manipulations, they can invoke the microservice that is in charge of managing the database directly. This has the advantage that connection pools are managed by a single application. Therefore, if there would be a change in the way the database is queried or accessed, only one microservice would need to be adjusted.

On the one hand, this compromise does not introduce an anti-pattern nor contradicts the fundamentals of a microservice-based architecture. In fact, single services still have their own business logic and are well-encapsulated. On the other hand the same database can be queried and navigated through different collections by using multiple instances. This allows an easy communication across different services and reduces complexity and negotiation effort to resolve a simple request.

3.5 Benchmark

In order to find a standard procedure to be able to compare the performance among different marketing activities, a normalized benchmark needs to be considered. In this respect, an overall scoring system has been created with the goal to assign a normalized score to each marketing activity. By doing so, activities which have been setup with different parameters (*e.g.*, with different radius of the measurement zone, in different industry fields or in different locations) can be compared with each other using a standardised benchmark. In summary, the proposed method can be broken down into three steps:

1. Adjusting the value or KPI of a measurement on a common scale to ensure consistency and comparability.
2. Normalization (scoring) of the adjusted values or KPIs.
3. Construction of an index from a weighted basket of scores for each measurement which is distinct for each type of activity. For instance, "Sampling and Giveaway" or "Product or Service Demonstration" or more types presented in section 3.1.

The practical implication of this approach is that there will be no benchmark or average score as such for any measure. Instead, a specific measurement (activity) can be said to be better than, for example, 65% of similar measurements. For instance, a measurement X which has a visibility of 50% might be given a score of 57, meaning its profile was better than 57% of similar measurements. On the other hand, a different measurement Y which has a visibility of 55% might receive a score of 34, meaning its profile was only better than 34% of similar measurements. It would however not be possible to make a statement about average visibility, or rather, the average visibility score is 50% (better than 50% of similar measurements).

3.5.1 Adjustment of values and KPIs

Count and Unique Count

In order to make individual metrics comparable they have to be adjusted. This implies, that dimensions, metrics and parameters presented in Section 3.2 need to be calibrated and normalized on a common scale to ensure comparability.

In fact, while absolute counts are significant factors, their measurement can be positively influenced by increasing the size of the respective measurement zone and the duration of a measurement. Therefore, a comparable measure has to be normalized by area and duration.

In this regard, the total absolute (unique) count can be skewed by the measurement zone area in square meters and the measurement duration in hours. It is defined as follows:

The adjusted (unique) count is the average count per square meter and hour for a given zone.

Let's consider two different measurements for two different activities. On the one hand, measurement A has a count of 5'000. The radius of the measurement zone *interactions* was 2 meters and the measurement was conducted over a period of 4 days for 6 hours each day. On the other hand, measurement B has a count of 20'000. The radius of the measurement zone *interactions* was 10 meters and the measurement was conducted over a period of 7 days for 6 hours each day. Table 3.3 illustrates the aforementioned scenario.

<i>Measurement</i>	<i>Zone Signal Strength</i>	<i>Zone Radius</i>	<i>Zone Area (m²)</i>	<i>Measurement Days</i>	<i>Measurement Hours</i>	<i>Zone Count</i>	<i>Adjusted Count</i>
A	-44	2	12	4	24	5'000	17.4
B	-57	10	314	7	42	20'000	1.5

Table 3.3: Unique Count Adjustment to ensure Comparability between Marketing Activities.

Consequently, measurement A has an average count of 17.4 per hour and square meter, while Measurement B has an average count of 1.5 per hour and square meter. Therefore, although measurement A has a lower overall count, it has a higher adjusted count.

Average dwell time

To make average dwell times comparable across measurements, parameters need to be considered that define and influence its calculation. For instance, a measurement with a higher visit gap will automatically increase the overall average dwell time. As a recall, the visit GAP is a parameter which expresses the length of time in seconds after which a visit ends if a signal is not reacquired.

To compare average dwell times, an adjustment is needed by the range of min/-max thresholds and the size of the visit gap.

Let's consider two different measurements of two different activities for which the measurement that has the higher adjusted average dwell time needs to be found. On the one hand, measurement A has a minimum dwell time threshold of 120 seconds, a maximum dwell time threshold of 10'800 seconds, and a visit gap of 300 seconds. The average dwell time is 617 seconds. On the other hand, measurement B has a minimum dwell time threshold of 120 seconds, a maximum dwell time threshold of 10'800 seconds, and a visit gap of 1'800 seconds. The average dwell time is 1'305 seconds.

First of all, in order to compute the higher adjusted average dwell time, min-max scaling needs to be adopted to put the average dwell time into a range between 0 and 1.

Table 3.4 illustrates the aforementioned scenario.

<i>Measurement</i>	<i>Min. Dwell Time</i>	<i>Max. Dwell Time</i>	<i>Visit Gap</i>	<i>Avg. Dwell Time</i>	<i>Scaled Dwell Time</i>
A	120	10'800	300	617	0.05
B	120	10'800	1800	1305	0.11

Table 3.4: Normalized Average Dwell Time.

With Min-max scaling values of 0.05 and 0.11 respectively for measurements A and B are computed. But Measurement B has a much higher visit gap which might be the reason for measurement B's higher average dwell time. Therefore, a further adjustment needs to be made to compensate for the larger visit gap.

Finally, the scaled dwell time is multiplied by the gap adjusted dwell time to get the adjusted dwell time, which is normalized by the visit gap and the dwell time intervals.

Therefore, as shown by Table 3.5, while measurement B had a higher average dwell time it performed worse than measurement A within the given constraints.

<i>Measurement</i>	<i>Avg. Dwell Time</i>	<i>Scaled Dwell Time</i>	<i>Gap adjusted Dwell Time</i>	<i>Adjusted Dwell Time</i>
A	617	0.05	3	0.14
B	1305	0.11	1	0.11

Table 3.5: Adjusted Dwell Time by Visit Gap and Dwell Time Thresholds.

Ratios

Ratios such as visibility, engagement, and interaction rate are relations between counts, which represent the stages of the offline funnel. By defining zone radiuses which are very close to each other, it is possible to influence ratios positively. For example, a views zone radius of 50 meters with a visibility of 100% is less significant than a visibility of 100% in a views zone with a radius of 5 meters.

Furthermore, when zone counts differ by minimum dwell time thresholds, a small delta between these thresholds will positively impact the ratio between their counts. For instance, if zone count visits is compared with zone count interactions and both zones differ by minimum dwell time, the interaction rate is inversely proportional to the thresholds delta between these zones. In other words, the lower the minimum dwell time delta, the better the interaction rate.

To be able to compare ratios, an adjustment is needed by taking the deltas of their maximum distance and minimum dwell time thresholds into account.

In this regard, there are two factors which might affect the most the computation of ratios. The first one consists of the difference in distance thresholds between the two zone of interests of the ratio. The second one consists of the difference in dwell time thresholds between the two zone of interests of the ratio.

Visibility

Let's consider two different measurements of two different activities for which the measurement that has the higher adjusted visibility needs to be found. Measurement A has a visibility of 35%. The radius of the views zone was 15 meters. The approximate range of the sensor is 50 meters. Measurement B has a visibility of 75% . The radius of the views zone was 40 meters. The approximate range of the sensor is 50 meters.

As explained in Chapter 3.2, visibility is defined as the ratio between the count of the total sensor range (*i.e.*, opportunities zone) and the count of the views zone. Consequently, a larger views zone radius might be biased towards a higher visibility. Again, min-max scaling can be used to derive a score for the delta of the zone radiuses in the ratio. On a scale between 0 and 1, a larger delta between zone radiuses will be closer to 1.

In general, a low visibility does not strictly imply a low adjusted visibility. Indeed, while Measurement A has a lower visibility ratio than Measurement B, its adjusted visibility is higher as shown by Table 3.6.

<i>Measurement</i>	<i>Views Radius</i>	<i>Opport. Radius</i>	<i>Count Views</i>	<i>Count Opport.</i>	<i>Visibility</i>	<i>Adjusted Visibility</i>
A	15	50	35	100	35%	24.5%
B	40	50	75	100	75%	15.0%

Table 3.6: Adjusted Visibility.

Interaction Rate

The interaction rate is the ratio between the count of the visitor zone and the count of the interaction zone. Generally, the radiuses of these zones are the same. The difference is in the minimum dwell time threshold.

As with zone radiuses, a small delta between dwell time thresholds will positively impact the ratio. As before, min-max scaling can be used to score the significance in the difference between dwell time thresholds on a scale between 0 and 1 where a larger delta is closer to 1.

Let's consider two different measurements of two different activities for which the measurement that has the higher adjusted interaction rate needs to be found. Measurement A has an interaction rate of 25%. The minimum dwell time for visitors is 60 seconds

and for interactions is 300 seconds. Measurement B has an interaction rate of 70%. The minimum dwell time for visitors is 60 seconds and for interactions is 120 seconds.

Firstly, the interaction rate for a measurement x is computed as the ratio between the interactions count and the visitors count.

Then, similarly to the approach aforementioned for computing the scaling factor for visibility, the scaling factor for the interaction rate of the measurement x needs to be computed.

These values are defined in the platform as a threshold based on a practical approach. This means, that users cannot setup the max. dwell time to more than 600 seconds and to less than 1 second.

Finally, as shown in Table 3.7, the adjusted interaction rate of the measurement x can be calculated as a multiplication between the scaling factor and actual interaction rate.

<i>Measurement</i>	<i>Visitors $\min(\Delta)$</i>	<i>Inter. $\min(\Delta)$</i>	<i>Count Inter.</i>	<i>Count Visitors</i>	<i>Inter. Rate</i>	<i>Adjusted Inter. Rate</i>
A	60	300	25	100	25%	40.0%
B	60	120	70	100	70%	10.0%

Table 3.7: Adjusted Interaction Rate.

3.5.2 Normalization

Standard Score (z-score)

The standard score algorithm can be used to determine the position of an adjusted value or ratio relative to all other observed, adjusted values or ratios. For example, a standard score of -1.9 means that the adjusted amount or ratio is better than 28.7% of all other adjusted values or ratios (the percentage equivalent -1.9 to can be derived from a z-score [96] table).

3.5.3 Index

Finally, the index I^a (or combined score) for a particular activity type (such as "Sampling and Giveaway" or "Product or Service Demonstration") is constructed from a weighted basket of individual scores from the amounts and ratios that are most relevant for the respective activity type.

3.6 Predictions

This section describes the overall approach used for predicting how different marketing activities would perform in a similar context based on historical data. Firstly, the features used to train the ML model are explained. Afterwards, the selection of the ML algorithm is described and finally the final adopted solution is presented.

3.6.1 Features Selection

Introduction

In ML, the process of selecting a subset of relevant features (or predictors) for model construction is also known as feature selection [89]. This process consists of finding a subset of features from the dataset, which contribute the most to the target prediction variable and influence the most the outcome of the model once it gets trained. The overall goal is to avoid irrelevant features which might decrease the accuracy of the model and to select the most relevant ones which minimize the error rate of it [90].

Usually, in order to select the best predictor subsets, search techniques for proposing new feature subsets in combination with an evaluation procedure which scores them can be adopted. One search technique consists of testing all different subsets of predictors and find the one which reduces the error rate the most. This procedure can be very expensive for large datasets in terms of computational power, since each possible subset combination need to be evaluated. However, for small datasets where the amount of features is limited, a simpler approach can be adopted. Indeed, feature selection for little datasets can be performed by human intuition and simple features correlation. This is only possible, when all features are known beforehand and when the correlation between non-target features and target variables follows an intuitive and logic structure.

In order to consider different scenarios and to train different models using different feature subsets combinations, tools like Rapid Miner [91] or Domino [92] can be used to speed up the feature selection process. These tools allow the automate creation of predictive models and the evaluation of the outcome of different ML algorithms in terms of error rate. Because of this, the best performing feature subset can be used to train the predictive model and the corresponding design can be used on an implementation level.

Data Quality

The data quality assessment process aims at discarding data columns (attributes) that provide less value for the model. In order to achieve this, data patterns need to be identified for extracting which attributes are valuable and which are worthless. In fact, data without any standard deviation or without any discernible patterns are considered to be likely irrelevant for the prediction model. In this regard, the following five quality measures extracted with Rapid Miner [91] have been identified in order to assess the dataset quality:

- **Correlation (C)**

It measures the linear correlation between the target variable and the concerned non-target feature, *i.e.*, how these two variables are dependent upon each other. Correlation can be either positive, when both variables change in the same direction, negative, when they are negatively correlated and change in opposite directions or neutral, which means that there is no relationships between them [93].

- **Uniqueness (U)**

It measures the extent to which the feature in question is similar to an ID. The ID-ness for a certain feature can be computed as the fraction of unique values by using the following Formula:

$$U = \frac{\# \text{ different values}}{\# \text{ data rows}}$$

- **Constancy (S)**

It measures how identical or constant the concerned feature column is. This quality measure firstly removes all missing values (*i.e.*, *null* values) from the feature and then can be computed as the fraction of constant non-missing values by using the following Formula:

$$S = \frac{\# \text{ data rows with the most frequent value}}{\# \text{ data rows}}$$

- **Absence (A)**

It measures the amount of null values for the concerned feature. This measure can be expressed as a fraction of missing values:

$$A = \frac{\# \text{ missing values}}{\# \text{ data rows}}$$

- **Text-ness (T)**

It indicates how text-like the concerned feature is. Text-ness can be computed as the average of the ID-ness, the fraction of cells containing token limiters and a length-based score to state whether the feature contains free text or not.

Typically, on the one hand, features columns with low values for Stability, ID-ness or Missing are preferred. This is because, these type of data are usually characterized by a low standard deviation and therefore provide less value for the training model. The more stability or constancy in the data, the more difficult would be to have a generalized and well-trained prediction model. On the other hand, features with a high Correlation might positively affect the outcome of the prediction. This only when, the high correlation does not occur because of a direct cause-and-effect relationship with the target value. This error occurs when the prediction model assumes that one event A must cause another event B simply because the events happen together. More formally, this error implies that the occurrence A is the cause of B simply because A and B are in regular conjunction.

In order to make a decision whether a certain feature has to be discarded or has to be part of the feature subset, three different categories have been considered according to [94]:

- **Poor quality features**

It indicates a feature column of poor quality, which usually needs be removed from the feature subset. A feature within this category has encountered one of the following concerns:

1. More than 65% of all values for the concerned feature in this column are *absent*.
2. The concerned feature has almost all values that resemble an *ID* and at the same time they are not *text-like* values.
3. The concerned feature has almost all values constant, *i.e.*, more than 90% of all values being *constant*.
4. The feature values have a correlation of lower than 0.0001% or higher than 95% with the target value.

- **Irrelevant or uncategorized features**

It indicates those features which are irrelevant for the predictive model and could be discarded without affecting its outcome. By removing these type of features, the model building process might speed up. A feature within this category has encountered one of the following concerns:

1. The concerned feature has more than 85% of all values that resemble an *ID* and at the same time they are *text-like* values.
2. The feature in question has correlation of less than 0.01%, which indicates that it is unlikely to contribute to the prediction outcome.
3. The feature values have a correlation of more than 40% with the target value. This is considered to be an *uncategorized* feature. As explained before, predictions outcome with features with a high correlation feature is tightly dependent on the dataset. This means, that the predictive model might perform better or worse when this feature is included.

- **Relevant features**

Features which are relevant and of high-quality for the predictive model. The elimination of one of these feature might tightly affect the model performance.

Activity-related dataset

In order to predict the outcome of marketing activities based on historical interactions, the dataset needs to be enriched with a set of activity-related attributes. To achieve this, different utilities tables in the DB have been already joined together, so that there is an overview of all possible activity-related data in one single view. In particular, historical weather data are fetched from Open Weather Map API [95] and subsequently mapped to the activity table using the *city_id* as foreign key. By doing so, the dataset has been enriched with historical weather data which might influence the amount of visitors in a certain event.

The dataset for marketing activities is composed by the following data attributes:

- ***id*** (*e.g.*, 7515aacb-314b-467f-afe1-f0f425b0cf5d)
The activity unique identifier, represented in a token-like format
- ***datetime*** (*e.g.*, 2019-06-06 00:00:00)
The start date of the activity. The date is expressed in a US-format and the counter unit is days.
- ***agency_id*** (*e.g.*, a5c02d91-d961-42b8-bf3c-c05f2b23dcd9)
The agency unique identifier, represented in a token-like format. The agency represents that entity which manages the brand that performs the promotional activity. It is used as foreign key to link agencies and activities.
- ***campaign_id*** (*e.g.*, 2dfae87a-868f-42c6-84c4-0cd837690fbe)
The campaign unique identifier, represented in a token-like format. The campaign represents that entity which contains the promotional activity. It is used as foreign key to link campaigns and activities.
- ***name*** (*e.g.*, Lounge (Village))
The activity name expressed in a free-text format.
- ***industry_type*** (*e.g.*, Food and Drink)
The industry field of the brand which is responsible to run the activity. Its value belongs to the enumeration: *Arts and Entertainment, Automotive, Beauty and Fitness, Books and Literature, Finance, Food and Drink, Home and Garden, Internet and Telecom, Jobs and Education, Law and Government, News, Online Communities, People and Society, Pets and Animals, Real Estate, Shopping, Technology, Other*.
- ***activity_type*** (*e.g.*, Sampling or Giveaway)
The activity type. Its value belongs to the enumeration of activity types presented in Section 3.1.
- ***activity_format*** (*e.g.*, Festival)
The activity format. Its value belongs to the enumeration: *Bar / Club, Cafe Restaurant, Retail, Festival, Street, Trade Show, Mall, Train Station / Airport, Other*.
- ***campaign_name*** (*e.g.*, Livealytics Festivals - Village)
The name of the activity's campaign expressed in a free-text format.
- ***zone_of_interest*** (*e.g.*, Opportunities)
The zone of interest of the corresponding activity. Each activity has at least five entries split by the zone of interests presented in 3.2.
- ***city_id*** (*e.g.*, 2660253)
The unique identifier of the city where the activity took place. Expressed in a numeric format.
- ***city*** (*e.g.*, Greenfield)
The name of the city where the activity took place. Express in a free-text format.

- **latitude** (*e.g.*, 46.67494)
Latitude coordinate value expressed in a numerical value for the city where the activity took place.
- **longitude** (*e.g.*, 7.8778)
Longitude coordinate value expressed in a numerical value for the city where the activity took place.
- **count** (*e.g.*, 2166)
Metric presented in 3.2, which expresses in a numerical value the amount of MAC addresses counted within a given time period for the concerned activity.
- **unique_count** (*e.g.*, 771)
Metric presented in 3.2, which expresses in a numerical value the amount of unique MAC addresses counted within a given time period for the concerned activity.
- **average_dwell_time** (*e.g.*, 244)
Metric presented in 3.2, which expresses with a numerical value the average visit length distributed over a time period for the concerned activity.
- **weather** (*e.g.*, cloudy)
The historical weather data recorded at the start date and in the city for concerned activity. Expressed in a large enumeration: *sunny, partially sunny, cloudy, partially cloudy, etc.*
- **temperature** (*e.g.*, 22.3)
The historical temperature data recorded at the start date and in the city for concerned activity. Expressed in a numerical value which ranges from -50 to +50.

In order to predict the outcome of marketing activities, numerical metrics need to be considered. Thus, the overall approach consists of setting as target columns for the predictive model counts, unique counts and average dwell time. For instance, the ML-based algorithm should be able to predict the unique counts or dwell-times for a certain zone of interest for a specific activity considering all its related data. Therefore, the overall goal consists of finding the best feature subset which most contribute to the prediction.

Based on the data quality assessment aforementioned, the most relevant attributes need to be extracted from this dataset in order to train the predictive model. In addition to this, a practical approach is conducted with RapidMiner in order to find the best feature subset.

Several attributes such as *id*, *agency_id*, *campaign_id*, etc. are unique to each activity. In fact, they are equivalent to IDs and the predictive model cannot learn anything from them. These attributes are part of the poor quality features category and therefore need to be removed from the feature subset.

Additionally, free-text columns such as *name*, *campaign_name*, etc. have a high stability and textness. The high stability in the data is due to the fact, that there are multiple entries in the dataset for the same activity. This is because, activities data are aggregated

per day. Therefore, if an activity lasts for two weeks, it will have 14 entries in the dataset. This makes the activity name to appear several times in the same dataset and thus to be considered a free-text constant in the data quality assessment process.

Data concerning geographical attributes such as *latitude*, *longitude* or *place* need to be enriched with statements that the predictive model is able to interpret. In this regard, statements regarding weather the activity took place close to a city with a high population density, and consequently with the possibility of having a higher influx of people, might enrich the outcome of the model. For instance, coordinates can be used to determine whether the location of an input city is within a radius of 30 kilometers of a high populated city.

Industry_type, *activity_format* or *activity_type* are highly correlated to the target columns. Indeed, these type of data are decisive in determining the number of people who have been reached through the activity. For instance, they might exist industry fields which are much more popular than others or formats that are designed for a larger mass of people (*e.g.*, Festivals) and thus might attract more people. Similarly, an activity type might be designed to unintentionally attract passers-by in an open area (*e.g.*, Giveaway) or to promote the product exposure without interacting with the consumer, where no active promotional staff involved (*e.g.*, Display and Window). Different activity types will result into different influx of people. These features are all relevant and of high-quality and need to be considered in the training process.

Finally, weather data might determine the outcome of an activity, especially if it has been organised outdoors. Indeed, a festival with nasty weather will most likely have a lower influx than another activity format arranged on a sunny day.

Chapter 4

Implementation

This chapter presents the concrete implementation details of the design proposed in the previous chapter. In order to achieve the design goals, the implementation consists of a compact Web application which interacts with the AWS serverless backend and the IoT system. Thus, this chapter firstly presents the front-end part, then it goes through the overall implementation of the AWS microservice-based architecture and finally it describes the IoT system.

4.1 User Interface

4.1.1 React

For the User Interface (UI) for the Web application the ReactJs Library [106] is used. ReactJs is a JavaScript lightweight library for building fast and efficiently web apps. In order to empower the library, React is used in combination with JSX [107], a XML-like syntax extension built on top of JavaScript. In this sense, JSX adds XML elements to JavaScript such as tags, which can have attributes, children and other properties. Each JSX element must be in line with the strictly hierarchical tags structure. Finally, JSX can be easily combined with CSS and HTML [108] and thus add power to the React rendering process.

Visuals

At the top of React, the front-end relies on the Argon Design System from Creative Tim [109] for the design of basic web components, such as Modals, Buttons, Input fields, Tables, Label, etc.

Concerning style maintenance and HTML tags management, the platform relies on the well-known external library *styled-components* [110]. This library, enhances CSS in order to style React component systems and provides several advantages [36], such as:

- No more bugs with unique Class Names
The library generates unique class names during the rendering process and ensures that there are no duplicates for different HTML tags.
- Conditional Styling
CSS elements are like React components. Using props [112] in combination with Template Literals syntax [111] CSS attributes can be dynamically styled without having to manually manage tons of external CSS classes.
- Maintenance
Searching for styled elements is the same as searching for React components. Everything is compact and stays in the same file. In fact, styled-components avoids time-consuming searches for finding the styling affecting a specific component.

4.1.2 Structure

The front-end structure is modular and well-encapsulated. The following directory tree shows a high-level overview of the front-end structure.

```
├── frontend
│   ├── api
│   ├── assets
│   ├── constants
│   ├── components
│   ├── helpers
│   ├── redux
│   ├── routes
│   └── services
```

API

The API sub-folder exposes a configuration file, containing a declarative interface for the communication with the AWS serverless API. In this regard, Axios [117] has been integrated to simplify the HTTP communication between the front-end and back-end. Axios is one of the most used promise based HTTP clients. It can be integrated with different browsers and Node.js applications.

As illustrated by Listing 4.1, the axios instance needs to be initialized by passing the relevant configuration parameters (lines 2-5). Required parameters are HTTP *headers*, *i.e.*, fields of the header section which are needed to request and response messages over the HTTPS protocol and *baseURL*, which contains the API end point path. The latter is kept dynamic (line 1) in order to be able to switch between different environments (*e.g.*, dev, stage and prod). Indeed, each API path specified in a *env.* file which points to a different end point, *i.e.*, to a different stack in AWS.

Afterwards, the API can be exposed and accessed in the whole client. In lines 6-15, an example can be found showing how to expose simple API calls for users end points.

```
1  const { REACT_APP_API_PATH_MAIN, REACT_APP_API_PATH_IOT } = process.env;
2  const instance = axios.create({
3    baseURL: `${REACT_APP_API_PATH_MAIN}`,
4    headers: { "Content-Type": "application/json" }
5  });
6  const API = {
7    users: {
8      getUsers: () => instance.get("/users"),
9      editUser: data => instance.put(`/users/${data.id}`, data),
10     deleteUser: ({ id, email }) =>
11       instance.post("/users/delete", { id, email }),
12     getMyUser: () => instance.get("/users/me"),
13     ...
14   }
15 }
```

Listing 4.1: API - Axios Instances

Assets

This category contains shared static assets for the whole front-end. Indeed, it includes static images, global CSS files, JSON files for rendering Lottie animations [118], icons and font files. Assets are a core part of the JavaScript bundle and yield some of the largest byte savings in order to faster the application. For instance, on-demand loading of assets might already improve the performance of the Web application.

Constants

This category consists of a variety of Javascript exported variables which can be statically shared along the platform. For instance, a string for a common message in a Modal or the colors palette.

Components

In general, this category follows the React pattern, which states that a Web application must be divided among different reusable Components. In this regard, a component can be either functional, *i.e.*, a stateless view [119] that displays a set of passed properties (props) [112] or a class, *i.e.*, a stateful component [119] that encapsulates and manages its own state [112] and logic.

Usually, the latter are those components which are in charge of sending asynchronous requests to the serverless back-end, change the state that is locally encapsulated within the component and pass the fetched data through the application tree using read-only props.

In other words, stateless components are responsible for *"How things look"* in the application, while stateful components are responsible for *"How things work"* by considering dynamic states and actions that can change at any time in the components life cycles.

Stateful components represents the logical layer of the platform. They are responsible for storing application data and defining the different behaviour of the child components based on an internal state change. They can be presentational (*i.e.*, , manage the state for an input field) or logical (*i.e.*, fetching data from the back-end and pass them along the application tree).

As mentioned before, React components are styled using styled-components, which represents the foundation of the application's design. Indeed, this module enhances normal CSS and removes the mapping between components and styles by creating normal React components instead of external CSS classes.

The application exploits this module in order to have reusable styled components that can be shared along the whole platform and thus improving style consistency. For instance, a Button or a Input field can be imported into different React components and have the same style structure. Based on different props, they can be conditionally styled as well.

For instance, consider the stateless component illustrated in Listing 4.2, which is charge of rendering images for different contexts (*e.g.*, user profile picture or the logo of a brand).

```

1  import React from "react";
2  import styled from "styled-components";
3
4  const Container = styled.div `
5    display: flex;
6  `;
7  const Figure = styled.img `
8    height: ${props => (props.height ? `${props.height}px` : "auto")};
9  `;
10
11 export const Image = ({ style, height, source, alt }) => {
12   return (
13     <Container>
14       <Figure style={style} height={height} src={source} alt={alt} />
15     </Container>
16   );
17 };

```

Listing 4.2: Reusable Image Component

As shown in the code snippet above, there is no mapping between the component and its styling using external CSS classes, but everything can be represented in the same file. Indeed, at lines 4-8, native HTML tags are overwritten with a styled-components instance called *styled* (lines 4 and 7).

The condition for having different styling behaviours can be found at the line 8. Indeed, once a *height* attribute is provided to the Image component from a parent node, its value in pixels is given to the *img* native HTML tag. For instance, a user profile picture which

has always the same height sets the value to *120px*.

```
1      height: 120px;
```

Otherwise, if no height is passed, the default behaviour is set to *auto*.

```
1      height: auto;
```

The condition at line 8 can be embedded using template literals syntax [111] and binary operators [120] without having to use external CSS classes. By doing so, stateless components can be conditionally styled, and therefore reused and anonymised, from parents nodes by passing a set of different props.

Helpers

This section consists of a variety of utility elements, *i.e.*, descriptive methods or variables that can be reused among the whole application. The majority of these elements, are highly related to the style and layout of the platform.

Redux

In order to easily manage different states simultaneously, Redux [113] was integrated. Redux is a popular well-known predictable state container for building state-full React and Javascripts applications. Indeed, it helps the application to maintain a global state in the Redux store [114] and to share it among different components in the hierarchy. Subsequently, all concerned units can be updated as a result of a change in the global state.

For this reason, most of the components in the front-end are subscribed to the store and are listening to state diffs. Once a certain attribute in the global state is changed, the concerned components are re-rendered and therefore updated.

Redux principles consists of having multiple Reducers [116] and Actions [115], and one Store. Actions are used for payloads of information that are directly sent to the global store, which in turn holds the different application states together. Reducers define how a state shall change when a certain action is executed. In other words, an action can be viewed as a trigger for a reducer which defines how an application's state changes. The store keeps track of the application states and gives access to different component to access them in the application tree.

Listing 4.3 illustrates the initialization of the Redux store. Lines 5-12 show how different reducers for core components (*e.g.*, users, clients, agencies, etc.) are combined with *combineReducers* from Redux. Subsequently, the *rootReducer* object containing a set of reducers is passed as an argument to *createStore* from Redux (lines 15-18).

```

1  import { createStore, combineReducers } from "redux";
2  ...
3  export const initStore = () => {
4    const rootReducer = () => {
5      combineReducers({
6        authentication,
7        users,
8        clients,
9        agencies,
10       campaign,
11       activities,
12     });
13   }
14
15   const store = createStore(
16     rootReducer,
17     ...
18   );
19
20   return store;
21 }

```

Listing 4.3: Redux Store Initialization

Routes

This category contains React Routers. React Routers are built at the top of React and are used for core dynamic routing in the front-end. They define navigation principles, such as render components based on a set of conditions (*e.g.*, matching a URL path).

For instance, as illustrated by Listing 4.4, private routes which require a valid session to be accessed can be protected by checking whether the passed user is logged in or not (line 5). If the user is successfully authenticated, the corresponding component is rendered (line 6), otherwise the user is redirected to the previous location (lines 8-10).

```

1  const PrivateRoute = ({ component: Component, ...user }) => (
2    <Route
3      {...user}
4      render={props => {
5        return user.isLoggedIn ?
6          <Component {...props} />
7        } : (
8          <Redirect
9            to={{ state: { from: props.location } }}
10         />
11        );
12      }}
13    />
14 );

```

Listing 4.4: React Navigation with Private Routes

Services

This category contains interfaces to AWS Amplify [121], which exposes AWS modules to manage authentication patterns with AWS Cognito [57] and other cloud services. Indeed, Amplify manages for whole web app users sessions and authorization tokens and provides a declarative interface.

As illustrated by Listing 4.5, this interface wraps authentication-related exported methods, such as *sign in*, *sign out*, *forgot password*, etc. Furthermore, it is able to return the current authenticated user, which can then be propagated through Redux in the whole app.

```
1 import { Auth } from "aws-amplify";
2
3 export const login = (username, password) => {
4   return Auth.signIn(username, password)
5     .then(user => user);
6 }
7
8 export const forgotPassword = (useremail) => {
9   return Auth.forgotPassword(useremail)
10    .then(data => data);
11 }
12 export const logout = () => {
13   Auth.signOut({ global: true })
14     .then(data => data);
15 }
```

Listing 4.5: AWS Amplify and Authentication Patterns

4.2 AWS Serverless Architecture

4.2.1 AWS Core Components

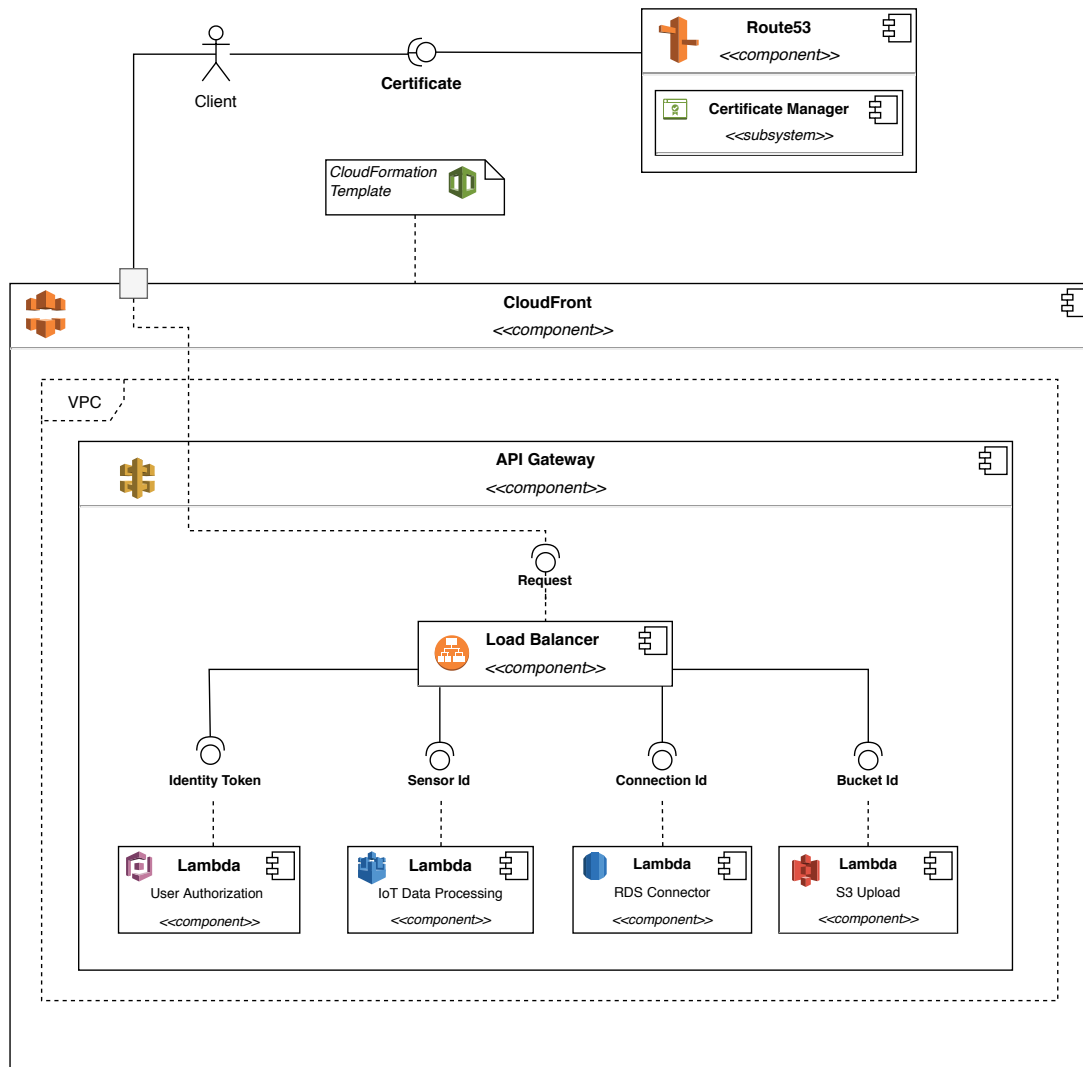


Figure 4.1: Microservice-oriented Architecture: API Gateway and Lambda Functions.

In order to achieve a microservices-based architecture and simultaneously rely on a state of the art infrastructure, the final solution has been designed within the AWS ecosystem [47]. Inside this ecosystem there exist a wide variety of services that can be used for building serverless architectures and are designed to harmoniously coexist. Figure 4.1 illustrates the concrete implementation using AWS as main infrastructure of the design shown in Figure 3.14.

Route 53

First of all, application clients need to be routed via a DNS web service in order to map human-friendly domains into numeric IP addresses that routers can use to connect to each other. In this sense, the AWS Route 53 [48] has been adopted to route end users to internet applications. AWS Route 53 is DNS web service designed to be fully compliant with IPv6 [62] protocols and can be used to configure DNS health checks to route traffic to different endpoints endpoints and monitoring the health of the application. Furthermore, it is meant to work well with other AWS features inside Cloud Front [50], is cost-effective and highly available and scalable.

AWS Certificate Manager

In order for Route 53 to be able to route end users to a secure internet domain, AWS Certificate Manager [49] has been integrated to manage the life cycles of internet certificates. Nowadays, SSL/TLS [64] certificates are industry standard protocols and security measures used for the encryption of network communication. In order to ensure a secure connection for sensitive data in transit, a trusted certification authority is needed to verify the identity of the website's owner [63]. In this regard, AWS Certificate Manager is used as a service that provisions and deploys public and private SSL/TLS certificates to secure network communication and establish websites identities. Additionally, this service handles certificates renewals and allows the creation of private certificates for internal resources and manages the certificate life cycles centrally.

AWS Cloud Front

All these internet resources are connected to each other live in Cloud Front [50], a highly secure and programmable Content Delivery Network (CDN). Cloud Front enables secure applications delivery with low latency and high transfer speed and is at the base of each serverless architecture within the AWS ecosystem.

AWS Virtual Private Cloud

Within Cloud Front, multiple AWS Virtual Private Cloud (VPC) [53] instances can be created to provision different logically isolated partitions of AWS. Each VPC instance has its own virtual networking environment, including the selection of different IP address ranges, creation of subnets and configuration of route tables and network gateways. The network configuration is highly customizable and allows to create public or private-facing subnets in order to differentiate which resources are able to access the internet (*e.g.*, web servers) and which back-end systems need to be encapsulated into a private-facing subnet with no internet access (*e.g.*, databases or application servers). In general, VPCs consist of different security layers and network access control lists.

AWS API Gateway

As mentioned before, a VPC is logically partitioned with a set of AWS resources, which need to communicate to each other using a well-defined structure. In a AWS serverless architecture, an AWS API Gateway [54] is the core of any API management solution in terms of creation, publication, maintenance and monitoring of secure REST [66] APIs. Indeed, it acts as the single entryway for every internet [65] request and is responsible for orchestrating multiple microservices living within the same application layer. Thanks to the API Gateway, these microservices can act cohesively since every API call is reliably processed by one single entity. In fact, API Gateway handles all requests involved in accepting and processing a large amount of concurrent API calls, including traffic management, authorization and access control for specific routes, monitoring, and API version management. Furthermore, it allows to run multiple versions of the same API, by making the testing and release of new API versions easier and quicker. This is because, developers can work and test the behavior of their code base in the cloud already by using a different API instance. This allows new features to be easily integrated in production by simply merging multiple API instances. The AWS API Gateway [54] has a flexible security control by authorizing users and groups with AWS Identity and Access Management (IAM) roles [61] and AWS Cognito [57].

AWS CloudFormation

A scalable microservice-based architecture needs a well-defined structure in order to be maintainable. In this regard, AWS Cloud Formation [51] provides a common language to describe and provision the infrastructure deployed to the cloud. Indeed, multiple templates can be modeled which contain declarative APIs that serve as input data for the API Gateway for quick and reliable provisioning of the services or applications. Based on the template code, a new *stack* is created within Cloud Formation, which can be viewed as a collection of AWS resources that can be managed as a single unit [52]. Stacks might be very useful for encapsulating different API version into a single container. For instance, if a certain API version is no longer required, by simply deleting the stack all of its related resources are accordingly removed from the cloud. Thanks to Cloud Formation, all resources needed and deployed to the cloud are declared and standardized using a single source of truth for the whole application.

AWS Elastic Load Balancing

In order for an application to be able to scale, the incoming network traffic must be intelligently distributed across multiple targets, such as microservices or application containers. In this regard, AWS Elastic Load Balancing [56] automatically distributes incoming application traffic across multiple lambda functions [55] and thus it increases availability and reliability of the system. The Application Load Balancer is one of three types of AWS Elastic Load Balancing and is best suited for balancing applications which communicate over the HTTP and HTTPS protocols and provides the ability to route traffic

within VPCs. Furthermore, it is capable of handling unexpected changes in network traffic patterns by auto scaling and thus ensuring enough application capacity.

AWS Lambda

Each AWS lambda function [55] represents one microservice and can be viewed as a serverless compute service which executes pieces of code in stateless containers only in response to events triggered by multiple clients. Each lambda function is decoupled from each other and therefore can be independently scaled based on the incoming network traffic. Once the code is uploaded to a lambda function, the service itself automatically handles all the capacity, patching, scaling and administration of the infrastructure in order to be able to run the containing code. Furthermore, in order to provide visibility in terms of performance, lambda functions publish real time metrics and logs to AWS Cloud Watch [58].

AWS Cognito

A set of lambdas are in charge of handling the communication with AWS Cognito [57]. AWS Cognito is a service which handles secure authentication and authorization patterns and allows to store user identities within AWS. Cognito consists of two services, *i.e.*, *Identity Pools* and *User Pools*. A UP is a user directory which allows users to sign in into the application, while an IDP provides AWS credentials for users who already have been authenticated and received a token. In other terms, UP are used for authentication and identify verification. With a UP, users sign in or federate through a third-party identity provider. Differently, IDP are used for authorization and access control and for generating unique user identities [67].

The use case of this thesis requires a more complex structure regarding access control. In fact, as shown in Table 3.1, new users can be invited into the system from users with certain roles who already are operating in the platform. Therefore, a common sign in and sign out procedure can not be adopted. In this regard, the approach consist of having a lambda function which is responsible to set up new accounts and thus control membership in the UP by only allowing users with certain credentials to create new members.

In order to simplify the identity check, different role-based Cognito User Groups [68] have been created to manage users permissions. The creation of a new user implies the assignation to a specific group based on its role. User Groups are useful to control permission with AWS API Gateway. Indeed, the group that a user is a member of together with an ID token is attached to each request sent by that user through the API Gateway. Therefore, each user can be uniquely and securely classified based on its role. Another lambda function is then responsible to send to new members a customized invitation with their user name and a temporary password.

AWS RDS

As explained in section 3.4.2, the main approach in terms of data structure consists of having a shared relational DB which is able to release and open different DB connections for multiple microservices. Amazon Relational Database Service (RDS) [59] allows to scale a relation DB in the cloud and offers a variety of different DBs.

In this regard, Amazon Aurora [69] is a cost-effective MySQL and PostgreSQL-compatible relational database built for the cloud. AWS Aurora enables a fast lookup among different collections, which suits the use case in this thesis since DB entities are close related to each other. Furthermore, it is serverless and therefore able to scale based on the network traffic.

AWS S3

As shown in section 3.3, the presence of UI cards for the layout aims at highlight colors and images for catching the attention of the user rather than highlighting textual details. Because of this, images upload and image processing must be reliable, fast and highly available. In this regard, Amazon Simple Storage Service (Amazon S3) [60] is an object storage service which offers different bucket and supports common API actions, such as copying images between buckets or replacing them with new ones. S3 is meant to work well with lambda Functions which can complex operations on the S3 objects, such as image processing or image upload.

AWS Cloud Watch

AWS CloudWatch [58] is a service which lives within the CloudFront ecosystem. It is used to monitor and observe in real-time the outcomes of all lambda functions by providing actionable insights to monitor the applications. In general, it gives an overview of the system's health and helps to detect anomalous behavior within CloudFront.

Processing Server

The Processing Server is an EC2 instance [70] which is triggered by those lambda Functions responsible for gathering, processing and analysing data from Wi-Fi sensors. Indeed, the processing server consists of three main components, such as (i) inbox server which is in charge of gathering raw data from Wi-Fi sensors, (ii) analytics server which is responsible to process the collected data and the (iii) portal server which is used for viewing aggregated event metrics. Details regarding IoT devices and their interactions within the AWS ecosystem are presented in section 4.3.

4.2.2 Node.js and AWS Microservices

The core principle of the serverless architecture adopted for this work consists of having exactly one application running for each microservice (lambda function). This means, that each lambda runs in its own container, with its own compiler runtime and is tightly decoupled from the others micro applications.

The programming language used to implement all microservice is Node.js [122], a well-known javascript runtime environment that executes code and can be used to build fast and scalable network applications. Lambda provides runtimes compilers for Node.js, which execute code in response to cloud events. The code itself runs in an environment that includes the AWS SDK for javascript [123].

Handler method

The *handler* method exists in every lambda function which is deployed to the cloud and is executed at runtime [124]. This method is in charge of processing events and to return handled responses to the front-end. Once a response is sent back, the handler is able to handle the next event.

When a lambda function is setup, the value of the handler setting needs to be specified. This consists of the file name, and the name of the exported handler module, split by a dot.

As illustrated by Listing 4.6, the handler method takes as input two arguments: an *event* and *context* object (line 1).

```
1 exports.lambdaHandler = async (event, context) => {  
2  
3   return {  
4     statusCode: 200,  
5     body: JSON.stringify({  
6       message: "Hello " + event.name  
7     })  
8   };  
9 };
```

Listing 4.6: AWS Lambda Function Handler Method

Events are JSON-like data structures which might be different based on the event type. These data contain information about the request that invoked the lambda function, such as the http method, url path, query string parameter, http headers and the body. Subsequently, lambda runtimes convert the event into an object and pass it to the function.

Context objects contain information about the invocation, function, and execution environment. Similarly to event objects, lambda context is JSON-like formatted. Within this object, properties such as function name, the amount of memory that's allocated for the function and the identifier of the invocation request are provided.

Response Handler

In order to standardize the way responses are sent back to the front-end, a global asynchronous response handler function has been implemented.

As illustrated by Listing 4.7, the function takes three variables as arguments. The first one is a boolean variable, defining whether the response handler has to return a successful response or throw an error. The second one is the message describing the handled scenario and the last one is the http status code. As default, 200 is sent for successful requests and 400 for catch exceptions.

Standard headers are attached to the response to avoid CORS problems. For instance, Access Control Allow Origin (line 2) is a response header which states whether the response data can be shared with requesting code from the requesting origin. In addition, at line 3 are specified which HTTP methods are allowed on the concerned lambda function for cross-origin requests.

```

1  const headers = {
2    "Access-Control-Allow-Origin": "*",
3    "Access-Control-Allow-Methods": "POST,GET,OPTIONS,PUT,DELETE"
4  };
5
6  const responseHandler = (success, message, code) => {
7    return {
8      statusCode: code || (success ? 200 : 401),
9      headers,
10     body: JSON.stringify({
11       success,
12       message
13     })
14   };
15 };

```

Listing 4.7: Shared Response Handler

DB Lambda

As mentioned in chapter 3.4, DB instances are managed by a single lambda function, called DB lambda. This implies, that if other lambdas need to query the DB, they do not have to subscribe to the DB directly, but must invoke DB lambda by passing the concerned query as event body.

DB lambda handles a pool of DB connections with RDS using *mysql* [125], a javascript client implementing the MySQL protocol.

As illustrated by Listing 4.8, every time that DB lambda is invoked, it either creates a SQL connection with RDS or it takes an already opened connection pool (lines 37-39). Both requests are asynchronous.

The initialization procedure of the connection pool can be found in lines 4-22. Firstly, a SQL pool is created with the *mysql* instance and a configuration object (lines 5-10). The configuration contains different parameters, such as the DB host, the DB user, the password and the DB port.

For security reasons, these variables are set as environment variables, *i.e.*, variables whose value is set outside the lambda function and can be accessed with the *process* object [126]. The process object provides information about, and control over, the current Node.js process and can always be accessed at runtime.

After the pool is created, a new promise [127] is either resolved with the sql connection pool (lines 17-19) or rejected with the catch error (*e.g.*, when the credentials are wrong). If the promise is resolved, the database instance is then assigned to the variable *connection* at line 37.

Similarly, if DB lambda is invoked and there is already a connection pool running, the instance is assigned to the same variable calling *getConnectionFromPool()* at lines 24-33.

```

1
2 let pool;
3
4 const initDB = async () => {
5   pool = mysql.createPool ({
6     host: dev ? "127.0.0.1" : process.env.SSH_DST_HOST,
7     user: process.env.DB_USER_NAME,
8     password: process.env.DB_PASSWORD,
9     port: process.env.DB_PORT,
10  });
11  return new Promise(async (resolve, reject) => {
12    pool.getConnection (function(err, conn) {
13      if (err) {
14        console.error("Error connecting DB: " + err.stack);
15        return reject(err);
16      }
17      console.log("DB connected with id: " + conn.threadId);
18      conn.query = util.promisify(conn.query);
19      return resolve(conn);
20    });
21  });
22 };
23
24 const getConnectionFromPool = () => {
25   return new Promise(async (resolve, reject) => {
26     pool.getConnection(function(err, conn) {
27       if (err) {
28         return reject(err);
29       }
30       return resolve(conn);
31     });
32   });
33 };
34
35 // handler method

```

```

36 exports.lambdaDB = async (event, context) => {
37   let connection = !pool
38     ? await initConnection()
39     : await getConnectionFromPool();
40   // more code..
41 }

```

Listing 4.8: DB Lambda Connection Pool

After the connection has been established, DB lambda is able to handle queries with mysql. Queries are sent from other lambda functions as payload in the *InvocationRequest* structure, and are then mapped to the event object. As shown in Listing 4.9 at line 4, queries can be accessed through the *query* key.

At line 5, potential SQL injection attacks are tackled, in a way that malicious SQL statements which might compromise the whole database cannot be executed. This is possible, due to the *escape* method, which removes ambiguity in quotes (and other characters) used in the passed query. At lines 6-7, an unsuccessful response handler is triggered when query parameters are not provided.

Code at lines 9-19 executes the query and returns the respective response handler functions. It is worth to point out, that in both cases (lines 13 and 18) the connection is released, *i.e.*, removed from the connection pool. By doing so, the pool does not need to maintain unused instances and has more capacity for different lambda functions which need to query the DB.

```

1 // handler method
2 exports.lambdaDB = async (event, context) => {
3   // more code..
4   const sqlRequest = event.query;
5   const escaped = mysql.escape(sqlRequest);
6   if (!escaped) {
7     return responseHandler(false, "no query provided", 401);
8   }
9   return await connection
10     .query(escaped)
11     .then(message => {
12       connection.release();
13       message = JSON.parse(message);
14       return responseHandler(true, message, 200);
15     })
16     .catch(e => {
17       connection.release();
18       return responseHandler(false, e, 400);
19     })
20 }

```

Listing 4.9: DB Lambda Queries

Listing 4.10 illustrates how DB lambda can be invoked by another lambda function. First of all, the database name is accessed through its environment variable (line 3) and subsequently, a query for retrieving all clients from its collection is created (line 5).

Afterwards, the invocation request object is defined (lines 6-11). Required parameters are the *function name*, *i.e.*, the ARN identifier, which uniquely identifies the function in the cloud so that it can be called from other microservices, the *invocation type*, which defines whether the request is asynchronous or synchronous, the *log type* which states whether the execution log is included in the response and finally the *payload*, *i.e.*, the object which is passed to the other lambda function as input. As shown at line 10, the value of the *Payload* key is the converted javascript representation into string of the query which wants to be executed.

Finally, at lines 13-23 the DB lambda is invoked by the `getClient` lambda. If the payload is successfully returned, the response handler sends back the queried resource to the front-end (line 18), otherwise an error is thrown (line 21).

```

1 exports.getClient = async (event, context) => {
2   const dbName = process.env.DB_NAME;
3
4   let query = `SELECT * FROM ${dbName}.clients`;
5
6   const invocationRequest = {
7     FunctionName: process.env.DBLambdaFunction,
8     InvocationType: "RequestResponse",
9     LogType: "Tail",
10    Payload: JSON.stringify({ query })
11  };
12
13  return lambda
14    .invoke(invocationRequest)
15    .promise()
16    .then(s => {
17      const clients = JSON.parse(s.Payload);
18      return responseHandler(true, clients, 200);
19    })
20    .catch(err => {
21      return responseHandler(false, err, 400);
22    });
23 };

```

Listing 4.10: Lambda Invoke Function

4.2.3 AWS CloudFormation Template

In a on-demand and event-driven applications context, each lambda function is declared and modeled in the Cloud Formation SAM template [129]. The template specifies a stack which contains specifications for each lambda function, such as where its code is located, the API path, the function name, what kind of permissions it has within AWS or which runtime compiler (depending on the programming language) it needs in order to be able to run.

Once the template is uploaded to the cloud, all resources are accordingly created in AWS

following the template specifications. Usually, one template reflects one single AWS stack.

Finally, lambda functions are able to handle in conjunction with AWS Cognito [57] authentication and authorization patterns and thus protecting routes based on user credentials. The Cognito authorizer is explained in the next section.

The template is written using the YAML syntax [128], a human-readable data-serialization language mainly used for configuration files. This markup language is tab-sensitive and allows a clear structure in the template, which can be divided into three main categories:

- Parameters

In this section is specified the environment of the stack (*e.g.*, dev, stage or prod) and mainly environment variables, represented as type-value pairs. For instance, as shown in Listing 4.11, the environment variable *DbPassword* is specified at line 5 and can be thereafter assigned to the concerned lambda functions, which will then in turn access it with the process object (as shown in Listing 4.8).

```

1      Parameters:
2          Environment:
3              Type: String
4              Default: dev
5          DbPassword:
6              Type: String
7              Default: "123456789!!"
8          CognitoArn:
9              Type: String
10             Default: "arn:aws:cognito-idp:ZZ:XX:userpool/YY"

```

Listing 4.11: Cloud Formation Template - Parameters

- Globals

Resources in a SAM template might have shared configuration such as Runtime, Memory, VPC Settings, Environment Variables, Cors, etc. The global section allows to couple this information together, instead of duplicating it in every resource. All under resources can then inherit from it.

Listing 4.12 illustrates the global section. At line 2 binary media types are listed, *i.e.*, types that are supported by the *Api* resource, such as multipart/form-data (line 4) for file upload. In addition, at lines 5-6 it is specified a global lambda timeout. This means, that each microservice has a default timeout of 15 seconds.

```

1      Globals:
2          Api:
3              BinaryMediaTypes:
4                  - multipart/form-data
5          Function:
6              Timeout: 15

```

Listing 4.12: Cloud Formation Template - Globals

- Resources

This category includes two types of resources, a serverless Function and a serverless API. Once a resource is declared, the corresponding type together with a set of properties need to be specified. Properties are related to the type.

For instance, as illustrated in Listing 4.13 the API resource is declared between lines 2-14 and is of type `AWS::Serverless::Api`. The stage name of the API resource is defined by the environment variable shown in Listing 4.11 and is accessed using a reference pointing to that variable (line 5). Furthermore, different CORS configurations are declared between lines 6-9 and apply for the whole API Gateway.

Additionally, a *Cognito Authorizer* is defined between lines 11 and 14. It will protect specific microservices based on authorization patterns, for instance by using bearer token authentication strategies such as authorization headers (line 14).

This implies, clients requesting resources without having a valid authorization token, will result in unauthorized exceptions. In order to link the authorizer with the corresponding UP in Cognito, the UP ARN need to be referenced (line 13). Similarly to the stage name, the Cognito ARN is accessed using a reference pointing to *CognitoArn*, declared in Listing 4.11.

The second resource presented in Listing 4.13 is a serverless function, provisioning the *getUsers* microservice (line 15). First of all, the location to the function code needs to be declared (line 18). Subsequently, the handler name and the compiler runtime environment (lines 19 and 20). Afterwards, a set of events need to be specified. Events are source objects which trigger the concerned lambda function.

For instance, the *GetUsers* event (line 22) is declared as API endpoint, since its type is referenced to the global variable *Api*, presented in Listing 4.12. Furthermore, events have a list of properties (lines 24-29), such as the API endpoint path, the required http method, the reference to the API resource and the Cognito Authorizer.

If the header of the Authorizer at line 30 would have been set to *none*, the concerned microservice would have been accessed without granting any type of authorization. Finally, a list of environment variables can be passed to the function (lines 31-33). For instance, the environment variable *DbPassword* specified in Listing 4.11 can be referenced and accessed within the lambda function with the process object.

```

1   Resources:
2       ApiGateway:
3           Type: AWS::Serverless::Api
4           Properties:
5               StageName: !Ref Environment
6               Cors:
7                   AllowMethods: "'OPTIONS,GET,POST,PUT,DELETE'"
8                   AllowHeaders: "'*'"
9                   AllowOrigin: "'*'"
10          Auth:
11              Authorizers:
12                  MyCognitoAuthorizer:
13                      UserPoolArn: !Ref CognitoArn
14                      Header: "Authorization"
15      GetUsers:
```

```

16      Type: AWS::Serverless::Function
17      Properties:
18        CodeUri: Users/getUsers/
19        Handler: app.lambdaHandler
20        Runtime: nodejs8.10
21      Events:
22        GetUsers:
23          Type: Api
24          Properties:
25            Path: /users
26            Method: get
27            RestApiId: !Ref ApiGateway
28            Auth:
29              Authorizer: MyCognitoAuthorizer
30              Header: "Authorization"
31      Environment:
32      Variables:
33        DbPassword: !Ref DbPassword

```

Listing 4.13: Cloud Formation Template - Resources

4.2.4 Predictions: AWS Time Series Forecasting

The implementation for forecasting the outcome of marketing activities relies on AWS Time Series Forecasting [97]. This service uses machine learning to deliver accurate forecasts based on historical series of data, which is called time series data. Furthermore, it is built within the AWS ecosystem and therefore can be integrated within the architecture presented in Figure 4.1.

When a forecast is created with AWS Time Series, the following resources need to be allocated [98]:

- *Datasets*

Datasets groups are a collection of datasets which need to be correctly aggregated per time unit (*e.g.*, day) in order to be able to be consequently interpreted. ML-based algorithms use then these groups to train custom forecasting models, called predictors. Each group can have up to three datasets, one of each type: target time series, related time series and item metadata.

The first one should be used when the training data is a time series and include the field for which the forecast is generated for (*i.e.*, target fields). The second one should be adopted when the training data is a time series, but does not include the target fields. The last one should be used when the training data is not a time series, but includes metadata information about the target fields.

Furthermore, for each dataset is required a schema mapping the fields in the training data so that target, related or other fields can be identified in the training phase.

Finally, when a dataset group is imported into AWS Time Series, the corresponding path to the CSV file needs to be specified.

- *Predictors*

Predictors are custom models trained based on the input dataset groups. They can be trained using different forecasting algorithms. There are five different algorithms that can be chosen for the training model:

- Autoregressive Integrated Moving Average (ARIMA) [99]
Generally, it is used for stationary time series, *i.e.*, independent of time. Usually, these dataset groups contain noise and therefore unambiguous pattern cannot be easily extracted.
- DeepAR+ [100]
ML-algorithm which uses a supervised learning algorithm for one-dimensional forecasting scalar time series. It uses a RNNs in order to train the model. Should perform better than ARIMA for datasets with consistent time series.
- Exponential Smoothing (ETS) [101]
Commonly-used local statistical algorithm for time-series forecasting. Typically, it is used for dataset groups which contain seasonality patterns in the data.
- Non-Parametric Time Series (NPTS) [102]
NPTS is probabilistic baseline forecaster. Through sampling from past observations, it estimates the future value distribution of a given time series. The values observed limit the predictions. In particular, it is useful if the time series is sporadic.
- Prophet [103]
Prophet is a popular time series model structured in a Bayes-like matter. Generally, it is used for datasets which contain an extended time period, have strong seasonalities and have null values or outliers.

Before a predictor is created, a feature configuration needs to be specified, which determines the forecast frequency (*e.g.*, daily or weekly) and provides information about how to transform the data before the model is trained. In addition, a forecast horizon (or prediction length) needs to be set in order for the predictor to know the number of time-steps to perform.

- *Forecasts*

Forecasts are then generated based on the predictors that have been trained. They can be queried and return a list of predictions based on the input data.

Generally, a forecast is a group of predictions, one for every item in the target time series dataset. Based on the forecast horizon specified in the predictor, the forecast returns the predictions with different confidence level, which represent the probability that an actual value will be below the listed value at the specified date and time.

The implementation details for predicting activities' outcomes consists of having one dataset group for each target field, *i.e.*, *count*, *unique_count* and *average_dwelling_time*. As mentioned before, dataset groups are the base for the prediction model. Thus, in order

to create them, files containing all dataset attributes need to be exported into S3 Storage in a CSV-like format.

Listing 4.14 illustrates how to create dataset groups within AWS Forecast. The input data for the `createDatasetGroup` function at line 9 consists of a javascript objects with three keys (lines 3-7): the ARN for the dataset, the dataset name and the domain of the forecast.

```

1      const forecastservice = new AWS.ForecastService();
2      const createDatasetGroup = async (datasetARN, groupName) => {
3          const params = {
4              DatasetArns: datasetARN,
5              DatasetGroupName: groupName,
6              Domain: "CUSTOM"
7          };
8          return await new Promise((resolve, reject) => {
9              forecastservice.createDatasetGroup(params, (err, data) => {
10                  if (err) reject("Error dataset group" + err);
11                  console.log("Create Dataset group response", data);
12                  resolve(data);
13              });
14          });
15      };

```

Listing 4.14: AWS Forecast Service: Create Dataset Groups

In parallel, another lambda function needs to be periodically executed (*e.g.*, on a weekly basis), for retrieving all new data from the corresponding DB table and consequently creating an updated CSV file. By doing so, the predictive model can always be fed and trained with new attributes. Simultaneously, another lambda function has to retrieve historical weather data from the Open Weather API and enrich the dataset before the file is created.

Subsequently, once the writing of the CSV file is completed, an additional lambda function needs to be triggered to retrieve the S3 file path and create new predictors with the attributes contained within them. The trigger is executed through a AWS Step function [104], which is monitoring each step of execution as it happens. This means, that if there would be a problem in the writing of the file, the step function automatically would catch it and retries after a predetermined time period.

As illustrated by Listing 4.15, in the creation request, one of the algorithms presented above need to be specified. In order to always ensure the selection of the best ML model in terms of minimization of the error rate, AWS Forecast provides an Auto-ML option for model training (line 19). Auto-ML allows the automation of complex machine learning tasks, such as algorithm selection, hyper-parameter tuning, iterative modeling, and model assessment [105]. Generally, DeepAR+ [100] tend to be more accurate when the datasets does not contain noise and has a consistent time series.

In addition to the ML-algorithm, a feature configuration (lines 3-21) needs to be attached once a predictor is created. The feature configuration contains two different parameters,

i.e., the forecast dimension and the forecast frequency. The latter is the predictions frequency (line 4) in a forecast and must be greater than the real time series frequency in the datasets. Valid ranges are year, month, day and hour. The forecast dimension (line 5) is a list of names of dimensions indicating how to group the forecast once it is generated. Thus, the forecast will be able to be queried using them as input parameter and afterwards return a list of predictions based on the target columns.

As shown between lines 5 and 13, the implementation includes *industry_type*, *weather*, *temperature*, *activity_type*, *activity_format*, *zone_of_interest*, *latitude* and *longitude* as forecast dimensions and *count*, *unique_count* and *average_dwell_time* as target fields. This implies, that changing input dimensions would result in the prediction returning different confidence levels. For instance, the returned target fields for the zone of interest *visits* should be completely different than the predictions for the zone *interactions*. Similarly, different input values for *activity_types* should influence the prediction results.

```

1  const forecastservice = new AWS.ForecastService();
2  const createPredictor = async (predictorName, DatasetGroupArn) => {
3    const params = {
4      FeaturizationConfig: {
5        ForecastFrequency: "D",
6        ForecastDimensions: [
7          "industry",
8          "activity_format",
9          "zone_of_interest",
10         "latitude",
11         "longitude",
12         "weather",
13         "temperature"
14       ]
15     },
16     ForecastHorizon: 10,
17     InputDataConfig: {
18       DatasetGroupArn
19     },
20     PerformAutoML: true,
21     PredictorName: predictorName
22   };
23   return await new Promise((resolve, reject) => {
24     forecastservice.createPredictor(params, (err, data) => {
25       if (err) reject("Error predictor" + err);
26       console.log("Create predictor", data);
27       resolve(data);
28     });
29   });
30 };

```

Listing 4.15: AWS Forecast Service: Create Predictor

Finally, the implementation includes another lambda function in charge of creating the forecasts based on the trained predictors. This function creates three different forecast and exposes them in the API Gateway. Consequently, the front-end can call it and based on the input dimensions sent in the request, different predictions can be retrieved.

4.3 Internet of Things (IoT)

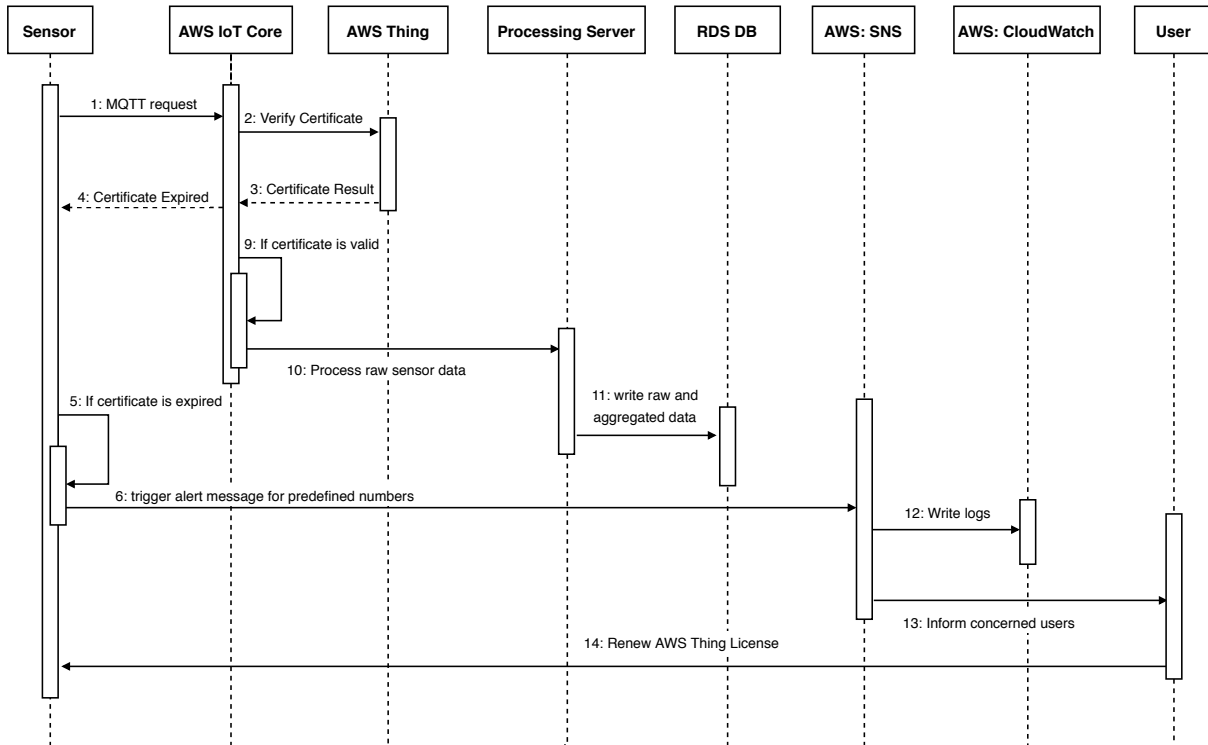


Figure 4.2: AWS IoT Concept: IoT Core, Sensors and Aggregated Data.

The IoT system is a core component in the overall architecture. Figure 4.2, illustrates the concept behind the IoT subsystems with a sequence diagram.

4.3.1 Information Gathering Process through IoT Sensors

The entry point of the IoT system can be found in the sensors themselves, as shown in Figure 4.2 and identified by the first step in the sequence diagram. In this step, GL-iNet MiFi 4G Smart Routers [34] are configured for being able to passively listen to emitted Wi-Fi signals and simultaneously for exchanging data with IoT Core [35].

Sensor OS and Firmware

Each sensor has its own Machine to Machine (M2M) SIM Card for mobile 3G/4G connectivity in Europe and a SD card for permanent archival and offline storage data. The OS of each IoT device consists of OpenWrt [71], an open source GNU/Linux project used for embedded devices with the main scope to route network traffic. It provides a writable file system which allows the installation of external packages and firmwares. Because of this, BlueMark Zero [72] has been installed as main firmware. BlueMark is a third-party

firmware which allows to detect, track and locate smartphones based on Wi-Fi signals. The firmware offers a license management system for generating different licenses and assign them to specific sensors. This is important because, the licenses are needed to unlock the firmware. Once they are installed, a script periodically checks if the license is still valid. If this is the case, the firmware keeps running correctly. Moreover, BlueMark provides accessibility to the Wi-Fi analytics settings of each sensor, which are stored in the corresponding configuration files and thus can be easily adjusted.

In order for the sensors to be able to communicate with IoT Core [35], a package implementing the Message Queuing Telemetry Transport (MQTT) protocol [73] has to be installed in their OS. Indeed, MQTT is a lightweight messaging protocol designed for unreliable networks (*i.e.*, with high latency due to bandwidth constraints) which allows a publish-subscribe mechanism for transporting messages between devices. The protocol usually runs over TCP/IP and is mainly used for M2M communication. Using MQTT, IoT sensors are able to publish information about a given topic to a specific channel in a server, which acts as a message broker [74]. Afterwards, clients can subscribe to channel they are interested for and get the information once the broker pushes it. In this regard, Eclipse Mosquitto [75] is an open source MQTT broker which is installed on each sensor to allow a MQTT-like communication between sensors and IoT Core. Mosquitto is mainly suitable for IoT messaging and can be installed as external package in OpenWrt.

AWS IoT Core and AWS Thing

In order for sensors to be able to publish their raw data through the MQTT protocol in IoT Core [35], they need to be registered as logical entities in the AWS thing registry [76] and have a valid certificate. A *thing*, can be viewed as the logical representation of a sensor in IoT Core. Things are identified by an Amazon Resource Name (ARN) [77] and have a thing type (*e.g.*, Sensor) which allows to store configuration and description information which can be shared with all things associated with the same type. Furthermore, things might have a list of attributes that are stored as name-value pairs. These pairs consists of sensor-related data, such as serial number (or sensor ids), license keys or manufacturer information.

Certificates

As illustrated in Figure 4.2, in order for a sensor to be able to publish data to IoT Core, a certificate [78] associated to a private key needs to be generated to authenticate the device's connection. In this regard, the communication between sensors and AWS IoT is protected through X.509 certificates [80]. These digital certificates use the X.509 public key infrastructure standard in order to link a thing to a public key. Furthermore, asymmetric encryption can be used with the sensors. This implies, that private keys can securely be stored in the sensors themselves and thus the authentication procedures can follow other mechanisms (*e.g.*, bearer tokens), since the private key never leaves the device.

Certificates allows to create AWS IoT Thing Policy Variables [79] which grant or deny permissions depending on the certificates attached to a specific thing. A sensor can therefore invoke AWS IoT operations only if it is granted the appropriate permissions. Thing policies enable specific AWS IoT policies based on thing properties such as, ARNs, thing types and thing attribute name-value pairs. For instance, the thing name is revealed by parsing the client ID contained in the MQTT message sent when a thing connects to AWS IoT.

In addition to policies and certificates, AWS things have thing *shadows* [82]. Sensor shadows are JSON [81] documents identified by the ARN of the thing they are associated to, which can be used to represent the current state (*i.e.*, current configuration) for a specific sensor. The Device Shadow service in AWS IoT core is used to maintain a shadow for each sensor and to retrieve a registry of attributes containing information such as the sensor id, the OS firmware version, the analytics firmware version, the SIM Card ICCID [83] or the Hardware model. Finally, the service can retrieve the current state over MQTT or HTTP, independently wheather the sensor has a stable internet connection or is offline.

As illustrated by the sequence diagram in Figure 4.2, when a certificate is revoked or expired, sensors are no longer able to publish data to AWS IoT Core until the license is renewed. When this happens, a message is triggered with SNS [88] to a list of predefined numbers. SNS is a publish-subscribe messaging for microservices which allows to send messages to a large number of subscriber endpoints simultaneously. It is compatible with lambdas and API Gateway and can be used to fan out notifications to end users using SMS, and emails.

Publishing Data to AWS IoT Core

Using the current analytics firmware, sensors are able to publish data over MQTT to an AWS endpoint. The relevant MQTT configuration settings such as AWS IoT Endpoint, relevant certificates, topic, etc. are stored in the device's configuration file. These settings are passed on to the Mosquitto message broker which is automatically launched when the sensor boots. At present, the system is designed that all data (from all sensors) are published to a single topic called *data*. Sensors are configured to passively listen to emitted Wi-Fi signals. Therefore, once moving devices send a probe request to an AP, sensors are able to catch that signal and send the corresponding raw data to the AWS endpoint. A JSON-like representation of the raw data sent to AWS can be found in Listing 4.16.

```

1 {
2   "type": "mac",
3   "sensorid": 2301,
4   "timestamp": 1572191072,
5   "mac": "a3ff3cbdb13049d1",
6   "rssi_mean": -80.9,
7   "rssi_max": -80,
8   "nr_packets": 2,
9   "oui": "A8BD27",
10  "channel": 5,
11  "localMAC": "b24e635c12c14d84",
12  "multicastMAC": 0,

```

```

13  "retry": 0,
14  "packet_type": 5,
15  "IE_hash": "200fc0e5",
16  "IE_length": 170,
17  "seq_nr": 2247,
18  "override_OUI": 0
19  }

```

Listing 4.16: Sample Wi-Fi Trace sent to AWS IoT Core

As illustrated in Listing 4.16, different information is sent to AWS. On the one hand, information purely related to the interested sensor can be retrieved, such as the sensor id, the channel where it is publishing the data and the local mac address. On the other hand, information related to the signal that has been intercepted is shown. For instance, the RSSI [84] (Received Signal Strength Indicator), indicates an estimated measure of signal strength level that the sensors has perceived.

This metric is important for the analytics part afterwards. In fact, in order to correctly classify visitors by zones of interest as shown in Chapter 3.2, the signal strength is used to reveal individual's position (*i.e.*, distance between the transmitter and the receiver) in a certain event. This is possible due to the fact that sensors are geometrically distributed among the event's area with a predetermined position.

It is worth to mention, that RSSI measurements might be affected by external factors influencing Wi-Fi signals which are strictly related to the target surface. Indeed, it is complex for Wi-Fi sensors to transit with fixed power due to different electromagnetic environments present in the same surface [86]. Nevertheless, time-varying RSSI values can be normalized with an average or sliding filter (*rssi mean* in Listing 4.16) which can reduce random fluctuations caused by the environment and therefore improve the positioning accuracy. In order to approximate individuals positioning within a target surface, the following simplified distance path-loss model [86] is applied:

$$RSSI = MP - 10n\log(d)$$

$$d = 10^{\frac{(MP-RSSI)}{10n}}$$

RSSI is the strength of the beacon's signal. It depends on the distance and the Broadcasting Power value. At maximum Broadcasting Power (+4 dBm) the RSSI ranges from -26, which means that the target is very close, to -100 (approx. 50m distance) [87]. Measured Power (*MP*) is a factory-calibrated constant. It indicates what should be the expected RSSI value perceived by the sensors at a distance of one meter from the target. *N* is a constant which strictly depends on the environmental factor related to the wireless transmission. It ranges from two to four and can be optimized by fitting a large number of experimental measurements.

By using this formula, individuals can be classified into different zone of interests based on their positioning and the dwell time intervals presented in section 3.2. In conclusion, different signal strength result in different distances between targets and the sensors which implies a different assignation in terms of zone of interest.

4.3.2 Processing Data through the Processing Server

As shown by the tenth step in Figure 4.2, the processing server subscribes using MQTT to IoT Core and receives all messages from the relevant topics. The processing server is a proprietary stack consisting of Docker [85] containers for inbox raw data, analytics and portal. The Docker stack is installed on an EC2 [70] instance and performs the following functions:

1. Collect raw data published by different sensors in the **Inbox** Server.
2. Process raw data into aggregated data according to event configurations in the **Analytics** Server and provides aggregated data for analytics output.

In addition, the stack provides a means to remotely manage sensors via shell scripts which are downloaded and executed on sensors. Furthermore, the stack must have the relevant IoT credentials and be configured as a subscriber to the IoT Core endpoint in order to properly work.

Inbox Server

This server (or container) collects the sensor data of the sensors and has the following functions:

- Receives sensor data using the MQTT protocol.
- A time server for sensors.
- A remote update server. Sensors periodically connect with the server and will download and execute corresponding scripts.

Analytics Server

This server processes the sensor data of the sensors and performs the following actions:

- Process received sensor data by the inbox server and store it in the *raw* data DB.
- Cleans the raw data by filtering away noise like random MAC addresses, non-smartphone Wi-Fi devices and static devices.
- Calculates visitor count and other metrics such as duration and stores the aggregated results in the data DB.

4.3.3 IoT Alerting Mechanism: How to track Sensors Status

The monitoring process is a crucial task during the measurement process. Indeed, once a sensor is no longer sending data, it is important to inform all relevant stakeholders about a possible incident. Therefore, if a sensor that should be online turns out to be offline, a text message must be sent to a list of predefined numbers. Figure 4.3 illustrates the approach regarding alerting mechanism for IoT devices.

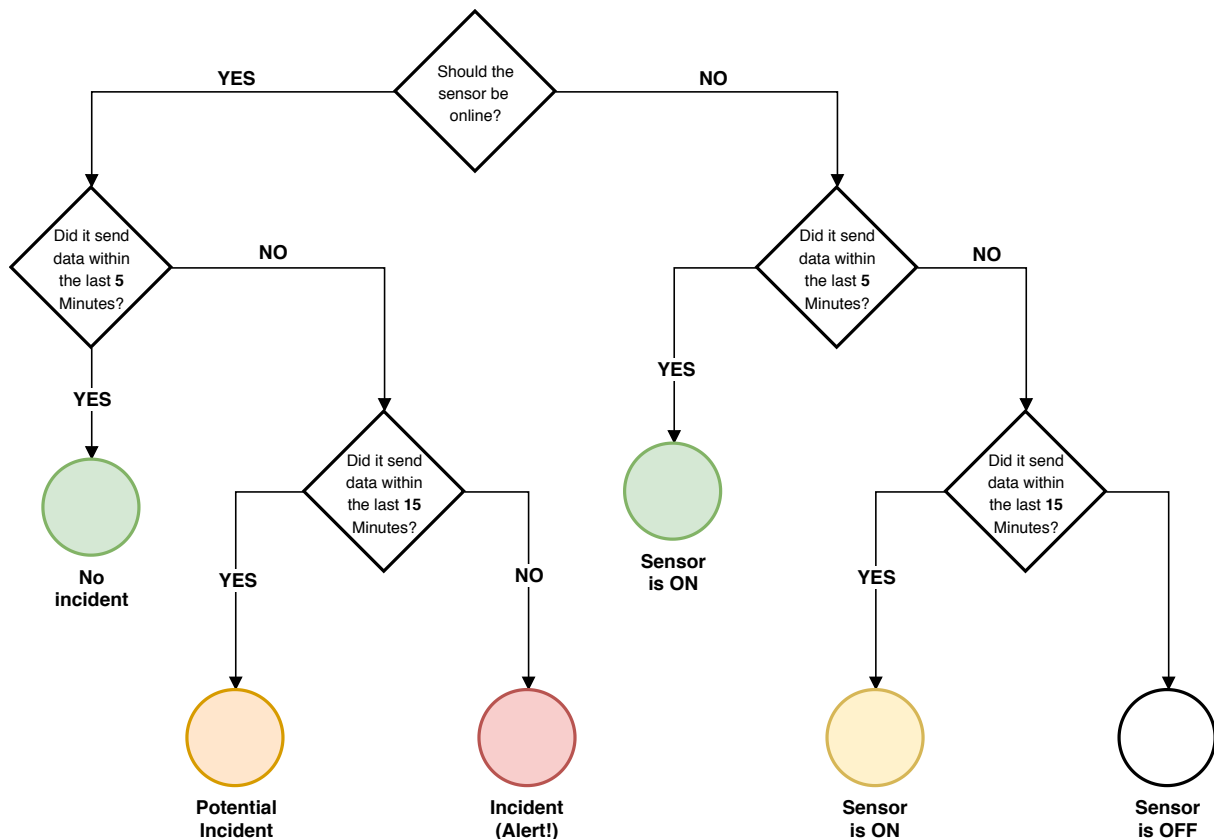


Figure 4.3: AWS IoT Alerting Mechanism.

First of all, a statement is made concerning whether a sensor should be online or not. From an architectural point of view, sensors that should be online are already assigned on the activity level in the relational database. This means, that when an activity id is mapped to a specific sensor id in the *sensors* table and exactly that activity has *running* as a status, the sensor is considered to be online. In addition to beacons packets, IoT devices periodically send a log file to AWS IoT Core containing a JSON-like structure with a timestamp and sensor id in order to track sensors status. Listing 4.17 illustrates the log file sent to IoT Core.

```

1 {
2   "type": "log",
3   "sensorid": 2400,
4   "timestamp": 1572941160
5 }

```

Listing 4.17: Log File sent to Track Sensors Status

In fact, every time that a new timestamp is registered in AWS IoT Core, the column *last_online* in the sensors *table* is updated for the corresponding sensor id. Accordingly, a new log is created in CloudWatch [58] as illustrated by the twelfth step in Figure 4.2.

Based on the last recorded timestamp, it can be inferred when was the last time the sensor sent data to IoT Core. The threshold which has been configured for the system is between five and fifteen minutes. This means, that if the sensor sent data within the last five minutes, the sensor is performing as desired. Between five and fifteen minutes, it is in a warning state and if the last sent data was more than 15 minutes ago, an alerting SMS is sent to a set of predefined numbers using SNS [88].

4.3.4 Remote Sensor Configuration

The firmware running on the sensors needs a well-defined configuration file in order to run properly. This configuration contains a list of parameters such as *channel_mode*, which determines whether the sensor listens for one channel or a random channel, *maintenance* which enables remote login for the sensor, *cron_profile* which determines how often periodical tasks are executed such as data upload, but also checking for updates, remote login etc.

Configuration files are mirrored by AWS *thing shadows* for each sensor. This means, that each sensor has its own configuration stored online which can be accessed and changed by a lambda function.

In order to always have the latest state, two scripts have been implemented to communicate with IoT Core. The first one, reacts on a possible state change in AWS and downloads the configuration diff onto the sensor and afterwards kills the BlueMark process. The second one, executes the downloaded configuration at boot so that the newest configuration is always stored in the sensors. Since all configuration files live in the AWS ecosystem and not in the devices, they can be changed even though the sensors are offline. By rebooting them, they will always get the latest state present in the corresponding AWS thing shadows.

Chapter 5

Evaluation

This chapter presents the evaluation of the platform for passive Wi-Fi tracking. The focus of the evaluation lies on the software side and presents three different evaluation methods. Firstly, the results of the System Usability Scale (SUS) applied to the platform is presented. Secondly, the accuracy and precision of the predictive model is evaluated. Finally, costs associated to the deployment, data storage, hosting and running of the whole infrastructure are described.

5.1 Usability

System Usability Scale [130] is a user-oriented testing strategy for measuring the usability of a certain system. With this testing approach, participants are asked to score different questions with one of five responses that range from Strongly Agree to Strongly disagree. The questionnaire and the corresponding scoring system are outlined in the System Usability Scale Template [131].

Due to the fact that the platform at the current stage can still be considered a minimum viable product for early customers, the SUS approach has been chosen, since it can be used on small sample sizes and obtaining reliable results. It is worth to point out, that the usability of any software has to be considered in terms of the context in which it is used and therefore its appropriateness has to be viewed in that context [131].

In order to obtain sensible results for the SUS test, ten people have been interviewed with different background and skill sets. Participants have been divided into three different groups based on the following criteria:

1. *Developers*

People who are technically knowledgeable and have a computer science background. Developers have been selected from Axelra [132], a tech venture builder based in Zurich. During the development of the platform, they have been involved in discussions regarding potential solutions and were aware of the progress of the system.

2. Promoters

People working for different brands and agencies who are part of their promotional teams. These people have a good understanding of how to setup and run a promotional activity or a marketing campaign. They have never worked before with the platform implemented in this thesis and have no technical background.

3. Brands and Agencies Managers

People at the head of different brands and agencies who are responsible for the outcome of their marketing campaigns or promotional activities. These people are usually not part of the promotional team and are mainly involved in high-level marketing strategies.

The SUS experiment has been executed after the first official release in November 2019. The results are described in the section below and can be found in Figure 5.1.

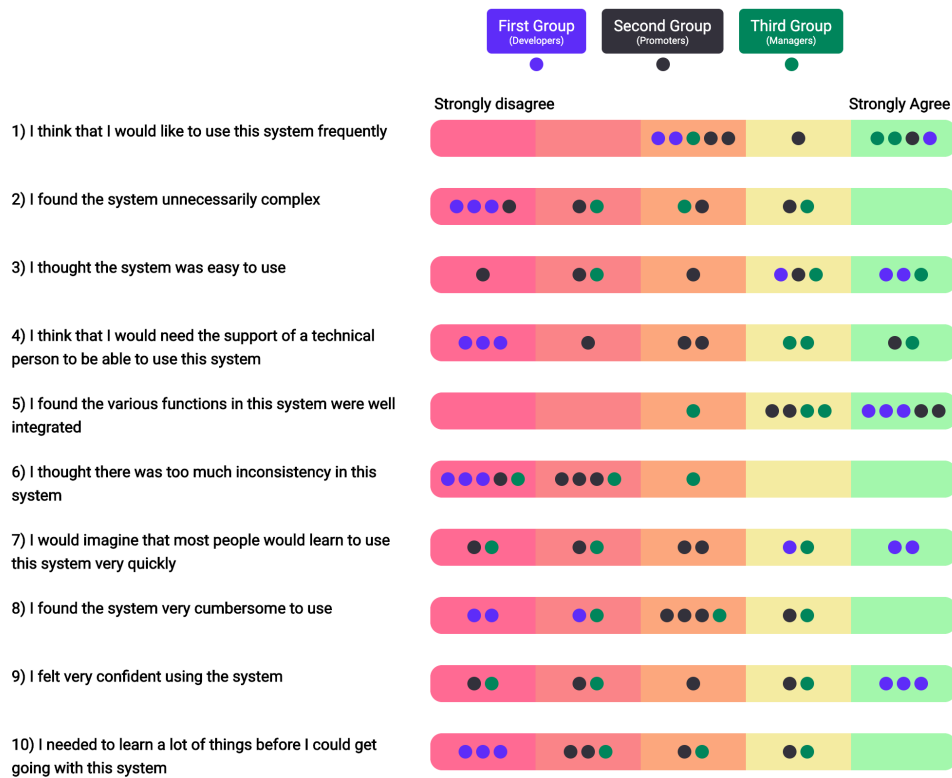


Figure 5.1: System Usability Scale Results

5.1.1 SUS Results and Personal Feedback

In general, all groups considered the system well integrated and free of inconsistency. Furthermore, almost all members would use the system frequently, independently to the group they belong to.

One issue which differentiates developers and non-technical people regards the complexity of the system in terms of technical understanding. Developers found the system easy to use

while promoters and managers were struggling with it at the beginning. After explaining to them the core functionalities and features they started to feeling more confident to use it.

For instance, most of the parameters presented in Section 3.2 need to be configured before an activity is created. Some of them, such as *Visit GAP* or *Min/Max Dwell Time* were not clear to them and lead to the insertion of random intervals or numbers. They were not aware, that based on those metrics they activity would have setup in completely different ways.

Another common issue was about the wording. Interviewed people were confused about the terminology applied to the Stakeholders presented in 3.1. Indeed, terms such as *activities*, *campaigns*, *brands* and *agencies* were new to them and lead to an initial disorientation while approaching the platform. For instance, they proposed to switch the term *activities* with *measurements*, since it is on the activity layer where sensors are configured and will afterwards result into measurements.

Most of the issues addressed by the participants of the SUS have been integrated in the final Release in December 2019.

5.1.2 Evaluated Screens

This section illustrates different screenshots extracted from the platform implemented in this thesis.

Figure 5.2 illustrates the first step for creating a new activity in the front-end. First of all, one of the activity types presented in Section 3.1 needs to be selected.

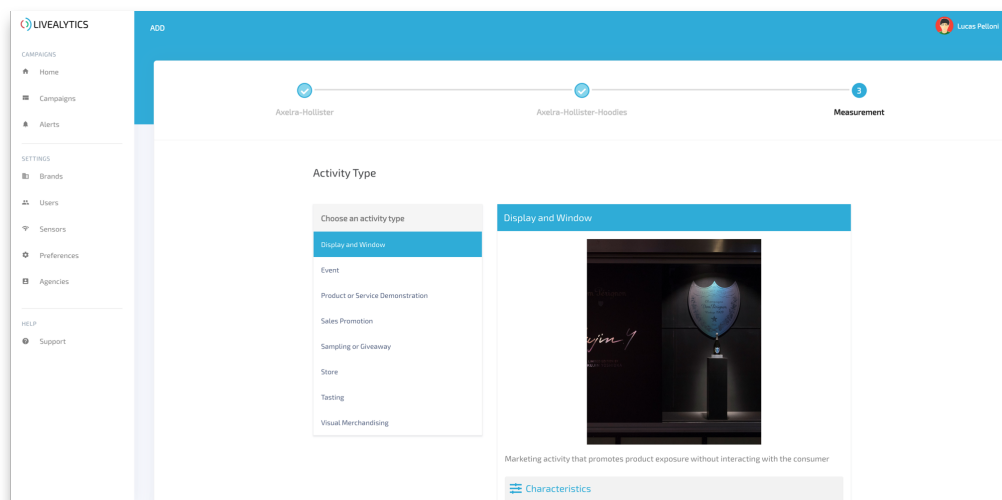


Figure 5.2: Activity Form - First Step

Afterwards, as shown by Figure 5.3 some activity details need to be configured. Firstly,

the activity name has to be inserted. Then, values such as the environment (*i.e.*, activity format) and surroundings (*e.g.*, Indoor or Outdoor) have to be chosen. Finally, the activity place has to be located. By manually moving the map sensor, the geographical coordinates of the chosen location are calculated automatically. These values will be relevant to locate the activity more specifically.

The screenshot shows the 'Activity Form - Second Step' in the LIVEALYTICS interface. The form is titled 'Activity Details' and includes the following fields:

- Title:** A text input field containing 'Axeira AG Activity'.
- Environment:** A dropdown menu with 'Retail' selected.
- Surroundings:** A dropdown menu with 'Open Area (e.g. Outdoor)' selected.
- Location:** A text input field containing 'Bahnhofstrasse 3, Zürich, Schweiz'.
- Latitude:** A text input field containing '47.3675143'.
- Longitude:** A text input field containing '8.539884599999999'.

A map is displayed on the right side of the form, showing the location of the activity. The map includes labels for 'TOWN HALL', 'Grossmünster', 'Ressenden', 'Openland Zürich', and 'Quai à l'Écluse'. The map is titled 'Map Satellite' and includes a 'Map data ©2019' and 'Terms of use' link.

The form is part of a multi-step process, with steps 1, 2, and 3 visible at the top. Step 1 is 'Axeira-Hollister', Step 2 is 'Axeira-Hollister-Hoodies', and Step 3 is 'Measurement'.

Figure 5.3: Activity Form - Second Step

The screenshot shows the 'Activity Form - Third Step' in the LIVEALYTICS interface. The form is titled 'Measuring' and includes the following sections:

- Distance:** A horizontal slider with three segments: 'Visitors / Interaction' (0-15m), 'Views' (15-30m), and 'Opportunities' (30-50m).
- Duration:** A horizontal slider with three segments: 'Bounces' (0-30s), 'Visitors' (30-180s), and 'Interactions' (180-3600s). A 'Cut-off' point is also indicated.
- Measure Interactions:** A checkbox that is checked.
- Your current configuration:** A summary box showing the current configuration of the activity, including the radius and duration of the activity.
- Sensors:** A section for assigning sensors to the activity, with a list of sensors and a 'Don't have enough sensors?' link.

The form is part of a multi-step process, with steps 1, 2, and 3 visible at the top. Step 1 is 'Axeira-Hollister', Step 2 is 'Axeira-Hollister-Hoodies', and Step 3 is 'Measurement'.

Figure 5.4: Activity Form - Third Step

Figure 5.4 illustrates the third step of the activity form. Within this step, the activity configuration is setup. In fact, using the coloured slider, parameters such as radius and mix-max dwell times for different zone of interests can be configured. Finally, a bunch

of available sensors can be assigned to the activity. These sensors will be then used for measuring metrics during the activity life cycle.

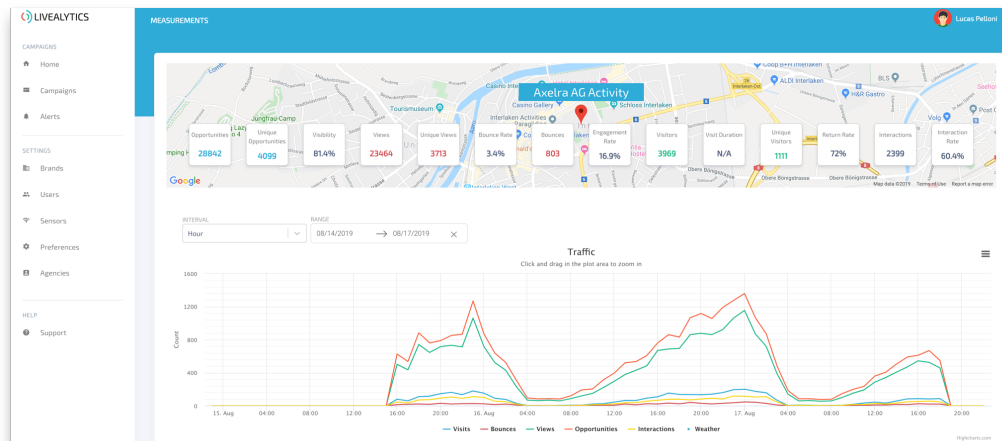


Figure 5.5: Activity Overview - First Part

Figure 5.5 and 5.6 show the activity dashboard. The charts and KPIs displayed in this overview have been presented in Section 3.2 respectively 3.3. Charts can be sorted by date and time and exported as PNG or PDFs.



Figure 5.6: Activity Overview - Second Part

Figure 5.7 illustrates the agency overview in the perspective of root user. As explained in Section 3.3, the overview follows a card-based design principle.

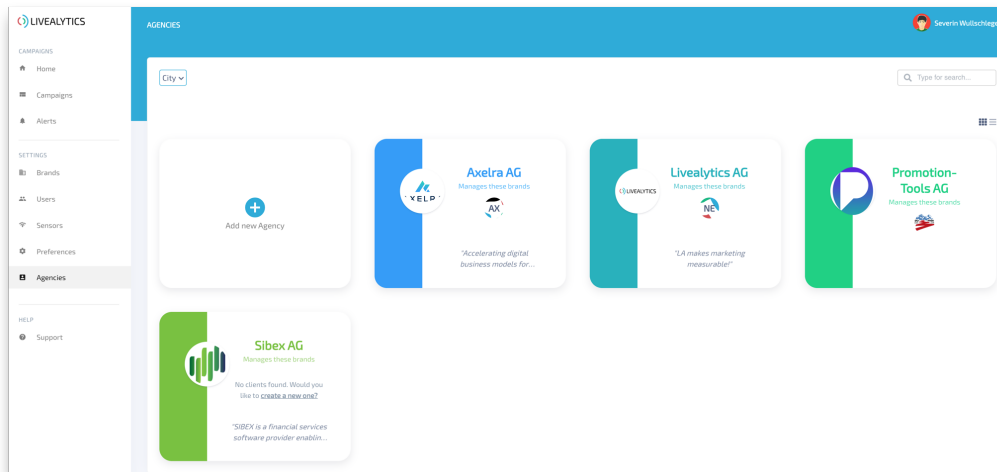


Figure 5.7: Agencies Overview

Figure 5.8 illustrates the campaign overview. Within this tab, all activities belonging to the concerned campaign can be tracked. Activities can be stored as drafts or have an active status.

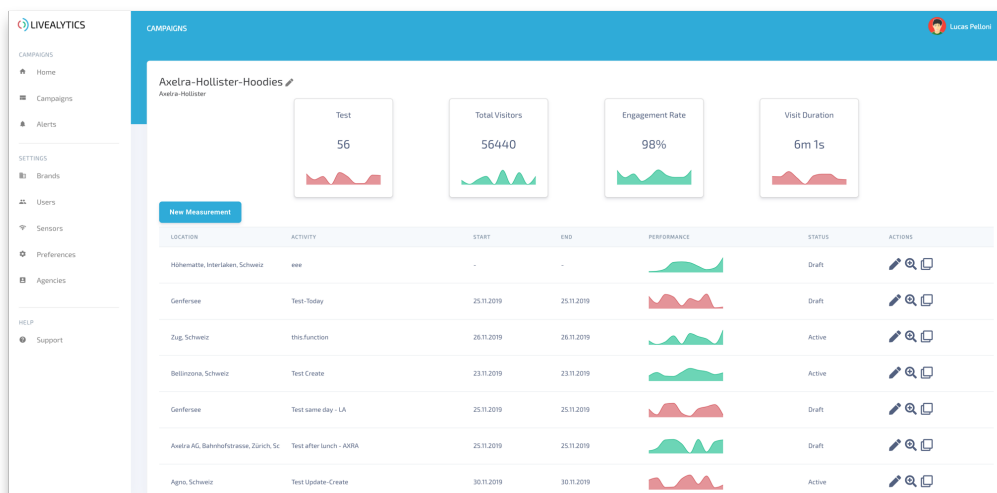


Figure 5.8: Campaign Overview

Finally, Figure 5.9 shows the sensors overview in the perspective of a root account. Within this tab, all sensors can be monitored, configured and assigned to other agencies. Furthermore, the current configuration for each sensor can be downloaded and executed locally.

SENSOR CODE	LAST ONLINE	STATUS	ACTIONS
#5678	Never online	Please configure sensor	
#123	Never online	Please configure sensor	
#2234	Never online	Please configure sensor	
#3334	2019-11-28 16:05	Offline	
#2222	2019-11-30 11:04	Online	
#2324	Never online	Please configure sensor	
#1122	Never online	Please configure sensor	
#1022	2019-11-28 09:51	Offline	
#140	Never online	Please configure sensor	
#1225	Never online	Please configure sensor	

Figure 5.9: Sensors Overview

5.2 Predictions Accuracy

In order to train the predictive model, activities run by Livealitics [3] have been collected and used as input data for the ML algorithm ¹. The model has been trained with activities for which their corresponding end dates were set before the 22th of November 2019.

Afterwards, one activity which took place between the 22th and 23th of November has been randomly extracted from the dataset. Finally, it has been excluded from the training procedure, and kept as a benchmark for evaluating the model precision retrospectively.

The input forecast dimensions used for the evaluation, as explained in Chapter 4.2.4, consist of features such as *zone_of_interests*, *activity_type* and *activity_format*. In this experiment, features such as *location* or *weather* have not be considered.

Table 5.1 shows the characteristics of the activity used for the evaluation.

Period	22.11 - 23.11.2019
Industry	Technology
Activity Type	Product or Service Demonstration
Activity Format	Trade Show
Measured Zones	Interactions, Visitors, Views, Opportunities
Computed Metrics	Count, Unique Count, Average Dwell Time

Table 5.1: Activity for Evaluating Prediction Model Precision.

¹<https://github.com/lucaspelloni2/master-thesis-evaluation-dataset>

Table 5.2 shows the actual results obtained by the activity for the two-day period, classified by zone of interests and days for the computed metrics. Table 5.3 illustrates the forecasts guessed by the ML algorithm for the same period.

Day	Zone	<i>Count</i>	<i>Unique Count</i>	<i>Average DT</i>
22.11	Interactions	38	18	85
22.11	Visitors	121	18	375
22.11	Views	342	134	500
22.11	Opportunities	7042	1059	576
23.11	Interactions	56	34	72
23.11	Visitors	136	37	355
23.11	Views	285	342	418
23.11	Opportunities	9072	1844	499

Table 5.2: Actual computed Metrics for a randomly selected Activity.

Day	Zone	<i>Count</i>	<i>Unique Count</i>	<i>Average DT</i>
22.11	Interactions	13.97	11.18	26.35
22.11	Visitors	43.09	15.44	36.38
22.11	Views	217.32	98.96	612.78
22.11	Opportunities	7566.14	2970.26	1048.05
23.11	Interactions	13.34	11.85	14.96
23.11	Visitors	41.15	28.11	21.69
23.11	Views	264.69	113.91	32.56
23.11	Opportunities	7296.79	3832.71	653.27

Table 5.3: Predicted Activity Metrics.

On the one hand, the ML algorithm was able to individuate a growing pattern for *count*, *unique_count* and *average_dwell_time* between the different zones of interest. Indeed, the interaction zone always shows the smallest amount, while the opportunities zone has the highest one. The same growing pattern applies for both, the actual computed metrics and the guessed forecast.

On the other hand, the accuracy of the prediction shows alternate behaviors depending on the concerned metrics and the zone of interest. For instance, the lower the count, the more difficult it is for the predictive model to forecast a reliable metric. In fact, the

forecast for the interactions zone shows the greatest variance for the count metric, *i.e.*, the highest delta between the actual and the predicted value. For example, on the 22nd of November, the count metric for the interaction zone recorded a value of 38, while the ML algorithm forecast a value of 14, which results in a difference of almost 300% times smaller. Differently, the delta between the actual and forecast value for the opportunities zone on the same day results in a difference of only 7%.

5.3 Costs

The costs involved using AWS services follow the principle "Pay Less by Using More", meaning AWS providing volume discounts for professional users spending more than a specific amount of money. Furthermore, AWS grants one year of free tiers making the experimentation of the multitude of its technical services more affordable for developers and organizations.

On the one hand, free usage encourages cloud adoption and reduce risk to be highly charged due to inexperience. On the other hand, it creates a high lock-in effect for organizations which might result into undesirable side-effect on the long run. Nevertheless, AWS services are well established in the market and provide an enormous amount of sophistication within their ecosystem.

5.3.1 AWS Pricing

The base for the costs calculation consists of the different pricing models for the used AWS services. This includes, AWS Cognito as identity manager [57], AWS RDS as relation database [59], Lambda functions as microservices [55], AWS API Gateway as main orchestration entity [54], S3 for data storage [60] and IoT Core as cloud application for interacting with sensors via MQTT [35].

The respective pricing tiers [133, 134, 135, 136, 137, 138] selected for the current status of the platform have been summarized in Table 5.4.

AWS Service	Price (monthly billed)
Cognito	$\frac{0.00550 \$}{\text{Monthly Active Users}}$
RDS	$\frac{0.11 \$}{1 \text{ GB storage}} + \frac{0.044 \$}{\text{hour}}$
Lambda	$\frac{0.000001667 \$}{100 \text{ ms running time}}$
API Gateway	$\frac{3.50 \$}{1 \text{ million requests}}$
S3	$\frac{0.022 \$}{1 \text{ GB storage}}$
IoT Core	$\frac{1.00 \$}{1 \text{ million messages}}$

Table 5.4: AWS Services Pricing Models.

Current Cost Calculation

At present, the system has one deployed API Gateway managing around 65 Lambda functions, for which it has been allocated 512 MB of memory each. Based on the API Gateway insight, each lambda function runs on average for 0.85 seconds each time.

Let's assume one single user performs on average 200 requests per month and that there are 25'000 monthly active users operating in the system. Therefore, the lambda functions in the API Gateway would have been asymmetrically executed 5'000'000 times per month. Consequently, the total computed running time would be (in seconds):

$$5'000'000 \times 0.85s = 4'250'000 s$$

This results into a total computational power of:

$$4'250'000 s \times \frac{512 MB}{1024} = 2'125'000 \frac{GB}{s}$$

Since AWS Lambda offers a free tier of 400'000 GB/seconds of compute time per month, the total amount of monthly chargeable compute time is:

$$2'125'000 \frac{GB}{s} - 400'000 \frac{GB}{s} (free\ tier) = 1'725'000 \frac{GB}{s}$$

Finally, the billable monthly amount is computed as follows:

$$\textbf{Lambdas: } 2'125'000 \times 0.00001667 \$ = 35.42 \$$$

In combination with the API Gateway and the 5'000'000 requests:

$$\textbf{API Gateway: } 3.50 \$ \times 5M = 17.50 \$$$

Regarding the usage of Cognito with the assumption of the 25'000 monthly users, the total costs for user management would be:

$$\textbf{Cognito: } 25'000 \times 0.00550 \$ = 137.50 \$$$

Concerning RDS usage, only 17 lambdas out of 65 are exposed as PUT or POST http methods in the SAM template and therefore change records in the database collections. Typically, these endpoints are less executed than for instance microservices exposed as GET methods. Therefore, a cheaper DB instance can be chosen (0.044 \$ pro hour). The DB storage is set by default to 100 GB. The monthly costs for RDS are computed as follows:

$$\textbf{RDS: } 0.044 \$ \times 24 (h) \times 30 (d) + 100 GB \times 0.11\$ = 42.68 \$$$

S3 is mainly used for data upload and for storing historical raw data sent from the sensors to AWS IoT Core. Storage costs increase as stored data increases. By default the tier is set to 100 GB, thus the monthly costs for S3 are calculated as follows:

$$\mathbf{S3:} \ 0.022 \$ \times 100 \text{ GB} = 2.20 \$$$

Costs associated to IoT Core are highly dependent from the amount of marketing activities that have been running during the billable period. Let's assume that 50 marketing activities of one week's duration have taken place, each with two sensors onsite measuring 8 hours per day. To simplify the scenario, assume that all sensors have the same transmission interval, meaning they have been sending data to IoT Core every two seconds.

This results in the following amount of messages sent to IoT core during the billable period:

$$50 \text{ activities} \times 2 \text{ sensors} \times 7 \text{ days} \times 2s \times 30 \times 60 \times 8 = 20'160'000 \text{ messages sent}$$

Finally, the amount of messages sent to AWS IoT Core would cost:

$$\mathbf{IoT \ Core:} \ 1 \$ \times 20.16M \text{ messages} = 20.16 \$$$

In conclusion, the total costs associated to the usage of AWS services in the system for a period of one month including some hypothetical scenarios would be:

$$\mathbf{Total \ costs:} \ 35.42 \$ + 17.50 \$ + 137.50 \$ + 42.68 \$ + 2.20 \$ + 20.16 \$ = 304.47 \$$$

Chapter 6

Final Considerations

In this master thesis, the system for passive Wi-Fi tracking has been introduced, designed, implemented and deployed. Furthermore, the implemented software has been evaluated using different evaluation methods from an usability and technical point of view.

The general objectives stated in this thesis were about to have a modular platform which uses gathered information from emitted wireless signals to provide various insights integrated into a single Web application. On the one hand, one specific goal was to have a functional dashboard able to manage all different stakeholders involved in the system and additionally to benchmark the performance of their marketing activities using a list of KPIs. On the other hand, another specific objective consisted of the design and implementation of a predictive model able to forecast activity metrics based on historical records.

Both objectives have been achieved and documented within this thesis. The main factors that contributed to their achievement is partly due to the selection of the infrastructure and technologies used for the implementation part. Another important aspect resides in the fact the hardware part, *i.e.*, sensors configuration, was already partially calibrated.

The choice of AWS as main infrastructure for the implementation part has been intensely discussed with the supervisor. In fact, AWS might be challenging for on-boarding new projects from scratch, and requires knowledge on multiple layers. Despite the difficulty of the initial configuration, the long-term advantages that it has brought for the coexistence of the different components of the system, have helped to achieve the objectives set. For example, the IoT part was well-integrated and manageable within IoT Core, whose services in part provided a pre-configuration for IoT devices. Another example can be found in the prediction part. Indeed, the predictive model was trained using ML algorithms that were already configured in AWS. This allowed to have a robust predictor without having to manually calibrate multiple models.

One of the main difficulties encountered in the execution of the work presented in this thesis was about implementing a system using new cloud technologies in a relatively short period of time. Since the timeline was tight, some trade-offs, as explained in previous Chapters, had to be taken in order to be able to deliver a functional system on time.

Furthermore, specific AWS knowledge has been acquired during this thesis. There was no past experience with this infrastructure.

6.1 Future Work

Although the platform presented in this thesis is already functional and has already been deployed, there are still several aspects that can be further explored in the future.

One of these consist of improving the precision and accuracy of the predictive model. Indeed, despite the efforts put into feature selection and optimization during the design and implementation part, there are still some challenges open. One key factor might be the expansion of the number of features selected for the training model. Another one could be to augment the precision of the current ML model by performing a more selective pre-processing procedure of the input data.

Finally, the current AWS architecture could be reviewed to optimize the ratio between costs and services used. Indeed. during this thesis, most of the AWS services adopted were relying on a free tier plan. Therefore, their usage costs were partially considered for the implementation part.

Bibliography

- [1] Juniper Networks: Understanding Wireless Scanning, https://www.juniper.net/documentation/en_US/junos-space-apps/network-director3.1/topics/concept/wireless-scanning.html, last accessed July 2019.
- [2] WiFi Networks: Understanding the Scan Modes, <https://www.adriangranados.com/blog/understanding-scan-modes-wifiexplorerpro>, last accessed July 2019.
- [3] Livealytics: We make Live Marketing measurable, <https://www.livealytics.com/>, last accessed June 2019.
- [4] Sapiezynski, P., Stopczynski, A., Gatej, R. and Lehmann, S., *Tracking human mobility using wifi signals*. PloS one, 10(7), p.e0130824, last accessed July 2019.
- [5] Musa, A.B.M. and Eriksson, J., 2012, November. *Tracking unmodified smartphones using wi-fi monitors*. In Proceedings of the 10th ACM conference on embedded network sensor systems (pp. 281-294). ACM, last accessed August 2019.
- [6] Chitra R. Karanam, Belal Korany, Yasamin Mostofi, *Multi-Person Passive Tracking with WiFi Magnitude Measurements*. IPSN '19 Proceedings of the 18th International Conference on Information Processing in Sensor Networks (pp. 181-192), last accessed August 2019.
- [7] Chilipirea, C., Baratchi, M., Dobre, C. and van Steen, M., 2018. *Identifying Stops and Moves in WiFi Tracking Data*. Sensors, 18(11), p.4039, last accessed August 2019.
- [8] Gosai, A. and Raval, R., 2014. *Real Time Location based Tracking using WIFI Signals*. International Journal of Computer Applications, 101(5), last accessed August 2019.
- [9] Scheuner, J., Mazlami, G., Schoeni, D., Stephan, S., De Carli, A., Bocek, T. and Stiller, B., 2016, November. *Probr-a generic and passive WiFi tracking system*. In 2016 IEEE 41st Conference on Local Computer Networks (LCN) (pp. 495-502). IEEE, last accessed August 2019.
- [10] Kalogianni, E., Sileryte, R., Lam, M., Zhou, K., Van der Ham, M., Van der Spek, S. and Verbree, E., 2015, June. *Passive wifi monitoring of the rhythm of the campus*. In Proceedings of The 18th AGILE International Conference on Geographic Information Science (pp. 9-14), last accessed August 2019.

- [11] Bonne, B., Barzan, A., Quax, P. and Lamotte, W., 2013, June. *WiFiPi: Involuntary tracking of visitors at mass events*. In 2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM) (pp. 1-6). IEEE, last accessed August 2019.
- [12] Cunche, M., 2014. *I know your MAC Address: Targeted tracking of individual using Wi-Fi*. Journal of Computer Virology and Hacking Techniques, 10(4), pp.219-227, last accessed August 2019.
- [13] Husted, N. and Myers, S., 2010, October. *Mobile location tracking in metro areas: malnets and others*. In Proceedings of the 17th ACM conference on Computer and communications security (pp. 85-96). ACM, last accessed August 2019.
- [14] Youssef, M., Mah, M. and Agrawala, A., 2007, September. *Challenges: device-free passive localization for wireless environments*. In Proceedings of the 13th annual ACM international conference on Mobile computing and networking (pp. 222-229). ACM, last accessed August 2019.
- [15] Privacy in WiFi analytics, Whitepaper, <https://www.privacysig.org/>, last accessed August 2019.
- [16] Wikipedia: Viterbi's algorithm, https://en.wikipedia.org/wiki/Viterbi_algorithm, last accessed August 2019.
- [17] Palma, A.T.; Bogorny, V.; Kuijpers, B.; Alvares, L.O. *A clustering-based approach for discovering interesting places in trajectories*. In Proceedings of the ACM Symposium on Applied Computing, Fortaleza, Brazil, 16-20 March 2008; pp. 863-868, last accessed August 2019.
- [18] Rocha, J.A.M.R.; Times, V.C.; Oliveira, G.; Alvares, L.O.; Bogorny, V. *DB-SMoT: A direction-based spatio-temporal clustering method*. In Proceedings of the IEEE International Conference on Intelligent Systems, IS 2010, Berks, UK, 1-2 September 2010; pp. 114-119, last accessed August 2019.
- [19] Kang, J.H.; Welbourne, W.; Stewart, B.; Borriello, G. *Extracting places from traces of locations*. In Proceedings of the 2nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots, Philadelphia, PA, USA, 1 January 2004; pp. 110-118, last accessed August 2019.
- [20] Li, Q.; Zheng, Y.; Xie, X.; Chen, Y.; Liu, W.; Ma, W. *Mining user similarity based on location history*. In Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Irvine, CA, USA, 5-7 November 2008; p. 34, last accessed August 2019.
- [21] F. Barcelo1-Arroyo et al, *Indoor Location for Safety Applications using Wireless Networks*, in Proceeding of the 1st ERCIM Workshop on Mobility, Portugal, 2014, last accessed August 2019.
- [22] Stackoverflow: WiFi Position Triangulation, <https://stackoverflow.com/questions/16485370/wifi-position-triangulation>, last accessed September 2019.

- [23] Dwell Time, <https://www.searchenginejournal.com/dwell-time-seo/294471/>, last accessed September 2019.
- [24] Livealitics Platform, <https://stage.livealitics.fr>, last accessed September 2019.
- [25] Wikipedia: IEEE 802.11, https://en.wikipedia.org/wiki/IEEE_802.11, last accessed September 2019.
- [26] IEEE Standards Association, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," in IEEE Standard for Information technology. New York: IEEE Computer Society, 2012, vol. 2012, no. March, ch. 11, last accessed September 2019.
- [27] C. Systems, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*, 20152020, Technical Report, CISCO, 2016, last accessed September 2019.
- [28] S. Du, J. Hua, Y. Gao, S. Zhong, Ev-linker: *Mapping eavesdropped WiFi packets to individuals via electronic and visual signal matching*, J. Comput. Syst. Sci. 82 (1, Part B) (2016) 156-172. Mobile Social Networking and computing in Proximity (MSNP): A Multi-disciplinary Inspired Approach <https://doi.org/10.1016/j.jcss.2015.06.005>, last accessed September 2019.
- [29] A. Di Luzio, A. Mei, J. Stefa, *Mind your probes: Deanonymization of large crowds through smartphone WiFi probe requests*, Computer Communications (INFOCOM), 2016 IEEE Conference on, (2016), last accessed September 2019.
- [30] WiGLE: Location-based Wireless Network Mapping, <https://wigle.net/>, last accessed September 2019.
- [31] M. Cunche, M.A. Kaafar, R. Boreli, *I know who you will meet this evening! linking wireless devices using WiFi probe requests*, World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a, (2012), pp. 1-9, <http://dx.doi.org/10.1109/WoWMoM.2012.6263700>, last accessed September 2019.
- [32] M. Cunche, M.-A. Kaafar, R. Boreli, *Linking wireless devices using information contained in wi-fi probe requests*, Pervasive Mob. Comput. 11 (2014) 56-69 <https://doi.org/10.1016/j.pmcj.2013.04.001>, last accessed September 2019.
- [33] Webopedia: *Entity Relationship Diagram*, https://www.webopedia.com/TERM/E/entity_relationship_diagram.html, last accessed September 2019.
- [34] GL-MiFi, 4G Smart Routers, <https://www.gl-inet.com/products/gl-mifi/>, last accessed September 2019.
- [35] AWS IoT Core. Easily and securely connect devices to the cloud, <https://aws.amazon.com/iot-core/>, last accessed September 2019.
- [36] What is a KPI?, <https://www.klipfolio.com/resources/articles/what-is-a-key-performance-indicator>, last accessed September 2019.

- [37] Design Evaluation - Wireframe draft, <https://xd.adobe.com/view/a5cec664-35a5-437c-7f1c-9c3d840e68da-d3ff/?fullscreen>, last accessed September 2019.
- [38] Highcharts: Funnel 3D, <https://www.highcharts.com/docs/chart-and-series-types/funnel-3d>, last accessed October 2019.
- [39] Highcharts: Bar Chart, <https://www.highcharts.com/docs/chart-and-series-types/bar-chart>, last accessed October 2019.
- [40] Highcharts: Heat Map, <https://www.highcharts.com/docs/chart-and-series-types/heatmap>, last accessed October 2019.
- [41] Highcharts: Histogram Series, <https://www.highcharts.com/docs/chart-and-series-types/histogram-series>, last accessed October 2019.
- [42] Highcharts: Line charts, <https://www.highcharts.com/docs/chart-and-series-types/line-chart>, last accessed October 2019.
- [43] Cards: UI-Component Definition, <https://www.nngroup.com/articles/cards-component/>, last accessed October 2019.
- [44] AWS: What are Microservices, <https://aws.amazon.com/microservices/>, last accessed October 2019.
- [45] Monolithic vs. Microservices Architecture, <https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59>, last accessed October 2019.
- [46] Wikipedia: Separation of Concerns, https://en.wikipedia.org/wiki/Separation_of_concerns, last accessed October 2019.
- [47] AWS Ecosystem, <https://aws.amazon.com/>, last accessed October 2019.
- [48] AWS: Amazon Route 53, <https://aws.amazon.com/route53/>, last accessed October 2019.
- [49] AWS: Amazon Certificate Manager 53, <https://aws.amazon.com/certificate-manager/>, last accessed October 2019.
- [50] AWS: Cloudfront, <https://aws.amazon.com/cloudfront/>, last accessed October 2019.
- [51] AWS: CloudFormation, <https://aws.amazon.com/cloudformation/>, last accessed October 2019.
- [52] AWS CloudFormation Documentation: Working with Stacks, <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html>, last accessed October 2019.
- [53] AWS: Virtual Private Cloud, <https://aws.amazon.com/vpc/>, last accessed October 2019.

- [54] AWS: Amazon API Gateway, <https://aws.amazon.com/api-gateway/>, last accessed October 2019.
- [55] AWS: AWS Lambda, <https://aws.amazon.com/lambda/>, last accessed October 2019.
- [56] AWS: Elastic Load Balancing, <https://aws.amazon.com/elasticloadbalancing/>, last accessed October 2019.
- [57] AWS: Amazon Cognito, <https://aws.amazon.com/cognito/>, last accessed October 2019.
- [58] AWS: Amazon CloudWatch, <https://aws.amazon.com/cloudwatch/>, last accessed October 2019.
- [59] AWS: Amazon Relational Database Service (RDS), <https://aws.amazon.com/rds/>, last accessed October 2019.
- [60] AWS: Amazon S3, <https://aws.amazon.com/s3/>, last accessed October 2019.
- [61] AWS: Identity and Access Management, <https://aws.amazon.com/iam/>, last accessed October 2019.
- [62] Wikipedia: IPv6, <https://en.wikipedia.org/wiki/IPv6>, last accessed October 2019.
- [63] Techopedia: Website Security Certificate, <https://www.techopedia.com/definition/29743/website-security-certificate>, last accessed October 2019.
- [64] Wikipedia: Secure Sockets Layer (SSL) and Transport Layer Security (TLS), https://en.wikipedia.org/wiki/Transport_Layer_Security, last accessed October 2019.
- [65] Apigee: API Gateway, <https://apigee.com/about/cp/api-gateway>, last accessed October 2019.
- [66] Wikipedia: Representational state transfer, https://en.wikipedia.org/wiki/Representational_state_transfer, last accessed October 2019.
- [67] AWS Docs: What's the difference between Amazon Cognito user pools and identity pools?, <https://aws.amazon.com/premiumsupport/knowledge-center/cognito-user-pools-identity-pools/>, last accessed October 2019.
- [68] AWS Docs: Adding Groups to a User Pool, <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-user-groups.html>, last accessed October 2019.
- [69] AWS: Amazon Aurora, <https://aws.amazon.com/rds/aurora/>, last accessed October 2019.
- [70] AWS: EC2, <https://aws.amazon.com/ec2/>, last accessed October 2019.

- [71] OpenWrt: Wireless Freedom, <https://openwrt.org/about>, last accessed October 2019.
- [72] BlueMark: WiFi analytics, <https://bluemark.io>, last accessed October 2019.
- [73] MQTT: a lightweight messaging protocol, <http://mqtt.org/>, last accessed October 2019.
- [74] TechTarget: What is MQTT, <https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport>, last accessed October 2019.
- [75] Eclipse Mosquitto: An open source MQTT broker, <https://mosquitto.org/>, last accessed October 2019.
- [76] AWS Docs: How to Manage Things with the Registry, <https://docs.aws.amazon.com/iot/latest/developerguide/thing-registry.html>, last accessed October 2019.
- [77] AWS Docs: Amazon Resource Names (ARNs), <https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html>, last accessed October 2019.
- [78] AWS Docs: Attach a Certificate to a Thing, <https://docs.aws.amazon.com/iot/latest/developerguide/attach-cert-thing.html>, last accessed October 2019.
- [79] AWS Docs: Thing Policy Variables, <https://docs.aws.amazon.com/iot/latest/developerguide/thing-policy-variables.html>, last accessed October 2019.
- [80] AWS Docs: X.509 Certificates, <https://docs.aws.amazon.com/iot/latest/developerguide/x509-certs.html>, last accessed October 2019.
- [81] Wikipedia: JSON, <https://en.wikipedia.org/wiki/JSON>, last accessed October 2019.
- [82] AWS Docs: Device Shadow Service for AWS IoT, <https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html>, last accessed October 2019.
- [83] Imei: What is ICCID?, <https://www.imei.info/faq-what-is-ICCID/>, last accessed October 2019.
- [84] EnGenius: What is RSSI and its acceptable signal strength?, <https://helpcenter.engenlustech.com/hc/en-us/articles/234761008>, last accessed October 2019.
- [85] Docker: Modernize your applications, accelerate, innovation, <https://www.docker.com/>, last accessed October 2019.
- [86] Li, Guoquan and Geng, Enxu and Ye, Zhouyang and Xu, Yongjun and Lin, Jinzhao and Pang, Yu, *Indoor positioning algorithm based on the improved RSSI distance model*, Sensors, 18(9), p.2820., last accessed November 2019.

- [87] How to Calculate Distance from the RSSI value of the BLE Beacon, <https://iotandelectronics.wordpress.com/2016/10/07/how-to-calculate-distance-from-the-rssi-value-of-the-ble-beacon/>, last accessed November 2019.
- [88] AWS: Simple Notification Service, <https://aws.amazon.com/sns/> last accessed November 2019.
- [89] Wikipedia: Feature Selection, https://en.wikipedia.org/wiki/Feature_selection, last accessed November 2019.
- [90] Medium: Feature Selection Techniques in Machine Learning, <https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e>, last accessed November 2019.
- [91] RapidMiner Studio: Visual workflow designer for the entire analytics team, <https://rapidminer.com/products/studio/>, last accessed November 2019.
- [92] Domino: One place for your data science tools, apps, results, models, and knowledge., <https://www.dominodatalab.com/>, last accessed November 2019.
- [93] Machine Learning Mastery: How to Calculate Correlation Between Variables, <https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/>, last accessed November 2019.
- [94] RapidMiner Community: Auto Model and Variables Quality, <https://community.rapidminer.com/discussion/54916/auto-model-and-variables-quality>, last accessed November 2019.
- [95] Open Weather Map API, <https://openweathermap.org/api>, last accessed November 2019.
- [96] Z-Score Table, <http://www.z-table.com/>, last accessed November 2019.
- [97] AWS: Forecast, <https://aws.amazon.com/forecast/>, last accessed November 2019.
- [98] AWS: Forecast, Developer Guide, <https://docs.aws.amazon.com/forecast/latest/dg/forecast.dg.pdf>, last accessed November 2019.
- [99] AWS: Autoregressive Integrated Moving Average (ARIMA) Algorithm, <https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-arima.html>, last accessed November 2019.
- [100] AWS: DeepAR Forecasting Algorithm, <https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>, last accessed November 2019.
- [101] AWS: Exponential Smoothing (ETS) Algorithm, <https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-ets.html>, last accessed November 2019.

- [102] AWS: Non-Parametric Time Series (NPTS) Algorithm, <https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-npts.html>, last accessed November 2019.
- [103] AWS: Prophet Algorithm, <https://docs.aws.amazon.com/forecast/latest/dg/aws-forecast-recipe-prophet.html>, last accessed November 2019.
- [104] AWS: Step Function, <https://aws.amazon.com/step-functions/>, last accessed November 2019.
- [105] Wikipedia: Hyperparameter Optimization, https://en.wikipedia.org/wiki/Hyperparameter_optimization, last accessed November 2019.
- [106] ReactJS: A JavaScript library for building user interfaces, <https://reactjs.org/>, last accessed November 2019.
- [107] JSX: Introduction, <https://reactjs.org/docs/introducing-jsx.html>, last accessed November 2019.
- [108] W3Schools: HTML Styles - CSS, https://www.w3schools.com/html/html_css.asp, last accessed November 2019.
- [109] Creative Tim: Argon Design System, <https://demos.creative-tim.com/argon-design-system/>, last accessed November 2019.
- [110] Documentation for Styled Components, <https://www.styled-components.com/docs/basics>, last accessed November 2019.
- [111] Template literals (Template strings), https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals, last accessed November 2019.
- [112] Agiliq: Understanding State and Props in ReactJS, <https://www.agiliq.com/blog/2019/05/understanding-react-state-and-props/>, last accessed November 2019.
- [113] Redux Framework: A JavaScript predictable state container, <https://redux.js.org>, last accessed November 2019.
- [114] Redux Store, <https://redux.js.org/basics/store>, last accessed November 2019.
- [115] Redux Actions, <https://redux.js.org/basics/actions>, last accessed November 2019.
- [116] Redux Reducers, <https://redux.js.org/basics/reducers>, last accessed November 2019.
- [117] Github: Axios, <https://github.com/axios/axios>, last accessed November 2019.
- [118] Lottie: Easily add high-quality animation to any app, <https://airbnb.design/lottie/>, last accessed November 2019.

- [119] Stateful vs. Stateless Functional Components in React, <https://code.tutsplus.com/tutorials/stateful-vs-stateless-functional-components-in-react--cms-29541>, last accessed November 2019.
- [120] Binary Operator, <https://www.techopedia.com/definition/23953/binary-operator>, last accessed November 2019.
- [121] AWS: Amplify, <https://aws-amplify.github.io/docs/js/authentication>, last accessed November 2019.
- [122] Node.js: a platform built on Chrome's JavaScript runtime, <https://nodejs.org>, last accessed November 2019.
- [123] AWS Javascript SDK, <https://aws.amazon.com/sdk-for-node-js/>, last accessed November 2019.
- [124] AWS Docs: AWS Lambda Function Handler in Node.js, <https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-handler.html>, last accessed November 2019.
- [125] Github: MySQL Package, <https://github.com/mysqljs/mysql>, last accessed November 2019.
- [126] Node.js: Process object, <https://nodejs.org/api/process.html>, last accessed November 2019.
- [127] Javascript Promise, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise, last accessed November 2019.
- [128] Wikipedia: YAML syntax, <https://en.wikipedia.org/wiki/YAML>, last accessed November 2019.
- [129] SAM Template, <https://github.com/aws-labs/serverless-application-model>, last accessed November 2019.
- [130] System Usability Scale (SUS), <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>, last accessed November 2019.
- [131] System Usability Scale Template, <http://www.uxforthemasses.com/wp-content/uploads/2011/04/SUS-System-Usability-Scale.doc>, last accessed November 2019.
- [132] Axelra: Accelerating digital business models for corporates and startups with skin in the game, <https://axelra.com/>, last accessed November 2019.
- [133] AWS Pricing: Cognito, <https://aws.amazon.com/cognito/pricing/>, last accessed November 2019.
- [134] AWS Pricing: Lambda, <https://aws.amazon.com/lambda/pricing/>, last accessed November 2019.

- [135] AWS Pricing: RDS Aurora, <https://aws.amazon.com/rds/aurora/pricing/>, last accessed November 2019.
- [136] AWS Pricing: API Gateway, <https://aws.amazon.com/api-gateway/pricing/>, last accessed November 2019.
- [137] AWS Pricing: IoT Core, <https://aws.amazon.com/iot-core/pricing/>, last accessed November 2019.
- [138] AWS Pricing: S3 Storage, <https://aws.amazon.com/s3/pricing/>, last accessed November 2019.

Abbreviations

AP	Access Point
API	Application Program Interface
ARNs	Amazon Resource Names
AWS	Amazon Web Services
CDN	Content Delivery Network
CORS	Cross-Origin Resource Sharing
DB	Database
DNS	Domain Name System
ERD	Entity Relationship Diagram
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
KPI	Key Performance Indicator
IAM	Identity and Access Management
ICCID	Integrated Circuit Card Identifier
IDP	Identity Pool
IoT	Internet of Things
JSON	JavaScript Object Notation
MAC	Media Access Control
MAU	Monthly Active Users
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
M2M	Machine to Machine
OS	Operating System
PIWN	Personally Identifying Wireless Networks
RDS	Relational Database Service
REST	Representational state transfer
RNN	Recurrent Neural Networks
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
SNS	Simple Notification Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
SUS	System Usability Scale
TLS	Transport Layer Security
UI	User Interface
UP	User Pool

UX	User Experience
VPC	Virtual Private Cloud
YAML	Ain't Markup Language

List of Figures

3.1	Stakeholders: Entity Relationship Diagram	11
3.2	Dimensions: Zone of Interests	15
3.3	GL-iNet MiFi 4G Smart Router [34]	18
3.4	Design Evaluation: Clients Concept	22
3.5	Design Evaluation: Campaigns Concept	23
3.6	Design Evaluation: Activity Concept	23
3.7	Charts universe: Funnel 3D	24
3.8	Charts universe: Bar Chart	24
3.9	Charts universe: Heat Map	25
3.10	Charts universe: Histogram Series	25
3.11	Charts universe: Line Chart	26
3.12	Card-based Design Principle	27
3.13	Table-based Design Principle	27
3.14	Serverless Architecture and Shared Database.	30
4.1	Microservice-oriented Architecture: API Gateway and Lambda Functions. .	50
4.2	AWS IoT Concept: IoT Core, Sensors and Aggregated Data.	66
4.3	AWS IoT Alerting Mechanism.	71
5.1	System Usability Scale Results	74
5.2	Activity Form - First Step	75
5.3	Activity Form - Second Step	76

5.4 Activity Form - Third Step 76

5.5 Activity Overview - First Part 77

5.6 Activity Overview - Second Part 77

5.7 Agencies Overview 78

5.8 Campaign Overview 78

5.9 Sensors Overview 79

List of Tables

3.1	User- and Role-based executable Actions within the Platform.	13
3.2	KPIs Formulas.	20
3.3	Unique Count Adjustment to ensure Comparability between Marketing Activities.	32
3.4	Normalized Average Dwell Time.	33
3.5	Adjusted Dwell Time by Visit Gap and Dwell Time Thresholds.	33
3.6	Adjusted Visibility.	34
3.7	Adjusted Interaction Rate.	35
5.1	Activity for Evaluating Prediction Model Precision.	79
5.2	Actual computed Metrics for a randomly selected Activity.	80
5.3	Predicted Activity Metrics.	80
5.4	AWS Services Pricing Models.	81

Appendix A

Installation Guidelines

A.1 Front-End

1) In the root folder install all dependencies with either *yarn* or *npm*.

```
$ yarn install
```

2) Create an *.env* file with yours API keys and Amplify Configuration in the root folder. For running it locally, an *.env.local* can be created with dummy data as well. For instance, a local *env* file could look like this:

```
1 NODE_PATH=src
2
3 REACT_APP_AWS_ACCESS_KEY_ID= 'AAABBBCCC'
4 REACT_APP_AWS_SECRET_ACCESS_KEY_ID= 'DDDEEEFFF'
5
6 REACT_APP_AWS_IDENTITY_POOL_ID= 'eu-west-GGGHHHHII'
7 REACT_APP_AWS_REGION= 'eu-west-XY'
8
9 REACT_APP_AWS_USER_POOL_ID= 'eu-west-1_ZZZ'
10 REACT_APP_AWS_USER_POOL_WEBCLIENT_ID= 'LLLMMMNNN'
11
12 REACT_APP_AWS_DOMAIN=FALSE
13
14 REACT_APP_API_PATH= 'http://127.0.0.1:3001'
```

3) Start the Web application (will be listening on *http://127.0.0.1:3001*)

```
$ yarn start
```

A.2 Back-End

A.2.1 Installation

- 1) Install and start Docker ¹
- 2) Install AWS CLI ²
- 3) Install AWS SAM CLI ³
- 4) Add AWS credentials to the AWS credentials file.

```
$ cd /.aws/credentials
```

- 5) Add default AWS region to the AWS configuration file.

```
$ cd /.aws/config
```

- 6) Create an *env.json* file for running the back-end locally.

```
1 {  
2   "HiWorld": {  
3     "EXAMPLE_ENV": "\nThis is an example for using a non-global env  
                     variable."  
4   }  
5 }
```

- 7) Install node modules. Node modules folders of other microservices are not ignored on purpose, since the lambda functions are using them.

```
$ npm install
```

- 8) Start the local (or dev) server.

```
$ npm run local | npm run dev
```

¹<https://www.docker.com/>

²<https://aws.amazon.com/cli/>

³<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html>

A.2.2 Deployment to AWS

1) Set *env.* variables (Set your *PREFERED_SUFFIX* or the whole *CLOUDFORMATION_NAME*). If AWS profile has not been specified, keep this *AWS_PROFILE* variable on *default*.

```
1 export ARTIFACT_STORE=XX-backend
2 export CLOUDFORMATION_NAME=YY-<PREFERED_SUFFIX>
3 export ENVIRONMENT=dev
4 export AWS_PROFILE=default
```

2) Build and publish the Stack to AWS.

```
$ aws cloudformation package -template-file template.yaml -output-template-file outputtemplate.yaml -s3-bucket XX-backend
```

```
$ aws cloudformation deploy -template-file outputtemplate.yaml -stack-name my-stack-name -capabilities CAPABILITY_IAM
```


Appendix B

Contents of the CD

The attached CD consists of the following files and directories:

- **Application:** Contains the source code of the application.
 - *FrontEnd*: Source code of the front-end Web application.
 - *BackEnd*: Source code of the microservice-based back-end.
- **Related Work:** Contains PDF documents of the related work material.
- **Thesis:** Contains documents of the written report.
 - *Zusfsg.txt*: Plain text version of the German abstract.
 - *Abstract.txt*: Plain text version of the English abstract.
 - *Sources*: Latex source files used to generate the content of the thesis.
 - *Thesis.pdf*: PDF version of the written report with sensitive content.
 - *Open_Thesis.pdf*: PDF version of the written report without sensitive content and that can be published.