



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Background Geolocation Features in the Context of Mobile Applications

TESI DI LAUREA MAGISTRALE IN COMPUTER
SCIENCE AND ENGINEERING-INGEGNERIA
INFORMATICA

Author: **Ludovico Righi**

Student ID:	10769559
Advisor:	Luciano Baresi
Co-advisor:	Julius Willems
Academic Year:	2022-23

Abstract

The construction industry faces challenges of health, safety, and efficiency for on-site workers. In Switzerland, regulations like the "Federal Labour Act" and "Landesmantelvertrag für das schweizerische Bauhauptgewerbe" ensure working conditions but also introduce administrative complexities. In particular, error-prone work hour recording methods prevail, lacking user support. Addressing these administrative obstacles and enhancing accuracy could significantly enhance industry efficiency, ultimately leading to financial benefits and sustainability improvements.

The research presented in this thesis emerged through collaboration with Benetics, an early-stage startup located in Zurich, Switzerland. Benetics specializes in mobile and web applications tailored for the construction industry. These applications encompass a wide array of purpose-built features including project and task management, a plan viewer with multimedia pin annotations, centralized communication, and more.

Within this framework, the primary focus was initially directed towards integrating a background geolocation framework. This involved a comprehensive evaluation, testing, and refinement process of various libraries. Subsequently, the emphasis shifted to the implementation of functionalities that leverage this type of data. Examples of such features include support for a time card management system and the capability to remotely verify on-site presence at specific construction sites.

This study contributes to the optimization of the industry through technology-driven approaches. It underscores the advantages of background geolocation in automating tasks, enhancing the monitoring of workers, and refining the management of construction sites. Additionally, the thesis underscores potential privacy concerns and investigates the feasibility of other location-aware features, designed to adhere to the business requirement of exclusively utilizing smartphones as the data source.

Keywords: background geolocation, time cards management, mobile applications development, construction industry, privacy and workers monitoring.

Abstract in lingua italiana

L'industria delle costruzioni affronta sfide legate alla salute, alla sicurezza e all'efficienza dei lavoratori in cantiere. In Svizzera, normative come la "Federal Labour Act" garantiscono le condizioni di lavoro, ma introducono anche complessità amministrative. In particolare, prevale l'uso di metodi di registrazione delle ore di lavoro soggetti a errori e privi di supporto per gli utenti. Riducendo gli oneri amministrativi e migliorando l'accuratezza, c'è il potenziale per aumentare notevolmente l'efficienza del settore, contribuendo così a guadagni finanziari e alla sostenibilità.

La ricerca presentata in questa tesi è emersa attraverso la collaborazione con Benetics, una startup in fase iniziale con sede a Zurigo, in Svizzera. Benetics è specializzata nello sviluppo di applicazioni mobili e web su misura per l'industria delle costruzioni. Queste applicazioni comprendono una vasta gamma di funzionalità specifiche, tra cui la gestione di progetti e attività, un visualizzatore di piani con annotazioni multimediali, comunicazioni centralizzate e altro ancora.

In questo contesto, il focus principale è stato inizialmente quello di introdurre un framework di geolocalizzazione in background. Questo ha coinvolto una valutazione completa e una fase di test di varie librerie. Successivamente, l'attenzione si è spostata all'implementazione di funzionalità che sfruttano questo tipo di dati. Esempi di tali funzionalità includono il supporto per un sistema di gestione dei cartellini orari e la capacità di verificare a distanza la presenza in cantiere presso siti di costruzione specifici.

Questo studio mira all'ottimizzazione dell'industria attraverso la tecnologia, sottolineando i vantaggi della geolocalizzazione in background nell'automazione delle attività, nel monitoraggio dei lavoratori e nel perfezionamento della gestione dei cantieri. Inoltre, la tesi evidenzia potenziali problemi legati alla privacy e indaga sulla fattibilità di altre funzionalità basate sulla localizzazione, progettate per rispettare l'esigenza aziendale di utilizzare esclusivamente gli smartphone come fonte di dati.

Parole chiave: geolocalizzazione in background, gestione cartellini orari, sviluppo applicazioni mobile, software per edilizia, privacy e monitoraggio lavoratori.

Contents

Abstract.....	i
Abstract in lingua italiana	iii
Contents	v
1. Introduction	1
1.1 Structure of the thesis.....	2
2. Context	5
2.1 The Startup: Benetics.....	5
2.1.1 Key Features.....	6
2.1.2 Market, Competitors and Vision.....	8
2.1.3 Business Model and Distribution.....	10
2.1.4 The Team	11
2.2 Efficiency Opportunity	11
2.3 Objectives and Scope.....	12
3. Background Geolocation	13
3.1 Geofencing.....	14
3.2 Libraries	14
3.2.1 Transistor Software Library	15
3.3 Data.....	19
3.3.1 Geolocation Data	19
3.3.2 Activity Data	19
3.4 Adoption of the Transistor Software Library	20
3.4.1 Parameters Configuration.....	20
3.4.2 Keeping the Library in Synch with the Database	23
3.5 Testing of the library	28
3.5.1 Testing on Simulators	28

4. Tech Stack	31
4.1 Backend	31
4.1.1 AWS Dynamo DB	31
4.1.2 Serverless Framework	35
4.2 Frontend	37
5. Implementation	39
5.1 Feature: Remote Personnel Presence Monitoring	39
5.1.1 Requirements	39
5.1.2 Design	40
5.1.3 User Interface	41
5.1.4 Testing	42
5.2 Feature: Time Cards Manager	42
5.2.1 Requirements	42
5.2.2 Design	43
5.2.3 User Interface	43
6. Data Models	53
6.1 Queries	53
6.2 Entities	53
6.2.1 Geofence	54
6.2.2 GeofenceEvent	55
6.2.3 Timecard	57
6.3 APIs	59
6.3.1 REST	59
6.3.2 API Models and Routes	60
6.4 Entities Clean-Up	67
6.4.1 Clean-Up Lambdas	67
6.4.2 Soft Deletions	73
7. Privacy Concerns	75
7.1 Data Access Permissions	75
7.2 Users Trust	77
7.3 Regulations	78

7.4 Mitigations..... 78

8. Conclusions 81

8.1 Summary of Contributions..... 81

8.2 Future Developments..... 81

8.2.1 General Improvements in Geofencing 82

8.2.2 Intelligent Location Based Task Management & Reminders..... 82

8.2.3 Location Based Safety Alerts 82

8.2.4 Human Activity Recognition Based Safety Alert 83

8.2.5 Manual Stopwatch for Time Cards Management 83

8.3 Learnings..... 84

Definitions..... 87

Acronyms 89

Bibliography..... 91

A. Appendix A 95

List of Figures..... 97

Acknowledgements 99

1. Introduction

The construction industry stands as a significant economic force across nations; however, it remains riddled with inefficiencies stemming from management and organizational shortcomings. A prevailing cause of this inefficiency is the absence of specialized software tailored ad-hoc to streamline processes and reduce overhead.

Traditional communication channels like emails, phone calls, and standard messaging applications persist as the dominant technologies, leading to time wastage due to misunderstandings [35], decentralized communication, and information sparsity [36]. In a sector laden with inefficiencies, McKinsey's projection of potential annual savings of \$1.6 trillion through efficiency improvements underscores the transformative impact that can be achieved [37].

Tight schedules in combination with hard physical labor render a health and safety hazard for many construction site workers worldwide [19, 20]. In Switzerland, the "Federal Labour Act" (ArG) and the "Landesmantelvertrag für das schweizerische Bauhauptgewerbe" (LMV) regulate the working and resting time requirements of construction site workers [41, 43]. While the primary objective of these protective guidelines is to ensure healthy working conditions, they entail a significant administrative overhead for both employees and employers.

The adoption of technology presents a solution: mobile and web applications emerge as indispensable tools to streamline operations, monitor progress, and facilitate communication. Through collaboration with the early-stage startup Benetics in Zurich, this thesis endeavors to implement features based on background geolocation technology within their existing mobile app, with the goal of enhancing construction sites efficiency.

Background geolocation technology holds promise, enabling devices to track their location even during periods of inactivity. Smartphones carried by workers can gather a series of time-stamped, geo-tagged data points alongside activity data from Inertial Measurement Units. This technology, coupled with the collected data, will aid in supporting and potentially automating various processes such as time card management, worker tracking, and remote presence verification on construction

sites. The extended app contributes to streamlining operations and elevating productivity within the construction industry.

The outcomes of this thesis will not only demonstrate the potential of background geolocation technology to enhance construction industry efficiency, but also highlight its limitations and principal challenges. By showcasing the potential of technology-driven industry advancement, this study's results underscore the transformative capacity of background geolocation in reshaping the efficiency landscape of the construction industry.

Furthermore, this study addresses considerations and concerns regarding worker privacy and outlines potential avenues for continued development within the framework established by this thesis. Notably, the nature of this thesis work remains deeply practical, experimental, and aligned with tangible, evolving business requirements.

This pursuit has offered a comprehensive experience in a real and empowering work environment, allowing me to engage across the entirety of the technology stack – from backend business logic to AWS serverless services and frontend interface development.

1.1 Structure of the thesis

This thesis is organized as follows:

- **Chapter 1 (Introduction)** introduces the covered topics and provides a high-level overview of the work. It also presents the thesis structure to guide the reader throughout the document.
- **Chapter 2 (Context)** first introduces the startup I collaborated with and the environment where I have developed my thesis work and then highlights the main opportunities and problems, as well as goals and objectives of the thesis.
- **Chapter 3 (Background Geolocation)** offers readers comprehensive introduction to background geolocation and an analysis of the libraries offering this kind of functionality; it also includes insights related to the adoption and testing of the chosen library.
- **Chapter 4 (Tech Stack)** presents the technological stack together with the most important concepts and characteristics of it.
- **Chapter 5 (Implementation)** delves into the details of the implementation of two main features, namely the Remote Personnel Presence Monitoring and the Time Cards Manager.
- **Chapter 6 (Data Models)** explains the design and implementation of the data models relevant to this project, emphasizing critical design decision trade-offs.

- **Chapter 7 (Privacy Concerns)** addresses potential privacy issues associated with background geolocation data usage. While not exhaustive, it provides thought-provoking insights on these pivotal technological aspects.
- **Chapter 8 (Conclusions)** offers conclusions, summarizing contributions and takeaways from this work. It also outlines possibilities for future development.

The reader can refer to the **Definitions** and **Acronyms** sections to clarify any uncertainties while navigating this thesis. Finally, the **Bibliography**, a concise **Appendix**, **List of Figures**, and **Acknowledgments** follow.

2. Context

This chapter sets the stage for the context in which my thesis work was undertaken: firstly, a quick overview of Benetics is offered and then two sections regarding the opportunities and objectives are presented.

2.1 The Startup: Benetics

In the pursuit of my thesis work, I had the privilege of collaborating with Benetics, a forward-looking software startup that became the backdrop of my work and contributions. As a Software Engineer Intern, I engaged with Benetics for three months, from May to July 2023.

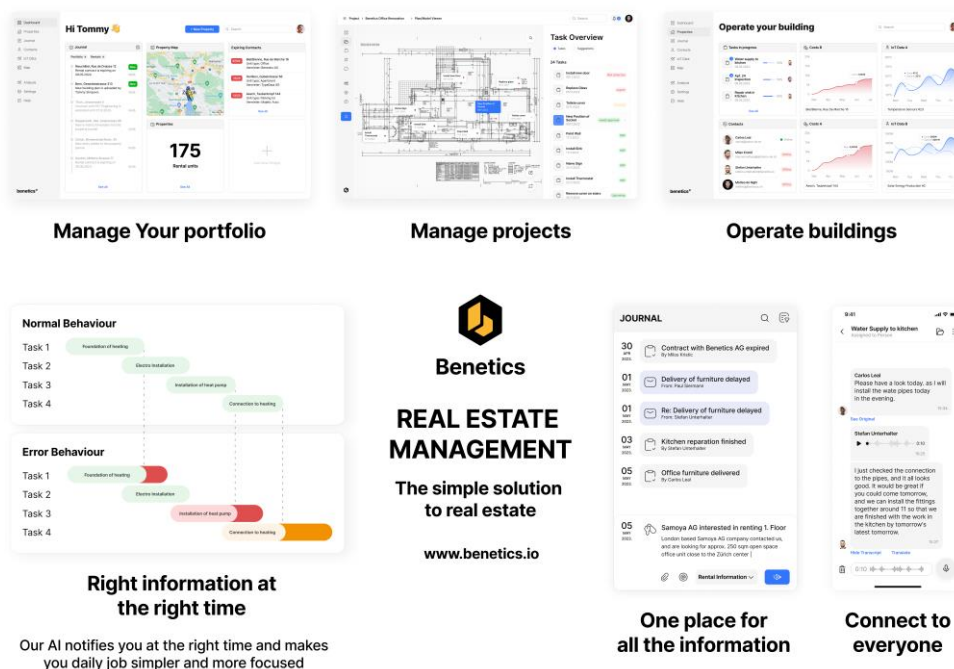


Figure 2.1: Benetics, the startup.

Benetics stands as a cutting-edge startup focused on revolutionizing the construction industry. The startup's primary objective revolves around tackling the challenge of

managing information complexity - streamlining data flow to offer users only the pertinent information while concurrently elevating their productivity through insightful guidance in their daily operations.

2.1.1 Key Features

At the time of my internship, the Benetics showcased several key features:

- **Project and Task Management:** The app encompassed a user-friendly yet robust task management workflow that facilitated efficient organization of workflows, task assignment, and task prioritization.
- **Plan Viewer with Pin Annotations:** A notable capability of the application lay in its ability to digitize project plans. This feature allowed users to incorporate images, text, and audio annotations, thereby fostering seamless information sharing within project-related chats and among team members.

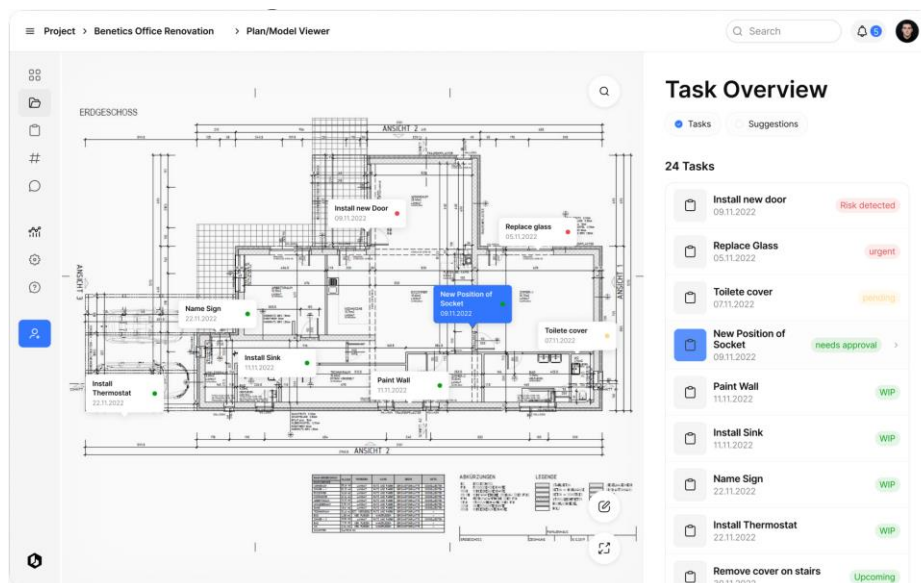


Figure 2.2: Benetics, project and task management and plan viewer.

- **Chats with Translations:** The app bridged language barriers by facilitating communication across diverse linguistic backgrounds. This feature enabled effective interactions and collaborations among individuals conversing in different languages.

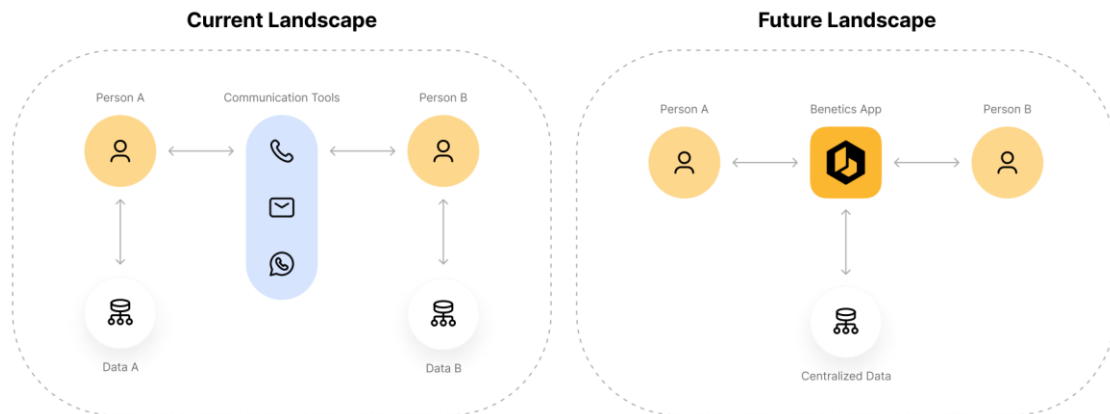


Figure 2.3: Benetics as communication tool.

A distinguishing characteristic of the app is its intuitive and user-friendly interface, aspects that have garnered significant acclaim from early adopters.

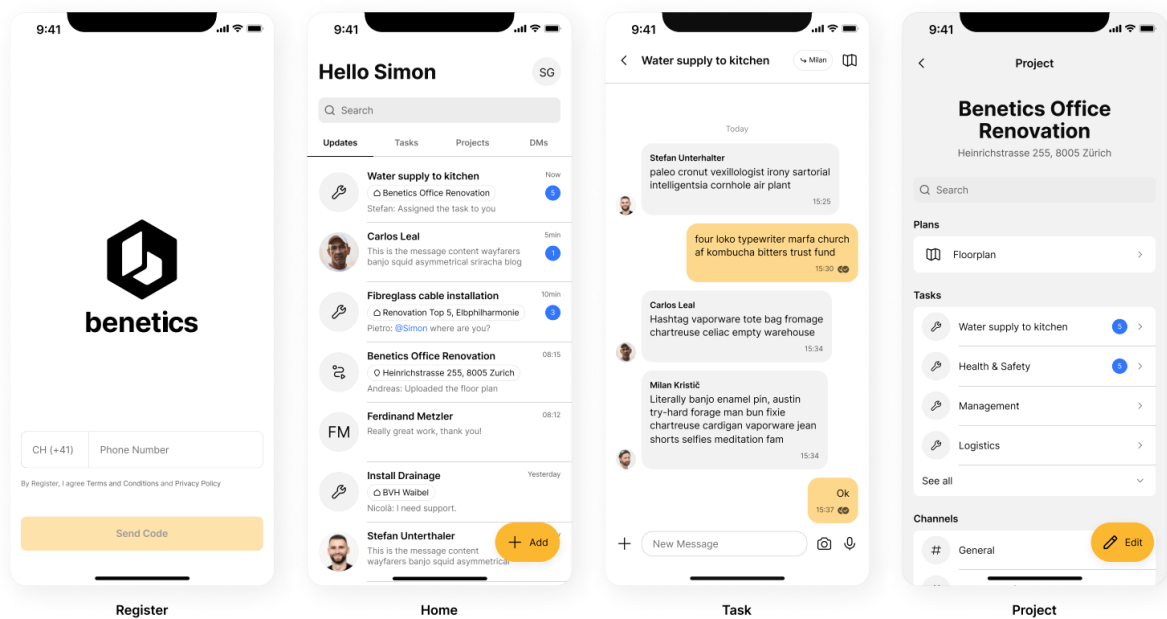


Figure 2.4: Benetics, some UIs.

Continuously evolving on a weekly basis, the application maintains a dynamic trajectory of enhancement and expansion. For more comprehensive information, please refer to the Benetics official website [4] or experience the application firsthand by downloading it from the App Store [6] or the Play Store [7].

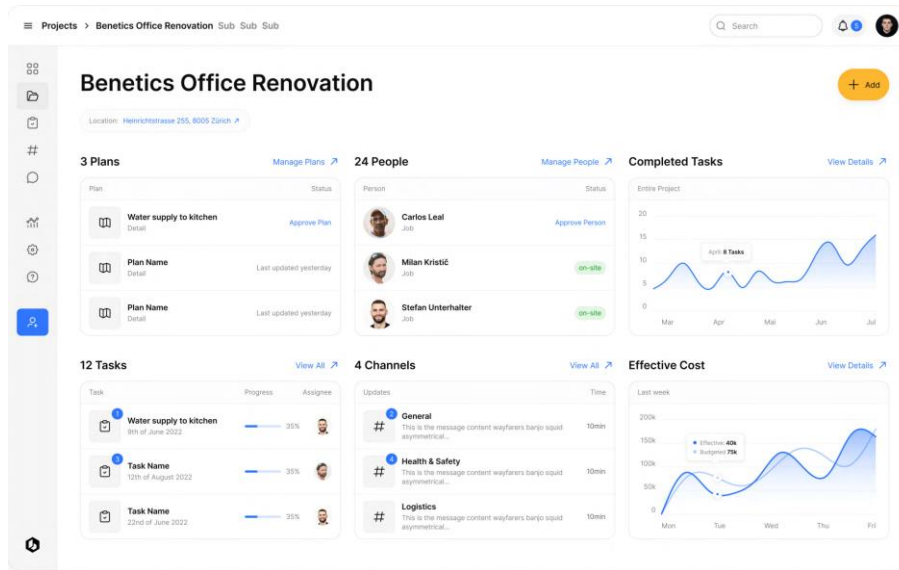


Figure 2.5: Benetics, analytics with the web app.

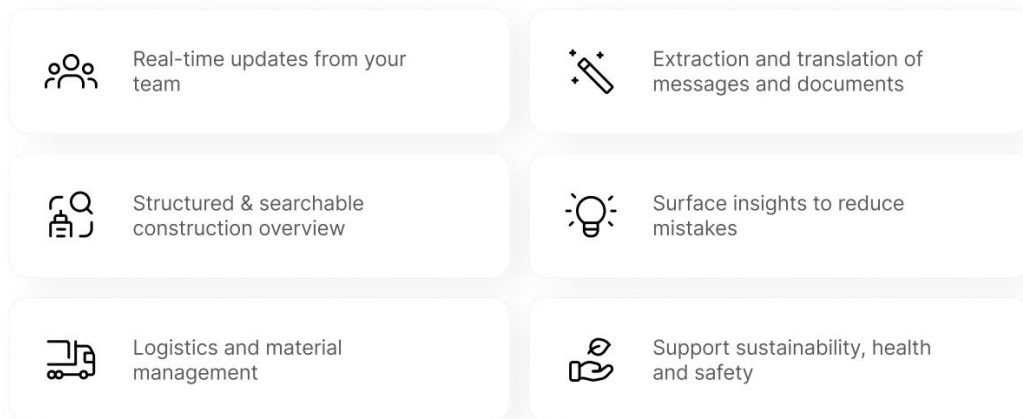


Figure 2.6: Benetics, key features summary.

2.1.2 Market, Competitors and Vision

The construction industry presents an expansive addressable market [37, 41] for software solutions like Benetics. With an ever-increasing demand for streamlined project management, enhanced communication, and information accessibility, the startup is well-positioned to cater to a wide array of construction professionals, from project managers to on-site workers.



Figure 2.7: Benetics, addressable challenges and market [38].

Benetics envisions a future where the construction landscape is seamlessly connected, communication barriers are eradicated, and project management becomes an exercise in precision and efficiency. The startup's vision aligns with the larger industry shift toward digitization, automation, data-driven [40] decision-making and attention towards environment [39].

With a user-centric approach and a commitment to addressing the industry's pain points, Benetics aspires to be the premier choice for construction professionals seeking innovative solutions that elevate their capabilities.

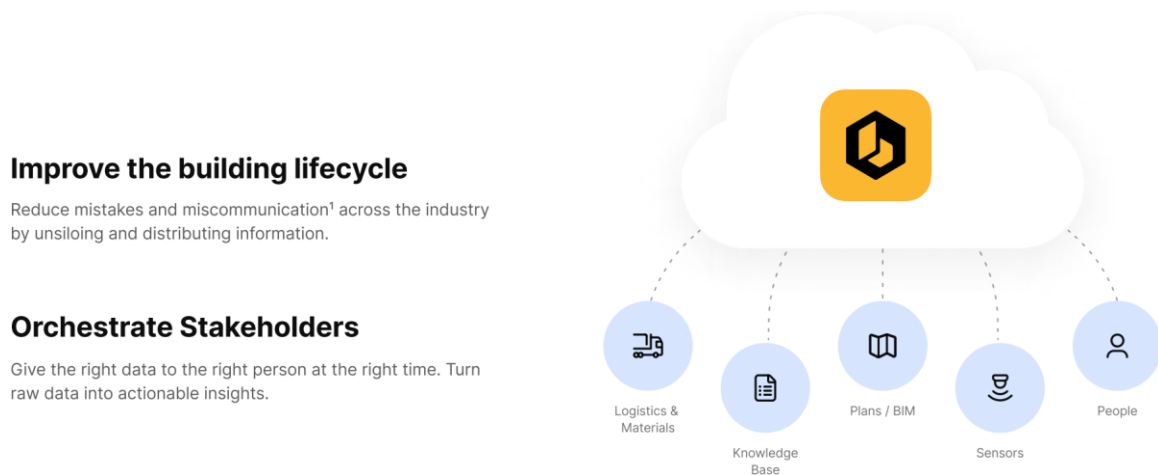


Figure 2.8: Benetics, the vision.

In the ever-evolving landscape of software solutions for the construction industry, Benetics encounters a diverse array of competitors, even if the already present solutions are generally not directly targeting some of the characteristic problems of the industry and in general are not so sophisticated.

The following figure compares Benetics with the main competitors, highlighting the features that the different applications offer.





	Background	Key Features	Target Customer	
	>2B active users Part of Meta Platforms Most used application in the construction industry	<ul style="list-style-type: none"> ✓ Chat ✓ Mobile first ✗ Project and task management 	<ul style="list-style-type: none"> ✗ Construction specific features ✗ Translations ✗ Data warehouse and analytics 	Everyone
	iPad application focusing on task and plan distribution Sold to Hilti in 2021 - 300M\$	<ul style="list-style-type: none"> ✓ Chat ✓ Mobile first ✓ Project and task management 	<ul style="list-style-type: none"> ✓ Construction specific features ✗ Translations ✗ Data warehouse and analytics 	Construction managers General contractors
	Integrated communication tool for the construction industry for Autodesk products, Sold to Autodesk in 2018 - 800M\$	<ul style="list-style-type: none"> ✓ Chat ✓ Mobile first ✓ Project and task management 	<ul style="list-style-type: none"> ✓ Construction specific features ✗ Translations ✗ Data warehouse and analytics 	Construction managers General contractors
	Communication platform focusing on the communication to the front line workers We build the data warehouse of the construction industry.	<ul style="list-style-type: none"> ✓ Chat ✓ Mobile first ✓ Project and task management 	<ul style="list-style-type: none"> ✓ Construction specific features ✓ Translations ✓ Data warehouse and analytics 	Construction workers Construction managers General contractors Owners Facility Operators

Figure 2.9: Benetics, the competitors.

2.1.3 Business Model and Distribution

Benetics operates on a Business-to-Business (B2B) Software-as-a-Service (SaaS) model that is contingent on functionality and the number of users. This model allows construction companies to tailor their subscription based on the specific features they require and the size of their workforce. The dynamic pricing structure ensures that businesses of all scales can benefit from Benetics' offerings.

The distribution strategy is focused on reaching front-line workers, a vital component of the construction workforce. Multiple channels are employed, including direct partnerships with construction firms, collaborations with industry associations, and outreach through online platforms. This multifaceted approach ensures that Benetics effectively reaches and supports the end-users who play a critical role in project execution.

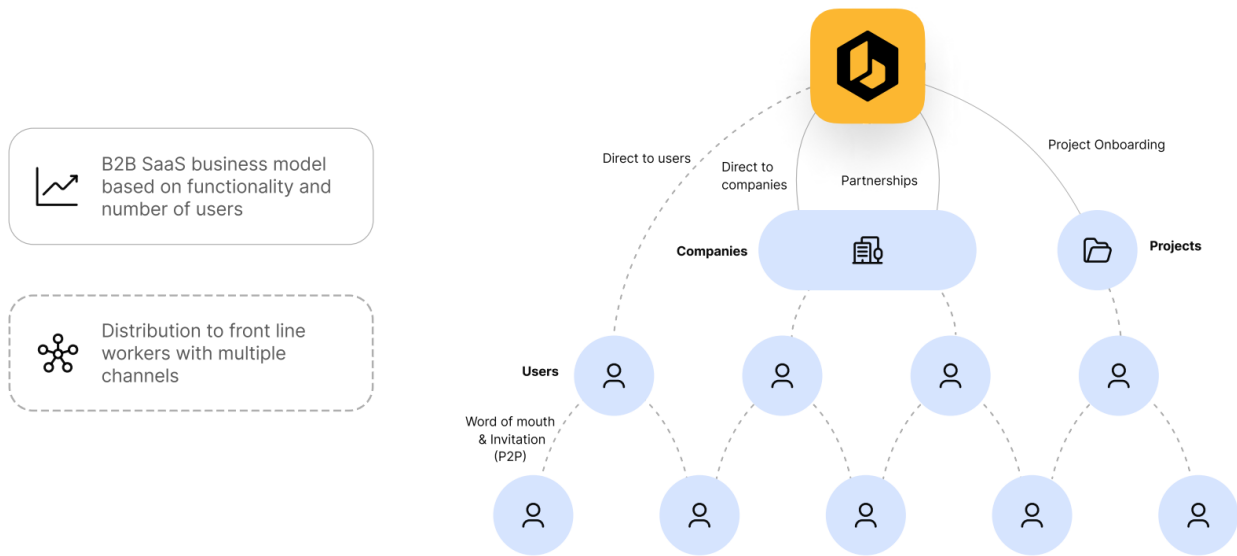


Figure 2.10: Benetics: business model and distribution.

2.1.4 The Team

At the time I joined the company, the team was still small, but really experienced as can be seen in the figure below. The team is, however, expanding very rapidly as the product is getting more ready and the customers base is growing in size month after month.




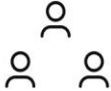



			
<p>Ferdinand Metzler Co-Founder and CEO Software entrepreneur with a successful exit to Zalando in 2020 MSc. in Mechanical Engineering ETH</p> 	<p>Dr. Aaron Shon Co-Founder and Co-CTO Director of software engineering at Google 14+ years Ph.D. Computer Science, University of Washington</p> 	<p>Johan Tibell Co-Founder and Co-CTO Staff software engineer at Google 15+ years MSc. in Computer Science, Chalmers University of Technology</p> 	<p>Team +2 software engineers +1 UI/UX designer</p>

Figure 2.11: Benetics, the team.

2.2 Efficiency Opportunity

In the dynamic realm of the construction industry, the significance of efficient time card management cannot be overstated. The financial remuneration of diligent

workers is intricately linked to the accurate recording of their toiled hours, making this process a linchpin of fairness.

However, despite the industry's advancement, the practice of time card management has, for the most part, remained wedded to traditional manual methods. This situation is not without its drawbacks; the reliance on manual input increases the vulnerability to errors and inaccuracies, leading to potential wage discrepancies and a compromised trust in the compensation system.

Additionally, this manual approach entails a substantial investment of time and effort, with administrative tasks often overshadowing more productive activities.

Hence, the prevailing approach, while familiar, is rife with shortcomings that impede efficiency and equity. The call for an evolution in time card management practices is evident, pointing toward a digital transformation that streamlines processes, minimizes errors, and optimizes workforce compensation.

2.3 Objectives and Scope

Within the ambit of my internship and thesis endeavor, the pursued objectives were as follows:

- **Introduction of Background Geolocation Library:** A pivotal objective lay in the adoption of a specialized library proficient in executing background geolocation within the mobile application. It is noteworthy that the integration of a background geolocation framework may be extremely useful for future development and other possible features; therefore, this task must be done in a general way and not directly targeting the Timecard Management System.
- **Implementation of a Timecard Management System powered by Background Geolocation:** A core directive involved the development of an integrated timecard management framework leveraging background geolocation capabilities. This system aimed to enhance efficiency by affording insights into worker presence at designated construction sites.
- **Enabling On-Site Presence Visibility:** An additional goal was to empower the system to furnish a comprehensive view of individuals physically located at specific construction sites, further enhancing project coordination and resource allocation.

3. Background Geolocation

In the context of mobile applications, Background Geolocation refers to the capability of tracking a device's location even when the app is running in the background or when the device is locked. This means that the app can continue to gather location data and respond to geofencing events, notifications, or other triggers even if the user is not actively using the app on their device.

Background Geolocation is important for various types of applications that require continuous location tracking, such as:

- **Navigation Apps:** apps like Google Maps or Waze need to track a user's location in the background to provide accurate turn-by-turn directions even when the app is not open on the screen.
- **Fitness Apps:** apps that track distance, speed, and routes during activities like running or cycling often rely on Background Geolocation to provide accurate data throughout the entire workout.
- **Location-Based Alerts:** apps that send notifications or alerts based on a user's location, such as geofencing notifications or location-specific reminders, require Background Geolocation to monitor the user's movements and trigger the appropriate actions.
- **Location Sharing:** apps that allow users to share their real-time location with friends or family members need Background Geolocation to maintain accurate and up-to-date location information.
- **Ridesharing and Delivery Services:** services like ridesharing or food delivery apps need Background Geolocation to track the driver's location and provide real-time updates to users.

Implementing Background Geolocation effectively requires careful consideration of factors like battery consumption, privacy concerns, and data usage. Mobile operating systems often have mechanisms in place to manage background processes, including location tracking, to balance the app's functionality with the device's performance and battery life.

It is important to adhere to best practices and guidelines to ensure a positive user experience while respecting the user's privacy and device resources.

3.1 Geofencing

One important notion in the context of background geolocation is the one of geofencing.

In the context of a mobile application, geofencing refers to the use of location-based technology to create a virtual boundary or perimeter around a physical geographic area. When the device crosses the virtual boundary, the application can trigger specific actions, notifications, or alerts.



Figure 3.1: Geofencing

This tracking mechanism is very important to collect raw data that is necessary to enable more sophisticated features.

3.2 Libraries

Before delving into library selection, a comprehensive evaluation of diverse alternatives was imperative. Navigating this landscape necessitated a dual consideration: prioritizing speed and maintaining cost-efficiency, aligning with the startup's needs. Although more feature-rich options like the costlier Radar [12, 29] and Mapbox [30] remained attractive, their affordability did not align with Benetics' current stage.

Notably, the design of features was agnostic to specific libraries, ensuring easy transition in the future. Therefore, data collection methods wouldn't disrupt feature enablement.

3.2.1 Transistor Software Library

The React Native Background Geolocation [2] from Transistor Software [1] is the library I decided to use after the initial evaluation phase.

3.2.1.1 *Philosophy*

The core philosophy of Background Geolocation is to track a device's location in the most battery-efficient manner possible. For this reason, motion detection - detecting when the device is still or moving - is central to this philosophy. Only when the device is detected to be moving will the plugin engage location-services. When the device is sitting still, location-services are off.

3.2.1.2 *Tracking Modalities*

Two key modalities characterize the library's operation:

- **[M1] Tracking Geofences and Location** (i.e., with “start()” method): in this modality, the library tracks entering and exiting events to and from geofences and also captures location data point every fixed interval.
- **[M2] Tracking Geofences Only** (i.e., with “startGeofences()” method): in this modality, the library tracks only geofence events; therefore, this modality is a “subset” of the first one (M1).

In this way, it is possible to implement the library in such a way that is toggling between the two modalities when certain events happen. For example, the switch can be performed whenever a geofence event occurs:

- If the user has just entered a new geofence and he was not in any other geofence (i.e., enter event), the library can switch from M2 to M1.
- If the user passes from a geofence area to a non-geofence area (i.e., exit event), the library can switch from M1 to M2.

Even if the modalities switch was initially considered in order to have a finer granularity within geofences, it was eventually decided to keep tracking only geofences in order not to impact too much phones battery life.

It is also possible to introduce additional logic: for example, in the code snippet below, the switch happens only for a specific geofence which represents a danger zone.

```
// Listen to geofence events
BackgroundGeolocation.onGeofence(geofence => {
  if (geofence.identifier == 'DANGER_ZONE') {
    if (geofence.action == 'ENTER') {
      // Entering the danger-zone, we want to aggressively track location.
      BackgroundGeolocation.start();
    } else if (geofence.action == 'EXIT') {
      // Exiting the danger-zone, we resume geofences-only tracking.
      BackgroundGeolocation.startGeofences();
    }
  }
})
```

3.2.1.3 Activity Recognition

The library also provides support for activity recognition; in particular, the followings are the different types of activities that the library supports:

- still
- walking
- on_foot
- running
- on_bicycle
- in_vehicle
- unknown

Moreover, the library always provides together with the activity type also a confidence number, which represents the confidence of the reported device motion activity in percentage.

Therefore, the library can be configured in a way that the activity type and the confidence level are collected and sent to the database every time that a geofence event occurs or, more in general, every time that a timestamped location point is registered.

After initially considering the possibility of using the activity recognition support in order to enable more advanced features, the decision not to use it was eventually taken. The main reasons in support of this are the following:

- The accuracy of the recognized activities was not high enough, especially in smartphones that are not so advanced and high quality, to support any kind of feature in a reliable way.
- The use of activity recognition does not really make sense if only geofence data points are monitored. This is because the number of geofence data points

highlighting, for example, entry points with green arrows, exits points with red arrows and geofence with circles.

It is straightforward to configure the console to receive data by using the “findOrCreateTransistorAuthorizationToken()” method provided by the library itself.

```
const authToken: TransistorAuthorizationToken =
  await BackgroundGeolocation.findOrCreateTransistorAuthorizationToken(
    'BENETICSORG',
    user.first_name.concat(' ').concat(user.last_name),
  );
```

The debugging dashboard provides support to filter on the names of the users from which data were collected – which is why in the code first and last names were added to the parameters - and on date and time as well.

Notice that, once the testing and debugging phase has been terminated, it is important to turn off this relay of data to avoid privacy problems. In fact, these data are accessible not only to TransistorSoftware itself, but also to anyone who accesses the URL identifier that has been configured (i.e., “BENETICSORG” in the example figure above.)

3.2.1.5 *Limitations of the Library*

Notably, the chosen library is not without limitations:

- **Only Circular-Shaped Geofences:** It is only possible to create circular shaped geofences and not arbitrary shaped polygonal geofences. It is however possible to aggregate multiple circular geofences for creating different shapes. This requires a bit more effort because multiple geofences must be considered as a single one.
- **Accuracy:** The library’s accuracy ranges between 150-200 meters. Having tried different libraries, I can quite confidently state that this is actually a limitation of smartphones precision, no matter what the library is declaring. Clearly, smartphones are not optimized for the task of background geofencing and therefore the data that is collected has a certain variance.
- **Occasional Redundant Geofence Events:** Sometimes the library collects repeated geofence events even if only a single one happened. For example, two enter (resp. exit) events in a row with respect to a single geofence, without any exit (resp. enter) event in between. This problem was mitigated by simply storing all the geofence events and then decide which geofence events to consider; for example, a heuristic approach which works well is to always take the first enter (resp. exit) geofence event of the series of repeated enter (resp, exit) events. Moreover, this series of repeated events have very similar

timestamps, so at the end of the day, the impact on the functioning of the feature is not even relevant.

Despite these limitations, the selected library aligned with project needs, underpinned by cost-effectiveness, longevity, and its supplementary debugging tool.

3.3 Data

This section delves into the explanation of the kind of data that are typically used by these kinds of libraries in order to perform background geolocation.

3.3.1 Geolocation Data

The most important kind of data for background geofencing are obviously location ones. Clearly, the higher the precision and accuracy of the data points that we want the higher the battery consumption [24]. It is therefore important to always keep in mind this tradeoff and understand what the best configuration is based on the features that want to be implemented. The following list shows the different sources of geolocation data from the most to the least accurate.

- **GPS (Global Positioning System):** precision between approximately 10-20 meters; the GPS also provides information about speed, heading and altitude.
- **Wi-Fi:** precision between approximately 30-500 meters.
- **Cell towers:** precision between approximately 300-3000 meters.
- **IP address:** precision between approximately 1000-5000 meters.

3.3.2 Activity Data

To make libraries more efficient, usually other sources of data than geolocation ones are accessed from the phone, such as Inertial Measurements Units (IMUs).

An IMU [25] in the context of mobile phones is a compact electronic device that combines various sensors to measure and report information about an object's orientation, position, and motion. IMUs are commonly used in smartphones and other portable devices to enable features like screen rotation, step counting, gaming interactions, augmented reality applications, and more.

Typically, an IMU in a mobile phone consists of a combination of the following sensors:

- **Accelerometer:** Measures acceleration forces along the three axes (X, Y and Z) in terms of gravity. It helps determine the phone's orientation, tilt, and movement speed.

- **Gyroscope:** Measures the angular velocity or rate of rotation around the device's axes. It provides information about the phone's rotational movements and helps maintain accurate orientation data.
- **Magnetometer:** Detects the Earth's magnetic field and provides a sense of direction. It is used to determine the phone's heading relative to magnetic north.

By combining the data from these sensors, the IMU can calculate and track the phone's position, orientation, and movement in real-time. This information is crucial for a variety of applications, such as changing the screen orientation from portrait to landscape mode, tracking steps for fitness monitoring, enabling motion-controlled gaming, and enhancing augmented reality experiences by overlaying digital content onto the real world.

It's important to note that while IMUs provide valuable data for these applications, they can also suffer from limitations such as drift over time and inaccuracies due to sensor noise. To mitigate these issues, IMU data is often fused with other technologies like GPS to create more robust and accurate positioning and orientation information for mobile applications.

While the activity data are not strictly necessary to perform background geolocation, they can make a big difference as most of the optimizations that help preserve battery consumption are based on them. For instance, as anticipated when discussing Transistor Software one, libraries are typically implemented in a way that:

- When the device is detected to be moving, plugins will automatically start the recording of location data point and geofence events.
- When the device is detected be stationary, plugins will automatically turn off location-services to conserve energy.

3.4 Adoption of the Transistor Software Library

3.4.1 Parameters Configuration

One of the strengths of the chosen library is the fact that it is highly configurable and therefore can be easily customized considering the needs of the application that has to be implemented.

The following are the most important parameters that can be configured:

- **desiredAccuracy:** specify the desired accuracy of the geolocation system.

Value	Location Providers	Description
NAVIGATION	(iOS only) GPS + Wi-Fi + Cellular.	(iOS only) Highest power; highest accuracy.
HIGH	GPS + Wi-Fi + Cellular.	Highest power; highest accuracy.
MEDIUM	Wi-Fi + Cellular.	Medium power; medium accuracy.
LOW	Wi-Fi (low power) + Cellular.	Lower power; no GPS
VERY_LOW	Cellular only.	Lowest power; lowest accuracy.
LOWEST	(iOS only) Cellular only.	(iOS only) Lowest power; lowest accuracy.

- **distanceFilter:** the minimum number of meters a device must move horizontally before an update event is generated.
- **disableElasticity:** by default, the SDK automatically increases distanceFilter as speed increases and decreases it as speed decreases to record fewer locations and conserve energy.
- **elasticityMultiplier:** controls the scale of automatic speed-based distanceFilter elasticity.
- **useSignificantChangesOnly:** set to true to disable constant background-tracking; locations will be recorded only periodically.
- **stationaryRadius:** the minimum number of meters the device must move beyond the stationary location for aggressive background-tracking to engage.

Tuning the parameters required extensive testing: following a trial-and-error approach, I was able to find a good configuration which works well for both Android and iOS and that fits with the requirements and with the kind of application that Benetics is.


```
await BackgroundGeolocation.ready(  
  {  
    transistorAuthorizationToken: authToken,  
    desiredAccuracy: BackgroundGeolocation.DESIRED_ACCURACY_NAVIGATION,  
    distanceFilter: 10,  
    elasticityMultiplier: 1,  
    geofenceProximityRadius: 5000,  
    stationaryRadius: 25,  
    stopTimeout: 15,  
    activityType: BackgroundGeolocation.ACTIVITY_TYPE_OTHER,  
    stopAfterElapsedMinutes: 0,  
    showsBackgroundLocationIndicator: true,  
    locationUpdateInterval: 1000,  
    geofenceModeHighAccuracy: true,  
    disableStopDetection: false,  
    stopDetectionDelay: 0,  
    motionTriggerDelay: 0,  
    autoSyncThreshold: 0,  
    disableMotionActivityUpdates: false,  
    deferTime: 0,  
    preventSuspend: true,  
    disableElasticity: false,  
    disableAutoSyncOnCellular: false,  
    desiredOdometerAccuracy: 100,  
    useSignificantChangesOnly: false,  
    disableLocationAuthorizationAlert: false,  
    enableHeadless: true,  
    notificationPriority:  
      BackgroundGeolocation.NOTIFICATION_PRIORITY_DEFAULT,  
    heartbeatInterval: 60,  
  }  
)
```



```
    debug: false,
    logLevel: BackgroundGeolocation.LOG_LEVEL_VERBOSE,
    stopOnTerminate: false,
    startOnBoot: true,
    backgroundPermissionRationale: {
      title: t('Allow Benetics to access to this devices location when
        closed or not in use?'),
      {ns: 'main'},
    },
    message: t(
      "Benetics collects location data to identify start and end times of
        workday and to show other users if you're in a construction site.",
      {ns: 'main'},
    ),
    positiveAction: t('Change to always', {ns: 'main'}),
    negativeAction: t('Cancel', {ns: 'main'}),
  },
  batchSync: false,
  autoSync: true,
},
state => {
  console.debug(
    '[GEOLOCATOR STATE] BackgroundGeolocation is configured and ready! ',
    state.enabled,
  );
},
);
```

For more information, please refer to the official library documentation [3].

3.4.2 Keeping the Library in Synch with the Database

Ensuring consistency between the geofences registered in the library and the ones in the database stands paramount. While library-registered geofences are monitored on users' devices, storing them in the database is vital for retrieval during phone changes or reinstalls.

In fact, it is important to store the registered geofences in the database because the user should be able to recover them when needed at any point in time. For example, if the user changes his phone, the first time that he logs in, the library will have to fetch the relevant geofences for that user and register them again in the phone to be monitored.

The pattern that was used to implement the logic is based on the cache invalidation. In general cache invalidation is a crucial aspect of designing efficient and responsive software systems that utilize cache mechanisms. The pattern deals with the problem of keeping cached data up-to-date and consistent with the underlying data source. When the underlying data changes, cached data must be invalidated to prevent serving stale or incorrect information to users.

Thanks to this cache invalidation mechanism, it is easier to understand when a fetch from the database is needed. Moreover, this allows to keep the logic for keeping the database and the library in synch centralized. The advantage of this is that in all the other points of the codebase it is sufficient to invalidate the cache to trigger the update, and it also prevents code duplication, making it more maintainable and simpler.

The following is the implementation.

```
import {useEffect} from 'react';
import BackgroundGeolocation, {
  Geofence,
} from 'react-native-background-geolocation';
import {useQuery} from 'react-query';
import {useApi} from '../context/ApiContext';
import {useAuth} from '../context/AuthContext';
import {useProfile} from '../context/ProfileContext';
export const GEOFENCES_STALE_TIME_MS = 5 * 60 * 1000; // 5 min
const useProjectGeofences = () => {
  const {user} = useProfile();
  const api = useApi();
  const {data: userGeofences} = useQuery(
    ['users', user.id, 'project-geofences'],
    () => api.users.getProjectGeofences(user.id),
    {
      staleTime: GEOFENCES_STALE_TIME_MS,
    },
  );
};
const {addListener} = useAuth();
useEffect(() => {
  const updateGeofences = async () => {
    // After updateGeofences, the plugin must reflect exactly the state in DB
    if (userGeofences === undefined) {
      return;
    }
  }
}
```

```

// copy the lists to sets for faster lookup
const userGeofencesIds: Set<string> = new Set(
  userGeofences.map(geofence => geofence.id),
);
const pluginRegisteredGeofences: Geofence[] =
  await BackgroundGeolocation.getGeofences();

const pluginRegisteredGeofencesIds: Set<string> = new Set(
  pluginRegisteredGeofences.map(geofence => geofence.identifier),
);
const geofencesToRegister: Geofence[] = [];
for (const curGeofence of userGeofences) {
  if (!pluginRegisteredGeofencesIds.has(curGeofence.id)) {
    geofencesToRegister.push({
      identifier: curGeofence.id,
      radius: curGeofence.radius,
      latitude: curGeofence.latitude,
      longitude: curGeofence.longitude,
      notifyOnEntry: true,
      notifyOnExit: true,
      loiteringDelay: 0,
    });
  }
}
void BackgroundGeolocation.addGeofences(geofencesToRegister);
// if a geofence is registered in the plugin, but not present in database, it
// must be removed
for (const curGeofence of pluginRegisteredGeofences) {
  if (!userGeofencesIds.has(curGeofence.identifier)) {
    void BackgroundGeolocation.removeGeofence(curGeofence.identifier);
  }
}
};
void updateGeofences();
}, [userGeofences]);
useEffect(() => {
  const unsubscribe = addListener({
    onSignOut: async () => {
      // clean the state of Background Geolocator plugin
      await BackgroundGeolocation.removeGeofences(
        () => console.debug('[GEOFENCES] All geofences have been removed'),
        () => console.error('[GEOFENCES] Error while remove all geofences'),
      );
      void BackgroundGeolocation.removeAllListeners(
        () => console.debug('[LISTENERS] All listeners have been removed'),
        () => console.error('[LISTENERS] Error while remove all listeners'),
      );
    };
  });
});

```

```
void BackgroundGeolocation.stop(
  () => console.debug('[PLUGIN STATE] The plugin successfully stopped'),
  err =>
    console.error('[PLUGIN STATE] Error while stopping plugin: ', err),
);
},
});
return unsubscribe;
}, [addListener]);
};
export default useProjectGeofences;
```

The following notes highlight the remarkable parts of the code snippet to better understand how it works:

- The “userGeofences” variable is initialized using the “useQuery” hook; this hook allows to specify a cache identifier. In general, the cache is used to avoid performing requests to the database every time that geofences are needed if the cacheTime (i.e., how long fetched data is retained in the cache) and staleTime (i.e., how long a piece of data is considered “fresh” or “valid” after it has been fetched from the API) have not expired yet. However, the data in the cache can also be arbitrarily invalidated programmatically: in this way it is possible to force a new request to the database.
- The “useEffect” hook has a dependency on “userGeofences” and therefore it is triggered every time that “userGeofences” changes; clearly this also includes the moment in which the cache is invalidated.
- Inside the first “useEffect”, the “updateGeofences” method is used to compare the geofences registered in the plugin with the data coming from the database and to eventually reflect the state of the database in the library.
- The second “useEffect” hook handles user sign-out. It removes geofences, listeners, and stops the “BackgroundGeolocation” plugin when the user signs out. It sets up an event listener using the “addListener” function from the “useAuth” hook and returns an unsubscribe function that will clean up the listener when the component is unmounted.
- The “useProjectGeofences” is a custom hook which is called from a component named “PostAuthNavigator” which defines the navigation within the app; it encapsulates geofence management logic for better organization and separation of concerns within the app.

One limitation of the design that has been presented is that changes to geofences determined by other users are not reflected immediately in the library. Examples of such changes include:

- An admin of the project modifies the address of the project.
- An admin of the project deletes the project.
- A user is added to a project.
- A user is removed from a project.

In all these cases, it may take up to the “cacheTime” defined for “userGeofences”, in the worst-case scenario, to have the new state of geofences reflected from the database to all the project participants’ phones plugins. For example, it may happen that after a project is deleted, for some minutes geofence events related to that project’s geofences are continuing to be received. Or similarly, if a project address changes, it will take this small delay period to switch from the monitoring of the old address to the new one.

Considering these limitations in the broader context, however, their impact is relatively inconsequential. Several reasons support this stance: changes to projects are expected to be infrequent based on how the app is designed, the number of geofence events expected during a day is quite limited and geofence events have an advisory nature and are therefore meant to be used as suggestions.

This approach adheres to a “self-correction” philosophy: it ensures accurate states are eventually attained within, in the worst-case scenario, a delay equal to the cacheTime, configurable as needed.

Another approach that I considered to accomplish the task of keeping geofences synchronized between the database and the users’ phones is to implement web sockets. In this way, every time that a change on a project affecting a geofence is performed, all the other project participants “clients” will be notified and will have changes reflected almost immediately.

It is evident that this is a tradeoff between accuracy and resources utilization; having data up to date all the time requires handling more complexity in the code base and it’s clearly more expensive in terms of number of requests.

Discussing this approach with the team, it turned out that this implementation idea is probably too complex for the time being and that the small advantage that can be gained is not so relevant to the app’s scope.

3.5 Testing of the library

3.5.1 Testing on Simulators

To thoroughly test and become acquainted with the library's functionality, it becomes imperative to establish a method for simulating locations on both mobile phone simulators.

While testing on actual devices remains a pivotal phase, it's important to acknowledge that such testing demands significantly more time and effort. Consequently, a logical approach involves conducting simulator-based testing prior to transitioning to real device testing.

3.5.1.1 Xcode

Within the Xcode [13] development environment, the process of generating debugging files to simulate locations proves highly efficient. These files, identified by the ".gpx" extension, essentially function as mappings of GPS coordinates across temporal intervals. By associating a timestamp with each GPS coordinate, the possibility of emulating activities like walking, biking, or highway driving becomes feasible.

Presented below is an illustrative depiction of the requisite format for these debugging files; for further information, please refer to the linked guide [14, 15].

```

<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
  <!--
    Provide one or more waypoints containing a latitude/longitude pair. If you
    provide one waypoint, Xcode will simulate that specific location. If you
    provide multiple waypoints, Xcode will simulate a route visiting each
    waypoint.
  -->
  <wpt lat="47.38591742273731" lon="8.521929159934638">
    <name>First Location Point</name>
    <!--
      Optionally provide a time element for each waypoint. Xcode will
      interpolate movement a rate of speed based on the time elapsed between
      each waypoint. If you do not provide a time element, then Xcode will
      use a fixed rate of speed. Waypoints must be sorted by time in
      ascending order.
    -->
    <time>2023-01-01T09:00:00Z</time>
  </wpt>
  <wpt lat="47.37463303976847" lon="8.54866151927547">
    <name>Second Location Point</name>
    <time>2023-01-01T09:00:30Z</time>
  </wpt>
  <wpt lat="47.39463303976847" lon="8.51866151927547">
    <name>Third Location Point</name>
    <time>2023-01-01T09:00:50Z</time>
  </wpt>
</gpx>

```

As can be seen, it is also possible to specify the arrival times for each of the designated location points. This facet facilitates the synchronization of feature testing with dynamic location changes directly within the simulator. For instance, during the evaluation of the "Remote Personnel Presence Monitoring" feature that will be later explained in a following chapter, the utility of this Xcode feature proved remarkably advantageous.

3.5.1.2 *Android Studio*

The same thing can be done for Android as well, using Android Studio [16] emulators. By accessing the location settings, it is possible to set the position or simulate paths by selecting the points directly from Google Maps; alternatively, it is possible to import and use ".gpx" files.

For more information, please visit the following resource [17].

4. Tech Stack

This chapter presents the technologies and frameworks employed in this project.

4.1 Backend

The backend was implemented using the followings:

- **Amazon Web Services (AWS)** [8]: The project adopts a serverless paradigm, harnessing the cloud computing services offered by Amazon to execute operations.
- **Python** [27]: The chosen programming language for the implementation of all business logic within the backend.
- **FastAPI** [28]: A modern, high-performance web framework for building Application Programming Interfaces (APIs) with Python; it is designed to be fast, easy to use, and to provide automatic validation and documentation for the APIs.
- **Pydantic** [32]: A vital component for data validation and settings management.
- **Docker** [33]: To package up and deploy the backend. Docker provides containers - “packages” of binaries and their dependencies - and interacts with Amazon’s Elastic Container Registry (ECR) to upload and distribute Docker images.
- **Serverless Framework** [18]: Used to simplify and automate the deployment, scaling, and management of serverless applications, abstracting away infrastructure complexities and enabling efficient development workflows.

4.1.1 AWS Dynamo DB

The database that is used within Benetics is DynamoDB.

Amazon DynamoDB is a fully managed NoSQL database service provided by AWS. It is designed to provide high-performance and scalable storage for applications that

require low-latency and seamless scaling. DynamoDB is built to handle various types of data, ranging from key-value and document data to time-series data.

Key features of DynamoDB include:

- **Scalability:** DynamoDB can automatically scale its capacity up or down based on the workload, ensuring consistent performance as the data and traffic grow.
- **High Availability:** It offers built-in data replication and multi-region support to ensure high availability and durability of data even in the face of hardware failures or region outages.
- **Low Latency:** DynamoDB provides single-digit millisecond response times for read and write operations, making it suitable for applications that require real-time data access.
- **Flexible Data Model:** It supports various data models, including key-value and document structures, allowing developers to choose the data format that best suits their application's needs.
- **Managed Service:** DynamoDB is fully managed by AWS, which means AWS takes care of administrative tasks such as hardware provisioning, software patching, and data backup, allowing developers to focus on building their applications.
- **Security and Access Control:** It offers robust security features, including data encryption at rest and in transit, fine-grained access control, and integration with AWS Identity and Access Management (IAM).
- **Automatic Scaling:** DynamoDB can automatically scale its resources to accommodate changes in demand without requiring manual intervention. This helps maintain optimal performance without over-provisioning resources.
- **Pay-as-You-Go Pricing:** DynamoDB follows a pay-as-you-go pricing model, where you pay just for the amount of data you store and the throughput you provide, making it cost-effective for a wide range of applications.

DynamoDB is commonly used for various use cases such as web and mobile applications, gaming, IoT applications, and more, where speed, scalability, and flexibility are crucial. Its managed nature and integration with other AWS services make it a popular choice for developers looking to build highly available and performant applications in the cloud.

Designing how to query data in DynamoDB is vital. Before building a non-relational database, it is important to think about the most frequent and resource-intensive queries that must be performed; this helps to shape data models and access patterns effectively. By identifying access patterns early, performances can be optimized, and

costs reduced. DynamoDB's NoSQL structure demands thoughtful schema design to match query needs. This optimizes data retrieval and minimizes costly operations, resulting in better latency and reduced expenses.

Creating an appropriate data model ensures even data distribution, preventing bottlenecks and enabling scalable growth. It also aids in selecting the right consistency levels for queries. In summary, pre-defining queries and structuring data accordingly enhances performance, scalability, cost efficiency, and maintenance of DynamoDB database.

As DynamoDB has a vast number of features that go out of the scope of this thesis, in this section I will just focus on the most important ones that were relevant to my project. For more information, please refer to the official documentation [31].

4.1.1.1 *Global Secondary Index*

Deciding which data paths should have priority over others may be tricky before the app has active users; moreover, often multiple query paths want to be optimized in order to consistently have great performance. When dealing with multiple queries that need optimization simultaneously in DynamoDB, **Global Secondary Indexes (GSIs)** offer a powerful solution.

A GSI is an index that can be created on a DynamoDB table, which provides an alternative way to query and access data in the table. While the primary key of a DynamoDB table uniquely identifies each item in the table, a GSI allows to define a different set of attributes as the index key.

This makes it possible to perform queries based on those attributes without the need to use the primary key.

Some key points about GSIs in DynamoDB:

- **Alternative Query Paths:** GSIs allow to perform efficient queries on attributes other than the table's primary key, which can help optimize various types of queries for different access patterns.
- **Distributed and Automatically Maintained:** GSIs are distributed across multiple partitions for improved performance, and DynamoDB automatically maintains them as data in the table is added, modified, or deleted.
- **Projection:** When creating a GSI, it is possible to specify which attributes from the table should be projected and included into the index.; this helps to reduce read costs and improve query performance by only retrieving the necessary data from the index.
- **Eventual Consistency:** Queries on GSIs might eventually provide consistent results, meaning that the index might not reflect the latest changes

immediately; however, it is possible to configure queries to use "strongly consistent" reads if the most up-to-date data are needed.

- **Write Capacity:** GSIs consume write capacity units for updates to the GSI itself as well as the underlying table.; this is an important consideration for provisioning throughput.

GSIs are useful when there are various query patterns beyond just retrieving items by their primary key, and they can help optimize the performance of those queries.

For more information, please refer to the official documentation. [10]

4.1.1.2 *ULIDs*

When designing the Primary Key of a data type, composed by Partition Key and Sort Key, it is important to be aware of which kind of identifiers may be the best fit for our use case; this section presents the type of identifiers which was particularly helpful in the scope of the project.

ULID stands for "Universally Unique Lexicographically Sortable Identifier" [11]. It is a type of identifier or unique key that is commonly used in software development to generate unique identifiers in a distributed system. ULIDs are designed to be both unique and sortable, making them suitable for various applications where identifiers need to be generated across multiple systems or instances and then sorted.

ULIDs are often used in scenarios where traditional auto-incrementing integers or UUIDs (i.e., Universally Unique Identifiers) might not be ideal. ULIDs combine a 48-bit timestamp with an 80-bit random value, resulting in a 128-bit identifier. The timestamp ensures that identifiers created at different times are inherently sortable, while the random component ensures uniqueness even when generated in parallel across multiple systems.

This makes ULIDs particularly useful in situations like database sharding, event sourcing, distributed systems, and scenarios where unique identifiers need to be generated in a way that avoids collisions and maintains some level of order.

4.1.1.3 *Lambdas*

A service of AWS which is worth mastering is the one of Lambda functions, serverless computational services that allow to run code without provisioning or managing servers.

In the context of DynamoDB, Lambdas are functions that can be associated with certain actions or events within a DynamoDB table. When used with DynamoDB, Lambda functions can be triggered in response to various DynamoDB events, such as inserts, updates, or deletions in a table.

Here is how the process typically works:

- **Event Trigger:** Lambda functions can be configured to be triggered by specific DynamoDB events. These could be events like creating, updating, or deleting an item in a DynamoDB table.
- **Lambda Function:** The code that contains the logic of the Lambda function can be in various programming languages, depending on what is supported by Lambda. The Lambda function can process the event data, perform additional logic, interact with other AWS services, and more.
- **Event Data:** When the configured event occurs in the DynamoDB table (i.e., an item is inserted), DynamoDB sends information about the event to the associated Lambda function.
- **Lambda Execution:** Lambda functions are invoked and executed with the event data; the code can process this data and perform actions based on the event that occurred in the DynamoDB table.
- **Response:** Lambda functions can optionally generate a response that might be sent back to the caller, depending on the use case.

Using Lambda functions with DynamoDB can help to implement various automation and business logic scenarios. For instance, Lambdas can be used to:

- Automatically update secondary indexes when data is inserted or updated.
- Perform data validation before allowing an insert or update.
- Notify other services or systems when specific data changes occur.
- Aggregate data and update summary records.
- Implement custom access controls and data transformations.

Lambda functions are a powerful way to extend DynamoDB's functionality: they allow to decouple application logic and run code in response to specific events, all without managing the underlying infrastructure.

4.1.2 Serverless Framework

This section quickly presents the Serverless Framework, as it will be referred to later in another chapter. It is a framework that simplifies the deployment and management of serverless applications, particularly popular for deploying serverless applications on AWS. Serverless computing is a cloud computing model where applications can be built and run without needing to manage the underlying server infrastructure. Instead, the cloud provider takes care of automatically scaling and managing the resources needed to run the application.

Key features of the Serverless Framework for AWS include:

- **Simplified Deployment:** The framework abstracts away many of the complex configuration tasks required for deploying serverless applications; it provides a more intuitive and straightforward way to define the application's resources and functions.
- **Infrastructure as Code:** Serverless allows to define the application's infrastructure using code, usually written in YAML or JSON; this makes it easier to version control, share, and manage the application's configuration.
- **Function Management:** Serverless provides tools to create, deploy, update, and manage serverless functions and their associated resources, such as databases, APIs, and event triggers.
- **Automatic Scaling:** Serverless applications automatically scale based on demand; when an event triggers a function the cloud provider provisions the necessary resources to handle the request and then scales down afterward.
- **Event-Driven Architecture:** Serverless encourages an event-driven architecture, where functions are triggered by various events, such as HTTP requests, database changes, file uploads, and more.
- **Offline Development:** The framework often provides emulation or local execution capabilities, allowing to develop and test functions locally before deploying them to the cloud.
- **Plugin System:** The Serverless Framework supports a plugin system that allows to extend its functionality or integrate with additional tools.

For AWS, the `serverless.yml` file is used to define serverless application's resources, functions, events, and various settings. It's a YAML-formatted file that provides a structured way to specify how the application should be deployed and managed by the Serverless Framework. Here's a basic overview of the content of a typical “`serverless.yml`” file for an AWS-based application:

- **Service Information:** Defines basic information about services, such as its name, provider (i.e., AWS), and runtime environment (i.e., Node.js, Python).
- **Provider Configuration:** configure AWS-specific settings, such as the region, runtime, IAM roles, and more.
- **Functions:** Define the individual serverless functions of the application. Each function has a name, handler and it is also possible to specify environment variables, memory allocation, and timeout settings.
- **Events:** Events trigger functions to run. They can be various AWS services' events, HTTP endpoints, jobs, and more. For example, it is possible to set up a function to be triggered by an HTTP request or when an S3 bucket receives a new file.
- **Resources:** Define additional AWS resources (like DynamoDB tables, S3 buckets, etc.) that the application might need.

- **Custom Variables:** Used throughout the configuration to make it more dynamic.

For more information, please refer to the official documentation [18].

4.2 Frontend

The frontend component was implemented using React Native, which is an open-source framework developed by Facebook in 2015. This framework empowers the implementation of mobile applications using JavaScript and React, a widely adopted JavaScript library specifically crafted for designing user interfaces. Furthermore, Typescript [44], a superset of JavaScript that adds static typing capabilities, was utilized in conjunction with React Native to enhance the development process.

The decision to employ React Native, along with the integration of Typescript, rests upon their combined inherent cross-platform capabilities. This rare attribute facilitates the development of mobile applications for various platforms, encompassing both iOS and Android, all while maintaining a unified codebase. The use of Typescript further contributes to the project's robustness by providing type checking and improved code quality, ultimately aiding in the creation of a more reliable and efficient mobile application.

5. Implementation

This chapter delves into the implementation of the two main features I have worked on during the internship, namely the Remote Personnel Presence Monitoring and the Time Card Manager.

5.1 Feature: Remote Personnel Presence Monitoring

This section details the proposed functionality designed to enhance real-time visibility of personnel at designated construction sites. The primary aim of this feature is to provide users with immediate awareness of individuals present at specific construction sites. The key beneficiaries are expected to be construction site managers, who can utilize this capability to achieve the following:

- **Effective Communication:** By promptly identifying relevant personnel presence, this feature enables construction site managers to quickly establish contact with individuals needing to be present at the site.
- **Clarity in Allocation and Ambiguity Mitigation:** In situations where workers are concurrently engaged across multiple construction sites, this feature eliminates potential confusion. It ensures that construction site managers can accurately determine worker availability and assign tasks based on predefined priorities.

Similarly, the feature will also enhance collaboration and efficiency among workers, especially in big construction site where it may be hard of keeping track of who is present and who is not.

5.1.1 Requirements

The feature must satisfy the following requirements:

- Create one or more geofences when a project is initiated, using the user-provided address.
- Dynamically determine in real-time which project participants are physically present in one of the geofences associated with it.

- Display this information in a concise and intuitive manner to all project participants through the app.

5.1.2 Design

Within the Benetics app, each project creation requires an associated address. This is especially crucial for geolocation-dependent features, including the one described in this section.

During the design phase, the most suitable entity for geofence association was discussed, with Projects and Tasks being the main candidates.

The key considerations include:

- Projects generally have much longer lifespans, that could be months or even years, while tasks usually last only a few days at most, although this clearly depends on user preferences.
- Projects typically involve more participants, whereas tasks are intended for smaller work units assigned to a limited set of individuals. Again, this can be decided arbitrarily by the user, so it may be false if users do not follow the guidelines proposed by the app.

Based on these considerations, the initial decision was to associate geofences exclusively with Projects. However, it's worth noting that thanks to the flexibility of DynamoDB, future associations with new entities like tasks could also be possible.

Another important consideration, which is probably going to be discussed again in the later months, regards who should have visibility on this kind of data. The considered possibilities were:

- Every participant of the project: this is clearly the most open approach and also the one that was initially adopted. In this case, the most value can be extracted from the feature, but more are also the privacy concerns.
- Only managers: in this case, privacy problems will be partially reduced, even if employees would probably not be so enthusiastic about this asymmetry in visibility.
- Only non-managers: similarly, also in this situation the privacy concerns will be less relevant and most of the employees will feel more comfortable in knowing that their employers cannot monitor them in real-time; however, it is worth noting that also most of the value of the feature itself would be lost.

5.1.3 User Interface

In collaboration with the UI/UX lead, it was decided to integrate this information into the Project Overview screen, as depicted in the figure below.

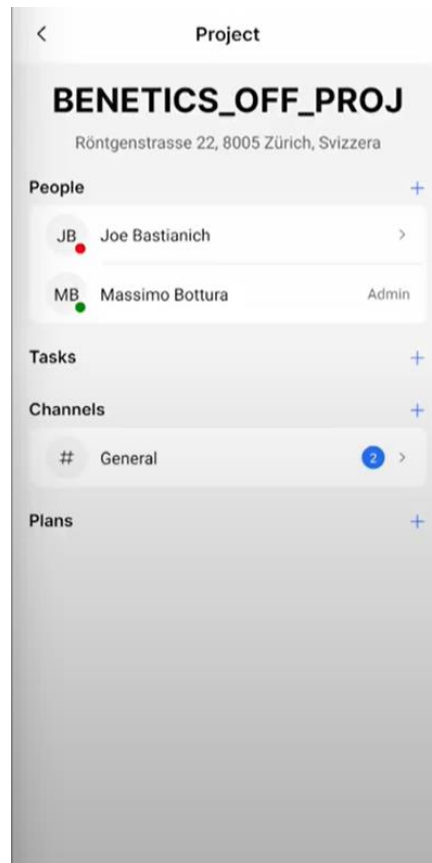


Figure 5.1: Remote Personnel Presence Monitoring feature.

The project overview screen already included a people section needed for managing administrative permission, and for adding and removing participants to the project. Therefore, the most intuitive extension to the section was to introduce a familiar notation on top of the present design. The designed notation involves a classical small colored dot near the person's avatar - profile picture or initials if no picture is set: specifically, a green dot signifies onsite presence, while a red dot indicates absence.

Additionally, when profile pages for individuals will be introduced within Benetics, this information could be displayed there as well, along with the corresponding project indication.

5.1.4 Testing

Internal testing of the feature was conducted with team members. Numerous real-world experiments were carried out to track and verify our presence in the office. The primary metrics considered were:

- **Accuracy:** how precise the collected data are. This is clearly extremely important to create a reliable system, which is one of the most important requirements to respect to gain the trust of customers and users.
- **Battery life:** how relevant the impact of the Background Geolocator is on the phone battery. This is also a fundamental metric for ensuring a positive user experience.

Balancing these two metrics requires careful trade-offs to achieve a favorable outcome. It's important to note that the testing conditions were relatively ideal, and further testing under less favorable conditions should be conducted. This includes scenarios involving cheaper smartphones, monitoring geofences in areas with low connectivity, and other potential obstacles to the feature's functionality.

5.2 Feature: Time Cards Manager

The aim of this feature is to precisely track the time spent by each worker on the construction site. The objective is not only to implement fundamental functionalities but also to leverage data collected by the background geofencing system introduced earlier in this thesis.

5.2.1 Requirements

Normal users should be capable of performing the following actions:

- Create, Read, Update and Delete (CRUD) time cards.
- Receive suggestions and support based on geofence events related to the day and project associated with the timecard. This feature is intended for users who have enabled background geolocation features.
- Have a clear overview of how many hours they have worked.
- Have a way to easily identify and correct errors.

Furthermore, project administrators should have the ability to:

- View time cards of all project participants in the project overview.
- Filter time cards based on attributes such as profession and time period.

- Export data in a textual format for use in other applications and services; note that Benetics plans to integrate this feature into their web app in the future, though support for this is not currently available.

5.2.2 Design

Initially, this feature was intended to eliminate the need for human intervention in tracking and reporting work hours. However, due to certain reasons and constraints, the focus shifted to developing a system to support time card management. These include the followings:

- Since background geolocation will be introduced in the app as an optional feature, it is important that users that will not grant permissions to the geofencing functionality still have a backup system for managing their time cards.
- The product team preferred a hybrid approach, avoiding sole reliance on geofencing data as the source of truth for crucial information. This approach allows refinement and comprehensive testing of data accuracy provided by the background geolocation library on a larger scale. Over time, automation will play a greater role, reducing the need for manual effort.
- Data from geofencing is not always complete or accurate. Consider for instance scenarios in which users experience problems like signal loss, connection issues, or low battery. Relying solely on such data would be unreliable and misleading.

5.2.3 User Interface

The Time Cards Manager is accessible in two main parts of the app:

- For all users, in a dedicated section accessible through the settings menu.
- For project admins, also on the Project Overview screen.

Upon accessing the Time Cards screen, standard users are presented with the following:

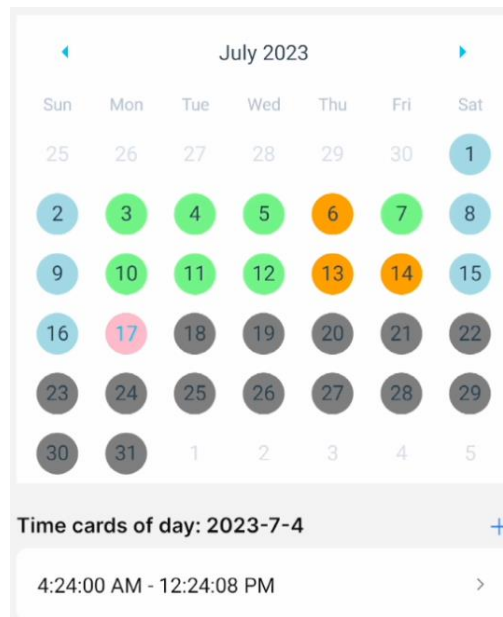


Figure 5.2: Time Cards overview calendar.

The notation and meaning of the different colors follow here:

- Green: past or present weekday for which it exists at least one time card.
- Orange: past weekday with no associated time cards.
- Light blue: weekend days; the assumption is that most of the workers are not active on weekends.
- Purple: simply highlights differently the current day, for better usability.
- Grey: future days; for these days it is not possible to create any time cards. When clicking on one of these, an information message is displayed to the user through a temporary Toast.

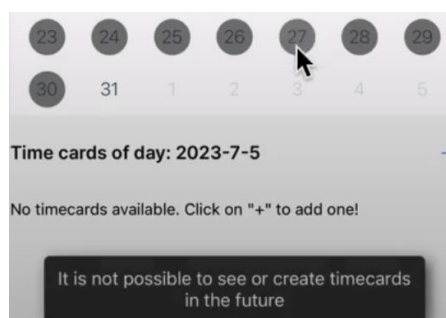


Figure 5.3: Error message time card in the future.

With these color notations the user can very easily check if for any weekday in the past he has not filled in any time card.

The user also has the possibility to filter time cards based on the project.

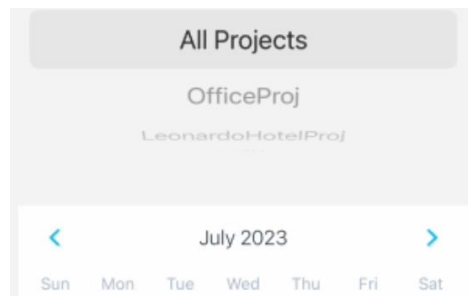


Figure 5.4: Filter time cards on project.

When clicking on the “+” button on the right, the user accesses the following screen where it is possible to create a new Time Card entry.

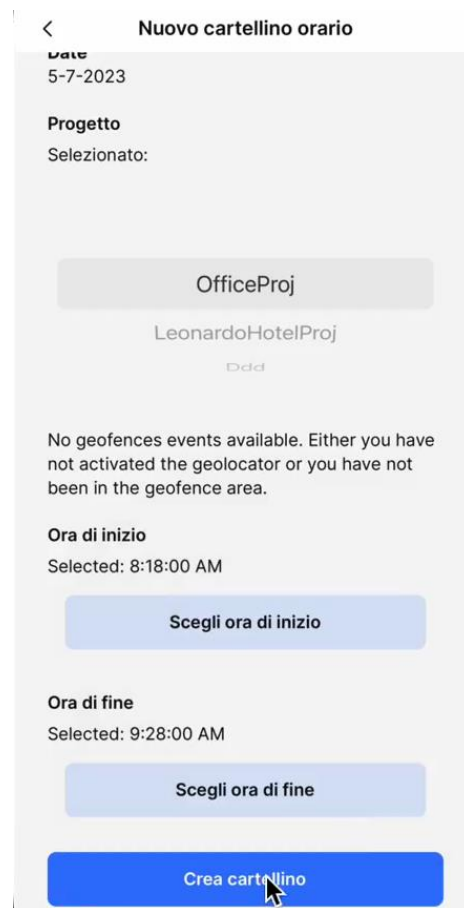


Figure 5.5: Creation of Time Card.

As can be seen, once the user has selected one of the project he is a participant of, a message related to the geofence events is displayed to the user. In this case, there were no geofence events available.

However, if the user has turned on the background geolocation and some geofence events have already been registered for that day and project, he will be displayed with the following for example:

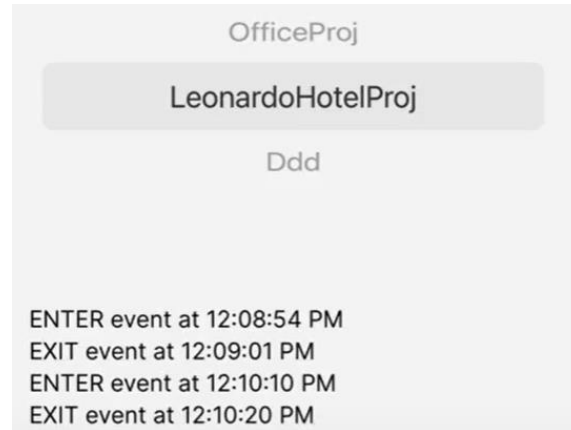
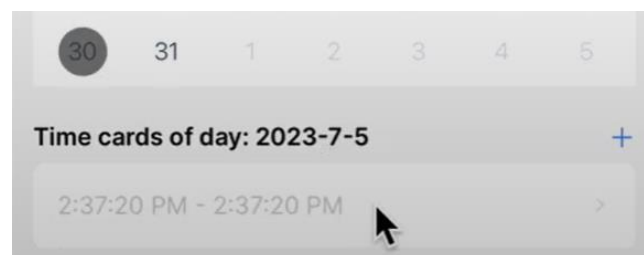


Figure 5.6: Geofencing suggestions for time card creation.

In this case, for instance, two enter events and two exit events were registered for that day; notice that the times do not look so meaningful in this case as these data were collected using the simulators.

This is meant to be a suggestion and support to help the user fill in the start and end times.

Moreover, in order to update a time card, the user should only click on one of the entries



in the list to access the following screen.

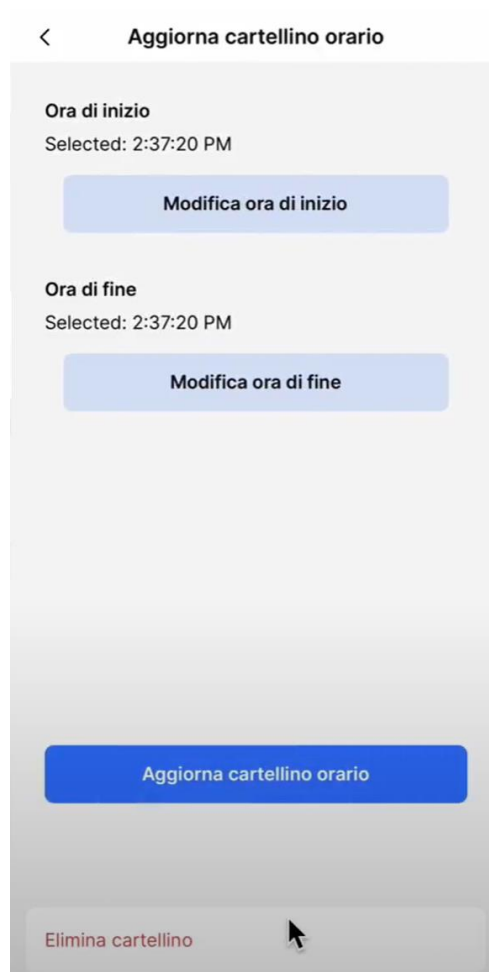


Figure 5.7: Update of Time Card.

In that screen, the user has the possibility to update the start date and the end date, or also to delete the time card.

Another important part, as can be seen in the figure below, is the overview graph that shows the number of hours that were worked during the past days of the current month; the graph is displayed directly below the calendar when scrolling down. Even if this information may somehow look redundant, it is still very helpful for the user because it better highlights possible errors in the data thanks to the compact format. For example, a user that is always working between 8 and 10 hours per day, will be able to easily notice a day for which he has only a time card for 3 hours and correct it if necessary.

To summarize, while the calendar is more intuitive for the users to create, read, update, and delete time cards, the overview graph offers more support when having to check the correctness of the data.



Figure 5.8: Time Cards overview graph.

Moreover, for a better user experience, graph highlights in green the last day that was selected on the calendar and the graph also has an automatic re-centering functionality to make sure that the selected day is visible. Notice in fact that, in order to accommodate all the days of the current month, the graph must be horizontally scrollable, and this means that only part of the days can be displayed at a time.

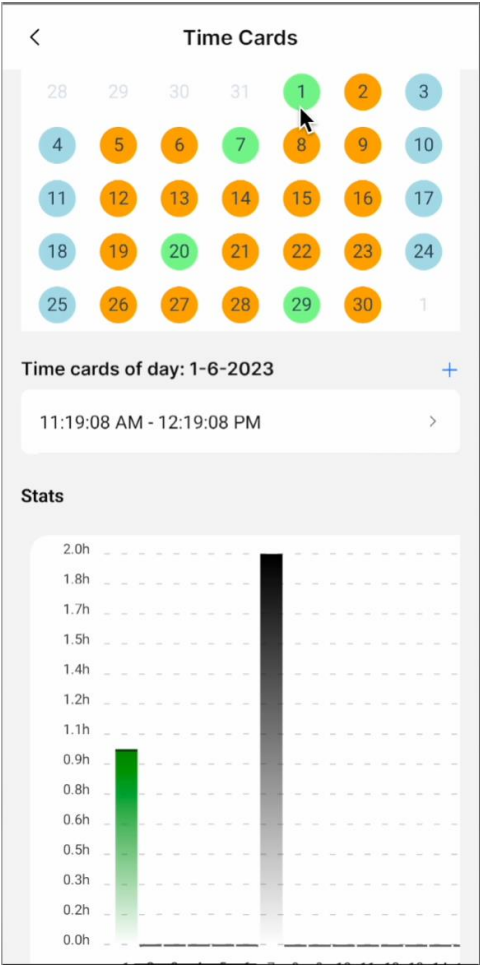


Figure 5.9: Interactive connection between calendar and graph.

The Time Cards Manager is not only present on the dedicated screens as shown until this point. In fact, it is important for the managers and admins of the project to have an overview of the work hours spent by the people working on that project.

Therefore, I have decided to introduce on the Project Overview screen a new section dedicated to Time Cards.

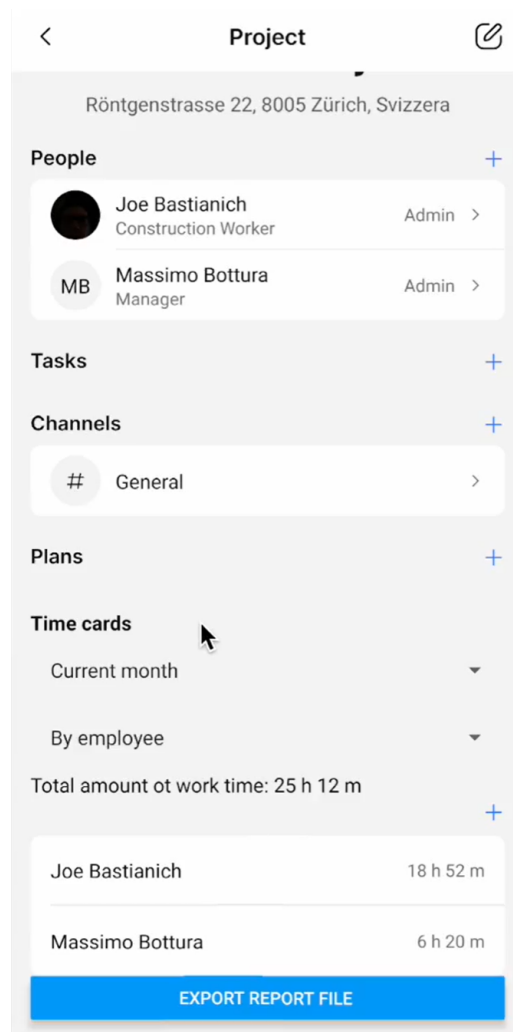


Figure 5.10: Time Cards section for managers on Project screen.

As it can be seen in the figure below, the manager can filter based on:

- the **period**: for example, “Current month”, “Previous Month”, “Since project start”. It would be straightforward to introduce the possibility to filter on a custom period (i.e., start and end dates).

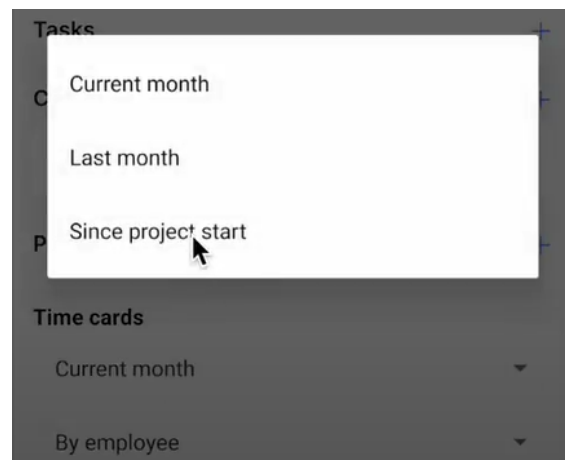


Figure 5.11: Period filtering for Time Cards.

- the **category**: for example, “By employee” or “By profession” in some cases, the list of people working for a certain project can be quite long and overwhelming and often it is instead more important to group by profession.

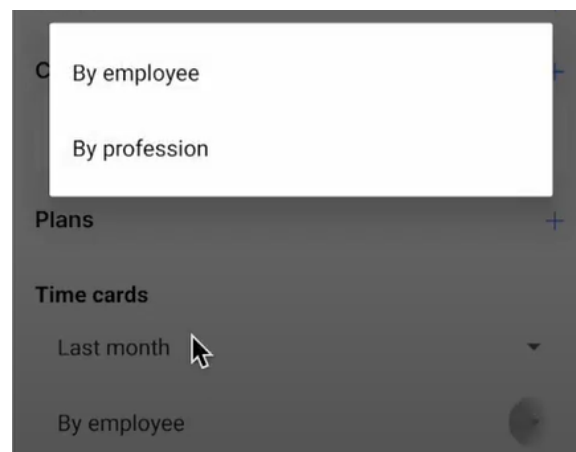


Figure 5.12: Category filtering for Time Cards.

As can be seen, the manager also has the possibility to export the data in a txt/csv format. Later, the idea of Benetics is to provide a complete management tool on the web app that is being developed, but for the moment, as it is not ready yet, exporting these data can be quite useful: for example, the manager may want to visualize them in a more detailed way, build graph and stats using other software or integrate this information with their existing systems.

The following is the code snippet that is used to generate the data.

```
def generate_csv_data(timecards: List[models.Timecard]) -> StringIO:
    csv_data = StringIO()
    fieldnames = [
        "Project ID",
        "User",
        "Started At",
        "Ended At",
    ]
    csvwriter = csv.DictWriter(csv_data, fieldnames=fieldnames)
    csvwriter.writeheader()
    for timecard in timecards:
        csvwriter.writerow(
            {
                "Project ID": project_item.Name,
                "User": name.full_display_name(
                    user_dict[timecard.user_id].FirstName,
                    user_dict[timecard.user_id].LastName,
                ),
                "Started At": timecard.started_at.isoformat(),
                "Ended At": timecard.ended_at.isoformat(),
            }
        )
    csv_data.seek(0)
    return csv_data
```

Following, an example of how exported data are formatted.

```
Project ID,User ID,Started At,Ended At
MyProj,Bastianich
Joe,2023-05-31T09:42:58.736000+00:00,2023-05-31T1
0:42:58.736000+00:00
MyProj,Bastianich
Joe,2023-06-01T09:19:08.916000+00:00,2023-06-01T10
:19:08.916000+00:00
MyProj,Bastianich
Joe,2023-06-07T07:54:42.083000+00:00,2023-06-07T0
9:54:42.083000+00:00
MyProj,Bastianich
Joe,2023-06-20T09:34:14.188000+00:00,2023-06-20T1
0:34:14.188000+00:00
MyProj,Bastianich
Joe,2023-06-29T08:54:15.822000+00:00,2023-06-29T0
9:54:15.822000+00:00
MyProj,Bastianich
Joe,2023-06-30T08:53:44.647000+00:00,2023-06-30T0
9:53:44.647000+00:00
MyProj,Bastianich
Joe,2023-07-04T10:35:52.838000+00:00,2023-07-04T2
0:35:52.838000+00:00
MyProj,Bastianich
Joe,2023-07-06T14:55:31.812000+00:00,2023-07-06T16
:55:31.812000+00:00
MyProj,Bastianich
Joe,2023-07-14T07:41:02.236000+00:00,2023-07-14T09
:41:02.236000+00:00
```

Figure 5.13: Time Cards exported data.

6. Data Models

This chapter focuses on the data model design which was adopted to implement the presented features.

6.1 Queries

As already mentioned before, when using non-relational databases like DynamoDB from AWS, it is extremely important to think about the ways data will have to be queried in order to have a great efficiency and to keep limited the cost to pay to the cloud service provider.

The following are the most important and frequent queries:

- Fetch all the users that are onsite for a certain project.
Notice that a user is considered onsite if he is present in any of the geofences associated to the project.
- Fetch all the timecards for a certain user and period, typically month-wise.
- Fetch all the timecards for all the participant of a certain project filtered on time period.
- Fetch all the geofence events for a certain user and day.

6.2 Entities

The following diagram shows the relationships among the main entities of this thesis project.

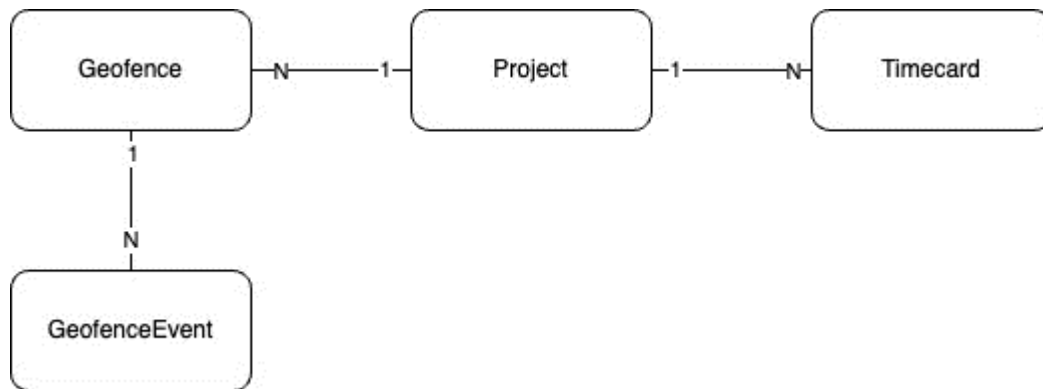


Figure 6.1: Entities relationships.

The following notes highlights relevant points regarding the diagram above:

- Each Project can be associated with multiple Geofences.
 - As already mentioned, this is to have more flexibility; since geofences offered by Transistor Software library can only be circular, sometimes it may be necessary to model areas using more of them resulting in different shapes.
- For a certain User, there can exist multiple Timecards associated with the same day. This is because of the following reasons:
 - Timecard represents only the time in which the worker was active.
 - This relationship is more flexible and can be useful to model worker breaks. For example, a User may have a Timecard for the morning from 08:00 to 10:30 and another Timecard for the afternoon from 15:30 to 18:45. This is particularly useful for those workers who do not have a fixed time schedule but that are working on-call instead.
 - A User may work on different Projects during the same day.

6.2.1 Geofence

A Geofence represents a circular area centered at a specified latitude and longitude with a certain radius.

The attributes of the Geofence data model are the followings:


```

class ProjectGeofence(core.Item):
    '''A geofence event.'''
    Id: str
    '''The ID of the geofence'''
    ProjectId: str
    '''Project the geofence refers to'''
    Latitude: str
    '''Latitude of the geofence'''
    Longitude: str
    '''Longitude of the geofence'''
    Radius: int = 200
    '''Radius of the circular geofence in meters
    200 is the minimum reliable value according to the
    Library Docs Link: https://transistorsoft.github.io/react-native-back
    ground-geolocation/interfaces/geofence.html
    #radius::~:text=exits%20this%20geofence.-,radius,-radius%3A%20number'''
    Type: Literal['PROJECT_GEOFENCE'] = core.EntityType.PROJECT_GEOFENCE.value

```

The primary key of the Geofence data type is the following.

```

@staticmethod
def key(project_id: str, project_geofence_id: str) -> dict[str, str]:
    '''The primary key of the item.'''
    return {
        'PK': f'{core.EntityPrefix.PROJECT}{project_id}',
        'SK': f'{core.EntityPrefix.GEOFENCE}{project_geofence_id}',
    }

```

This key allows to query efficiently for all the geofences associated with a certain project. It is therefore very important because it is triggered every time that we want to retrieve the users that are onsite. This query is expected to be used very often as it is triggered every time that the main Project Overview page is accessed, and this explains the choice of having the “project_id” as Partititon Key.

An important remark is that in Benetics code base, as can be seen, all the entities inherit from “core.Item”; this parent class enforces several design principles; for example, it adds PK and SK attributes and methods for reading and writing data and it enforces the presence of “Type” attribute, which is safe to have as all entries are stored in a unique table.

6.2.2 GeofenceEvent

A GeofenceEvent represents either an entering or exiting event from a certain Geofence. The attributes of the GeofenceEvent data model are the followings:

```

class GeofenceEventType(StrEnum):
    ENTER = 'ENTER'
    EXIT = 'EXIT'

class GeofenceEvent(core.Item):
    '''A geofence event.'''
    Id: str
    '''The ID of the geofence event
    We use ULID to sort them and perform range query'''
    UserId: str
    '''User the geofence refers to'''
    GeofenceId: str
    '''Geofence the event refers to'''
    GeofenceEventType: GeofenceEventType
    '''Specifies if the event was ENTER or EXIT'''
    CreatedAt: str
    '''Creation time, in UTC and using ISO 8601.'''
    GSI1PK: str
    '''Lets us easily look up all geofence events bound to a specific user.'''
    GSI1SK: str
    '''Lets us easily look up all geofence events bound to a specific user.'''
    Type: Literal['GEOFENCE_EVENT'] = core.EntityType.GEOFENCE_EVENT.value

```

The primary key of the Geofence data model is the following.

```

@staticmethod
def key(geofence_id: str, geofence_event_id: str) -> dict[str, str]:
    '''The primary key of the item.'''
    return {
        'PK': f'{core.EntityPrefix.GEOFENCE}{geofence_id}',
        'SK': f'{core.EntityPrefix.GEOFENCE_EVENT}{geofence_event_id}',
    }

```

This key allows us to fetch the last GeofenceEvents for each participant of the Project in order to infer if he is onsite or not. Simply, the last GeofenceEvent is considered for performing this classification:

- if the last GeofenceEvent is a “ENTER” event, the user is considered onsite.
- if the last GeofenceEvent is a “EXIT” event, the user is considered as not onsite.

The GeofenceEvent data type also defines the following GSI:

```

@staticmethod
def key_GSI(
    user_id: str, geofence_id: str, geofence_event_id: str
) -> dict[str, str]:
    return {
        'GSI1PK': f'{core.EntityPrefix.USER}{user_id}',
        'GSI1SK': (
            f'{core.EntityPrefix.GEOFENCE_EVENT}{geofence_id}#{geofence_event_id}'
        ),
    }

```

The reason why this GSI was introduced is due to the supportive suggestions that the user must receive when filling in a new time card. In that case, in fact, it is important to be able to efficiently query all the geofence events associated with a certain user.

Notice also the structure of the Sort Key: the “geofence_id” is the first part of this key because when the user is creating a timecard, we first ask him to input the project he want to associate the time card with. In this way, we can then fetch all the geofences associated with the project and use them to find more quickly the location where meaningful information are stored. Lastly, notice that “geofence_event_id” is a ULID built using the timestamp at which the event was recorded: this means that, for each user, all the geofences events will be already sorted by timestamp geofence-wise, significantly speeding up the data retrieval. Notice that the standard that was used in order to save timestamps is ISO 8601.

6.2.3 Timecard

A Timecard represents a time window during which a person was working. The data model defines the following attributes:

```

class Timecard(core.Item):
    '''A timecard entry.'''
    Id: str
    '''The ID of the timecard
    We use ULID to sort them and perform range query'''
    UserId: str
    '''User the timecard refers to'''
    ProjectId: str
    '''Project the timecard refers to'''
    StartedAt: str
    '''Timestamp at which employee started working in UTC and using ISO 8601.'''
    EndedAt: str
    '''Timestamp at which employee stopped working in UTC and using ISO 8601.'''
    CreatedAt: str
    '''Creation time, in UTC and using ISO 8601.'''
    GSI1PK: str
    '''Lets us easily look up all timecards bound to a specific user.'''
    GSI1SK: str
    '''Binding ID that links to a user.'''
    Type: Literal['TIMECARD'] = core.EntityType.TIMECARD.value

```

The primary key of the Timecard data model is the following.

```

@staticmethod
def key(project_id: str, timecard_id: str) -> dict[str, str]:
    '''The primary key of the item.'''
    return {
        'PK': f'{core.EntityPrefix.PROJECT}{project_id}',
        'SK': f'{core.EntityPrefix.TIMECARD}{timecard_id}',
    }

```

The choice for such a primary key was made because it is crucial that timecards can be retrieved quickly project-wise. As already presented, in fact, every time that a project admin accesses the project overview screen, a timecards section is presented to him: this includes the timecards of all the participants of the project which explains the need for such a primary key.

Moreover, “timecard_id” is a ULID and therefore allows fast queries on ranges; in particular this ULID was built using as timestamp the start time, rather than the creation time or the end time. This is because it is always convenient to display timecards to the user ordered by their start times.

The Timecard data type also defines the following GSI.

```
@staticmethod
def key_GSI(user_id: str, timecard_id: str) -> dict[str, str]:
    return {
        'GSI1PK': f'{core.EntityPrefix.USER}{user_id}',
        'GSI1SK': f'{core.EntityPrefix.TIMECARD}{timecard_id}',
    }
```

This GSI is in fact needed in order to efficiently retrieve Timecards for a specific user. Again, as the “timecard_id” used in the GSI1SK is a ULID, Timecards are sorted temporarily.

6.3 APIs

In this section, an overview of the most important APIs (Application Programming Interface) is presented.

Before delving into the details, a short introduction of the REST (Representational State Transfer) paradigm is presented.

6.3.1 REST

REST is an architectural style to create scalable and loosely coupled web services. It defines a set of principles for building networked applications and APIs that communicate over the HTTP protocol.

Key principles of REST include:

- **Statelessness:** Each request from a client to the server must contain all the information needed to understand and process the request. The server should not store any client context between requests. This design helps to improve scalability and reliability.
- **Client-Server Separation:** The client and the server are separate entities that communicate through well-defined interfaces. This separation allows for independent evolution and development of both the client and server components.
- **Cacheability:** Responses from the server can be cached by the client or intermediary servers, such as proxy servers. Cacheable responses help reduce network traffic and improve performance.
- **Uniform Interface:** RESTful APIs use a uniform set of methods to interact with resources. The most common HTTP methods used in REST are GET (retrieve), POST (create), PUT (update), and DELETE (remove). Resources are identified using URIs (Uniform Resource Identifiers).

- **Layered System:** Intermediary servers, such as proxy servers or gateways, can be used to improve scalability and security. Clients are unaware of the presence of these intermediaries and communicate directly with the server.

RESTful APIs often use JSON (JavaScript Object Notation) as the data format for communication, but they can also use XML or other formats. Over the years, REST has become a popular choice for building APIs due to its simplicity, scalability, and compatibility with the HTTP protocol that underlies the World Wide Web.

6.3.2 API Models and Routes

This section presents some of the most important API models and routes used in the project.

First of all, it is important to make a clear distinction between the data models already presented and the API models that are discussed in this section:

- The data models, in fact, represent the ways in which entities are stored in the database; each instance of a specific data model will in fact have all the mandatory attributes specified in the model itself and possibly the optional ones.
- The API models, instead, represent the data that has to be sent from the database layer to the client (i.e., the mobile and web applications) or vice versa. It is therefore possible to have multiple representations for the same entity in different resources, optimizing therefore the data fetches in the backend to only fetch the needed data.

Notice that all the API models are inherited from either “Request” or “Response” in order to enforce the Pydantic validation checks.

```
class Request(pydantic.BaseModel):
    '''Base model for all request body models.
    All models that are used directly or indirectly as request body
    (https://fastapi.tiangolo.com/tutorial/body/) should derive from this class.
    ...

class Response(pydantic.BaseModel, extra=pydantic.Extra.forbid):
    '''Base model for all response body models.
    All models that are used directly or indirectly in the response model (i.e.
    return type) (https://fastapi.tiangolo.com/tutorial/response-model/) should
    derive from this class.
    ...
```

The followings are examples of API models for the Timecard entity.

```
class CreateTimecard(core.Request):
    '''A timecard creation request.'''
    started_at: datetime
    '''Timestamp at which employee started working in UTC and using ISO 8601.'''
    ended_at: datetime
    '''Timestamp at which employee stopped working in UTC and using ISO 8601.'''

class Timecard(core.Response):
    id: str
    '''The ID of the timecard'''
    user_id: str
    '''User the timecard refers to'''
    project_id: str
    '''Project the timecard refers to'''
    started_at: datetime
    '''Timestamp at which employee started working in UTC and using ISO 8601.'''
    ended_at: datetime
    '''Timestamp at which employee stopped working in UTC and using ISO 8601.'''

class CreatedTimecard(core.Response):
    '''A created timecard.'''
    id: str
    '''The timecard ULID'''
    created_at: datetime
    '''When was the timecard created (in UTC)?'''
    @staticmethod
    def from_item(timecard_item: timecards.Timecard) -> CreatedTimecard:
        return CreatedTimecard(
            id=timecard_item.Id,
            created_at=datetime.fromisoformat(timecard_item.CreatedAt),
        )

class UpdateTimecard(core.Request):
    '''A timecard update request.
    All fields are optional. If a field is explicitly set to None, it will be
    removed. If it's not set, it will be left unchanged.
    ...
    '''
    started_at: datetime | None
    '''Timestamp at which employee started working in UTC and using ISO 8601.'''
    ended_at: datetime | None
    '''Timestamp at which employee stopped working in UTC and using ISO 8601.'''
```

As anticipated in the Entities section of this chapter, the fact that all the data models are derived from “core.Item” makes the reading and writing operations very handy.

For example, to read an item of type timecards, the code is as simple as this.

```
timecard = timecards.Timecard.get_item(
    timecards.Timecard.key(project_id, timecard_id)
)
```

Similarly, for deleting an item:

```
timecards.Timecard.delete_item(
    timecards.Timecard.key(project_id, timecard_id)
)
```

In the following the implementation of some of the Timecard APIs are presented as an example. For instance, the following is the route to create a new timecard.

```
@router.post("/{project_id}/timecards', status_code=201)
def create_timecard(
    project_id: str,
    new_timecard: models.CreateTimecard,
    user_id: str = Depends(dependencies.jwt_user_id),
) -> models.CreatedTimecard:
    participation = db_participants.ProjectParticipant.get_item(
        db_participants.ProjectParticipant.key(project_id, user_id)
    )
    if not permissions.can_create_read_update_delete_timecard_to_project(
        participation
    ):
        raise fastapi.HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden"
        )
    timecard_item = _timecard_item(
        project_id, user_id, new_timecard.started_at, new_timecard.ended_at
    )
    timecards.Timecard.put_item(timecard_item)
    return models.CreatedTimecard.from_item(timecard_item)
```

Notably, it is important to always check permission before performing an operation. In this case, for instance, we are making sure the user is a participant of the project to which he wants to associate the timecard with.

Similarly, for deleting and retrieving a specific Timecard using the id:


```

@router.delete('/{project_id}/timecards/{timecard_id}', status_code=204)
def delete_timecard(
    project_id: str,
    timecard_id: str,
    user_id: str = Depends(dependencies.jwt_user_id),
) -> None:
    participation = db_participants.ProjectParticipant.get_item(
        db_participants.ProjectParticipant.key(project_id, user_id)
    )
    if not permissions.can_create_read_update_delete_timecard_to_project(
        participation
    ):
        raise fastapi.HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden"
        )
    timecards.Timecard.delete_item(
        timecards.Timecard.key(project_id, timecard_id)
    )

@router.get('/{project_id}/timecards/{timecard_id}')
def get_timecard(
    project_id: str,
    timecard_id: str,
    user_id: str = Depends(dependencies.jwt_user_id),
) -> models.Timecard:
    participation = db_participants.ProjectParticipant.get_item(
        db_participants.ProjectParticipant.key(project_id, user_id)
    )
    if not permissions.can_create_read_update_delete_timecard_to_project(
        participation
    ):
        raise fastapi.HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden"
        )
    timecard = timecards.Timecard.get_item(
        timecards.Timecard.key(project_id, timecard_id)
    )
    if timecard is None:
        raise HTTPException(status_code=404, detail="Not Found")
    return models.Timecard(
        id=timecard.Id,
        user_id=timecard.UserId,
        project_id=timecard.ProjectId,
        started_at=timecard.StartedAt,
        ended_at=timecard.EndedAt,
    )

```

Clearly, sometimes certain operations cannot be performed with the simple use of

“core.Item” support but need the use of more advanced queries. The following is an example of a route using a query that fetches all the timecards between two given timestamps.

```
@router.get('/{project_id}/timecards')
def get_all_project_timecards(
    project_id: str,
    start_date: str = Query(None, alias='startDate'),
    end_date: str = Query(None, alias='endDate'),
    user_id: str = Depends(dependencies.jwt_user_id),
) -> list[models.Timecard]:
    participation = db_participants.ProjectParticipant.get_item(
        db_participants.ProjectParticipant.key(project_id, user_id)
    )
    if not permissions.can_create_read_update_delete_timecard_to_project(
        participation
    ):
        raise fastapi.HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden"
        )
    if start_date is None or end_date is None:
        results = project.fetch_all_timecards_since_project_creation(project_id)
    else:
        date_format = "%d-%m-%Y"
        parsed_start_date = datetime.strptime(start_date, date_format)
        start_datetime = parsed_start_date.combine(parsed_start_date, time.min)
        start_ulid = ulid.from_timestamp(start_datetime)
        parsed_end_date = datetime.strptime(end_date, date_format)
        end_datetime = parsed_end_date.combine(parsed_end_date, time.max)
        end_ulid = ulid.from_timestamp(end_datetime)
        results = timecards.Timecard.query_list(
            Key('PK').eq(f'{core.EntityPrefix.PROJECT}{project_id}')
            & Key('SK').between(
                f'{core.EntityPrefix.TIMECARD}{start_ulid}',
                f'{core.EntityPrefix.TIMECARD}{end_ulid}',
            )
        )
    return [
        models.Timecard(
            id=tc.Id,
            user_id=tc.UserId,
            project_id=tc.ProjectId,
            started_at=tc.StartedAt,
            ended_at=tc.EndedAt,
        )
        for tc in results
    ]
```

The parent class also provides methods to read multiple entries – also representing

different entities, with the same call to the backend by specifying multiple keys; this helps improve performance.

Another example is the one of the route used to fetch all the geofences associated with a specific project.

```
@router.get('/{user_id}/project-geofences')
def get_project_geofences(
    user_id: str,
    jwt_user_id: str = Depends(dependencies.jwt_user_id),
) -> list[models.ProjectGeofence]:
    if user_id != jwt_user_id:
        raise fastapi.HTTPException(status_code=403, detail="Forbidden")
    project_participations = get_project_participations(user_id)
    if project_participations is None:
        raise fastapi.HTTPException(status_code=403, detail="Forbidden")
    results = [
        list(
            geofences.ProjectGeofence.query(
                Key('PK').eq(
                    f'{core.EntityPrefix.PROJECT}{prj_part.project.id}'
                )
                & Key('SK').begins_with(core.EntityPrefix.GEOFENCE.value)
            )
        )
        for prj_part in project_participations
    ]
    return [
        models.ProjectGeofence(
            id=project_geofence.Id,
            project_id=project_geofence.ProjectId,
            radius=project_geofence.Radius,
            latitude=project_geofence.Latitude,
            longitude=project_geofence.Longitude,
        )
        for res in results
        for project_geofence in res
    ]
```

Another significant example, very relevant to the Time Card Manager feature is the following: it is used to fetch all the geofence events related to a specific user and project. This is relevant to display suggestions to the user while he is creating a new timecard.

```

@router.get('/{user_id}/geofence-events/{project_id}/date/{date}')
def get_geofence_events(
    user_id: str,
    project_id: str,
    date: str,
    jwt_user_id: str = Depends(dependencies.jwt_user_id),
) -> list[models.GeofenceEvent]:
    if user_id != jwt_user_id:
        raise fastapi.HTTPException(status_code=403, detail="Forbidden")
    participation = db_participants.ProjectParticipant.get_item(
        db_participants.ProjectParticipant.key(project_id, user_id)
    )
    if not permissions.can_create_read_update_delete_timecard_to_project(
        participation
    ):
        raise fastapi.HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden"
        )
    project_geofences_items = project.fetch_project_geofences(project_id)
    date_format = "%d-%m-%Y"
    parsed_date = datetime.strptime(date, date_format)
    start_datetime = parsed_date.replace(
        hour=0, minute=0, second=0, microsecond=0
    )
    start_ulid = ulid.from_timestamp(start_datetime)
    end_datetime = start_datetime.replace(
        hour=23, minute=59, second=59, microsecond=999999
    )
    end_ulid = ulid.from_timestamp(end_datetime)
    project_geofence_events = [
        geofences.GeofenceEvent.query_list(
            Key('GSI1PK').eq(f'{core.EntityPrefix.USER}{user_id}'),
            & Key('GSI1SK').between(
                f'{core.EntityPrefix.GEOFENCE_EVENT}{prj_geo.Id}#{start_ulid}',
                f'{core.EntityPrefix.GEOFENCE_EVENT}{prj_geo.Id}#{end_ulid}',
            ),
            index_name=constants.TABLE_GSI1,
        )
        for prj_geo in project_geofences_items
    ]
    return [
        models.GeofenceEvent(
            id=geo_evt.Id,
            geofence_id=geo_evt.GeofenceId,
            geofence_event_type=geo_evt.GeofenceEventType,
            created_at=geo_evt.CreatedAt,
        )
        for geofence_events in project_geofence_events
        for geo_evt in geofence_events
    ]

```

The following examples are not meant to be a complete discussion of the implementation of this thesis project. For a deeper analysis, I strongly recommend to download and explore Benetics code repository linked in Appendix A.

6.4 Entities Clean-Up

One important part of the project was to implement a mechanism to delete entities on cascade. In fact, DynamoDB does not have a straightforward mechanism to use the classical "ON DELETE CASCADE" typical of relational databases.

6.4.1 Clean-Up Lambdas

The way cascade deletions are typically implemented in DynamoDB is through Lambda functions.

DynamoDB does not support the concept of cascading deletes out-of-the-box like traditional relational databases. In relational databases, a cascading delete refers to the automatic deletion of dependent records in related tables when a record in the parent table is deleted.

However, in DynamoDB, each item in a table is treated as an independent entity, and there is no inherent mechanism for managing relationships or enforcing cascading behavior.

If cascading delete-like behavior wants to be achieved in DynamoDB, it is necessary to implement it from scratch in the application code. Here's a general outline of how this might be approached:

- **Identify Dependencies:** determine which items in the DynamoDB table have dependencies on other items; these dependencies could be represented as attributes of items.
- **Application Logic:** when an item has to be deleted, the application logic should first identify the dependent items and delete them before deleting the main item.
- **Batch Deletes:** DynamoDB supports batch operations, which allow to delete multiple items in a single operation. This can help to significantly improve efficiency and keep costs reduced.
- **Error Handling:** since DynamoDB is a distributed and highly available service, there could be cases where some items are deleted successfully while others fail due to network issues or other factors. The application needs to handle such scenarios and possibly retry or log errors.

DynamoDB's design philosophy is different from traditional relational databases, and it's optimized for high scalability, low-latency performance, and availability. This means that some of the patterns and practices used in relational databases might need to be adapted or rethought when working with DynamoDB.

It is important to decide which events are the triggers of the lambdas.

Following, as an example, is the definition of some of the lambdas in the "serverless.yml" file.

```
# Cleans up after deleted geofence.
GeofenceDeleter:
  image:
    name: backend
    command:
      - benetics.clean_up.geofence_deleted_handler.handle
  timeout: 300 # 5 minutes
  events:
    - stream:
        type: dynamodb
        arn:
          { "Fn::GetAtt": ["DynamoDBTable", "StreamArn"] }
        filterPatterns:
          - eventName: [REMOVE]
            dynamodb:
              OldImage:
                Type:
                  S: [PROJECT_GEOFENCE]
        batchWindow: 10
        batchSize: 10
        functionResponseType: ReportBatchItemFailures
  environment:
    POWERTOOLS_SERVICE_NAME: GeofenceDeleter
```

Similarly for the project:

```
# Cleans up after deleted projects.
ProjectDeleter:
  image:
    name: backend
    command:
      - benetics.clean_up.project_deleted_handler.handle
  timeout: 300 # in seconds (5 mins)
  events:
    - stream:
        type: dynamodb
        arn:
          { "Fn::GetAtt": ["DynamoDBTable", "StreamArn"] }
        filterPatterns:
          - eventName: [REMOVE]
            dynamodb:
              OldImage:
                Type:
                  S: [PROJECT]
          # We use a batchWindow of 0 as we want deletion to be reflected in the
          # UI for other users as soon as possible. This requires a max batch size
          # of 10.
            batchSize: 10
            functionResponseType: ReportBatchItemFailures
  environment:
    POWERTOOLS_SERVICE_NAME: ProjectDeleter
```

This is clearly just the definition and configuration of the lambda functions. The logic is instead contained in the Lambda handlers, as shown below in the case of geofences.

```

def _parse_geofence(record: DynamodbRecord) -> geofences.ProjectGeofence:
    return geofences.ProjectGeofence.parse_obj(
        TypeDeserializer().deserialize({'M': record['dynamodb']['OldImage']})
    )
def _delete_geofence_event(
    geofence_event: geofences.GeofenceEvent,
) -> None:
    geofences.GeofenceEvent.delete_item(
        key=geofences.GeofenceEvent.key(
            geofence_event.GeofenceId, geofence_event.Id
        )
    )
def _fetch_all_geofence_events(
    geofence_id: str,
) -> list[geofences.GeofenceEvent]:
    '''Fetch all the GeofenceEvents associated with the Project.
    Args:
        project_id: The Project's ID.
    Returns:
        The list of GeofenceEvent.
    ...
    return geofences.GeofenceEvent.query_list(
        Key('PK').eq(f'{core.EntityPrefix.GEOFENCE}{geofence_id}')
        & Key('SK').begins_with(core.EntityPrefix.GEOFENCE_EVENT.value)
    )
def _fetch_and_delete_geofence_events(
    geofence_id: str,
) -> None:
    geofence_events = _fetch_all_geofence_events(geofence_id)
    delete_geofence_events_futs = []
    for evt in geofence_events:
        delete_geofence_events_futs.append(
            global_executor.submit(_delete_geofence_event, evt)
        )
    _ = [fut.result() for fut in delete_geofence_events_futs]

```



```
@lambda_metrics.log_metrics(capture_cold_start_metric=True)
@logging.getLogger().inject_lambda_context(clear_state=True)
def handle(
    event: events.DynamoDBStreamEvent, context: context_typing.Context
) -> None:
    '''Handle DynamoDB events where a entity of type PROJECT_GEOFENCE
    is deleted.
    ...
    del context # Unused

    lambda_metrics.add_dimension('Stage', constants.STAGE)
    logging.info('Deleting geofence')

    for record in event['Records']:
        try:
            deleted_geofence = _parse_geofence(record)
            _fetch_and_delete_geofence_events(deleted_geofence.Id)

        except pydantic.ValidationError:
            logging.exception(
                'Could not parse ProjectGeofence from event: %s',
                record['dynamodb'],
            )
```

As can be seen, it is necessary to implement the code for retrieving the entries related to the one that has to be deleted: in this case, for instance, GeofenceEvents associated with the deleted Geofence has to be deleted programmatically.

Similarly, the Lambda handler with the logic for handling project deletion follows.

```
def _parse_project(record: DynamodbRecord) -> Project:
    return Project.parse_obj(
        TypeDeserializer().deserialize({'M': record['dynamodb']['OldImage']})
    )

def _delete_project_geofence(
    project_geofence: geofences.ProjectGeofence,
) -> None:
    geofences.ProjectGeofence.delete_item(
        key=geofences.ProjectGeofence.key(
            project_geofence.ProjectId, project_geofence.Id
        )
    )

def _fetch_and_delete_project_geofences(
    project_id: str,
) -> None:
    project_geofences = project.fetch_project_geofences(project_id)
    delete_project_geofences_futs = []
    for geof in project_geofences:
        delete_project_geofences_futs.append(
            global_executor.submit(_delete_project_geofence, geof)
        )
    _ = [fut.result() for fut in delete_project_geofences_futs]

def _delete_timecard(
    timecard: timecards.Timecard,
) -> None:
    timecards.Timecard.delete_item(
        key=timecards.Timecard.key(timecard.ProjectId, timecard.Id)
    )

def _fetch_and_delete_timecards(
    project_id: str,
) -> None:
    timecards = project.fetch_all_timecards_since_project_creation(project_id)
    delete_timecards_futs = []
    for tc in timecards:
        delete_timecards_futs.append(
            global_executor.submit(_delete_timecard, tc)
        )
    _ = [fut.result() for fut in delete_timecards_futs]
```

```
@lambda_metrics.log_metrics(capture_cold_start_metric=True)
@logging.getLogger().inject_lambda_context(clear_state=True)
def handle(
    event: events.DynamoDBStreamEvent, context: context_typing.Context
) -> None:
    del context # Unused
    lambda_metrics.add_dimension('Stage', constants.STAGE)
    for record in event['Records']:
        try:
            deleted_project = _parse_project(record)
            _fetch_and_delete_project_geofences(deleted_project.ProjectId)
            _fetch_and_delete_timecards(deleted_project.ProjectId)
            # (omitted the part for deleting all other entities like tasks,
            # participations, plans...)
        except pydantic.ValidationError:
            logging.exception(
                'Could not parse Project from event: %s', record['dynamodb']
            )
```

6.4.2 Soft Deletions

This section quickly presents another approach of handling deletions, which Benetics is planning to transition towards: the one of soft deletions.

Soft deletions are a concept often used in software development to provide a way to "delete" data without permanently removing it from the system. Instead of immediately erasing the data, soft deletions involve marking the data as deleted or inactive while still retaining it in the database. This approach offers several advantages for users and the overall system:

- **Data Recovery:** is the ability to recover accidentally deleted data. Since the data is not immediately removed, users and administrators have the chance to restore it if it was deleted unintentionally.
- **Audit Trail:** soft deletions enable the creation of an audit trail, providing transparency and accountability for data modifications. Users can track when an item was deleted, who performed the deletion, and potentially the reason behind it.
- **Legal and Compliance Requirements:** in many industries, there are regulations that require organizations to retain certain data for a specific period of time, even if it's no longer actively used. Soft deletions can help organizations comply with such requirements by keeping the data accessible while still considering it "deleted."

- **User Confidence:** knowing that data can be recovered if needed can enhance user confidence in the system. Users might be more willing to engage with the platform, knowing that their data is not permanently lost if they make a mistake.
- **Faster Recovery:** in case of accidental data loss due to system failures or human errors, soft deletions can simplify recovery processes. Instead of relying on complex data restoration procedures, the system can simply "undelete" the data by marking it as active again.
- **Testing and Development:** soft deletions can be useful for testing and development environments. Developers can simulate real-world scenarios by performing deletions and then restoring data without relying on backup/restore processes.
- **Data Analysis:** soft deleted data can be useful for data analysis and reporting. It can help track trends, user behaviors, and patterns, even if the data is no longer in active use.
- **User Control:** soft deletions empower users by allowing them to have a degree of control over their data. They can choose to delete data they no longer need while still having the option to retrieve it if circumstances change.

However, implementing soft deletions comes with its own challenges, including more complex database management, potential performance impacts, and the need for a well-designed user interface to manage and recover deleted data. Security concerns are also important – soft deleted data should be properly protected to prevent unauthorized access.

7. Privacy Concerns

The utilization of geolocation data has transformed modern mobile applications, enabling innovative features and personalized experiences.

However, it is important to acknowledge that the use of such data is inherently linked to potential privacy concerns. These concerns become even more pronounced when geolocation is employed not only in the foreground—when the user actively engages with the application—but also in the background, when the application operates while closed or terminated. This extended access to location data raises questions about continuous user tracking and the implications for individual privacy.

Before starting to implement invasive features like these it is always good to ask questions like “Will the users be concerned about sharing this information with the app?”, “What is the cost that they are paying in order to gain access to the value the app is offering?”.

This chapter delves into the intricate landscape of privacy issues surrounding background geolocation and discusses strategies to address them.

7.1 Data Access Permissions

Before accessing any user data, obtaining proper permissions is paramount. In the realm of geolocation, acquiring explicit consent from users is essential due to the sensitive nature of location data. The application must prompt users to grant access to their location information.

Upon initiation, the user is presented with the option to grant either "Precise" or "Approximate" location permissions. This initial choice empowers users to decide the granularity of their location data shared with the application.

Moreover, the user has to decide whether to grant the permissions and, in this case, if he wants to do that “Only this time” or “While using the app”, the typical options for this question.

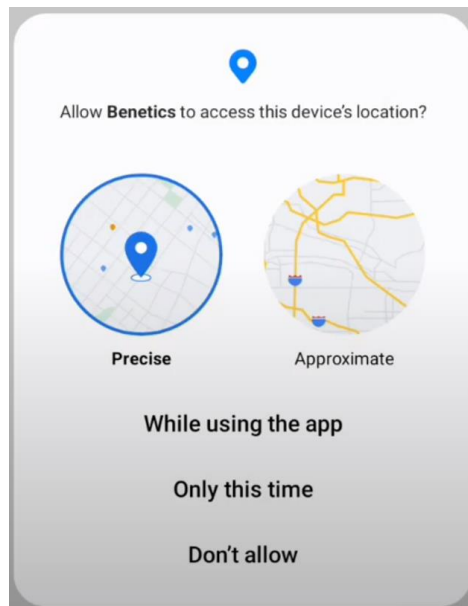


Figure 7.1: Location permissions request.

Subsequently, the application seeks permission for physical activity data: this is particularly relevant when using optimized libraries, such as those from Transistor Software, which thrive on such data. It's noteworthy that the application can continue to function even if users deny access to this specific privilege, as elaborated in the dedicated chapter: in this case, however, performances will be worse.

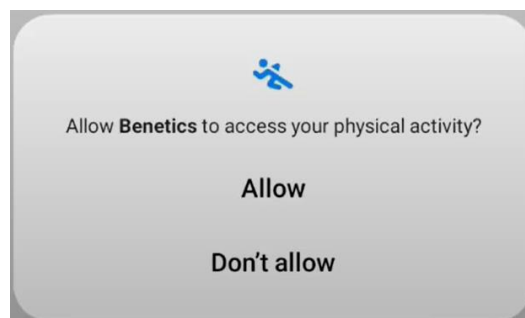


Figure 7.2: Physical activity permissions request.

The crucial aspect of background permissions follows, allowing the application to access location data even when it is not in active use ("Allow all the time").

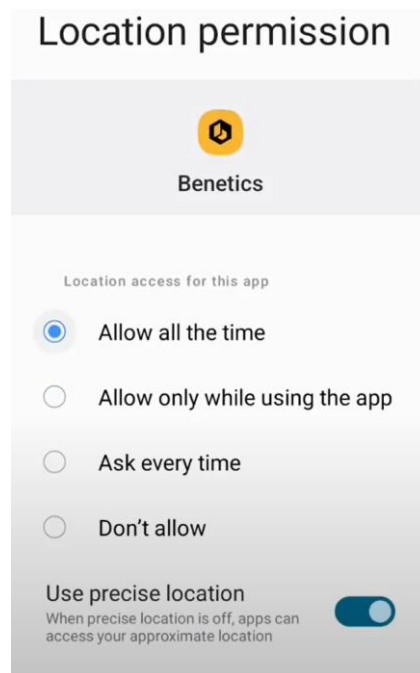


Figure 7.3: Background permissions request.

This permission drastically affects the effectiveness of the features implemented. Users who restrict background permissions to “Only while using the app” for instance will significantly limit the application's functionality, capturing data only during app usage. However, securing these permissions is a delicate balance; hence, it is necessary to provide users with clear explanations of how and why their data will be used.

Both the App Store and Play Store maintain stringent criteria regarding background geolocation permissions [9]. A range of demands must be satisfied, including module integration, declaration submissions, demonstrative videos, and comprehensive documentation, to successfully navigate the review process and introduce such features to their production versions.

7.2 Users Trust

The introduction of invasive features always necessitates careful consideration. During my internship, the team faced the challenge of introducing background geolocation-based features while maintaining users' trust and minimizing disruption.

Several factors informed the decision to delay full-scale deployment, which can be summarized in the following two prudent reasons:

- **Early Product Stage and Limited User Base:** With the product in its infancy and a limited user base, introducing such a powerful feature could raise undue concerns among users.
- **Building User Trust:** In industries like construction, where skepticism towards technology and concerns about privacy are common, gaining user trust is essential. Moreover, these working environments are characterized by a lack of trust between workers and employers, often belonging to very distant social categories, which makes the task even more difficult.

Originally, the plan was to introduce these features directly in the production version of the app; however, the approach evolved to prioritize a more cautious route. This decision represented a trade-off between offering advanced features and mitigating the risk of alienating users apprehensive about privacy breaches.

As we contemplate these topics, we are compelled to delve deeper into the broader landscape of privacy concerns, permissions, legal frameworks, user education, and ethical considerations. Each facet of this intricate puzzle contributes to the responsible and effective deployment of background geolocation within mobile applications.

7.3 Regulations

The landscape of geolocation data privacy is shaped by legal and regulatory frameworks, such as the General Data Protection Regulation (GDPR) in Europe and the California Consumer Privacy Act (CCPA) in the US. These regulations impose stringent requirements on the collection, storage, and usage of user data. Compliance with these regulations ensures user rights are respected and that data protection measures are robust.

7.4 Mitigations

To mitigate the problems related to privacy, several techniques may be considered:

- **Data Encryption:** Encrypting location data during transmission and storage safeguards it from unauthorized access, enhancing overall data security.
- **Data Collection Minimization:** Minimizing the collection of sensitive geolocation data is a fundamental privacy principle. For example, the library for performing geofencing may be configured in a way that it is actively monitoring the user only during certain moments and not 24/7. For instance, for workers during the day, it may be turned off during hours at night, during weekends, holidays, and in general at any moment the user is not meant to be

at work. The integration of the app with the workers' calendars may facilitate this task.

- **Data Retention Minimization:** Similarly, another important principle in data privacy is to keep the data the minimum amount of time that they are needed and discard it as soon as they are not anymore.

Moreover, it is important to underline again the importance of user awareness and control: this means providing extensive information to the users and letting them opt in and out at any point in time if they change their mind.

8. Conclusions

This final chapter contains the conclusions and possible future developments of my work.

8.1 Summary of Contributions

In this section I want to remark again the results that I was able to achieve highlighting the main contributions for Benetics:

- Analyzed different background geolocation libraries.
- Adopted, configured, and tested Transistor Software library.
- Designed, implemented, and tested prototypical versions of:
 - Remote Personnel Presence Monitoring
 - Time Cards Manager

Other secondary contributions involve:

- Researched solutions and the feasibility of features, like the use of human activity recognition to enable safety notification system.
- Handled background geolocation privacy permissions on Apple Store and Play Store.

8.2 Future Developments

Asking for background geolocation permissions can be intimidating for users. It is therefore important to make sure that the added value they can take advantage of is more than the fear they may have towards this kind of technology.

For this reason, the following ideas may be considered before introducing the feature in the production version of the app.

8.2.1 General Improvements in Geofencing

The implementation of the geofencing mechanism can still be improved and the following are some ideas that may help in that perspective:

- Give more control over geofences to the user; for the time being, geofences are only created after a new project is created and the user has no possibility to change the shape of the area.
- Allow the user to draw geofences on the map of any shape and size and automatically cover them using circular shaped geofences. If this task becomes complex, consider switching to a library that already supports geofences of arbitrary shapes.

8.2.2 Intelligent Location Based Task Management & Reminders

As part of the challenging task to reduce information complexity for workers, using the infrastructure that I have developed, it would be possible to integrate tasks with the location of the user.

For example, it would be possible to define geofences where certain actions must be taken and send reminders to the users whenever he enters or exits one of these. For example:

- Remind the user to turn off machinery when he leaves as last the work area.
- Invite the user to start a certain task saving the time for him to understand it from the tasks list.

8.2.3 Location Based Safety Alerts

In 2023, the safety levels guaranteed on construction sites are unfortunately still not enough. Especially in less developed counties, serious injuries and deaths to the work environment are still happening too often, as highlighted in a report from Procore [42].

Similarly to 7.2.1, introducing in the app a safety alert mechanism could reduce the accidents and guarantee better conditions for the workers. With a small effort, it would be possible to define geofences that represent areas of danger and send notifications to the user whenever he enters one of these. For example:

- Remind the user to wear adequate Personal Protective Equipment (PPE) in certain environments.

Clearly, the accuracy of the geolocation is crucial; in this respect, the use of Bluetooth Low Energy (BLE) beacons or proximity sensors at specific positions of the

construction site may be beneficial to improve the precision and recall of such a notification system. This would come with the additional cost of building, deploying, and maintaining the physical hardware infrastructure.

8.2.4 Human Activity Recognition Based Safety Alert

Another idea involves introducing and exploiting human activity recognition [22]; this way, it would be possible to check if workers are respecting the time limits imposed by the law to prevent accidents and ensure safety in the work places. As highlighted previously, the accuracy of human activity recognition by only using data coming from a single smartphone may be not enough, even if some research studies suggest the opposite [21, 23]; therefore, other electronic devices may be considered in order to enhance this kind of features.

Examples of safety alerts based on human activity recognition are the followings:

- Remind the user to take a break and sit for a while after a period of work. Moreover, increase the frequency of the breaks if the user is using tools like a drill, which may be quite easy to recognize by a smartphone.
- If the user has a very extended period of inactivity, send a notification to his workmates to check his health conditions and provide help if necessary.

A final note for all the proposed alerting and notification systems is that the algorithm must clearly be able to run synchronously and process the data in real time in order to send notifications in a timely manner. This introduces more challenges, and it is important to take this aspect into account when designing the whole service.

8.2.5 Manual Stopwatch for Time Cards Management

The Background Geolocation feature will be introduced as optional: this means that the users can decide whether to enable or not the functionality.

This also means that the support of the Background Geolocation to the Time Cards Manager is not always guaranteed. Therefore, it is important that the latter can continue to work even without the geofences data coming from the library, which may in fact be not in use for certain users. In this scenario, the Time Card Manager will have to function in a more standard way and information about the worked hours will be filled in directly by the users without any suggestion or support coming from the geofences data.

For these users, something that may be useful and that many other systems online have implemented is a simple stopwatch which allows the user to monitor his

working sessions. By starting the stopwatch, the user is telling the app that is clocking in and by stopping it that he is instead clocking out.

Clearly, this is not really a sophisticated way of implementing a Time Cards Manager; however, his importance relies on the fact that, in general, when features depend on optional data that the user may decide or not to grant, a backup solution must be present in order to have a fully functional app for everyone.

8.3 Learnings

This section outlines the significant insights gained from the culmination of my thesis work and internship experience. From a technical standpoint, the following key learnings emerged:

- Enhanced my expertise in cross-platform mobile app development, particularly over the React Native framework.
- Deepened my understanding and expertise of AWS, crucial in today's competitive job market as predominant cloud services platform.
- Navigated the dynamics of a small team, shouldering responsibility and achieving accelerated personal growth.
- Engaged in the collaborative evolution of an ongoing product, a challenge that brings both complexity and fulfillment.
- Acquired insights into the process of releasing a live mobile application on platforms like the App Store and Play Store.
- Recognized the paramount significance of maintaining high interoperability between modern and legacy versions of the app and backend, emphasizing the importance of backwards compatibility.

Moreover, broader takeaways and reflections emerged from this experience:

- Acknowledged the intricacies of making long-term decisions within the realm of software engineering. Adapting to a product that is still a work in progress and being receptive to potential changes in direction based on forthcoming customer feedback, underscores the need for a flexible mindset. It's essential to uphold a clear vision of the product's problem-solving mission while being open to necessary adjustments, such as the alteration prompted by background geolocation requirements.
- Internalized the principle that "done is better than perfect," particularly applicable in startup environments. Swift decision-making and efficient execution, emphasizing speed and momentum, can significantly influence the success trajectory.

- Recognized the foundational role of customer trust and data privacy in the attainment of success. The assurance of these aspects is integral in shaping positive user experiences and fostering sustainable growth.

In summation, this journey has fortified my technical proficiency, honed my collaborative skills, and nurtured a holistic perspective on software engineering dynamics. These insights, ranging from the technical to the strategic, shall undoubtedly resonate throughout my future endeavors.

Definitions

- **Benetics**: startup I worked with for my thesis work or homonymous mobile app, depending on the context.
- **Geofence**: data type representing a circular area with a certain radius located at a certain longitude and latitude.
- **GeofenceEvent**: data type representing an entering or exiting event from a certain Geofence.
- **React Native Background Geolocation**: React Native library used for enabling background geolocation features in the app.
- **Timecard**: data type representing a time window, characterized by a start and end time, during which a person was working
- **Transistor Software**: company which developed the React Native Background Geolocation library.

Acronyms

- **API:** Application Programming Interface.
- **APK:** Android Package Kit.
- **AWS:** Amazon Web Services.
- **BLE:** Bluetooth Low Energy.
- **GDPR:** General Data Protection Regulation.
- **GPS:** Global Positioning System.
- **GSI:** Global Secondary Index.
- **IMU:** Inertial Measurement Unit.
- **PK:** Partition Key.
- **REST:** Representational State Transfer.
- **SK:** Sort Key.
- **ULID:** Universally Unique Lexicographically Sortable Identifier.
- **UUID:** Universally Unique Identifiers.

Bibliography

- [1] Transistor Software
<https://www.transistorsoft.com/>
- [2] React Native Background Geolocation GitHub
<https://github.com/transistorsoft/react-native-background-geolocation>
- [3] React Native Background Geolocation Documentation
<https://transistorsoft.github.io/react-native-background-geolocation/index.html>
- [4] Benetics Website
<https://benetics.io/>
- [5] Benetics GitHub Repository
<https://github.com/benetics-ag/benetics>
- [6] Benetics App Download for iOS
<https://apps.apple.com/us/app/benetics/id1622611171>
- [7] Benetics App Download for Android
<https://play.google.com/store/apps/details?id=com.benetics&hl=it&gl=US&pli=1>
- [8] Amazon Web Services
<https://aws.amazon.com/it/>
- [9] Understanding Location in the Background Permissions - Play Store
<https://support.google.com/googleplay/android-developer/answer/9799150?hl=en>
- [10] Global Search Index
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSIL.html>
- [11] ULID
<https://victoryosayi.medium.com/ulid-universally-unique-lexicographically-sortable-identifier-d75c253bc6a8>
- [12] TransistorSoftware vs Radar
<https://radar.com/content/alternatives/react-native-background-geolocation-vs-radar>
- [13] Xcode

- <https://apps.apple.com/it/app/xcode/id497799835?mt=12>
- [14] Testing and Location Simulations with Xcode
<https://betterprogramming.pub/how-to-simulate-location-in-xcode-and-simulator-94ce701f5e1f>
- [15] Simulating a Moving Location in iOS
<https://digitalbunker.dev/simulating-a-moving-location-in-ios/>
- [16] Android Studio
<https://developer.android.com/studio>
- [17] Location Simulations with Android Studio
<https://hussainhabibullah.medium.com/simulate-location-in-android-emulator-289fd6de0432>
- [18] Serverless Framework
<https://www.serverless.com/>
- [19] Work-related Stress, Psychophysiological Strain, and Recovery Among On-site Construction Personnel
<https://www.sciencedirect.com/science/article/abs/pii/S0926580521000807>
- [20] Worker Psychological Well-Being in the Construction Industry
<https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29ME.1943-5479.0001074>
- [21] Development of iPhone Based Mobile App for Risk Assessment using IMUS
<https://www.diva-portal.org/smash/get/diva2:1653717/ATTACHMENT01.pdf>
- [22] How to Model Human Activity from Smartphone Data
<https://machinelearningmastery.com/how-to-model-human-activity-from-smartphone-data/>
- [23] Gait Analysis using Accelerometry Data from a Single Smartphone
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8622042/>
- [24] GPS Tracking: the Balance between Tracking Intervals and Battery Life
<https://www.brickhousesecurity.com/gps-trackers/tracking-intervals>
- [25] Inertial Measurement Units
https://en.wikipedia.org/wiki/Inertial_measurement_unit
- [26] Python
<https://www.python.org/>
- [27] FastAPI
<https://fastapi.tiangolo.com/>
- [28] Radar
<https://radar.com/>
- [29] Mapbox
<https://www.mapbox.com/>

- [30] DynamoDB
<https://aws.amazon.com/it/pm/dynamodb/>
- [31] Pydantic
<https://docs.pydantic.dev/latest/>
- [32] Docker
<https://www.docker.com/>
- [33] Transistor Software Debugging Console
<https://github.com/transistorsoft/background-geolocation-console>
- [34] Improving Value by Eliminating Errors in Construction Industry
<https://getitright.uk.com/live/files/reports/3-giri-research-report-revision-3-284.pdf>
- [35] Construction Disconnected: Rethinking the Management of Project Data and Mobile Collaboration
https://pg.plangrid.com/rs/572-JSV-775/images/Construction_Disconnected.pdf
- [36] The Next Normal in Construction by McKinsey & Company
<https://www.mckinsey.com/~media/McKinsey/Industries/Capital%20Projects%20and%20Infrastructure/Our%20Insights/The%20next%20normal%20in%20construction/The-next-normal-in-construction.pdf>
- [37] Construction Industry Statistics
<https://constructionblog.autodesk.com/construction-industry-statistics/>
- [38] Seizing the Decarbonization Opportunity in Construction
<https://www.mckinsey.com/industries/engineering-construction-and-building-materials/our-insights/call-for-action-seizing-the-decarbonization-opportunity-in-construction>
- [39] Harnessing Data Advantage in Construction
<https://construction.autodesk.com/resources/guides/harnessing-data-advantage-in-construction/>
- [40] Construction and Design Software Market
<https://www.grandviewresearch.com/industry-analysis/construction-design-software-market-report>
- [41] Recording of Working Time Regulation in Switzerland
<https://www.seco.admin.ch/seco/it/home/Arbeit/Arbeitsbedingungen/Arbeitnehmerschutz/Arbeits-und-Ruhezeiten/Arbeitszeiterfassung.html>
- [42] Construction Safety Statistics for 2023
<https://www.procore.com/library/construction-safety-statistics>
- [43] Recording Working Hours Correctly by PwC
<https://www.pwc.ch/en/insights/disclose/24/recording-working-hours-correctly.html>

A. Appendix A

This appendix is dedicated to how to install and try the application.

As already highlighted, the application is compatible both with Android and with iOS.

In order to distribute the beta versions of the app, however, some limitations apply to the iOS environment and therefore I have decided to explain here only how to install the app on Android.

In fact, on Android the only steps that are necessary are:

1. Download the APK file from the following link:
<https://drive.google.com/drive/folders/1brff8F6vYwgFWJd4-s06pLsRTbo1FYLu>
2. Click on the APK and accept the prompt to install the app.
 - a. Note: it may be necessary to enable Developer Mode in order to install an APK downloaded from the Internet.

Moreover, at the following link

<https://drive.google.com/drive/folders/1MKgbw5RsiYZU88NA3ya9r2nT9mrKr6fs>

other relevant resources can be found:

1. The entire repository of Benetics and in particular the branch where I have developed the presented features. Notice that this repository contains confidential information and that it is prohibited to share them. For this reason, an access request has to be sent and will be approved by me after evaluation.
2. Video demos that highlight the functioning of the features.

List of Figures

Figure 2.1: Benetics, the startup.....	5
Figure 2.2: Benetics, project and task management and plan viewer.....	6
Figure 2.3: Benetics as communication tool.....	7
Figure 2.4: Benetics, some UIs.	7
Figure 2.5: Benetics, analytics with the web app.	8
Figure 2.6: Benetics, key features summary.	8
Figure 2.7: Benetics, addressable challenges and market [38].	9
Figure 2.8: Benetics, the vision.	9
Figure 2.9: Benetics, the competitors.	10
Figure 2.10: Benetics: business model and distribution.....	11
Figure 2.11: Benetics, the team.....	11
Figure 3.1: Geofencing	14
Figure 3.2: Background Geolocation Console for debugging purposes.....	17
Figure 5.1: Remote Personnel Presence Monitoring feature.	41
Figure 5.2: Time Cards overview calendar.....	44
Figure 5.3: Error message time card in the future.	44
Figure 5.4: Filter time cards on project.	45
Figure 5.5: Creation of Time Card.....	45
Figure 5.6: Geofencing suggestions for time card creation.	46
Figure 5.7: Update of Time Card.....	47

Figure 5.8: Time Cards overview graph..... 48

Figure 5.9: Interactive connection between calendar and graph..... 49

Figure 5.10: Time Cards section for managers on Project screen. 50

Figure 5.11: Period filtering for Time Cards..... 51

Figure 5.12: Category filtering for Time Cards. 51

Figure 5.13: Time Cards exported data. 52

Figure 6.1: Entities relationships. 54

Figure 7.1: Location permissions request..... 76

Figure 7.2: Physical activity permissions request..... 76

Figure 7.3: Background permissions request. 77

Acknowledgements

Ringrazio la mia famiglia per avermi sempre supportato e per essermi vicina ogni giorno.

Ringrazio il mio relatore Luciano Baresi e ringrazio i ragazzi di Benetics per avermi dato l'opportunità di lavorare con loro ad un progetto interessante.

Ringrazio mia nonna.

Ringrazio Charlie, la mia ragazza.

Ringrazio Ghera, mio motivatore, compagno ed amico di questi anni.

Ringrazio me stesso.

