Location-Based Alerts for Passengers in Public Transportation

Antonia Stieger

BACHELORARBEIT

eingereicht am

Fachhochschul-Bachelorstudiengang

Mobile Computing

in Hagenberg

im März 2025

Advisor:

FH-Prof. Dr.-Ing. Jens Krösche

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, March 15, 2025

Antonia Stieger

Contents

Abstract

This should be a 1-page (maximum) summary of your work in English.

Kurzfassung

An dieser Stelle steht eine Zusammenfassung der Arbeit, Umfang max. 1 Seite. ...

Chapter 1

Introduction

1.1 Motivation

In today's world, climate change is a pressing global issue, driving efforts to reduce carbon footprints. One effective strategy to mitigate environmental impacts is increasing public transportation use over private vehicles. Despite the clear environmental advantages, such as reduced fuel consumption and lower emissions, many commuters still favor private cars due to their convenience and flexibility. According to the United Nations [47], switching to public transport could reduce an individual's carbon footprint by up to 2.2 tons annually.

However, as Zheng and Krol [42] note, widespread adoption will only happen if public transit becomes the "most convenient option for getting around." While service coverage and infrequent schedules are well-known barriers, another common issue is that passengers often miss their stop due to distractions. Whether listening to music, reading or using their smartphones, commuters can easily lose track of their location. To address this issue, this thesis explores the development of a location-based alert system that notifies users as they approach their destination. By helping passengers stay aware of their stop, such a system could make public transportation more accessible and user-friendly, ultimately increasing rates of usage.

1.2 Challenges

One significant challenge to this goal is the strong competition from private vehicles, particularly in suburban and rural regions, where car ownership is widespread, and public transport struggles to provide a viable alternative. While urban areas often benefit from more comprehensive public transport networks, less populated regions face specific difficulties with limited routes, infrequent schedules and longer distances between stops. In these areas, missing a stop can lead to long waits for the next transport or even stranding passengers.

Beyond challenges related to public transport, several technological challenges arise in developing an effective alert system. Apple's app development guidelines impose strict restrictions on background activity, limiting how and when alerts can be triggered. Geofencing requires continuous location tracking, which impacts battery life and may

raise privacy concerns, as noted by Shevchenko and Reips [14]. Additionally, defining an appropriate geofence radius is important, as a poorly chosen size can cause inaccuracy or latency. Some devices may lack real-time location capabilities, meaning geofencing might not work reliably or at all for those users.

1.3 Goals

This thesis aimes to develop a user-focused app-based alert system in iOS that integrates schedule data and geofencing. Passengers are provided with a public transport app that will notify them along their journey when their destination approaches. This approach, particularly valuable in rural areas, ensures that regular commuters as well as tourists, children, and the elderly can navigate public transportation networks with ease.

The app offers users the flexibility to choose from three distinct alert modes based on their preferences. The first mode sends alerts according to the scheduled arrival times of transports. The second uses three geofences along the route to progressively notify passengers as they near their desired stop. The third relies on nested geofences around the final destination to trigger alerts. Additionally, the user will be alerted with three escalating levels of urgency, ensuring passengers are notified effectively. Furthermore, the app aims develop a test feature for wizard-based design that uses voice control for setting up reminders.

1.4 Structure

Before diving into the implementation, Chapter 2 introduces the core technologies relevant to this project, focusing on positioning methods and the systems that implement them. The chapter also discusses alerts on smartphones and their restrictions within iOS. Chapter 3 analyzes existing public transportation apps and how they inform users of approaching destinations. Chapter 4 presents the app concept, detailing the prototype design, proposed reminder behavior and data persistence strategy. Chapter 4 discusses the implementation in iOS, covering system architecture, geofence management, reminder triggering and voice-controlled alert setup. This thesis closes with Chapter 5, which describes what has been achieved and outlines potential next steps.

Chapter 2

Fundamentals

This chapter provides an overview of the core concepts relevant to the thesis. It begins with a review of various positioning methods and systems for both indoor and outdoor environments. Next, it examines geofencing technology, exploring its use cases and limitations. Finally, the chapter discusses how smartphones handle alerts, focusing on different types of notifications and the factors influencing them in iOS.

2.1 Positioning Methods

The term "positioning" refers to the ability to determine an object's location within a defined space. According to Küpper [7], positioning relies on the measurement of observables, which describe the spatial relationship between a target and its reference points. Depending on the positioning technology used, observables can include angles, ranges, range differences or velocity and they are measured by analyzing the properties of pilot signals. Pilot signals, such as radio, infrared or ultrasound signals, are reference signals transmitted from a known source and by identifying their origin, a fixed point with known coordinates, the position of the target can be calculated using positioning techniques. These techniques vary based on the type of observable measured and include proximity sensing, lateration, angulation, pattern matching and inertial navigation. The computed position is expressed relative to a chosen reference system. Küpper [7] states that this system could be descriptive such as identifying a specific cell in a grid, or geodetic where the position is represented as two- or three-dimensional coordinates based on WGS-84 or UTM. Further details on the different positioning techniques are provided in the following sections.

2.1.1 Proximity Sensing

Proximity sensing, as described by Küpper [7], is the simplest positioning method. It leverages the limited coverage range of pilot signals to detect the presence or absence of a terminal within a specific area. In this context, a terminal refers to a device or object whose position is being determined, such as a mobile or Internet of Things (IoT) device. The nearest base station to the terminal is identified based on Received Signal Strengths (RSS) and since pilot signals have a limited range, the terminal's position is assumed to match the location of this base station. Militaru et al. [9] illustrate this

concept in Fig. 2.1 where the terminal (UE) is surrounded by three base stations BS_1, BS_2 and BS_3. The terminal receives signals from all three base stations, but BS_1 has the strongest RSS, indicating it is the closest. As a result, BS_1's coordinates are assigned as the position of the terminal.

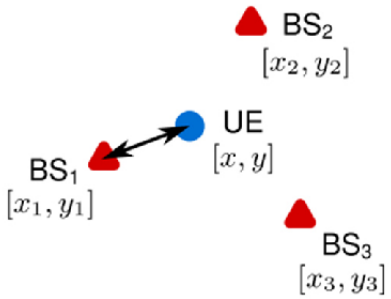


Figure 2.1: Proximity sensing with BS_1 as the nearest base station [9]

In the words of Küpper [7], proximity sensing in cellular networks is often referred to as Cell of Origin (CoO), Cell Global Identity or Cell-ID. A "cell" is a geographic area within the cellular network, managed by a base station that serves as a local signal hub for devices within that area. The accuracy of the location determined through CoO depends largely on the size and shape of the cell, as highlighted by Grejner-Brzezinska and Kealy [4] and Lee et al. [8]. Smaller cells allow for better location estimates, often reaching accuracy within 100 meters in urban areas where cell towers are densely placed. In rural areas, where cell coverage spans several kilometers, the location estimate becomes less accurate.

CoO is considered an inexpensive solution, as noted by Grejner-Brzezinska and Kealy [4] and Küpper [7], because it is compatible with the existing infrastructure. A terminal can determine its location using CoO either through an active connection to a base station or by passively receiving broadcast signals while in an idle state. When actively connected, the terminal's location is identified using the coordinates of the serving base station. In idle mode, the terminal can either query a remote database to retrieve the base station's location using its cell ID or rely on the base station to include its location directly in the broadcast signal, reducing the need for external lookups.

2.1.2 Lateration

Lateration, the most widely used method for localization according to Lee et al. [8], determines a target's location by measuring distances or distance differences from multiple reference stations. These measurements are referred to as "pseudoranges" because they include errors that distort the true ranges. Common error sources in lateration include clock errors, atmospheric effects or multipath propagation. Lateration techniques are categorized into circular lateration, which relies on absolute distance measurements and hyperbolic lateration, which uses differences in these distances across stations.

Circular Lateration

Circular lateration uses distance measurements to multiple base stations to derive a location. Assuming the base stations are at the same elevation, knowing the distance between the target and a single base station places the target somewhere on a circle centered on that station. Introducing a second base station allows for two possible positions where the two circles intersect. Adding a third base station resolves this uncertainty, pinpointing the target's exact location at the single intersection point of all three circles, according to Küpper [7]. This process, known as trilateration, utilizes the Pythagorean theorem for calculations and is illustrated in Fig. 2.2 (a).

In three-dimensional space, each distance measurement defines a sphere around a base station. Kolodziej and Hjelm [6] note that with only three base stations, the target's position is narrowed down to two possible points where the spheres intersect. Typically, one of these points can be dismissed as implausible, such as a location in outer space. To eliminate any ambiguity, a fourth base station is introduced as can be seen in Fig. 2.2 (b), ensuring a unique position fix. Additionally, since circular lateration determines position based on absolute distances, the fourth base station is necessary to synchronize clocks and correct for clock offset errors between the base stations and the target. Any clock offset would introduce range errors leading to incorrect positioning, which makes circular lateration more demanding in terms of synchronization requirements.

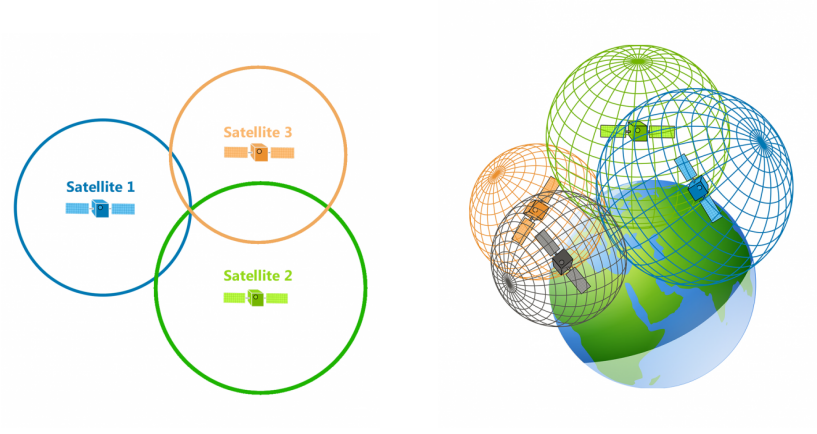


Figure 2.2: Circular lateration with three (a) and four (b) base stations (satellites) [23]

Hyperbolic Lateration

In contrast to circular lateration, hyperbolic lateration determines a position by measuring differences in distance rather than absolute distances. A hyperbola represents all points that maintain a constant range difference relative to two fixed points. Küpper [7] explains that with the known range difference between the target and two base stations, the target's possible locations are constrained along a hyperbolic path between them. This method, illustrated in Fig. 2.3 (a), uses two base stations to determine a single

hyperbolic path. However, with just two base stations, the target's precise location cannot be unambiguously determined, as it could lie anywhere along the hyperbola. To resolve this ambiguity, a third base station is introduced, as shown in Fig. 2.3 (b). By adding this third base station, a second hyperbola is created and the target's position is estimated at the intersection of these two hyperbolas. In three-dimensional space, the principle extends to hyperboloids, requiring at least three base stations for an unambiguous position fix. The most important advantage of hyperbolic lateration, according to Werner [18], is that it only requires synchronization among the base stations' clocks, rather than between the stations and the target, because it relies on Time Difference of Arrival (TDoA) measurements. With TDoA, only the relative arrival times at the base stations matter, making the exact emission time from the target irrelevant, as it is the same for all signals and cancels out in the calculation.

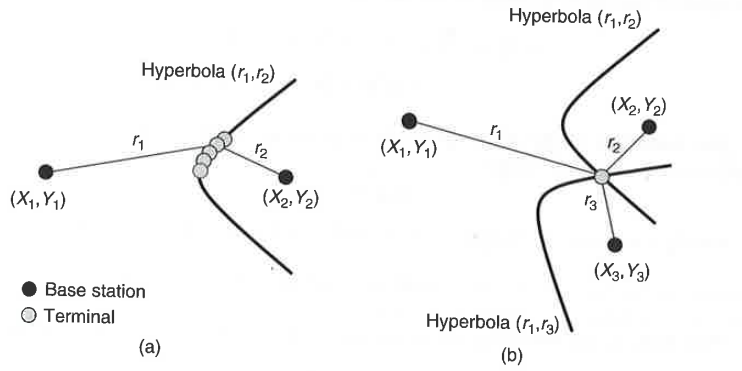


Figure 2.3: Hyperbolic lateration with two (a) and three (b) base stations [7]

2.1.3 Angulation

Angulation is a positioning technique that determines an object's location based on the angles at which signals are received from multiple reference points, rather than measuring distances. According to Küpper [7] it is also called Angle of Arrival (AoA) or Direction of Arrival (DoA). To determine the angle of incoming pilot signals, either the base station or the terminal must be equipped with antenna arrays. However, Küpper [7] highlights that angulation is predominantly used as a network-based method today, meaning that the arrays are more commonly installed at the base station rather than the terminal due to cost and complexity considerations. Once the angles from at least two base stations are known, the object's position can be determined by plotting lines along these angles, the point where the lines intersect reveals the target's location.

Technically, two angle measurements are sufficient to determine a position in 2D, but angulation is sensitive to errors caused by the resolution of antenna arrays, which can lead to approximations of the actual angle rather than precise measurements. These errors increase when the target is farther from the base station, making measurements less reliable at long distances. Additionally, in non-line-of-sight conditions, multipath propagation introduces further inaccuracies as signals reflect off obstacles and arrive

at the base station from unintended directions. To mitigate these errors, measurements from at least three base stations are recommended. Militaru et al. [9] illustrate in Fig. 2.4 how angle measurements from three base stations intersect to determine the terminal's position.

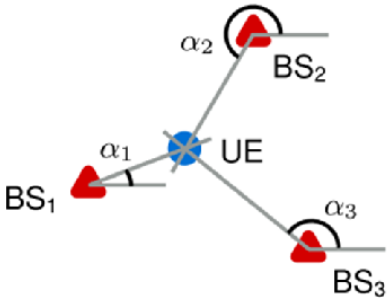


Figure 2.4: Angulation using three base stations [9]

2.1.4 Pattern Matching

Pattern matching is a method of finding a target's position by observing its surroundings and identifying patterns in the environment to determine where it is. Küpper [7] divides the matching of a target pattern with a known reference into two types: optical and nonoptical pattern matching.

Optical Pattern Matching

Optical pattern matching, also known as scene analysis, determines the position of a target by capturing and comparing images of a scene using a camera. This method can be categorized into two types: static and dynamic scene analysis, as described by Küpper [7]. In static scene analysis, a single image of the scene is compared to a database of pre-recorded images taken from various positions and angles. The target to be located could be the observer or another object within the scene. By matching the current image with the database, the target's position is identified. Dynamic scene analysis, on the other hand, determines the target's position by analyzing changes between consecutive images captured over time.

Nonoptical Pattern Matching

Nonoptical pattern matching identifies a target's position by analyzing measurable physical properties of the environment. When these properties are based on radio signal characteristics, the technique is commonly referred to as fingerprinting, as noted by Küpper [7] and Werner [18]. At the time of these publications, fingerprinting combined with Wireless Fidelity (Wi-Fi) was a widely used method for indoor positioning due to its inherent resilience to multipath propagation. Unlike lateration or angulation which suffer from signal reflections and interference, fingerprinting captures these effects within the

radio maps themselves using them as an important part for positioning. Furthermore, alternative technologies such as Ultra-Wideband (UWB) had not yet emerged and as a result, much of the focus on indoor positioning research was on Wi-Fi-based techniques due to the widespread deployment of Wi-Fi infrastructure.

Fingerprinting consists of two main phases: the offline phase and the online phase. In the offline phase, the area is divided into a grid and at each grid point, RSS from nearby base stations are recorded. These measurements create unique "fingerprints" that are stored in a database, each linked to its respective grid location. During the online phase, the terminal collects RSS values at its current location to form a sample vector. This vector is then transmitted to a server, which compares it with the database of fingerprints recorded during the offline phase. By identifying the closest match, the system estimates the target's position. However, a significant drawback of fingerprinting is the overhead of creating radio maps and maintaining them to account for changes in the signal landscape.

2.1.5 Inertial Navigation

Inertial navigation approximately calculates the current position by starting from a known location and using measurements of direction, velocity and time to compute the path traveled. This method represents a modern approach to dead reckoning, which was widely used by explorers during the Middle Ages. They navigated using tools such as compasses, log lines and estimates of speed and time, as detailed by Taylor [16]. Figure 2.5 illustrates this process of approximating position through dead reckoning.

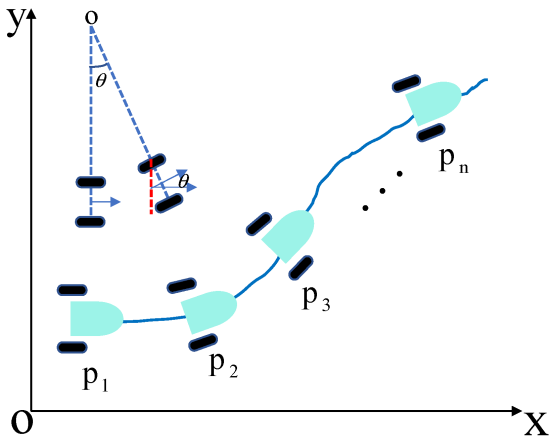


Figure 2.5: Principle of dead reckoning [17]

While inertial navigation shares these conceptual roots, it leverages modern inertial sensors to measure motion far more accurately. Unlike traditional dead reckoning which required frequent manual corrections, an Inertial Navigation System (INS) relies on data from an Inertial Measurement Unit (IMU), which contains accelerometers to measure linear acceleration and gyroscopes to track angular rotation. Inertial navigation errors increase over time due to sensor drift, so regular positions updates from the outside are needed. However, INS are covered in more detail in Subection 2.2.4.

2.1.6 Overview of Positioning Methods

The previous sections have introduced a range of fundamental positioning methods and to provide an overview, Table 2.1 summarizes the key characteristics of the methods. The table is based on the works of Küpper [7] and Werner [18] and outlines the specific features each method observes and the tools as well as techniques used for measurement.

Table 2.1: Positioning methods in comparison

Method	Observable	Measurement
Proximity Sensing	Physical proximity to a reference	Sensing pilot signals
Lateration	Distance and distance difference to reference point	Travel time and RSS of pilot signals (or their differences)
Angulation	Angle to reference point	Antenna arrays
Pattern Matching	Comparison to reference patterns	Camera and RSS
Inertial Navigation	Acceleration, angular rotation and time from a reference	Accelerometers, gyroscopes and additional sensors depending on the infrastructure

2.2 Positioning Systems

As highlighted by Küpper [7], a target cannot determine its location on its own. Instead, it relies on a distributed infrastructure that implements positioning methods to compute its position. This infrastructure typically consists of components like base stations, which are fixed points with known coordinates, and terminals, whose positions are initially unknown. Depending on the type of system, satellite, cellular or indoor, the base stations may include satellites, cellular towers, Wireless Local Area Network (WLAN) access points or tag readers. Terminals can range from mobile phones and laptops to vehicles and Radio Frequency Identification (RFID) tags, as illustrated in Figure 2.6.

In most cases, base stations either assist terminals in performing measurements or perform the measurements themselves. Additional components, such as databases and control units, may also be required for managing, processing and distributing positioning data. This section provides an overview of positioning systems, including Cellular Positioning System (CPS), Global Navigation Satellite System (GNSS), Wi-Fi and Bluetooth positioning systems, as well as INS, all of which rely on the positioning methods

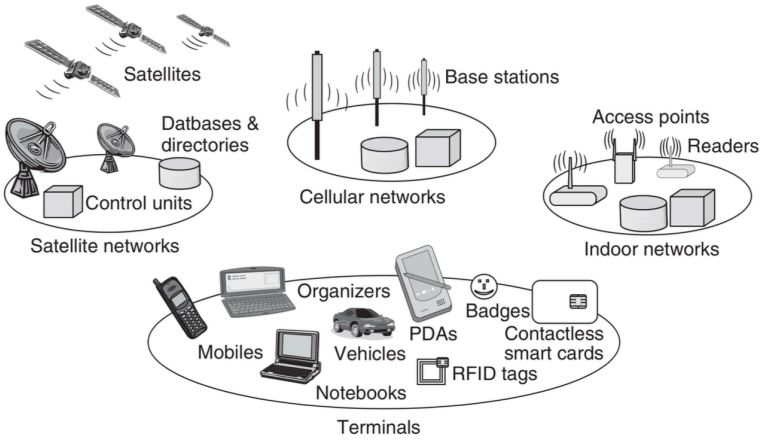


Figure 2.6: Infrastructures used in positioning [7]

described in Subsection 2.1.

2.2.1 Cellular Positioning Systems (CPS)

A CPS uses the existing infrastructure of cellular mobile phone networks to estimate the location of a device by analyzing signals exchanged between terminals and base stations. Küpper [7] highlights that the first generation of Location-Based Services (LBS) mainly used proximity-based positioning methods, such as CoO, because they were simple to implement and required minimal changes to the existing network infrastructure. Another benefit of CoO is that it is network-based, so it doesn't need any special functionality on the mobile devices, meaning it works even on legacy terminals. However, it soon became clear that basic cell-based positioning was not accurate enough to meet the demands of many LBS. In 1996, the Federal Communications Commission (FCC) issued a mandate, known as Enhanced 911 (E-911), requiring mobile operators to determine emergency callers' locations and send them to Public Safety Answering Points (PSAPs). Küpper [7] further points out that this pushed the development of more accurate, lateration-based positioning methods into cellular networks.

Cellular networks were not originally designed for positioning and their effectiveness in determining location is constrained by factors such as cell size, signal propagation characteristics and network density. These challenges were particularly evident in earlier generations, where the cells were larger and infrastructure was sparse, especially in rural areas. However, as mobile networks evolved, the shift to higher frequency bands necessitated the deployment of smaller cells and denser infrastructure to maintain effective coverage, as noted by Mushiba [10]. Despite these advancements, cellular positioning is still often used as a complementary approach to GNSS, providing coverage where GNSS signals are unavailable or unreliable, such as indoors or in urban canyons.

2.2.2 Global Navigation Satellite Systems (GNSS)

A GNSS is a network of satellites providing real-time positioning and timing data to users worldwide and offering higher accuracy compared to cellular approaches. The most widely used system is the Global Positioning System (GPS) and was developed by the United States Department of Defense (DoD) in the 1970s. Küpper [7] states that GPS was initially intended for military purposes before being opened for civilian use and gaining mass adoption as low-cost GPS receivers became accessible in the 1990s. Other GNSS systems include GLONASS (Russia), Galileo (European Union) and BeiDou (China). These positioning systems consist of constellations of satellites orbiting Earth, which transmit signals to receivers that calculate their position using circular lateration and time measurements.

The constellation of GPS consists of 24 satellites distributed across six orbital planes with four satellites per orbit, each spaced 60 degrees apart. This ensures that at least four satellites are always within the receiver's line of sight from any location on Earth, according to Winters et al. [19]. Three satellites are used to calculate the receiver's 3D position, while the fourth is necessary for time synchronization between the receiver and the satellites. Since GPS applies Time of Arrival (ToA), which is based on circular lateration, accurate timing is important. Satellites are equipped with highly accurate atomic clocks, whereas receivers use lower-cost quartz-crystal clocks, which tend to drift over time, as highlighted by Küpper [7]. To compensate for this drift, the receiver treats the time offset as an unknown variable and incorporates it into its position calculations. This method assumes that the satellites are perfectly synchronized, allowing the receiver to adjust its own time to match them.

However, since satellite signals are relatively weak and struggle to penetrate walls and ceilings, a receiver must have a clear view of the sky to detect and process them. Urban canyons, mountain valleys, tunnels and dense forests can create radio shadows that block satellite signals as per Zhang et al. [20]. Furthermore, as Enge [2] and Ruegamer et al. [12] highlight, GNSS is vulnerable to jamming, where interference blocks satellite signals, and spoofing, where fake signals deceive receivers into showing incorrect positions or times.

2.2.3 Wi-Fi and Bluetooth Positioning Systems

While outdoor positioning systems such as GNSS work reliably in open environments, they struggle indoors due to the obstruction of signals caused by solid objects like walls. Hence, Indoor Positioning Systems (IPS) have been developed to provide locating and navigation capabilities within buildings using technologies like Wi-Fi and Bluetooth, both of which operate in the 2.4 GHz frequency band.

Wi-Fi-based positioning is widely used due to the ubiquity of Access Points (APs) in indoor spaces which therefore suggests that positioning comes at no extra cost. According to Küpper [7], Wi-Fi positioning relies on measuring the RSS, Signal-to-Noise Ratio (SNR) or proximity to beacons, which can either be passively received from access points in the downlink or actively transmitted by devices in the uplink. The two primary methods used in Wi-Fi positioning are lateration and fingerprinting which are further described in Subsections 2.1.2 and 2.1.4. Unlike optical technologies such as infrared, Wi-Fi does not require a direct line of sight which makes it scalable. Wi-Fi signals can

also cover relatively long ranges of 50 to 100 meters, reducing the number of APs needed for coverage in large indoor spaces such as shopping malls or airports. Nonetheless, they are highly susceptible to multipath caused by obstacles like walls or structures and interferences from devices operating on the same frequency, which poses challenges for lateration as discussed by Grejner-Brzezinska and Kealy [4]. Fingerprinting, however, requires an extensive initial setup to map the environment along with periodic updates to account for changes in the signal landscape. Additionally, privacy concerns arise as users frequently connect to Wi-Fi networks which raises potential issues regarding tracking and data collection.

Bluetooth is a short-range wireless standard that typically determines a target's general location or proximity rather than absolute positions. Bluetooth Low Energy (BLE) uses beacons that continuously broadcast signals for nearby devices to receive or sensors, which detect transmissions from BLE devices. When only one beacon is available, the device's position is approximated by assuming the location of the detected beacon. If multiple beacons are detected, the position can be calculated using Received Signal Strength Indicator (RSSI) measurements and lateration which calculates distances from signal strength measurements, or AoA which determines the signal's direction. In addition, Bluetooth devices can form ad hoc networks, where they connect and exchange location information which allows them to detect their proximity to each other. However, as stated by Kolodziej and Hjelm [6], to connect the network to the physical environment, at least one terminal must have a known absolute position.

2.2.4 Inertial Navigation Systems (INS)

The term "dead reckoning" comes from "deduced reckoning" which refers to estimating the current position by extrapolating from a previously known location which is adjusted by the observed direction of movement and velocity over time, as described by Grejner-Brzezinska and Kealy [4]. As discussed in Subsection 2.1.5, the core of any INS is the IMU that combines inertial sensors like accelerometers, gyroscopes and sometimes magnetometers and barometers depending on the use case. Odometers are separate sensors but can be integrated into navigation systems alongside an IMU.

Accelerometers measure linear acceleration relative to their own inertial frame. They contain a proof mass attached to a spring and when acceleration on the vertical axis occurs, inertia causes the mass to displace. This displacement is used to calculate acceleration, according to Werner [18]. However, this principle faces two challenges: the influence of the earth's gravitational pull and the measurements being limited to a single axis. Accelerometers cannot distinguish between gravitational and motion-based acceleration which is why other sensors are needed. To resolve the second problem, three accelerometers are arranged orthogonally to measure acceleration along three axes. Grejner-Brzezinska and Kealy [4] explain that in static conditions, the axis with the highest acceleration value corresponds to the direction of gravity, typically toward the ground. However, during motion the highest acceleration may appear along a different axis.

Gyroscopes measure angular velocity and their data can be combined with accelerometer data to separate movement from gravity as highlighted by Werner [18]. They work based on the conservation of angular momentum, which means a spinning

object resists changes to its orientation. In a mechanical gyroscope, a rotating mass is mounted within a system of three concentric rings (gimbals) so it can rotate freely in all directions. As the device moves, the spinning mass maintains its original orientation relative to the surroundings and the rotation of the gimbals can be tracked which allows for the measurement of inertial rotational motion.

Magnetometers detect magnetic fields using Hall sensors which measure the voltage generated by the deflection of electrons when exposed to a magnetic field.

Barometers measure air pressure which can be used to estimate altitude changes or weather conditions based on the relationship between atmospheric pressure and height.

Odometers calculate the distance traveled by counting wheel rotations from a known starting point.

Once an initial position is established using methods like lateration or angulation, an INS operates independently and doesn't rely on external electromagnetic signals like radio signals. This self-contained nature makes inertial navigation highly robust against signal obstructions and manipulation, but they still are highly susceptible to errors. El-Sheimy [13] explains that position errors grow quadratically as a result of velocity errors that stem from biases or inaccuracies in accelerometer measurements during the initial integration process. He further points out that errors in gyroscopic data are even more impactful, as they first produce angular inaccuracies, which then propagate into velocity errors that grow quadratically and ultimately result in cubic error in position. These errors can accumulate over time due to the lack of external reference points. To mitigate this, inertial navigation is often only applied in short-term uses, such as to bridge periods when line-of-sight to satellites is obstructed or to refine position fixes determined by GNSS. By combining the last known location obtained via GNSS with measurements captured by an IMU chip, a vehicle's movement can still be tracked during GNSS outages, e.g. driving through a tunnel.

2.3 Geofencing

Shevchenko and Reips [14] define geofencing as the automated triggering of actions when "virtual boundaries around specific locations" are crossed. It works by defining a point on a map with latitude, longitude and a radius to form a circular boundary. When this boundary is entered or exited, an event can be triggered. iOS provides this feature through the Core Location framework, as stated on the Apple Developer Pages [32]. This framework mainly supports geofencing with circular regions, defined by a center point and radius, but does not natively support polygonal geofences. However, developers can implement polygonal geofencing through custom solutions or third-party libraries. For instance, the React Native Background Geolocation plugin enables polygonal geofencing by approximating polygons with a minimum enclosing circle, as documented by Transistor Software [45]. Over time, geofencing has become a valuable tool across various industries.

In marketing, businesses use geofencing to send location-based advertisements or discounts to customers near their stores. In workforce management, geofencing automates attendance tracking by allowing employees to clock in and out automatically upon entering or leaving designated work zones. In IoT, geofencing can enhance home automation. For example, smart devices can adjust lighting, heating or security systems when res-

idents enter or leave their homes. Similarly, Kadam et al. [5] explore how geofencing combined with IoT can transform agriculture by helping farmers protect crops from wild animal intrusions through real-time alerts. Geofencing is also improving safety in educational settings. Takyiwa-Debrah [15] highlights how GPS-enabled wearable devices help track children's locations to reduce the risk of abduction in schools. In logistics, geofencing ensures vehicles adhere to designated routes and enables businesses to monitor fleet locations, as stated by Reclus and Drouard [11]. In healthcare, it assists caregivers monitor dementia patients by setting up safe zones and alerting them when patients move beyond these boundaries to ensure the patients' safety, as noted by Arora and Deswal [1].

Despite its advantages, geofencing raises privacy concerns, as highlighted by Greenwald [3]. Since it relies on continuous location tracking, users' sensitive data may be vulnerable to misuse or security breaches. This is particularly concerning when monitored regions overlap with personal or sensitive areas, such as homes or healthcare facilities.

2.4 Alerts on Smartphones

Alert types, trigger conditions and restrictions define how and when users are notified of specific app events, ensuring that alerts provide value without disrupting user control. The following sections outline the types of alerts available, the mechanisms that trigger them and the specific guidelines imposed by Apple to maintain a responsible alert experience towards users.

2.4.1 Alert Types

Smartphones offer different ways to alert users, including visual, sound and vibration cues. These alerts help confirm actions, share important information and improve user experience. Notifications are one of the most common types of alerts and can be split into local and push notifications. According to Apple's documentation [36][37], local notifications are created by the app itself to inform users of events, even when the app runs in the background. Push notifications are sent from a server and are often used for messages or reminders and updates. Another type of alert is haptic feedback, which uses vibrations to notify users physically. In iOS, tools like the `UIImpactFeedbackGenerator` provide vibrations for various levels of physical impact such as light, medium or heavy taps. The `UISelectionFeedbackGenerator` signals changes in selection, like scrolling through a picker, while the UINotificationFeedbackGenerator conveys success, warning or error notifications to clarify important feedback, as noted by the Apple Dev-Pages [34]. Sound feedback is also available and is more noticeable, making it effective for capturing users' attention. According to Apple's guidelines [26], iOS sounds are divided into system sounds, used for standard actions like errors or confirmations and custom sounds, often used in games or specific app events to create unique auditory experiences. For visual feedback, Apple's `UIAlertController` [25] provides pop-up alerts with titles, messages and action buttons, often used for critical alerts or confirmations to ensure user attention. Toasts, which are brief messages that appear and disappear without user interaction, are another form of visual feedback. While not native to iOS, third-party frameworks

can enable toasts, offering a less intrusive way to share short information. Subtler visual cues, like progress indicators and badges, are also effective for showing ongoing tasks or highlighting unread notifications within app sections. With iOS 16, Apple introduced the Live Activities feature [31], which displays real-time updates on the lock screen or in the Dynamic Island. This feature keeps users informed about ongoing events, such as tracking a food delivery or monitoring live sports scores without needing to open the app.

2.4.2 Triggers

In iOS, alerts are not limited to user input, they can also be triggered by automated or event-based conditions. Apps, for example, can use scheduled timers to generate alerts or notifications at specific times for reminders, countdowns or periodic updates. This is often implemented using Swift's `Timer` class [38]. Alerts can also respond to app state changes, such as when an app transitions to the background, returns to the foreground or closes. For instance, an app may display a confirmation alert if the user attempts to leave a page with unsaved data or notify them of an important update. Device sensors like accelerometers and gyroscopes detect motion or orientation changes, which can also trigger alerts. Location-based triggers, which are central to this thesis, use geographical boundaries to deliver alerts when a user moves into or out of a specified region. According to Apple's Developer Pages [27, 28], the CoreLocation framework provides tools like `CLLocationManager` for managing these triggers, while `CoreMotion` supports motion event detection. Similarly, HealthKit [30] enables alerts based on health metrics, such as notifying users if their heart rate exceeds a specified threshold. To manage in-app notifications, Apple's `NotificationCenter` [33] class is frequently used to broadcast data changes, enabling observers to update alerts or badges in real time. For notifications, iOS provides three main trigger types. `UNTimeIntervalNotificationTrigger` [41] schedules alerts after a specified time interval. `UNCalendarNotificationTrigger` [39] triggers notifications at specific dates or times, with options for recurring alerts and `UNLocationNotificationTrigger` [40] sends alerts when a user enters or leaves predefined geographical areas.

2.4.3 Restrictions in iOS

When it comes to responsible app development, Apple has many guidelines to maintain user control and prevent disruptive app behavior. Generally, apps are restricted from triggering alarms or alerts when not actively running in the foreground, with exceptions for specific cases like Voice over IP, music streaming and location-tracking apps. These applications can run background audio, but this capability is not intended for alarms and comes with strict guidelines to prevent misuse. For instance, while a music app may play in the background, an alarm app that attempts to exploit this mode to ring an alarm would be rejected from the App Store. Additionally, iOS does not provide any public API for third-party apps to set, modify or access alarms in the native Clock app, which is entirely separate and inaccessible for tasks like scheduling alarms or timers. When delivering notifications, apps must obtain user permission. Users can choose to allow, mute or disable notifications from each app, and Apple discourages excessive notifications, especially those lacking immediate or relevant value. Apps that

send spammy notifications risk penalties or even removal from the App Store. Custom sounds for local notifications are limited to 30 seconds, if a sound exceeds this limit, iOS defaults to the standard notification sound, as discussed on the Apple Developer website [26]. Local notifications also respect Silent Mode and Do Not Disturb settings, meaning sounds will not play if these modes are active. According to Apple's documentation [29], critical alerts are allowed to bypass these settings, but this permission is reserved for essential use cases, such as health monitoring or emergency alerts, not for general alarms or reminders. Location-based triggers, such as geofencing, also require explicit user permission. iOS [35] limits the frequency of location-based notifications and may throttle frequent triggers. Excessive location tracking can lead to App Store rejection unless it is essential to the app's functionality. Additionally, Apple states on its Dev Pages [32] that the use of geofencing is limited to a maximum of 20 active geofences per app to allow all apps to access condition monitoring and prevent excessive battery drain or performance issues, as these features rely on shared hardware resources.

2.5 Conclusion

This chapter covered the fundamental concepts necessary for developing a location-based reminder system. Since the system relies on iOS's Core Location framework, understanding how positioning works is important. However, Apple does not disclose the exact mechanisms behind how location data is determined, only stating on the Developer Website [27] that a combination of GPS, Wi-Fi, Bluetooth, inertial sensors and cellular data is used. For this reason, this chapter examined the basic positioning methods, along with the positioning systems that implement them.

Geofencing was explored as the primary method for triggering location-based reminders, highlighting both its functionality and limitations such as iOS's restriction of 20 simultaneously monitored geofences per app. This constraint directly impacts how many reminders can be actively tracked at a time. While this app also includes a time-based reminder mode, this approach is fundamentally different from geofencing. Unlike location-based triggers, time-based reminders are straightforward to implement, whereas geofencing requires deeper background knowledge due to its reliance on continuous location monitoring. Additionally, an overview of smartphone alerts was provided to establish how reminders are delivered to users and the restrictions imposed by iOS.

Chapter 3

Related Work

Public transportation navigation apps help commuters plan, track and complete their journeys more efficiently. While many apps provide route tracking, the challenge of knowing exactly when to get off at the right stop persists for passengers. Without clear alerts, passengers must constantly monitor their route which can be inconvenient or impractical, especially in unfamiliar areas. This chapter examines three popular transit apps and their approaches to notifying users when to exit.

3.1 Google Maps

Google Maps, developed by Google, is a mapping service that offers route planning across different transportation modes including driving, walking, cycling and public transit. The platform partners with numerous public transportation providers worldwide to integrate their data while also collecting independent information to facilitate trip planning for users. Google Maps provides real-time transportation information that allows users to view live arrivals for buses, metros and subway systems and alerts them to canceled routes.

In 2017, Google introduced a feature for Android that sent users notifications when to transfer or exit their bus or train. Shortly after, various online sources reported the feature's availability, and in the 2017 version of the Google Maps app, it could be enabled after selecting a public transit route. As noted by The Verge [24], users could toggle on reminders for departure and transfers under the "Add a Google Maps reminder" section. However, in the 2025 version of the app, only the "Remind you to leave on time" toggle remains, with no trace of the stop alert feature in the settings. This suggests that Google Maps has discontinued the feature.

According to 9to5Google [21], Google Maps began testing the Live Activities feature on iPhones in 2023, enabling users to receive navigation updates directly on their lock screens and in the Dynamic Island. While helpful for navigation, this feature does not include explicit alerts for when to get off a bus or train. Additionally, Live Activities is not yet available globally and its rollout has been limited. Some users have reported seeing it appear intermittently, but Google has not provided an official method to manually enable it. When activated automatically, it appears as shown in Figure 3.1 (right).

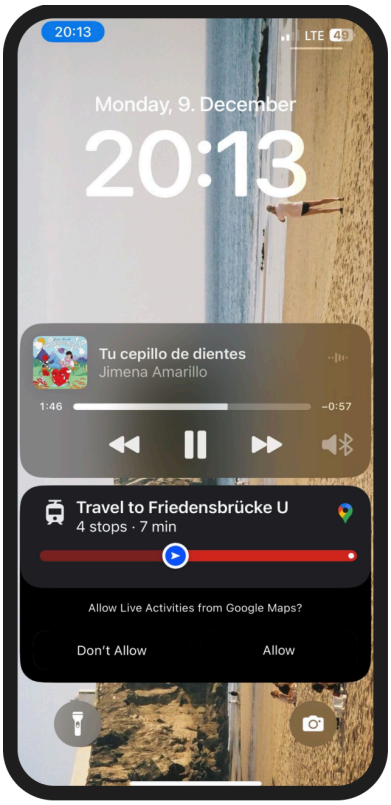


Figure 3.1: Google Maps' Live Activities feature on iOS

3.2 Citymapper

Citymapper was founded in 2011 by former Google employee Azmat Yusuf, who aimed to simplify navigation for London's public transport system, as highlighted by TechRound [46]. Initially launched as Busmapper, the app focused on London's bus network but later expanded to include the underground, trams and other transport modes, eventually becoming Citymapper. As of 2025, the platform has over 50 million users worldwide and operates in over 100 cities worldwide including New York, Paris, Tokyo, Vienna and Sydney, according to their website [22]. Additionally, it supports entire regions like Scotland, the Basque Country and the Balearic Islands.

The app suggests the fastest, most convenient public transport routes while considering delays and disruptions and offers users the possibility to receive notifications for service alerts. After selecting a public transportation route, users can activate the Live Activities feature and receive notifications reminding them when to get off a bus or train. Citymapper refers to these notifications as "Get Off Alerts", as stated on their website [44], and users receive three notifications at different points along their journey for better awareness. However, testing revealed that Citymapper frequently lost location signal even in open-sky conditions, leading to missed notifications. Additionally, while Get Off Alerts notify users when to exit, Citymapper does not provide an actual alarm, meaning users must rely solely on notifications that could be missed if the device is on

silent mode or if location tracking fails. Citymapper's Live Activities feature is displayed in Figure 3.2 (left) and a stop notification is shown in Figure 3.2 (right).

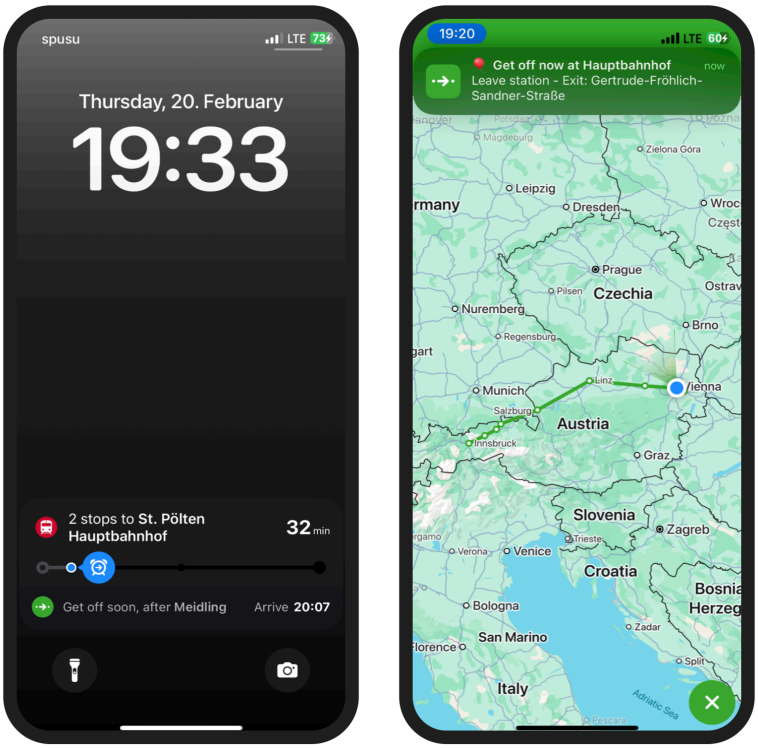


Figure 3.2: Citymapper's Live Activities (left) and Get Off Alerts (right) on iOS

3.3 Moovit

Moovit is a public transportation navigation app launched in 2012 for iOS, Android and web browsers. It aims to help users navigate cities by integrating multiple forms of transport including public transit, shared bikes, ride-hailing services and scooters into a single application. As of 2025, Moovit serves over 1.7 billion users in 3,500 cities across 112 countries and supports 45 languages, according to their website [43].

When a user selects a public transit route, Moovit provides Live Activities and notifications to alert them when to get off a bus or train. Three notifications are sent at different points along the journey, with increasingly urgent messages to ensure the user is informed in time. During testing, Moovit maintained a stable location signal on the same routes where Citymapper lost connection and effectively communicated the approaching destination both through Live Activities and notifications, as shown in Figure 3.3. However, Moovit also does not provide an audible alarm.

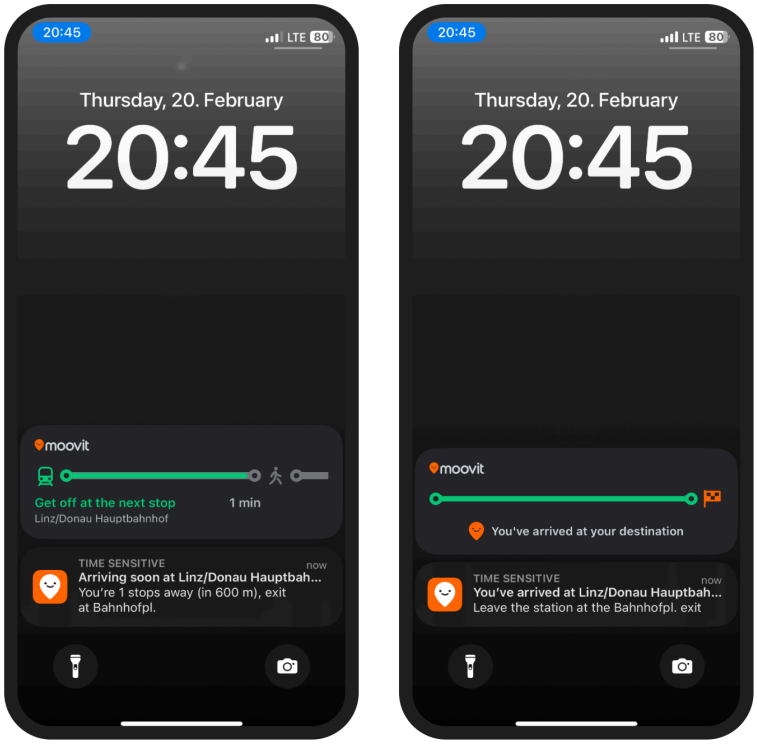


Figure 3.3: Moovit's Live Activities and stop notifications on iOS

3.4 Conclusion

The reviewed apps all provide route planning and tracking but differ in how they notify users when to exit public transportation. Google Maps offers Live Activities on iPhones for navigation updates on the lock screen, though this feature is not globally available. While stop notifications were reportedly available in 2017, they can no longer be found in the 2025 version of the app. Citymapper includes stop notifications and Live Activities but frequently lost location tracking, resulting in missed alerts. Moovit reliably provides stop notifications and Live Activities, making it the most effective at alerting users of their approaching destination. However, none of these apps offer vibrations or alarms in addition to stop notifications, as proposed in this thesis.

Chapter 4

Concept

This chapter presents the conceptual approach for addressing the previously described challenges in public transportation. It considers how a mobile application could be designed to assist users in reaching their destinations without missing their stops.

A sketched prototype serves as a reference for illustrating how such an application might be structured and what functionalities could be incorporated. In particular, the concept of reminder behavior is examined, exploring different triggering mechanisms and the potential use of visual, haptic and acoustic feedback to effectively notify users. To enable these functionalities, the system would likely require the ability to store and manage essential data, such as user preferences and reminder settings. The possible structure and handling of this data are considered in this chapter.

4.1 Prototype

The proposed system is a mobile application, likely targeting iOS, that aims to allow commuters to plan their public transport journeys and receive reminders for their selected routes. The idea is that users should be able to search for public transport connections by specifying a departure and arrival stop, along with the desired date and time of travel. Based on these inputs, the system would provide a selection of available routes, from which users can choose the most suitable option. Once a route is selected, a reminder mechanism is intended to ensure that users do not miss their desired destination stop. The idea of how such a reminder system could work is explored in more detail in Section 4.2.

The main screen of the proposed application is envisioned as the central interface for the trip planning process. It is structured around a selection section where the user can specify their journey details. At the top of this screen, two fields represent the departure and destination stops, displayed in an “A to B” format. Below these fields, a date and time selection is displayed with the default value set to “Depart Now”, indicating immediate departure. Beneath this section, a search button is prominently placed, allowing users to request available public transport connections for the specified parameters. When a user taps on either the departure or destination field, a dropdown menu is envisioned to appear, providing a list of selectable public transport stops. The exact filtering process for these stop suggestions is not further defined in this concept.

However, it is anticipated that an open Application Programming Interface (API) for public transport data endpoints would be required to dynamically populate the list with available stops.

If users wish to modify their departure time, tapping on the “Depart Now” button is expected to open a selection screen that slides up from the bottom of the screen, allowing them to specify a custom date and time. Once all trip planning parameters have been set, the system should search for available public transport routes matching the user’s request. The results screen displays the retrieved connections in a structured format. Each entry should present departure and arrival times, the transport service’s identification (such as the line number or train designation) and the duration of the trip. From this list, users should be able to select a preferred route, after which the reminder system will be activated for their journey. The sketched prototype in Figure 4.1 provides a visual representation of this envisioned process.

Figure 4.1: Prototype of a reminder app for public transportation

4.2 Reminder Behaviour

Once a user selects a public transport connection, the app should provide a reminder mechanism of some sort. The proposed concept aims to prioritize user control over how and when reminders are triggered, ensuring they are noticed even if the user is asleep or distracted. The reminder behavior can be divided into escalation and triggers.

Escalation refers to how reminder intensity increases as the journey progresses. This could involve a gradual shift from subtle visual cues to more intrusive haptic or acoustic signals. Triggers determine when these reminders activate. Different approaches can be considered, relying either on arrival times or the user's proximity to their destination. The following subsections explore these aspects in detail.

4.2.1 Escalation

Unlike conventional transit apps discussed in Chapter 3, which primarily provide route and schedule information and, at best, basic notifications or live activity updates, this approach emphasizes ensuring that users receive reminders in a way that effectively captures their attention. To make sure that alerts are reliably noticed, different types of feedback mechanisms can be considered, including visual, haptic and acoustic signals. A visual alert could appear as a notification, while a haptic signal might involve the phone vibrating. An acoustic signal could take the form of a loud and disruptive alarm, serving as a last-resort alert to ensure the user's awareness. These feedback methods may be used individually or in combination depending on the user's preference.

The way reminders escalate can follow different design approaches. Is a sequential model, reminders activate at multiple predefined points throughout the journey. This could involve placing triggers at specific stops before the destination or by progressively increasing alert intensity as the user nears arrival based off of fixed arrival times. Alternatively, a concentric structure can be used, where nested zones trigger increasingly urgent alerts as the user nears their stop. In this case, the triggering would be based on proximity to the destination, requiring a system capable of estimating the user's position along the route. These two approaches are illustrated in Figure 4.2 where (a) represents the sequential model and (b) the concentric model.

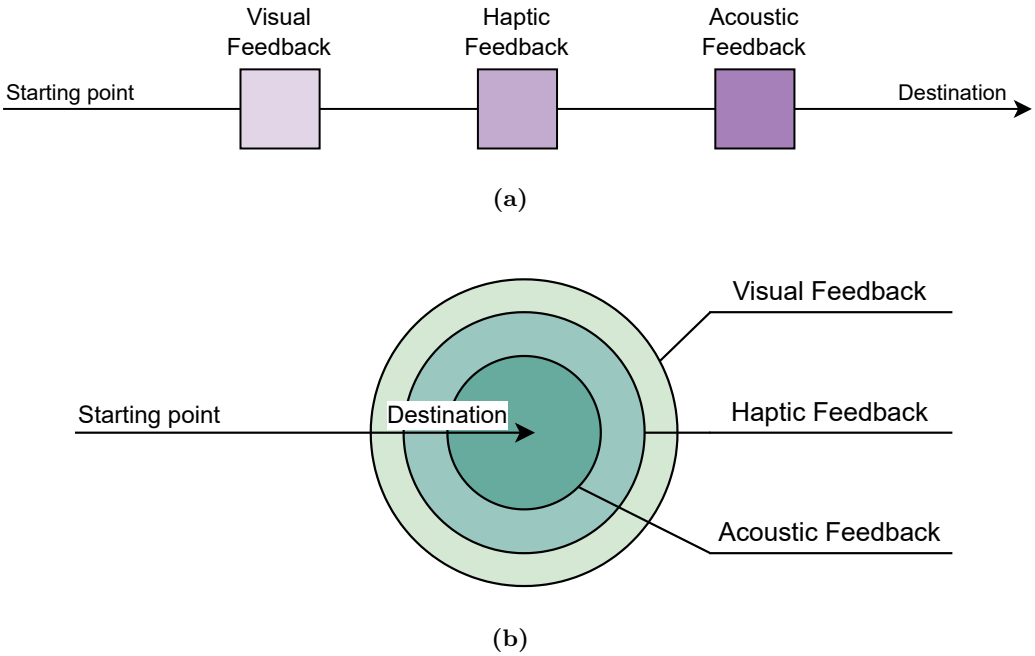


Figure 4.2: Escalation of reminders following a sequential model (a) and a concentric model (b)

4.2.2 Triggers

Beyond how reminders are structured, it must also be considered how they could be triggered in the first place. The timing of these reminders plays an important role in ensuring that users are notified at the right moment during their journey. Different approaches could be explored, depending on whether reminders should be based on time, location or distance to the destination.

One possible approach would be to rely on fixed arrival times, where reminders could be triggered at predefined intervals before reaching the stop. This would ensure that users receive alerts at specific time points, regardless of external factors such as travel speed or delays. Another option could be to determine triggers based on the user's location along the route, so that reminders are activated as they move closer to their destination. Unlike the time-based approach, this method would not depend on scheduled arrival times but rather on the user's actual position relative to their stop. These two approaches might follow a sequential model as discussed in Subsection 4.2.1, activating reminders at distinct points along the journey. Alternatively, a concentric approach could be considered, where reminders are triggered based on distance to the destination, where different zones around the destination are used to trigger reminders at increasing proximity. In this context, it could be considered whether geofencing might be used to allow the system to detect when the user crosses predefined distance markers.

4.3 Data Persistence

It is being considered that the app will require some form of data persistence to store user data beyond the app's runtime. The system envisions two main types of persistent data: user preferences and reminder objects. User preferences define how reminders behave, whereas reminder objects should store trip-related details. While the concept foresees that users can configure preferences for each reminder individually, there is also the idea that frequently used settings could be saved and easily reapplied. The following subsections outline these data structures in more detail.

4.3.1 User Preferences

For user preferences, it is being considered that the reminder behavior, whether time-based or location-based, could be represented through a mode variable that is then stored in a database. Additionally, an interval setting could define when the user receives reminders, depending on the selected mode. For example, if reminders are scheduled in a time-based manner, one possible approach could be to define the interval in minutes before arrival at the destination, such as triggering a reminder five minutes before. In the case of a location-based mode, the interval could instead be expressed in proximity, for instance 500 meters before reaching the destination. Furthermore, in alignment with the escalating reminder behavior, reminders could appear in different forms. These could be visual, haptic, acoustic, or a combination of them. However, if a user prefers only a non-intrusive visual cue such as a notification, the app would need to account for this preference to avoid unwanted alerts. All of these user preference settings would need to be stored to remain consistent beyond the app's runtime.

4.3.2 Reminders

When it comes to the reminder object itself, several parameters would need to be considered. The required information depends on whether the user prefers time-based or location-based reminders. For a time-based reminder, storing a specific time and date would be essential. For a location-based reminder, the system would need to store the latitude and longitude of the destination. This should allow the user's current position to be compared against the target location, ensuring the reminder is triggered at the right moment. Additionally, the names of the starting and destination stations should be stored for display in an overview of active reminders. The approach also aims to allow unscheduling of reminders, requiring each reminder object to have a unique identifier to locate the correct object and modify it.

Chapter 5

Implementation

This chapter presents the app's software basics with a focus on the User Interface (UI) framework and data persistence. It also outlines the implementation of geofencing and reminder behavior, concluding with a test feature designed to make the app more user-friendly.

5.1 Software Basics

The app is built using iOS technologies. SwiftUI was chosen as the framework for building the UI, while SwiftData is used for managing and persisting data across app sessions.

5.1.1 SwiftUI

SwiftUI is Apple's declarative UI framework, introduced in 2019, which allows developers to create user interfaces with less boilerplate code compared to UIKit. One of the advantages of SwiftUI is that UI updates occur automatically when the underlying data changes, instead of using view controllers and manual UI updates. This makes the framework particularly powerful when combined with SwiftData as the UI can react dynamically to stored data.

5.1.2 SwiftData

SwiftData simplifies data storage and retrieval in Swift applications, allowing developers to define models directly in Swift using macros instead of external files like Core Data. A `ModelContainer` manages persistence, determining whether data is stored locally or in iCloud. Within the app, it acts as a central storage hub for all models. To enable persistence for a specific model, the container is initialized as shown in Program 5.1.

```
1    let modelContainer: ModelContainer = ModelContainer(for: Reminder.self)
```

Program 5.1: Initializing the `ModelContainer` for SwiftData persistence

The `ModelContainer` stores the data models, but the actual operations such as inserting and deleting data are performed using a `ModelContext`. The `ModelContext` serves as the interface between the app's logic and the database, allowing modifications to stored data during runtime. Marking a class with the `@Model` macro signals to SwiftData that it should be persistently stored. At compile time, Swift automatically generates a schema and essential methods for managing data without adding runtime overhead. To fetch and track changes in stored data, the `@Query` macro can be used alongside `@Model`. It automatically retrieves data from the `ModelContainer` and updates the UI whenever changes occur without manual database queries. In the app, `ModelContext` is set as an environment value, and `@Query` dynamically accesses the array of reminder objects as illustrated in Program 5.2, which demonstrates deleting and inserting a reminder into the database.

```
1    @Environment(\.modelContext) private var modelContext
2    @Query var reminders: [Reminder]
3    ...
4    modelContext.delete(reminders[index])
5    modelcontext.insert(reminder)
```

Program 5.2: Using `ModelContext` for modifying persistent data and `@Query` for fetching data

In addition to data persistence with SwiftData, the app also incorporates `AppStorage` for lightweight preference storage. `@AppStorage` is a SwiftUI property wrapper built on top of `UserDefaults` and is commonly used for saving app settings or user preferences to persist across app launches and any updates trigger a UI refresh, ensuring the latest data is always displayed.

```
1    @AppStorage("ReminderMode") static var reminderMode: ReminderMode = .distance
2    @AppStorage("TimeInterval") static var timeInterval: TimeInterval = ._7min
3    @AppStorage("StationInterval") static var stationInterval: StationInterval = .
     _4stations
4    @AppStorage("DistanceInterval") static var distanceInterval: DistanceInterval =
     ._1km
5    @AppStorage("Vibration") static var vibration: Bool = true
6    @AppStorage("Alarm") static var alarm: Bool = true
```

Program 5.3: Using `@AppStorage` to persist user preferences in the app

5.2 User Interface

The app's UI consists of three main views, accessible via a tab bar at the bottom. In addition to these primary views, the app includes several secondary views. Figure 5.1 presents a diagram illustrating the hierarchical structure of these views within the app. The following sections provide an overview of each view and explain the user journey through the application.

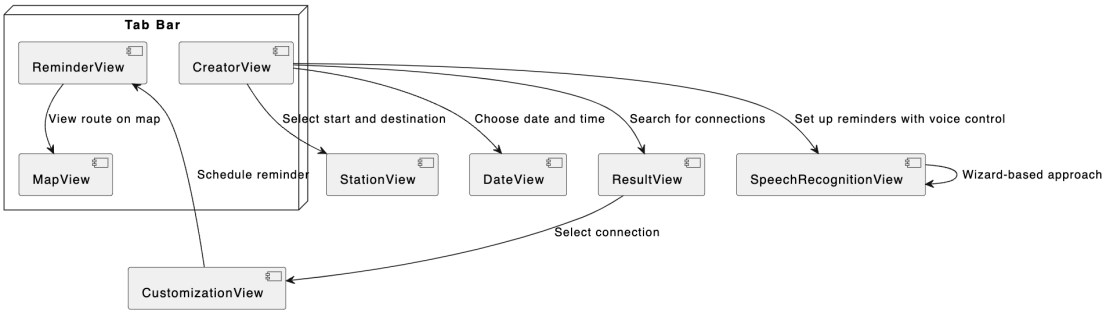
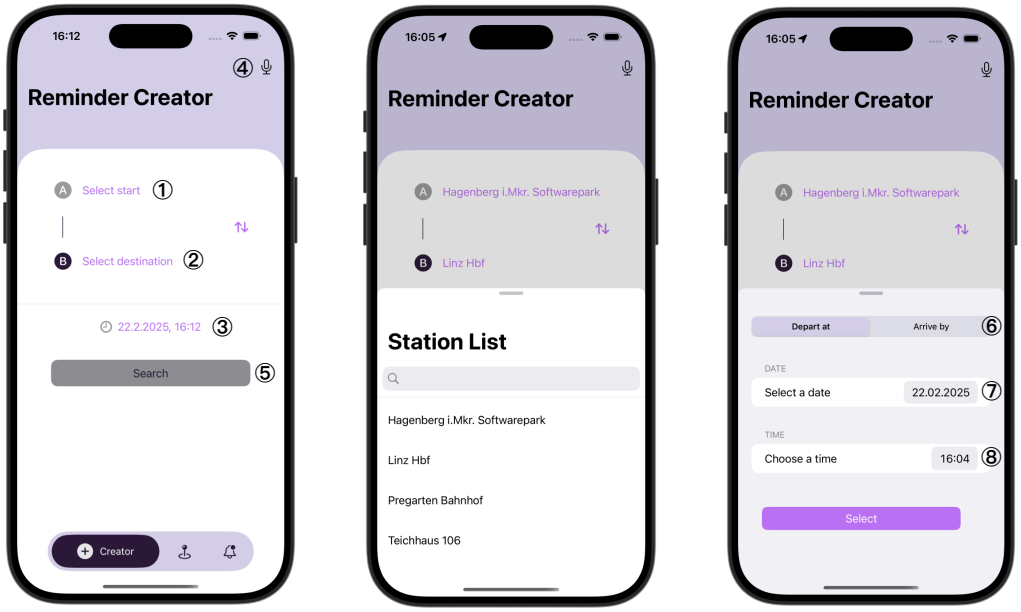


Figure 5.1: Diagram illustrating the hierarchical structure of main and secondary views

5.2.1 CreatorView

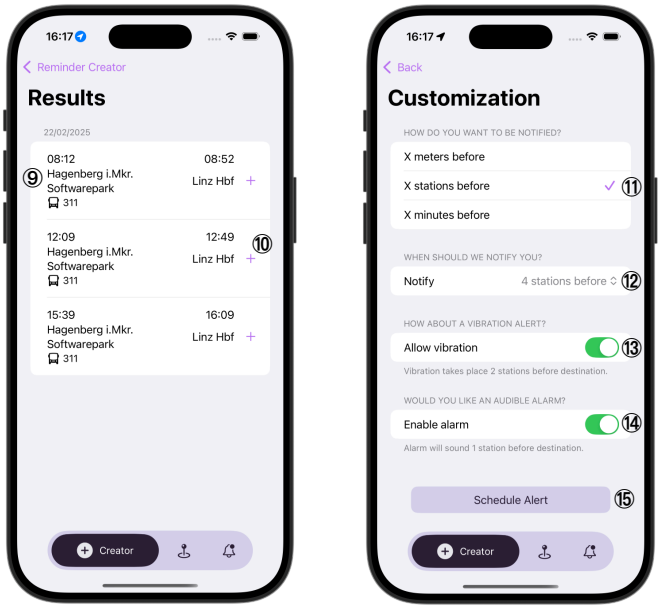
As introduced in Chapter 4, the `CreatorView` serves as the app's entry point, guiding users through the trip planning process. This view is illustrated in Figure 5.2 (a), where users can select their start location ① and destination ② by tapping the corresponding buttons. This action opens `StationView` in a sheet from the bottom, where users can search for a station and select it as seen in Figure 5.2 (b). Once a station is chosen, its name replaces the button label. Below the station selection, the interface provides a date and time selector ③ which defaults to immediate departure. Tapping it opens `DateView` in a sheet as depicted in Figure 5.2 (c), where users can choose between departure or arrival time ⑥ and set the date ⑦ and time ⑧. Additionally, a microphone icon button ④ in Figure 5.2 (a) provides access to the `SpeechRecognitionView`, which offers a wizard-based approach for setting up reminders through AI-assisted voice processing. This test feature is discussed in detail in Subsection 5.2.4.

The search button ⑤ in Figure 5.2 (a) allows users to retrieve simulated public transport connections based on the selected parameters. Upon pressing the button, the app navigates to the `ResultView` in Figure 5.3 (a), which displays details such as departure and arrival times, station names and line ID as seen in ⑨. Since no suitable open API for public transport data could be found, the displayed connections are simulated. Each entry includes a plus button ⑩ and pressing this button signals the app to navigate to the `CustomizationView`, where users can configure their reminder preferences. As depicted in Figure 5.3 (b), this view provides options to select a reminder mode ⑪ and set the corresponding interval ⑫. Users can also choose additional alert settings, such as vibration ⑬ and an audible alarm ⑭. When enabled, these alerts are scheduled to occur at a predefined interval. These preferences are stored, ensuring they are pre-filled when the `CustomizationView` is accessed again. Pressing the 'Schedule Alert' button ⑮ schedules the notification, vibration and alarm. The app then transitions to the `ReminderView`, which is described in the following section.



(a) (b) (c)

Figure 5.2: `CreatorView` guides users through trip planning with the main interface (a), `StationView` sheet (b) and `DateView` sheet (c)



(a) (b)

Figure 5.3: `ResultView` (a) displays transport connections and `CustomizationView` (b) allows users to configure reminder settings

5.2.2 ReminderView

The `ReminderView` displays all currently scheduled reminders, grouped by date as seen in Figure 5.4. Each entry (16) includes the destination name, arrival time, line ID and the interval at which notifications will be sent, and features two buttons: a map icon (17) and a trash can icon (18). The map button triggers the system to display the public transportation route on the map and switches to the MapView, the app's third main screen which is discussed in the following section. The trash can icon removes the reminder from the database, unschedules the notification and cancels the vibration and alarm if previously enabled. A plus button (19) opens the `CreatorView`, allowing users to schedule a new reminder.

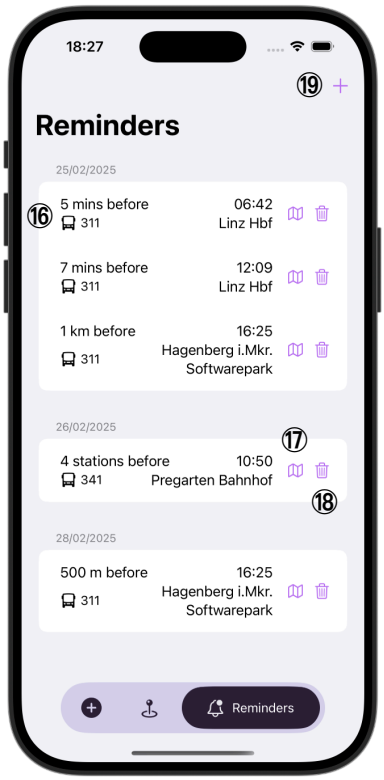


Figure 5.4: `ReminderView` displays scheduled reminders with options to view routes on the map or delete entries

5.2.3 MapView

The `MapView` displays the user's location and the selected public transportation route on a map. Depending on the chosen reminder mode, visual indicators are added to represent when alerts will be triggered. If the reminder is based on distance, circles (radii) (20) are drawn around the destination to indicate the alert zones. If it is based on the number of stops, the intermediate stations (21) where alerts will occur are marked. However, for time-based reminders, no visual representation is provided in the app. Figure 5.5

illustrates these visualizations: (a) represents the radii, (b) highlights the intermediate stops and (c) displays the standard route. The map icon button (22) toggles the map layout to satellite view, while the cursor icon button (23) zooms into the user's current location.

Figure 5.5: `MapView` displays routes for connections with active reminders showing alert radii (a), intermediate stops (b) or the standard route (c)

5.2.4 SpeechRecognitionView

The `SpeechRecognitionView` implements a wizard-based approach for setting up reminders through voice input processing. The user's speech is transcribed and sent to an AI model along with a structured prompt, which interprets the input and determines the necessary reminder details. This feature is primarily an experimental implementation intended for exploring how the reminder setup process can be made more accessible and user-friendly for a broader audience. It is not a fully developed feature but serves as a prototype for usability testing, further discussed in Section 5.4.

The process is structured into four consecutive views, each prompting the user with a specific question to progressively gather the required details for scheduling a reminder. However, all views follow the same workflow as illustrated in Figure 5.6. In the top-right corner, a microphone button (24) allows the user to start and stop the transcription process. Below, an options section (25) presents the user with three predefined choices as possible answers to the current question. As the user speaks, the input is processed and refined to improve accuracy before the transcript is displayed as seen in (26).

Once the transcript is available, the user can take further actions using the three buttons at the bottom of the screen. Pressing (27) resets the transcript, allowing the

user to reattempt the input. Pressing (28) sends the transcript along with the structured prompt to the AI model, which analyzes the response and attempts to match it to one of the predefined options. The selected option is then displayed in (30). Finally, pressing (29) checks whether an AI response has been successfully received. If so, the system navigates to the next step in the wizard-based flow, progressively collecting all the necessary details until the last view, where the reminders are scheduled.

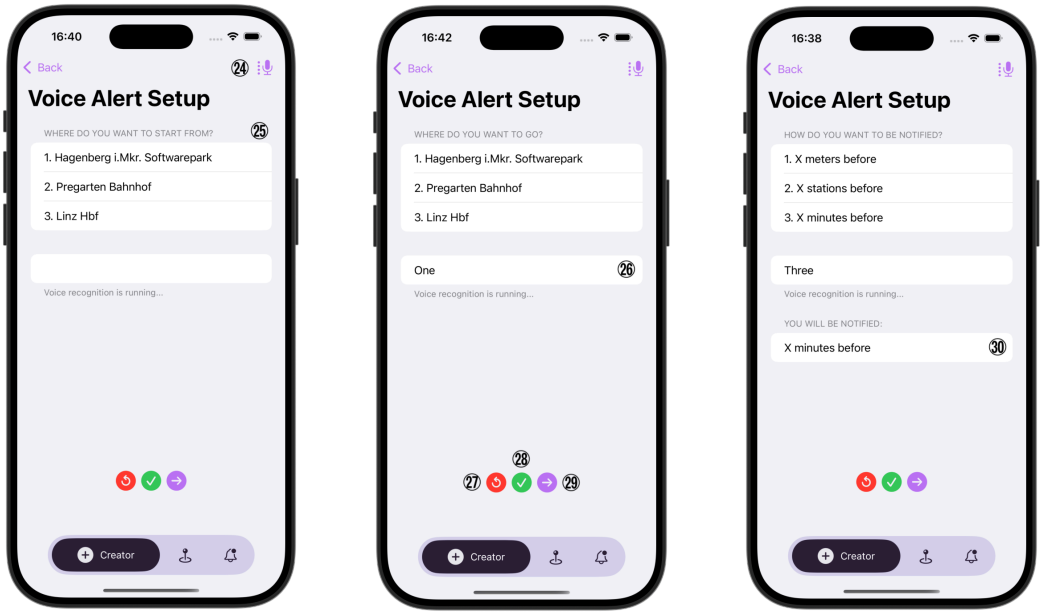


Figure 5.6: `SpeechRecognitionView` guides the user through AI-assisted voice processing for reminder setup

5.3 Reminder Behaviour

The app triggers reminders such as notifications, vibrations and alarms when users enter predefined geographic zones. This is handled by a custom `LocationManager` class, which extends Apple's `CLLocationManager` and follows a singleton pattern to prevent redundant tracking. The following subsections outline how regions are monitored and alerts are triggered in iOS.

5.3.1 Geofencing Methods

The geofencing process consists of monitoring regions and detecting user entry. Figure 5.7 provides an overview of this process in a flowchart.

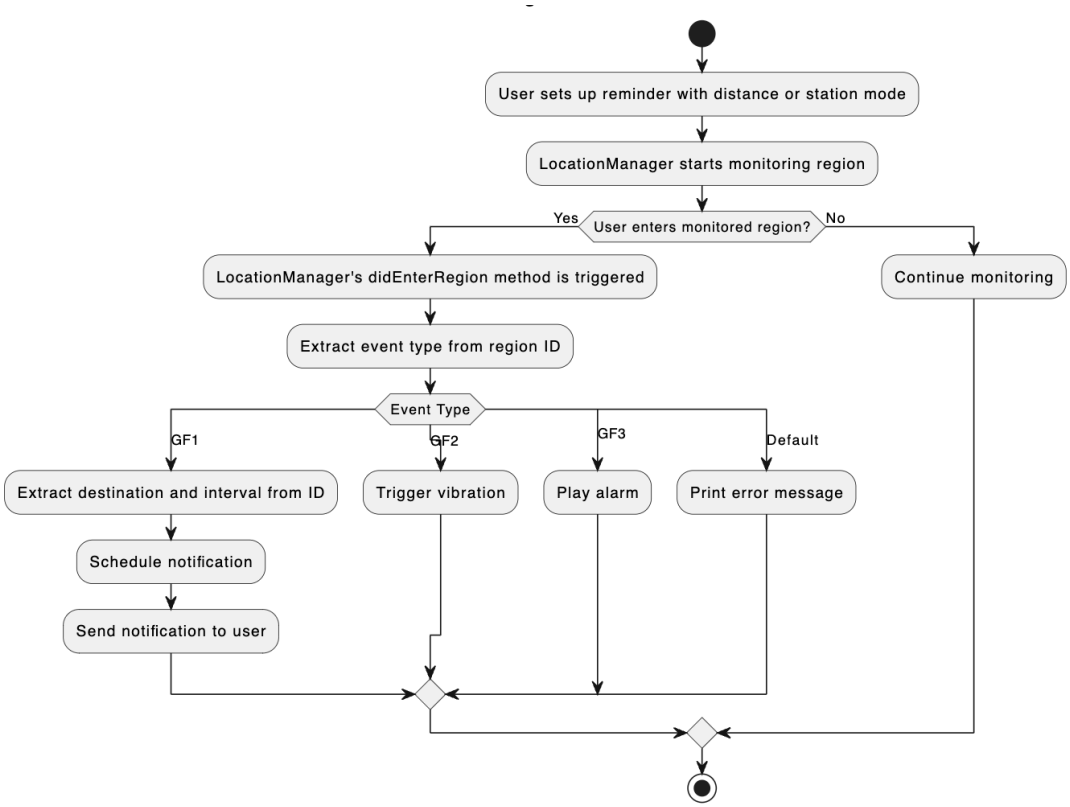


Figure 5.7: Flowchart illustrating the geofencing process

Start Monitoring

The function in Program 5.4 demonstrates how a geofence is created and registered for monitoring. Before creating the geofence, the function first checks whether geofencing is supported on the device. This ensures that the app does not attempt to monitor geofences on unsupported hardware. If monitoring is available, a `CLCircularRegion` object is created using a central coordinate (latitude and longitude), a radius that determines its size and a unique identifier. The properties `notifyOnEntry` and `notifyOnExit` determine whether notifications are triggered when the user enters or exits the region. In this app, only entry notifications are relevant, meaning `notifyOnEntry` is set to true, while `notifyOnExit` remains false. Once the `CLCircularRegion` is configured, the function registers the geofence with the system.

Stop Monitoring

To stop monitoring a geofenced region, the function takes the identifier of the region as a parameter. The `CLLocationManager` maintains a list of currently monitored regions, which can be accessed through its monitoredRegions property. Using this, the function searches for a geofence that matches the given identifier. If a corresponding region is found, monitoring is stopped as shown in Program 5.5.

```
let locationManager = CLLocationManager()
func startMonitorRegionAtLocation(center: CLLocationCoordinate2D, radius: Double,
    identifier: String) {
    if CLLocationManager.isMonitoringAvailable(for: CLCircularRegion.self) {
        let region = CLCircularRegion(center: center, radius: radius, identifier:
    identifier)
        region.notifyOnEntry = true
        region.notifyOnExit = false
        locationManager.startMonitoring(for: region)
    }
}
```

Program 5.4: Function for starting region monitoring

```
func stopMonitorRegionAtLocation(identifier: String) {
    if let region = manager.monitoredRegions.first(where: {$0.identifier ==
    identifier}) {
        manager.stopMonitoring(for: region)
    }
}
```

Program 5.5: Function for stopping region monitoring

Region Entry

In order to trigger an action based on a user entering a monitored region, the `CLLocationManager` provides the delegate method `didEnterRegion`, which is automatically called when a user enters a geofenced area. However, this method does not allow for additional parameters to be passed, meaning that only the region object is available within the function. To ensure the correct action is taken upon entering a geofenced area, the app must differentiate between different geofencing levels, each corresponding to a different type of alert. To achieve this, the simplest approach is string concatenation, where a Universally Unique Identifier (UUID) is combined with a prefix that indicates the type of geofence. In this implementation, the following naming convention is used: GF1 represents Geofence Level 1 which triggers a notification, GF2 for a vibration and GF3 for an alarm. For notifications, the identifier string also includes the destination name and the reminder interval, which are extracted to generate the notification content. Program 5.6 demonstrates how the `didEnterRegion` function processes a geofence entry event and triggers the corresponding action.

5.3.2 Triggered Actions

When a monitored region is entered, a geofence event triggers predefined actions. These actions include sending notifications, triggering vibration feedback and playing an audible alarm. These alerts ensure that users are notified in time for their stop. In contrast, the time-based mode schedules these actions for a specific time and date, independent of location. The following subsections detail the implementation of these triggered actions

```swift
func locationManager(_ manager: CLLocationManager, didEnterRegion region: CLRegion){
    guard let eventType = extractEventType(from: region.identifier) else {return}
    switch eventType {
    case "GF1":
        if let idInfo: [String] = extractIdInfoForNotification(using: region.
    identifier) {
            scheduleNotification(destination: idInfo[0], interval: idInfo[1], id:
    idInfo[2])}
    case "GF2": vibrate()
    case "GF3": playAlarm()
    default: print("Unknown event type: \(eventType)")
    }
}
```

Program 5.6: Function for handling monitored region entry events

in SwiftUI.

Notification

Notifications serve as a subtle first reminder to users as they approach their destination. The `scheduleNotification` function in Program 5.7 creates a notification request using `UNUserNotificationCenter`. The `UNMutableNotificationContent` object is populated with a title containing the reminder interval and a body displaying the final destination. To ensure the notification is noticeable, a critical sound alert is enabled using `UNNotificationSound.defaultCritical`. The notification trigger is set to fire after a one-second delay, ensuring immediate execution when the function is called.

To allow cancellation of a previously scheduled notification, the `cancelNotification` function removes pending notification requests associated with a given identifier. This ensures that users do not receive outdated alerts if their travel plans change. Figure 5.8 illustrates how a received notification appears within the app.

Figure 5.8: Notification alerting the user about their stop

For the time-based mode, the notification must be scheduled to fire at an exact time. Instead of an immediate trigger, the app constructs a `UNCalendarNotificationTrigger` using an instance of `DateComponents`, as seen in Program 5.8. Although the trigger type differs, the rest of the notification setup including the content configuration and request creation remains the same.

```swift
func scheduleNotification(destination: String, interval: String, id: String) {
    let content = UNMutableNotificationContent()
    content.title = "Please get off \(interval)!"
    content.body = "\(destination) is your stop."
    content.sound = UNNotificationSound.defaultCritical
    let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1, repeats: false)
    let request = UNNotificationRequest(identifier: id, content: content, trigger:
    trigger)
    UNUserNotificationCenter.current().add(request) { error in
        print(String(describing: error?.localizedDescription))
    }
}
func cancelNotification(for id: String) {
    UNUserNotificationCenter.current().removePendingNotificationRequests(
    withIdentifiers: [id])
}
```

Program 5.7: Triggering and canceling a local notification

```swift
let trigger = UNCalendarNotificationTrigger(dateMatching: dateComponents, repeats:
    false)
```

Program 5.8: Scheduling a local notification at a specific time

Vibration

Vibration feedback serves as a more noticeable second reminder to the user. The function shown in Program 5.9 utilizes the AVFAudio framework to initiate a short vibration pulse.

```swift
@objc func vibrate() {
    AudioServicesPlayAlertSound(kSystemSoundID_Vibrate)
}
```

Program 5.9: Triggering a vibration

For the time-based mode, a `Timer` object is used instead of a geofence event to trigger the vibration at a designated time as described in Program 5.10. The `Timer` object is initialized with an instance of `DateComponents` and once the time arrives, the `Timer` executes the function. The vibrate function in Program 5.9 is marked with `@objc` because it is referenced by `#selector` in the `Timer` object initialization. In Swift, methods used as selectors must be exposed to Objective-C runtime. Without `@objc`, the Swift compiler would not be able to locate the function at runtime when triggered by the `Timer` class.

To allow for cancellation of a previously scheduled vibration, `RunLoop.main.cancelPerform` is used. This method ensures that if a scheduled vibration is no longer needed, such as when a reminder is canceled, the function call is removed from the execution queue.

```
let timer = Timer(fireAt: dateComponents, interval: 0, target: self, selector: #
    selector(vibrate), userInfo: reminder.id, repeats: false)
RunLoop.main.add(timer, forMode: .common)
RunLoop.main.cancelPerform(#selector(vibrate), target: self, argument: id)
```

Program 5.10: Scheduling and canceling a vibration at a specific time

Alarm

The audible alarm acts as a last-resort alert to ensure the user is fully aware that they need to get off. Unlike notifications and vibrations, the alarm is highly intrusive making it difficult to miss. The function presented in Program 5.11 uses the `AVAudioPlayer` class from the AVFAudio framework to play a MP3 sound file. The function retrieves the file from the app bundle and initializes the audio player. If the file is missing or the initialization fails, an error is printed to aid debugging. For the time-based mode, the alarm can be scheduled and canceled just like in Program 5.10 by switching the selector to `#selector(playAlarm)`.

```
@objc func playAlarm() {
    var audioPlayer: AVAudioPlayer?
    let soundFileName = "alert_extreme.mp3"
    guard let soundURL = Bundle.main.url(forResource: soundFileName, withExtension:
    nil) else {
        fatalError("Unable to find \(soundFileName) in bundle")
    }
    do {
        audioPlayer = try AVAudioPlayer(contentsOf: soundURL)
    } catch {
        print(error.localizedDescription)
    }
    audioPlayer?.play()
}
```

Program 5.11: Triggering an alarm

5.4 Improving User Experience of App

To make the reminder setup more user-friendly and intuitive, two approaches have been tested and combined into a test feature in the app. The first approach is a step-by-step wizard, which guides users through the setup in a structured way. The second allows users to set reminders using speech input. The sequence diagram in Figure 5.9 breaks down how the system components interact during speech-based input, from recognizing speech and processing it with AI to displaying the recognized option and finally scheduling the reminder. Testing has shown that while speech input has potential, Apple's Speech framework struggles with consistently accurate transcriptions. This limitation is one of the reasons why the wizard-based approach was explored as well, as it reduces

the need for free-form input and instead guides users through predefined choices.

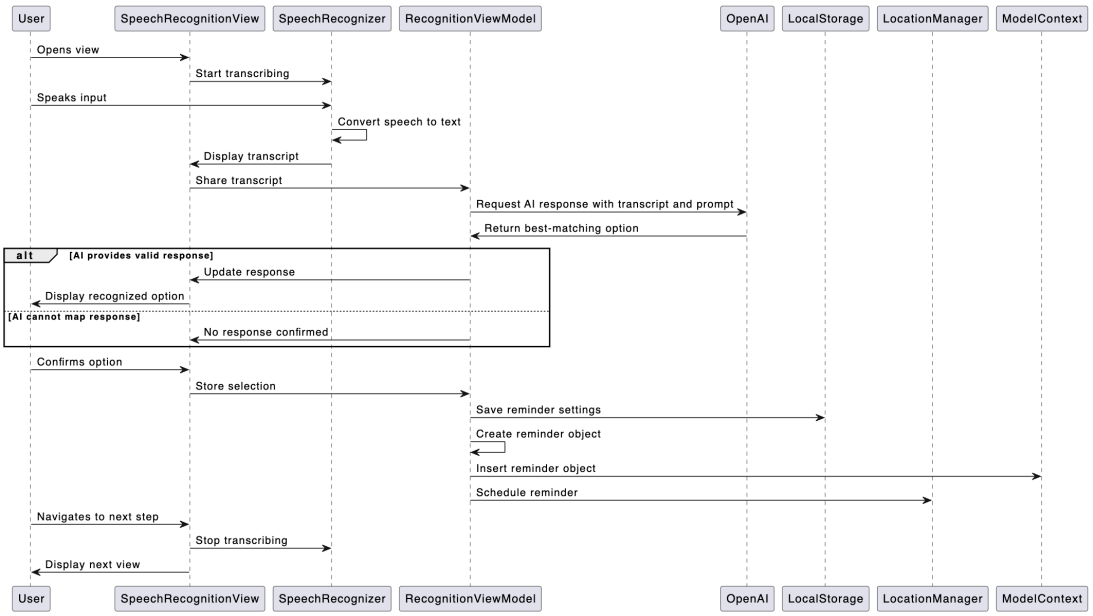


Figure 5.9: Sequence diagram for the voice-controlled reminder setup

Chapter 6

Conclusion

6.1 Resume

6.2 Next Steps

Appendix A

Acronyms

API Application Programming Interface
APs Access Points
AoA Angle of Arrival
BLE Bluetooth Low Energy
CoO Cell of Origin
CPS Cellular Positioning System
DoA Direction of Arrival
DoD Department of Defense
E-911 Enhanced 911
FCC Federal Communications Commission
GNSS Global Navigation Satellite System
GPS Global Positioning System
IMU Inertial Measurement Unit
INS Inertial Navigation System
IoT Internet of Things
IPS Indoor Positioning Systems
LBS Location-Based Services
PSAPs Public Safety Answering Points
RFID Radio Frequency Identification
RSS Received Signal Strengths
RSSI Received Signal Strength Indicator
SNR Signal-to-Noise Ratio
ToA Time of Arrival
TDoA Time Difference of Arrival
UI User Interface

UUID Universally Unique Identifier

UWB Ultra-Wideband

Wi-Fi Wireless Fidelity

WLAN Wireless Local Area Network

References

Literature

[1] Parul Arora and Suman Deswal. "Location Tracking Mechanisms for Dementia Patients". *Recent Patents on Engineering* 17.5 (2023), pp. 69–80 (cit. on p. 14).

[2] Per K Enge. "The Global Positioning System: Signals, Measurements, and Performance". *International Journal of Wireless Information Networks* 1 (1994), pp. 83–105 (cit. on p. 11).

[3] Amie Greenwald et al. "An economically viable solution to geofencing for mass-market applications". *Bell Labs Technical Journal* 16.2 (2011), pp. 21–38. DOI: 10.1002/bltj.20500 (cit. on p. 14).

[4] Dorota Grejner-Brzezinska and Allison Kealy. "Positioning and tracking approaches and technologies". *Advanced Location-Based Technologies and Services* (2004), p. 1 (cit. on pp. 4, 12).

[5] Ashwini L Kadam, Hoon Lee, and Mintae Hwang. "Implementation of IoT application using geofencing technology for protecting crops from wild animals". *Asia-pacific Journal of Convergent Research Interchange* 6.6 (2020), pp. 13–23 (cit. on p. 14).

[6] Krzysztof W Kolodziej and Johan Hjelm. *Local positioning systems: LBS applications and services*. CRC press, 2017 (cit. on pp. 5, 12).

[7] Axel Küpper. *Location-based services: fundamentals and operation*. John Wiley & Sons, 2005 (cit. on pp. 3–7, 9–11).

[8] Junggoo Lee et al. "An Efficient Localization Method Based on Adaptive Optimal Sensor Placement". *International Journal of Distributed Sensor Networks* 2014 (Aug. 2014). DOI: 10.1155/2014/983618 (cit. on p. 4).

[9] Andreea-Valentina Militaru et al. "Positioning Key Elements for Increasing Localization Precision of the VRUs in 5G NR Environments". *Bulletin of the Polytechnic Institute of Iași. Electrical Engineering, Power Engineering, Electronics Section* 68 (Feb. 2022), pp. 57–76 (cit. on pp. 3, 4, 7).

[10] Abel Mushiba. *ADVANCED TELECOMMUNICATIONS TECHNOLOGIES - A Comprehensive Guide to GSM, UMTS, LTE, and 5G*. Jan. 2024 (cit. on p. 10).

[11] Fabrice Reclus and Kristen Drouard. "Geofencing for fleet and freight management". In: *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*. 2009, pp. 353–356. DOI: 10.1109/ITST.2009.5399328 (cit. on p. 14).

[12] Alexander Ruegamer, Dirk Kowalewski, et al. "Jamming and spoofing of gnss signals–an underestimated risk?!" *Proc. Wisdom Ages Challenges Modern World* 3 (2015), pp. 17–21 (cit. on p. 11).

[13] Naser El-Sheimy. "Inertial techniques and INS/DGPS integration". *Engo 623-Course Notes* (2006), pp. 170–182 (cit. on p. 13).

[14] Yury Shevchenko and Ulf-Dietrich Reips. "Geofencing in location-based behavioral research: Methodology, challenges, and implementation". *Behavior Research Methods* 56.7 (2023), pp. 6411–6439 (cit. on pp. 2, 13).

[15] Akosua Takyiwa-Debrah. "A geofence and IoT-based solution for child abduction in Ghanaian schools" (2023) (cit. on p. 14).

[16] E. G. R. Taylor. "Five Centuries of Dead Reckoning". *Journal of Navigation* 3.3 (1950), pp. 280–285. DOI: 10.1017/S0373463300034871 (cit. on p. 8).

[17] Xiaofeng Wei et al. "Positioning algorithm of MEMS pipeline inertial locator based on dead reckoning and information multiplexing". *Electronics* 11.18 (2022), p. 2931 (cit. on p. 8).

[18] Martin Werner. *Indoor Location-Based Services: Prerequisites and Foundations*. Springer, 2014 (cit. on pp. 6, 7, 9, 12).

[19] Phil Winters, Sean Barbeau, and Nevine Georggi. "Smart Phone Application to Influence Travel Behavior (TRAC-IT Phase 3)" () (cit. on p. 11).

[20] Guohao Zhang et al. "Prediction on the Urban GNSS Measurement Uncertainty Based on Deep Learning Networks With Long Short-Term Memory". *IEEE Sensors Journal* 21.18 (2021), pp. 20563–20577. DOI: 10.1109/JSEN.2021.3098006 (cit. on p. 11).

Online sources

[21] 9to5Google. *Google Maps begins testing Live Activities on iPhone*. Accessed: 2025-02-08. 2024. URL: https://9to5google.com/2024/04/24/google-maps-live-activity-iphone-testing/ (cit. on p. 17).

[22] Citymapper. *Citymapper Supported Cities*. Accessed: 2025-02-08. 2025. URL: https://citymapper.com/cities (cit. on p. 18).

[23] GIS Geography. *Trilateration vs Triangulation - How GPS Receivers Work*. Accessed: 2025-01-05. URL: https://gisgeography.com/trilateration-triangulation-gps/ (cit. on p. 5).

[24] Andrew J. Hawkins. *Google Maps will now tell you when it's time to get off your train or bus*. Accessed: 2025-02-28. Dec. 2017. URL: https://www.theverge.com/2017/12/15/16763340/google-maps-train-bus-stop-transfer-alerts (cit. on p. 17).

[25] Apple Inc. *Alerts*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/design/human-interface-guidelines/alerts (cit. on p. 14).

[26] Apple Inc. *Audio Services: Play Alert Sound*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/audiotoolbox/1405202-audioservicesplayalertsound (cit. on pp. 14, 16).

[27] Apple Inc. *Core Location*. Accessed: 2024-11-03. 2024. URL: https://developer.apple.com/documentation/corelocation (cit. on pp. 15, 16).

[28] Apple Inc. *Core Motion*. Accessed: 2024-11-03. 2024. URL: https://developer.apple.com/documentation/coremotion (cit. on p. 15).

[29] Apple Inc. *Critical Alert Setting*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/usernotifications/unnotificationsettings/criticalalertsetting (cit. on p. 16).

[30] Apple Inc. *HealthKit*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/healthkit (cit. on p. 15).

[31] Apple Inc. *Live Activities*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/activitykit (cit. on p. 15).

[32] Apple Inc. *Monitoring the user's proximity to geographic regions*. Accessed: 2025-02-14. 2024. URL: https://developer.apple.com/documentation/corelocation/monitoring-the-user-s-proximity-to-geographic-regions (cit. on pp. 13, 16).

[33] Apple Inc. *Notification Center*. Accessed: 2024-11-03. 2024. URL: https://developer.apple.com/documentation/foundation/notificationcenter (cit. on p. 15).

[34] Apple Inc. *Playing Haptics*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/design/human-interface-guidelines/playing-haptics (cit. on p. 14).

[35] Apple Inc. *Region Monitoring*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions (cit. on p. 16).

[36] Apple Inc. *Scheduling a Notification Locally from Your App*. Accessed: 2024-11-02. 2024. URL: https://developer.apple.com/documentation/usernotifications/scheduling-a-notification-locally-from-your-app (cit. on p. 14).

[37] Apple Inc. *Setting Up a Remote Notification Server*. Accessed: 2024-11-02. 2024. URL: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server (cit. on p. 14).

[38] Apple Inc. *Timer*. Accessed: 2024-11-02. 2024. URL: https://developer.apple.com/documentation/foundation/timer (cit. on p. 15).

[39] Apple Inc. *UNCalendarNotificationTrigger*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/usernotifications/uncalendarnotificationtrigger (cit. on p. 15).

[40] Apple Inc. *UNLocationNotificationTrigger*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/usernotifications/unlocationnotificationtrigger (cit. on p. 15).

[41] Apple Inc. *UNTimeIntervalNotificationTrigger*. Accessed: 2024-11-04. 2024. URL: https://developer.apple.com/documentation/usernotifications/untimeintervalnotificationtrigger (cit. on p. 15).

[42] Massachusetts Institute of Technology. *Public Transportation*. Accessed: 2024-08-26. 2023. URL: https://climate.mit.edu/explainers/public-transportation (cit. on p. 1).

[43] Moovit Inc. *About Moovit*. Accessed: 2025-02-10. 2024. URL: https://moovit.com/about-us/ (cit. on p. 19).

[44] Moovit Inc. *Citymapper'S Get Off Alerts*. Accessed: 2025-02-10. 2024. URL: https://citymapper.com/news/823/get-off-alerts (cit. on p. 18).

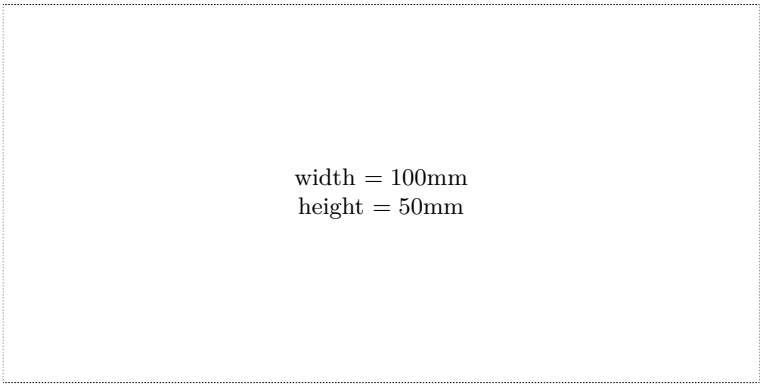
[45] Transistor Software. *Geofence Interface — React Native Background Geolocation*. Accessed: 2025-02-14. 2024. URL: https://transistorsoft.github.io/react-native-background-geolocation/interfaces/geofence.html (cit. on p. 13).

[46] TechRound. *Startup Profile: Citymapper*. Accessed: 2025-02-08. 2023. URL: https://techround.co.uk/startups/startup-profile-citymapper/ (cit. on p. 18).

[47] United Nations. *Act Now: Transport*. Accessed: 2024-08-26. 2023. URL: https://www.un.org/en/actnow/transport#:~:text=If%20your%20destination%20is%20too,tons%20of%20carbon%20emissions%20reduced. (cit. on p. 1).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —