BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

The Use of Augmented Reality in Locating and Virtually Replicating Landmarks

Supervisor
Lect. Dr. Lazăr Ioan

Author
Lazăr Răzvan-Andrei

2023

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Folosirea Realității Augmentate pentru Localizarea și Replicarea Virtuală a Reperelor Geografice

Conducător științific
Lect. Dr. Lazăr Ioan

Absolvent
Lazăr Răzvan-Andrei

2023

ABSTRACT

Recent developments in the field of navigation focus on creating a more and more immersive experience for the users, one that requires as little effort as possible and is as effective and reliable as it can be. Augmented reality is one of the core technologies used in this evolution due to its ability to create aiding elements that seemingly blend with the real world. While this direction centres on ease of use and time-saving, it takes away from the human creativity, curiosity, and experiences that some travellers are seeking when finding themselves in a new environment. Moreover, studies have shown that over-reliability on such navigation systems is detrimental to the innate abilities of wayfinding and spatial orientation, critical in a lot of limit situations.

This thesis proposes an original solution for both issues, which takes the very same technology, and uses it to create a similarly immersive experience that allows, in contrast, every individual more freedom to make their own decisions while navigating: AR landmark navigation. This method emphasizes the end goal of the journey by presenting spatially accurate outlines of a landmark, based on the geographic data of the camera and landmark. This projection can be seen through physical obstacles even if the user has no line of sight of the target. This offers enough cues to convey the surrounding context while still offering a challenging experience that forces users to engage with the world around them. In the long term, this approach is believed to improve spatial recognition and orientation.

In the first two chapters of this thesis, we will study the theoretical aspects of this method: the developments of the technology on top of which it is built and the issues it tries to resolve, and we lay down the mathematical foundations that allow us to convert real geographical data to the virtual world. In the third and fourth chapters, we present WanderAR, an Android application that exemplifies how such a technology can be used to create a new exciting travelling experience for users by allowing them to create routes composed of multiple landmarks, visualise them in different modes and even create their own, personal landmarks for places of interest.

The implementation is client-server to allow for scalability and to delegate the depositing of hefty landmark models away from the storage limitations of mobile devices. The client was implemented using Unity and the server exposes an API written with ASP.NET Core 6 which connects the client to a SQL Server database.

Contents

Chapter 1

Introduction

Navigation has been one of the core needs of the human race ever since its conception. Over the years, this process has gotten easier and easier. From the ancient star navigation, to the invention of maps, to modern GPS-based navigation, explorers have always been looking for a way to improve the reliability and speed of their routes. Each step in this development required less human effort and more reliability on external tools. In this day and age, location-based, real-time navigation has become irreplaceable and the classical 2D top-down map approach is transitioning into a more immersive and effective Augmented Reality version. Despite this natural development aimed at simplifying the way people travel every day, a negative side effect may be observed on spatial orientation, route learning, and environmental awareness. Studies carrier both in real [MZS+06] and virtual environments [RCRSC19, HOM+18] suggest that subjects who greatly rely on GPS navigation rarely engage with items that aid environmental learning and score lower at way-finding or route recalling in further iterations.

This thesis explores a modality in which people can improve these skills by offering an accurate location relative to the user of certain landmarks. This approach can later lead to assisted navigation in which only selected points of interest are displayed, leaving the task of wayfinding in the hands of the user, thus requiring a greater amount of conscious decision-making, ultimately leading to a better understanding of the route and the area.

This novel concept of navigation can be reduced to three simple steps:

1. Select a *destination* in the form of a landmark;
2. A three-dimensional model of the landmark is *displayed* on the screen in the direction where the real object can be found with an accurate scale based on distance;
3. *Navigate* towards the destination while being presented with the visual cue all along the way.

The final step can be chosen to be ignored and the model can be observed from a purely orientational perspective.

This idea leads to a new technology based on existing ones like AR navigation [Ran20] and spatial positioning [LLC23a], which is anchoring a 3D model in a virtual world only knowing the geographic coordinates of the observer and those of the target. While we have seen in past applications based on geographic data, they mostly rely on visual information and are optimised to work in an area relatively close to the observer.

What we propose is a system that can predict the location of an object without having a direct line of sight on it. This idea can be greatly improved by developments in accurate localisation and orientation, and due to its focus on visual cues that are unseen with the naked eye, can go beyond its initial scope of being used in environment exploration and aid in areas such as safety by presenting hazards in situations with limited visibility. Moreover, this technology is not based on a two-dimensional map representation of the Earth but takes into account the curvature when representing landmarks past a significant distance. This way we separated the positioning system into two modes:

- *A short-range mode* - for distances smaller than 50 kilometres, where the curvature of the earth is disregarded, the reference frame is the one of the device and the direction vector is composed using an intermediary point with the latitude of the device and the longitude of the landmark. This way better accuracy can be obtained in a small area;
- *A long-range mode* - for distances greater than 50 kilometres, where a set of transformations between coordinate systems are conducted to finally obtain a position of the landmark in the reference frame of the observer, followed by a scale adjustment to bring the 3D model to a manageable distance.

Although, as stated before, the focus is on visual representation, because we allow users to mark a location as a landmark and because we support long-distance targets, some adjustments had to be made when displaying the marker for the physical object. Thus, the 3D models of landmarks will be replaced with a placeholder (a circular point) under the following circumstances:

- The 3D model does not exist or is faulty;
- The angular size of the object is smaller than the human angular resolution of about 1 arcminute.

In support of these concepts, we will present an example of an application that encapsulates the main ideas presented above. **WanderAR** will provide users with

the base experience of AR landmark navigation. The name is a word-play from the action of wandering and the main used technology of augmented reality. Users will be presented with three types of navigation: *landmark* - that will target a single landmark, *route* - which will allow users to create a route formed by a succession of landmarks and visualise them in different ways, and *proximity* - which will allow users to visualise all the registered landmarks in a proximity area. WanderAR will be a client-server application in order to combat storage limitations on mobile devices, and to offer a scalable platform for future attempts.

For the implementation of the **client application**, we will use the *Unity* game engine and for the augmented reality integration on Android, the *AR Core* plugin. This application will allow users to create and manage accounts, landmarks and routes, and to visualise landmarks, in the modes presented above, in the navigation screen. The user can use the application as a guest and will have access to all the default landmarks, while not being able to create personalised content.

The **server side** will be used for storing all the data about accounts, landmarks and routes in a database and the connection will be provided by an API. The database of choice is Microsoft SQL Server and is accessed from an ASP.NET Core 6 web application using Entity Framework Core 6.

The thesis is structured into 6 chapters, the second and third chapters contain the theoretical part, while the fourth and fifth, have details about the implementation of **WanderAR**:

- **Chapter 1 - Introduction**;
- **Chapter 2 - Literature review**, presents the state of the art in the field of navigation, augmented reality for mobile devices and navigation, and discusses the issue of spatial-orientation loss caused by the overuse of GPS navigation systems;
- **Chapter 3 - Coordinate systems**, presents the necessary coordinate systems and transformations for building an AR application that is strongly linked to geographical data;
- **Chapter 4 - WanderAR: Server Side**, contains the architecture and implementation details for the server of the application. Here we can find information about the database, database access and how we structured the API that provides the main application with all the required data about users, landmarks and routes;
- **Chapter 5 - WanderAR: Client Application**, presents the functionalities of the application, the architecture for the mobile app and implementation details

about data access from the server, navigation modes and the flow for the main navigation component;

- **Chapter 6 - Conclusions and future work**, contains the conclusion about the positives, struggles and limitations, and ideas for further research in the area and application that can use similar concepts to the one presented in this thesis not just in the navigation area but for safety and other industries such as automotive.

Chapter 2

Literature review

In this chapter, we explore the current developments of the key components that drive this thesis: **navigation** and **augmented reality**, and how they interact with each other to create possible life-changing technologies. We inspect the state of the art for these areas, the global-scale issues that modern navigation systems might cause, and finally propose a solution that combines the two into a new AR landmark navigation model.

2.1 Modern navigation

The task of accurately locating a place on Earth and wayfinding between two points has been a priority for human society for a long time. Ever since the popularisation of long-distance and fast travel, the need for real-time precise localisation has pushed for new technological breakthroughs in this field. As expressed in [GAB20] there can be distinguished five navigation modes: pilotage, celestial navigation, dead reckoning, radio navigation (Section 2.1.2) and inertial navigation (Section 2.1.1). Modern systems use a combination of the previously mentioned methods to provide users with an efficient and reliable experience.

2.1.1 Inertial navigation

Inertial navigation systems (INS) make use of the forces produced by the motion of the subject. They have been in a constant state of development and innovation and play an important role in navigation, especially in locations where satellite or radio coverage is hardly reachable. This technology has been used to enable precise localisation underwater, in remote areas, in the air and even in space. Nowadays, INSs are used more and more in the field of indoor navigation, as well as being paired with satellite data for increasing the location precision of outdoor pedestrian and vehicle navigation and autonomous driving.

Inertial navigation is based mainly on the action of two forces: gravity and inertia, which can be measured with the use of relatively inexpensive equipment to a fairly small error. This makes its use in the realm of mobile devices affordable and convenient.

Modern smartphones come equipped with Micro-Electrical-Mechanical Systems (MEMS) gyroscopes and accelerometers. Different studies [CHYJ18, GZL+23, KGR21] have tackled the issue of the accuracy of smartphone-integrated inertial sensors and concluded that, while there are important differences from one device to another and the average error is still significant, today's flagships are getting close results to dedicated sensors.

Other developments in this field include **visual-inertial navigation**, where a calibrated camera is used to concurrently measure the movement of observed features [Hua19, MR07]. Future integration of deep learning in this area aims to better predict the mobility and action of the traced subjects [CZL+20, SZG+20].

2.1.2 Radio navigation

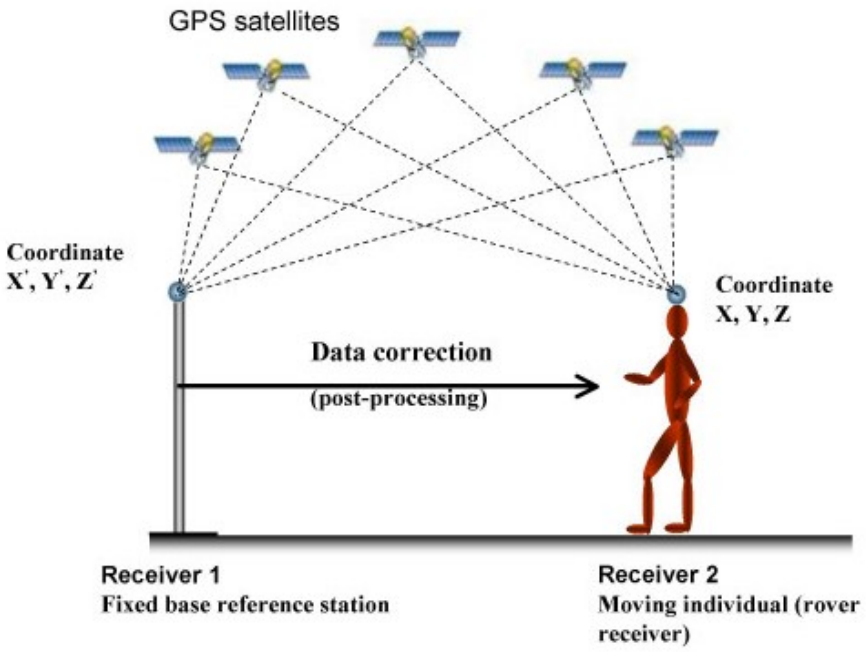


Figure 2.1: Differential GPS principle - The fixed receiver 1 calculates the error from the satellites and computes the correction which then sends to the mobile receiver 2. *Source: [TS05]*

Radio navigation can be ground or space-based, the latter of which is represented by the widely used **satellite navigation (SatNav)**. Satellite navigation is based on the concept of *Standard Point Positioning (SPP)*, which uses passive trilateration to

determine the position and velocity of a receiver by sending synchronised signals from the satellites in the system [MvDSJ+21].

SatNav systems are formed by more than 4 satellites (the minimum number for SSP [MvDSJ+21]) and can be either regional or global, based on the area that it covers. Currently, there are four available global systems (also known as GNSS): Global Position System (GPS) - USA, GLONASS - Russia, BeiDou - China and Galileo - Europe and two regional constellations: QZSS - Japan and NAVIC - India [Hei20].

Satellite-Based Augmentation Systems (SBAS) are small networks of geostationary satellites tasked with correcting the impact of the ionosphere (2.1) and providing integrity for civil aviation use (due to the military origins of the GNSS there was a need to separate the civilian infrastructure). Figure 2.2 shows the operational and future SBASs.

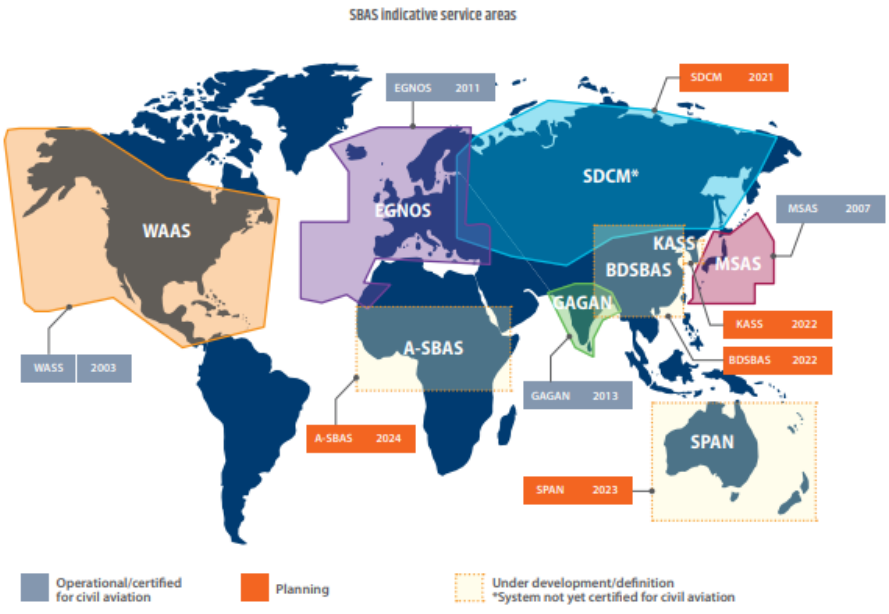


Figure 2.2: Satellite-Based Augmentation Systems (SBAS). *Source: EUSPA User Technology Report 2020*[1]

2.2 Effects of GPS navigation on spatial orientation

GPS navigation is nowadays a useful and easily accessible tool for a large proportion of the population. In a society that has grown accustomed to assisted navigation, certain concerns regarding the overreliance on such technology were raised: Have we become way too dependent on this technology to guide us at every step? Are we slowly losing our natural ability to orientate and navigate? What will happen if this, now widely available commodity, will suddenly become inaccessible?

[1]https://www.euspa.europa.eu/sites/default/files/uploads/technology_report_2020.pdf

Various studies tackled these topics so far. For the purpose of this thesis, three studies were selected. All of them consist of a learning or navigation phase, and a recall phase, in which the human subjects were tested on different aptitudes of wayfinding, spatial orientation and route recognition. One of the studies was conducted in a real environment [MZS+06] while the other two were in virtual environments [HOM+18, RCRSC19].

The aim of [MZS+06] was to compare different degrees of assistance: visual + context, auditory + context, auditory and paper map (where context means the user's location on the entire map). The recall phase consisted in remembering the right choice of direction in each point of decision on the given path and placing images on a mute map. It was concluded that the paper map users scored better both in route recognition (an average error of 7% for map users compared to 25% for the highest level of assistance) and survey knowledge (on average 75 to 192 pixel-deviation for the same two categories).

In [HOM+18] we find a different approach in the shape of *"there-and-back"* tasks where, in the learning phase, the participants were guided on a path, always longer than the optimal path, between two points. The recall phase consisted of three tasks: the participants were asked to point towards the starting location, and then navigate to it where they had to point towards the end location. In addition, the subjects were required to fill a blank map given the representation of landmarks found on the way. Thus, the speed, focus, accuracy, path and time redundancy gave a final score for each participant.

Finally, [RCRSC19] has tackled the issue of mental rotation and perspective-taking, where, after a similar learning process of guided navigation in the virtual environment and assessing different landmarks, the subjects were required to select rotated photographs of landmarks and point towards the location of different objects in the virtual space.

All of the above studies have concluded that the external navigation aid affects negatively the capacity to recall certain aspects of the route and diminishes spatial cognitive abilities due to decreased awareness of the surroundings and the lack of need for landmark and environmental assessment.

2.3 Use of landmarks in improving spatial recognition

The term **"landmark"** is a term that has been broadly used in navigation to describe standout characteristics in an environment. In this thesis, the term will be used to describe monuments or buildings of high importance for a specific area or culture, or that are easily recognisable due to architectural particularities. Monuments like the Eiffel Tower, Paris or the Statue of Liberty, New York and buildings like The

White House, Washington, D.C. or the Palace of Parliament, Bucharest are examples of such landmarks.

In human navigation, landmarks represent staples of orientation and are key when identifying and mapping a new area by providing an association for new places with recognisable, familiar constructs. Landmark navigation does not target optimisation, either when talking about distance, or speed, but about naturally learning a new region. Thus, studies have shown that this manner of navigation leads to improvements in spatial orientation and wayfinding over time [LSL22, MGJ+00]. A recent study showed that, compared to unassisted navigation, a group of people aided by VGL systems displaying outlines of the landmarks in their location performed better at perspective and spatial orientation tasks [LSL22]. The study presents a comparable method to the one presented by this thesis, and the results highlighting the improvements are presented in Table. 2.1.

Group	Women/Men	Age(years)	SBSOD	PTSOT
VGL	15/13	27.57 ($\pm$4.78)	0.66 ($\pm$0.15)	37.51 ($\pm$26.49)
NVGL	9/18	28.19 ($\pm$5.33)	0.67 ($\pm$0.17)	26.31 ($\pm$17.1)

Table 2.1: Participant demographics and average orienting ability test scores. *Source: [LSL22]*
Standard deviations are shown in parenthesis
$SBSOD$ Santa Barbara Sense of Direction scale,
$PTSOT$ Perspective Taking/Spatial Orientation Task

2.4 Augmented Reality for mobile devices

Due to the increasing demand for portable communication products, in particular, touchscreen smartphones, mobile phones and tablets have emerged as the primary output medium for augmented reality (AR), which has experienced a significant increase in popularity and usage among the general public. During the conception of this thesis, we have seen a major innovation in this area in the form of **Geospatial Creator** [LLC23b], which, by using geographic anchors, allows developers to create unique AR experiences based on physical locations using Street View data and models. This, along with AI-assisted technology for recognising and labelling real street geometry such as cars, buildings or trees, is bundled in the latest versions of *Google's* platform for mobile augmented reality applications: **ARCore** [LLC22].

Small display-based AR (handheld mobile display), in contrast to desktop and head-mounted display-based AR systems, requires various interaction techniques, most of which rely on single-handed interaction due to the restrictions of small screen displays and short activity times caused by the battery life limitations of the

mobile devices.

[GSI19] defines three different interaction modes and 3D object manipulation for mobile AR devices: touch-based, mid-air gestures-based, and device-based techniques. **Touch-based** interaction has been explored since the introduction of touch-screen devices but with the limitation of only two dimensions in the phone's plane, ignoring the depth component. **Mid-air gesture-based** interaction represents the next step in the credibility of AR systems and represents visually interacting with the object on every dimension in the user's frame of reference rather than the one of the phone. Finally, **device-based** involves using the device as a controller in the interaction with the virtual object.

Another step back for large-scale mobile AR applications is represented by the limitations in device localisation and rotation. An offset by just a few degrees in rotation can cause errors of hundreds or thousands of kilometres when representing objects at the scale of the Earth. In the past few years several methods of using a fusion of sensors to create a more appropriate representation of the phone's orientation relative to its environment using data from the gyroscope, GPS and camera.

All this data gathered by the device needs to be transposed to virtual reality by creating a frame of reference that facilitates this translation. [Shi05] presents a few examples of such frames with their evolution and relationships. For the purpose of this thesis, we highlight the **earth-frame**, which has its origin in the centre of mass of the Earth, and the **navigation-frame**, with the origin in the location of the observer. These two frames are key when translating real coordinates to positions in the virtual world, which we will study in a later chapter.

2.5 Use of Augmented Reality in navigation

With the popularisation of GPS-assisted navigation, it is natural that the next step is replacing the old two-dimensional directional indicators with three-dimensional signs, overlapped on top of real roads, obstacles or junctions. This is where augmented reality comes into the scene, as a way of further enhancing the user's experience and ease of use while navigating.

In the last couple of years, there has been continuous progress in the AR navigation scene with tools like **Live View by Google** [Ran20] coming as an addition to already one of the most popular navigation apps for Android users, combining the classical 2D maps with AR directions (Fig. 2.3).

Augmented reality is an important prospect in automotive navigation where, if precise, localised instructions can take away the strain from the driver in high-stress situations such as heavy traffic or settings that require quick decision-making. As expressed in the study [BGB20] there is an increasing demand and research for aug-

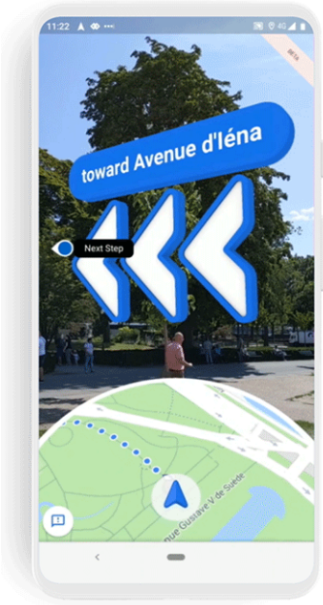


Figure 2.3: Live view for Google Maps showing AR directions near Eiffel Tower in Paris. *Source: Introducing Live View, the new augmented reality feature in Google Maps*[2]

mented reality technology in the field, with some research showing that, although not yet a big improvement over traditional GPS navigation, drivers take less time to react to directions while using AR assist [YKA22].

For pedestrian navigation, AR can create a completely different way of experiencing travelling and exploring both urban and natural landscapes. As we can see from [DWQ+21], while comparing AR to traditional 2D navigation for pedestrians we cannot find any great improvements in terms of speed, number of errors or hesitations. Moreover, when it comes to route drawing the study concludes that AR users performed much worse, which stays consistent with the concerns that the ease of use reduces the engagement of the user. However, augmented reality can be used in other interactive ways for enhancing the user's experience and combat the very problem it causes, the lack of engagement and need for spatial orientation. This thesis proposes a solution based on landmark navigation, that, by using augmented reality, reverts to a more simple, yet fun and challenging way of exploring unknown areas. In this approach, the users are given visually the location of the target landmark and finding the path toward it is left completely up to them.

[2]https://support.google.com/maps/thread/11554255

Chapter 3

Coordinate systems

An essential component of the technology presented in this thesis is the localisation and positioning of virtual components in relation to the geographical data available. Before jumping into the details of what an application based on our idea of AR landmark navigation can look like, we first need to define the coordinate systems and mathematical transformations that will be used to build a reliable methodology for tracking real objects in the virtual world. In this chapter, we will present the coordinate systems of both worlds and study how we can convert the geographic coordinates of landmarks to Cartesian coordinates that we can use in our virtual environment to depict the direction and distance from the observer to said landmark.

3.1 Virtual Cartesian systems

There are two important Cartesian coordinate systems that define the location of every virtual object in an AR application: **world** and **session**. Every geographical data will be translated relative to one of these two systems. Considering our choice of implementation, we will explain the two systems as they are represented by the **Unity** game engine.

3.1.1 The World Origin system

The world origin represents the centre of the virtual world, the point of coordinates $(0, 0, 0)$, with the three axes X, Y and Z defined with respect to the screen as follows:

- $+X$ - the horizontal axis of the screen, pointing from left to right.
- $+Y$ - the vertical axis of the screen, pointing from down to up.
- $+Z$ - the depth axis, pointing away from the viewer.

In other words, the plane formed by axes X and Z forms the ground plane, while the Y axis points upwards, as opposed to a conventional mathematical system when the Z axis is the one that points up.

The **world origin system** defines a static system and all the global transformations in the virtual world will be defined with respect to this coordinate system.

3.1.2 The Session Origin system

The session origin is the container for the AR camera in the virtual world. It is used to position the camera, transform and scale objects without affecting the motion tracking of the Camera object ([Fin22]). This system is used to adjust the rotation and motion of the camera tracking system. The rotation of the session system will be measured relative to the local tangent plane. Due to the definition of the axis systems in Unity, the local tangent plane will follow the *East-Up-North (EUN)* system rather than the classical *East-North-Up (ENU*, Subsec. 3.2.4*)*.

The **AR Camera** object is a child of the session origin object, together with other tracked objects to ensure consistency to the real world, and matches the rotation and movement of the physical device. Its axes with relation to the physical device (phone) are represented in Fig. 3.1.

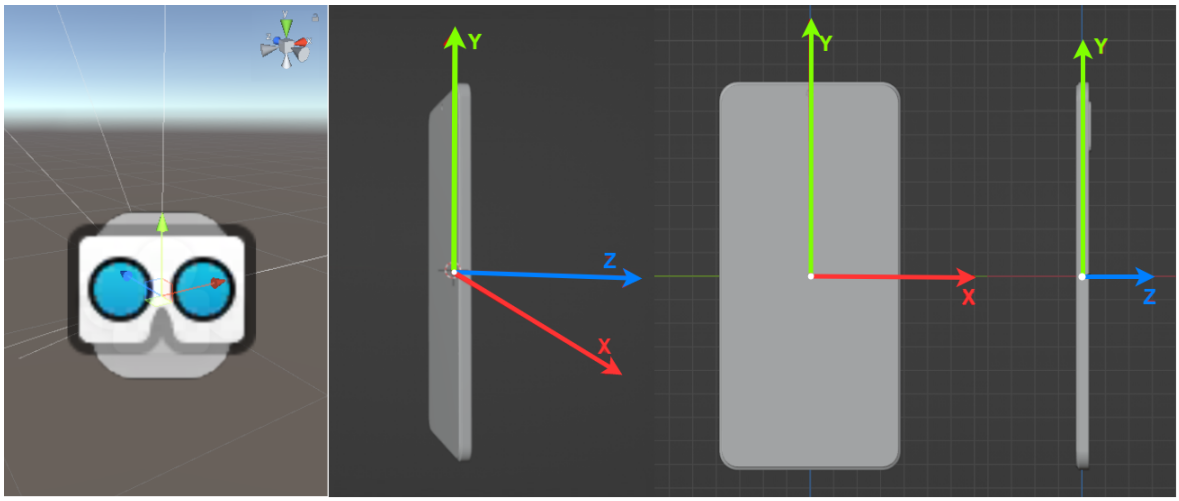


Figure 3.1: Virtual axis of the camera overlapped on the physical AR device.
+X - to the right facing the smartphone screen;
+Y - up;
+Z - pointing from the camera forwards.

3.2 Earth Coordinate systems

3.2.1 The WGS 84 standard

Since modelling a completely accurate representation of the Earth would be too complex, in geodesy, the Earth is approximated by an *oblate spheroid*. When working with geographical coordinates, we need to keep in mind that they are coordinates on the surface of a reference *Earth ellipsoid* defined in a geodetic system. The one used nowadays for most navigation-related tasks is the **World Geodetic System 1984 (WGS84)** (Fig. 3.2), defined by the parameters from the table 3.1 and the axis system defined as follows [Age14]:

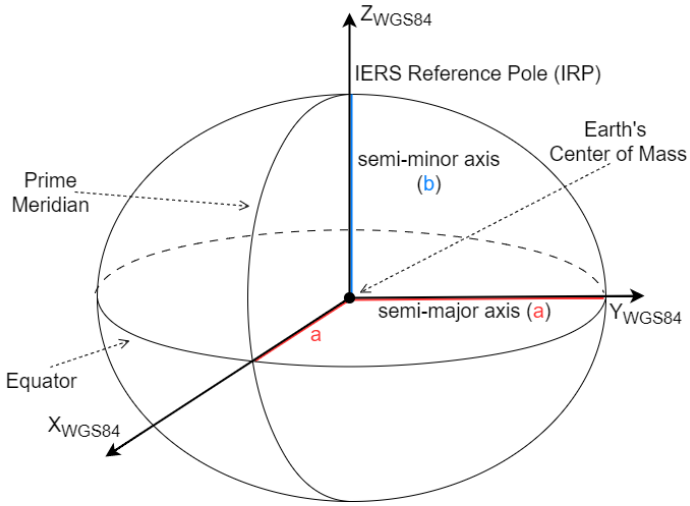


Figure 3.2: World Geodetic System 1984

Origin - Earth's centre of mass;

Z-Axis = towards the North Pole, more exactly going through the International Reference Pole (IRS);

X-Axis = going through the intersection of the prime meridian and the equator plane (the plane passing through the centre of mass and normal to the Z-axis);

Y-Axis = completes the coordinate system following the right-hand rule.

Parameter	Symbol	Value	Units
Semi-major Axis	a	6378137.0	m
Semi-minor Axis	b	6356752.314	m
Flattening Factor of the Earth	1/f	298.257223563	
Geocentric Gravitational Constant	GM	$3.986004418 * 10^{+14}$	m^3/s^2
Nominal Mean Angular Velocity of the Earth	ω	$7.292115 * 10^{-5}$	$rads/s$

Table 3.1: WGS 84 Defining Parameters. *Source: [Age14]*

3.2.2 The geodetic coordinate system

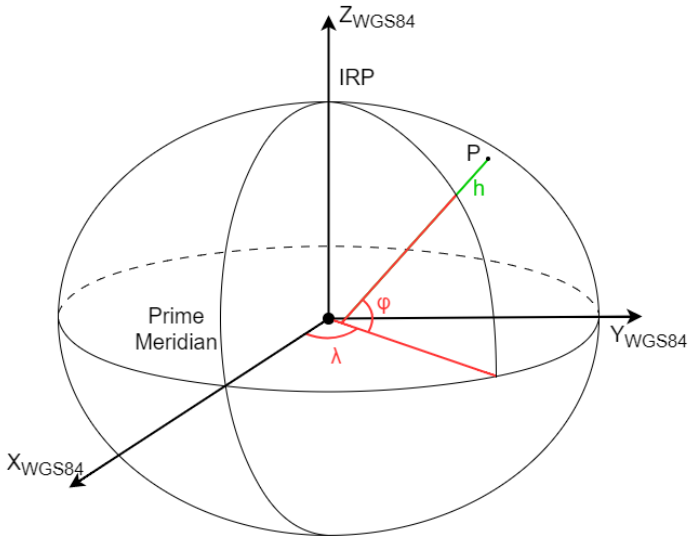


Figure 3.3: The geodetic coordinate system reported to the **WGS84** earth ellipsoid.
λ - longitude;
φ - geodetic latitude;
h - geodetic altitude.

The geodetic coordinate system (or elliptical coordinate system) (Fig. 3.3) defines the location of a point by *geodetic latitude*, *longitude* and *geodetic or ellipsoidal height*. It is the most common system for expressing an object's location on Earth's surface and the one that we will use as a base for all the locations in the implementation.

Geodetic latitude (φ) - or simply *latitude* is the angle formed by the equatorial plane and the normal on the surface in the measurement point. The latitude is measured in angles from $-90°$ to $+90°$, where $0°$ points to the **equator**;

Longitude (λ) - is the angle formed by the rotation around the Z-axis. It is measured from $-180°$ to $+180°$, where $0°$ points to the **prime meridian**;

Geodetic height (h) - or *altitude* is the distance between the measured point and the surface of the ellipsoid. Points inside the ellipsoid will have a negative height while the ones outside will have a positive value.

3.2.3 The Earth Centre, Earth First (ECEF) coordinate system

The Earth Centre, Earth First coordinate system (Fig. 3.4), also known as the *Conventional Terrestrial Reference System* (TRS) [SZHP11a], represents an earth-fixed Cartesian system suitable for the **earth frame** model.

The ECEF system is based on the **WGS84** representation, sharing its origin and axes, and allows for fast transformations to and from the geodetic coordinates.

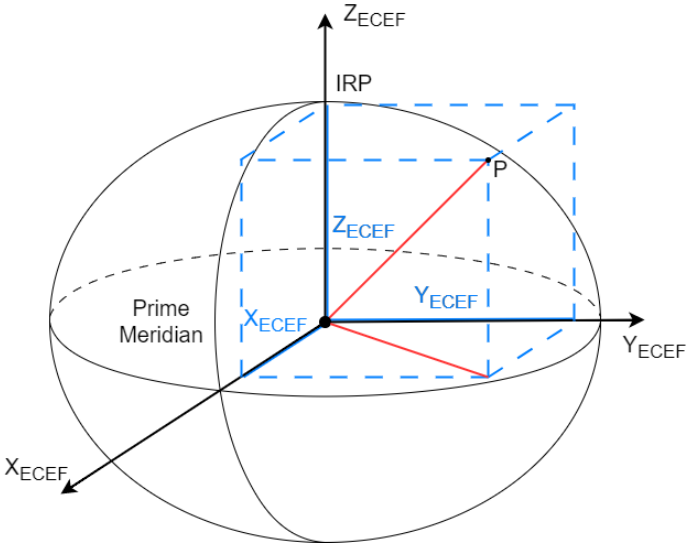


Figure 3.4: The Earth Centre Earth First coordinate system

3.2.4 The East-North-Up (ENU) coordinate system

The East-North-Up (Fig. 3.5) is a tangent plane coordinate system. This means that the base plane is tangent to the surface of the *earth ellipsoid*. This type of coordinate system is usually used in tracking applications in correlation with the **navigation frame** because it revolves around the observer, who becomes the centre of the system. The *origin* is fixed to a specific coordinate and the axes are defined as follows:

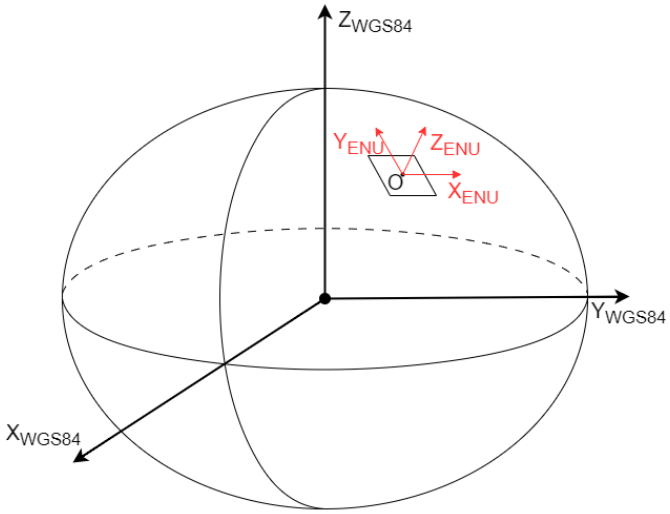


Figure 3.5: The East-North-Up coordinate system

X-axis - pointing **east**, tangent to the geographic parallels;

Y-axis - pointing **north**, tangent to the geographic meridians;

Z-axis - pointing **up**, the normal on the XY base plane (and on the surface of the ellipsoid).

3.2.5 Coordinate system transformations

As stated before, when talking about a location on Earth we are referencing it in terms of latitude, longitude and altitude, in other words, using the geodetic coordinate system. This format is used by localisation APIs and is the easiest to use in order to store location data about landmarks. However, when attempting to place objects in the virtual world we need Cartesian coordinates. Thus, we will use a set of transformations between the geodetic, ECEF and ENU systems. The ECEF coordinates are necessary for straight-line distance computation as well as directions on a small scale when the curvature of the Earth is negligible. We will use ENU coordinates to transform objects from the **earth frame** to the **navigation frame** using an ENU system with the origin at the location of the observer.

For the purpose of this thesis, where we consider the geodetic coordinates of landmarks as the known variables, we only need transformations from geodetic to ECEF and from ECEF to ENU.

Geodetic to ECEF transformations

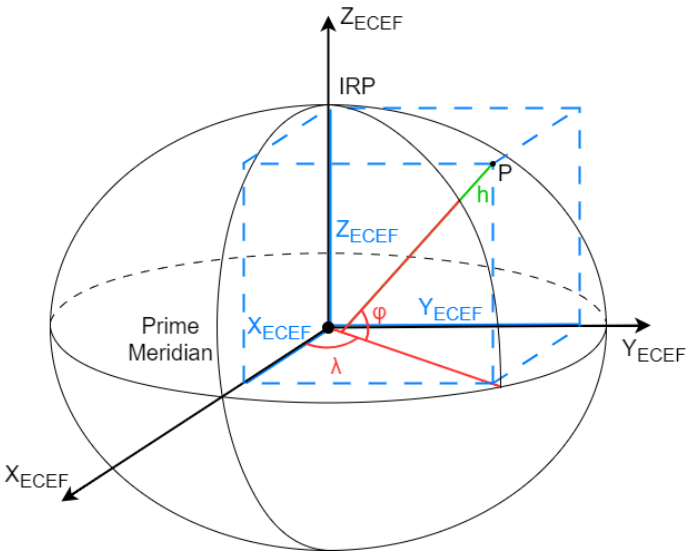


Figure 3.6: Relation between geodetic and ECEF coordinates

For this transformation (Fig. 3.7) we know the geodetic coordinates (φ, λ, h) of a point P, where φ - latitude, λ - longitude, h - altitude. We also know a = semi-major axis of the earth and b = semi-minor axis of the earth (Tab. 3.1). In order to find the ECEF Cartesian coordinates we have the following relations [SZHP11b]:

Let e = eccentricity of the Earth ellipsoid, with e^2 being the eccentricity squared:

$$e^2 = \frac{a^2 - b^2}{a^2} \tag{3.1}$$

and N = the radius of curvature in the prime vertical:

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \varphi}} \tag{3.2}$$

Thus, we have the ECEF coordinates of the point $P = (x_{ecef}, y_{ecef}, z_{ecef})$:

$$\begin{aligned} x_{ecef} &= (N + h) \cos \varphi \cos \lambda \\ y_{ecef} &= (N + h) \cos \varphi \sin \lambda \\ z_{ecef} &= ((1 - e^2)N + h) \sin \varphi \end{aligned} \tag{3.3}$$

ECEF to ENU

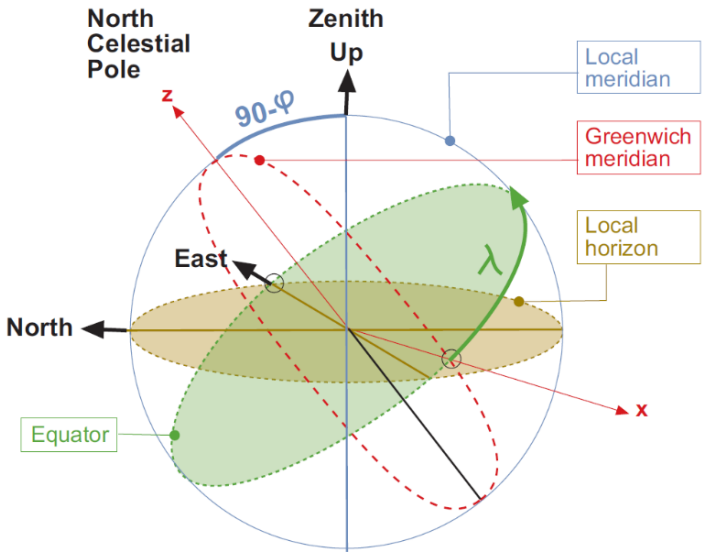


Figure 3.7: Rotation relation between the ECEF and ENU coordinate systems. *Source: [SZHP11c]*

In this transformation, we have two points: $P = (x_{Pecef}, y_{Pecef}, z_{Pecef})$ - the observed point and an observer $O = (x_{Oecef}, y_{Oecef}, z_{Oecef}) = (\varphi, \lambda, h)$. O will become the origin of the ENU system, thus we have to take into account the change in origin and rotation of the systems.

We begin by rotating the system using the rotation matrices:

$$\begin{aligned} R_x(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}; R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}; \\ R_z(\theta) &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}; \end{aligned} \tag{3.4}$$

In order to align the ECEF on top of ENU [SZHP11c] we need to perform a rotation of $\pi/2 - \varphi$ radians over the *X-axis*, followed by a rotation of $\pi/2 + \lambda$ radians over the *Z-axis*. Thus, using the R_x and R_z rotation matrices from (3.4) we obtain the transform matrix:

$$R_r = R_x[\pi/2 - \varphi]R_z[\pi/2 + \lambda] = \begin{bmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\cos\lambda\sin\varphi & -\sin\lambda\sin\varphi & \cos\varphi \\ \cos\lambda\cos\varphi & \sin\lambda\cos\varphi & \sin\varphi \end{bmatrix}; \tag{3.5}$$

To obtain the ENU coordinates for the point $P = (x_{enu}, y_{enu}, z_{enu})$ in the ENU system of the point P, we need to apply the rotation transform matrix R_r from (3.5) to the vector $\overrightarrow{OP}$.

$$\begin{aligned} \begin{bmatrix} x_{enu} \\ y_{enu} \\ z_{enu} \end{bmatrix} &= R_r * \overrightarrow{OP} = R_r * \left(\begin{bmatrix} x_{Pecef} \\ y_{Pecef} \\ z_{Pecef} \end{bmatrix} - \begin{bmatrix} x_{Oecef} \\ y_{Oecef} \\ z_{Oecef} \end{bmatrix} \right) \\ &= \begin{bmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\cos\lambda\sin\varphi & -\sin\lambda\sin\varphi & \cos\varphi \\ \cos\lambda\cos\varphi & \sin\lambda\cos\varphi & \sin\varphi \end{bmatrix} * \left(\begin{bmatrix} x_{Pecef} \\ y_{Pecef} \\ z_{Pecef} \end{bmatrix} - \begin{bmatrix} x_{Oecef} \\ y_{Oecef} \\ z_{Oecef} \end{bmatrix} \right) \end{aligned} \tag{3.6}$$

3.2.6 Adjusting coordinates transformations for Unity

As discussed in the section 3.1 there is an important difference between the systems presented in this section and the ones used by **Unity**, namely the switch between the Y and Z axis. Due to this, we need to make adjustments to the transformation formulas, such that they match the new axes definition.

Thus, the system transforming geodetic coordinates to ECEF (3.6) becomes:

$$\begin{aligned} x &= (N + h)\cos\varphi\cos\lambda \\ y &= ((1 - e^2)N + h)\sin\varphi \\ z &= (N + h)\cos\varphi\sin\lambda \end{aligned} \tag{3.7}$$

The transform matrix aligning the rotation of the ECEF and ENU coordinate systems (3.5) becomes:

$$R'_r = \begin{bmatrix} -\sin\lambda & 0 & \cos\lambda \\ \cos\lambda\cos\varphi & sin\varphi & \sin\lambda\cos\varphi \\ -\cos\lambda\sin\varphi & \cos\varphi & -\sin\lambda\sin\varphi \end{bmatrix}; \tag{3.8}$$

And substituting it in (3.6) yields the following formula for finding the coordinates of a point P in the **EUN** (East-Up-North) system of an observer O:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\sin\lambda & 0 & \cos\lambda \\ \cos\lambda\cos\varphi & sin\varphi & \sin\lambda\cos\varphi \\ -\cos\lambda\sin\varphi & \cos\varphi & -\sin\lambda\sin\varphi \end{bmatrix} * \left(\begin{bmatrix} x_{Pecef} \\ y_{Pecef} \\ z_{Pecef} \end{bmatrix} - \begin{bmatrix} x_{Oecef} \\ y_{Oecef} \\ z_{Oecef} \end{bmatrix} \right) \tag{3.9}$$

Chapter 4

WanderAR: Server Side

In this chapter, we see how the server that supplies WanderAR was designed and implemented. We begin presenting the architecture, both internal and deployment, and the main areas of:

- *Database* - the storage for all the data about users, landmarks and routes, we will present its structure and how we can access it;
- *Data security* - how we protect each user's personal data and the integrity of the entire application;
- *Error handling* - how we handle unexpected server errors or exceptions induced by faulty user input;
- *Web API* - what does the interface provide and what actions is the client allowed to execute on the data stored on the server.

4.1 Architecture

The server side of **WanderAR** is built using ASP.NET Core 6.0 and provides a complete API that facilitates the connection to the database. This supplies the mobile application with data about user accounts, landmarks with their positioning and 3D models, and routes created by the users.

4.1.1 Layered architecture

There are three packages that compose the back-end (Fig. 4.1):

- **Application Core**: containing the models of the database classes, exceptions, interfaces of services and repositories, and services;
- **Infrastructure**: the connection with the database, repositories and data access;

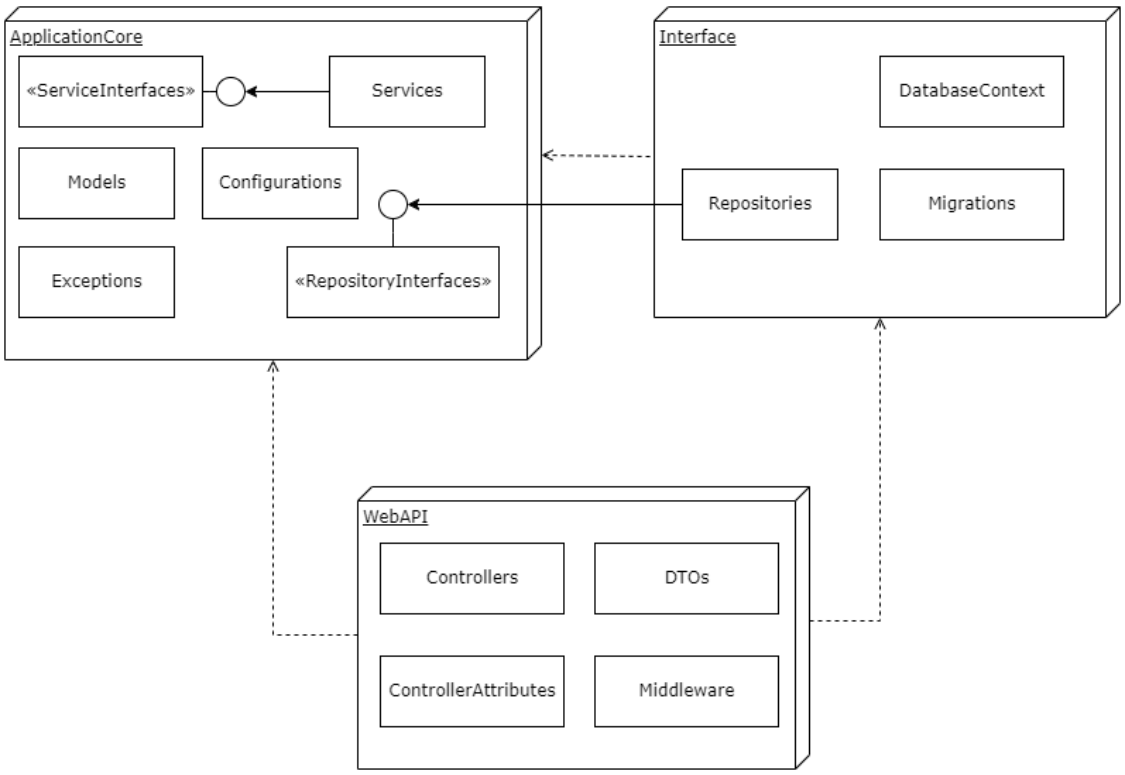


Figure 4.1: The structure of WanderAR server-side

- **WebAPI**: the API that provides the interface for the client side of the application, alongside data transfer objects (DTOs), middleware, and controller attributes.

The connections between the interfaces and classes for the servers and repository are done using dependency injection:

```
1     services.AddTransient<IUserService, UserService>();
```

This allowed for optimal separation between the responsibilities of each package. This way the **Application Core** defines the main components and logic of the application, the **Infrastructure** package handles only the communication with the database, and the **Web API**'s only concern is to expose an infrastructure to which other applications can connect. This separation allows for an easy change if the database provider is changed or other methods of data storage are considered, and it allows for API changes without having to touch the application logic.

4.1.2 Deployment architecture

The server application is built to run on a Windows machine and was deployed (Fig. 4.2) to a local *IIS Server* which can currently be accessed by any device in the same network.

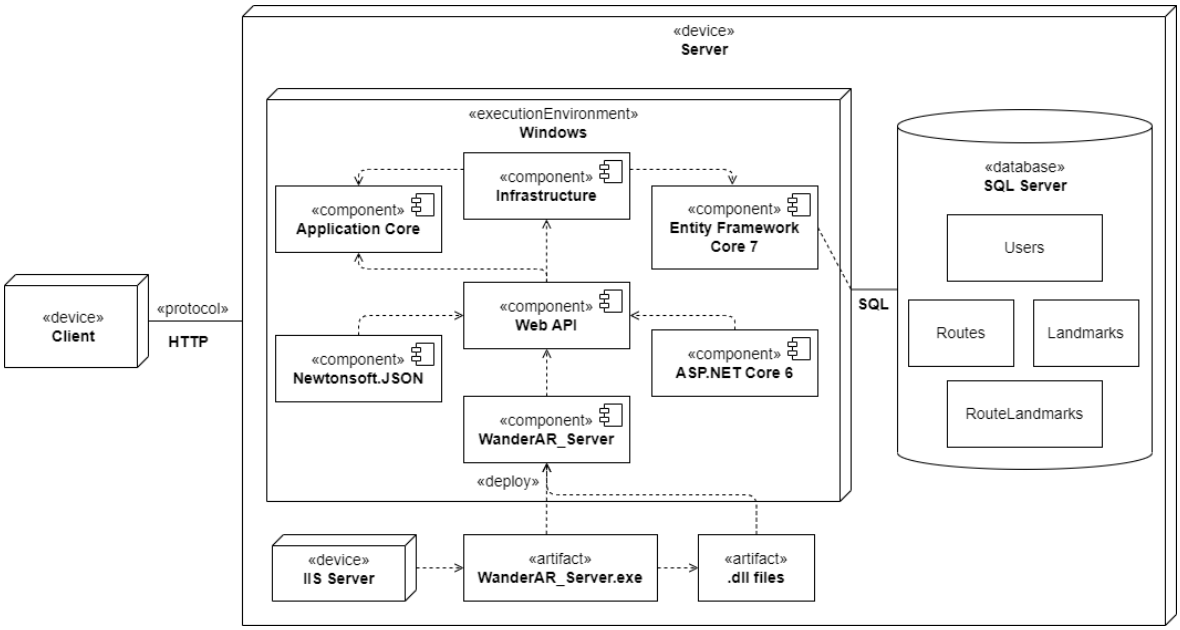


Figure 4.2: Server deployment diagram

The publishing was done using the built-in publishing mechanism of **Visual Studio** via the *File System* publishing method (Fig. 4.3).

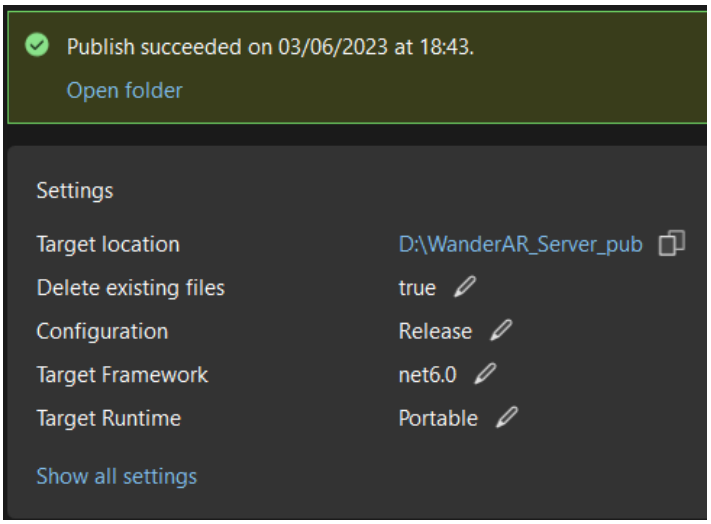


Figure 4.3: Publishing the API from Visual Studio

From there, an IIS site was configured to run on port *80* with the physical path *"D:/WanderAR_Server_pub"*. In order to grant access to the database, the *Application-Pool* under which the new IIS site was created was configured to use the **identity** of Windows user coinciding with the owner of the SQL Server database instance.

After the execution of these steps, every client that wishes to connect to the API can do so if they are connected to the same internet network by accessing: *http://<server_ip>:80/api/<endpoint>*.

4.2 Database

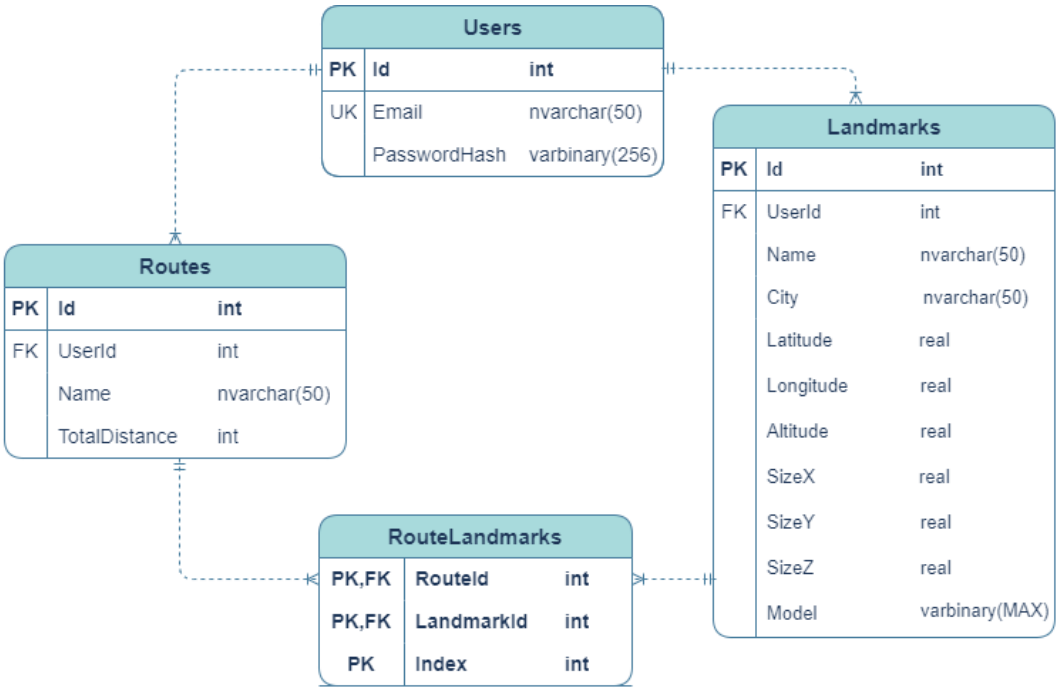


Figure 4.4: WanderAR database structure

The database was implemented using *Entity Framework Core 7* [Cor23], with the code-first approach[Kum20]. For the database server, Microsoft's **SQL Server** was chosen.

Setting up the database and the connection to it implied following the next steps:

1. Install the following required NuGet Packages in the **Infrastructure** package:
   - Microsoft.EntityFrameworkCore
   - Microsoft.EntityFrameworkCore.Tools
   - Microsoft.EntityFrameworkCore.SqlServer
2. Create the models of the database entities, including routing (relationships with other tables) and certain constraints

```
1 public class Route {
2     [Key]       // primary key constraint
3     public int Id { get; set; }
4     [Required]  // not null constraint
5     [Column(TypeName = "nvarchar(50)")]
6     public string Name { get; set; }
7     public float TotalDistance { get; set; }
8     public int UserId { get; set; }
9     // Routing
```

```
10      public User User { get; set; } // Many-to-One relation with
            Users table
11      public List<RouteLandmarks> RouteLandmarks { get; set; } //
            One-to-Many relations with RouteLandmarks table
12  }
```

Listing 4.1: Example of database entity: Route

3. Set up the DbContext (database context), which will act as a DB session through which we can query the database

```
1  public class WARDbContext : DbContext {
2      public WARDbContext(DbContextOptions options): base(options)
           {}
```

Add a DBSet for each table in the database

```
1      public DbSet<User> Users { get; set; }
2      public DbSet<Landmark> Landmarks { get; set; }
3      public DbSet<Route> Routes { get; set; }
4      public DbSet<RouteLandmarks> RouteLandmarks { get; set; }
```

Add additional constraints:

```
1  // UniqueKey constraint for User.Email
2  modelBuilder.Entity<User>()
3      .HasIndex(u => u.Email).IsUnique();
4  // Composed PrimaryKey constraint for RouteLandmarks
5  modelBuilder.Entity<RouteLandmarks>()
6      .HasKey(rl => new { rl.RouteId, rl.LandmarkId, rl.Index });
7  // ForeignKey constraint for RouteLandmarks.RouteId
8  modelBuilder.Entity<RouteLandmarks>()
9          .HasOne(rl => rl.Route)
10         .WithMany(r => r.RouteLandmarks)
11         .HasForeignKey(rl => rl.RouteId)
12         .OnDelete(DeleteBehavior.NoAction);
```

4. Add the connection string in the **appsettings.json** file

```
1      "ConnectionStrings": {
2          "Default": "Server=<server_name>;Database=WanderAR_DB;
               Trusted_Connection=True;TrustServerCertificate=True;"
3      },
```

Listing 4.2: Example of connection string with trusted connection

5. Add the DBContext to the build services in the **Program.cs** file

```
builder.Services.AddDbContext<WARDbContext>(options =>
{
    options.UseSqlServer(builder.Configuration.
        GetConnectionString("Default"));
});
```

6. Open the Package Manager Console and set the Default Project to Infrastructure. Then run the following commands:

   - **Add-Migration "[name]"**: to add a new migration
   - **Update-Database**: to update the local database

After setting up the connection, the DBContext object can be injected in repositories and used to query the database.

```
public class LandmarkRepository : ILandmarkRepository {
    private readonly WARDbContext _context;
    public LandmarkRepository(WARDbContext context)
    { _context = context; }
```

Listing 4.3: Example of database context injected in LandmarkRepository.

The database for the WanderAR application consists of three entities: **Users, Landmarks and Routes**. There are two one-to-many relationships between **Users** and **Landmarks**, and **Users** and **Routes**. In addition, there is a many-to-many relationship between **Routes** and **Landmarks**, which results in an additional link table **RouteLandmarks**. The final structure of the database with the fields and constraints is displayed in Figure 4.4.

4.3 User Data Security

The main concerns about data safety while using **WanderAR** are covered by password protection and authorization to execute operations on data owned by a user. The authentication and authorization of users are made via *JSON Web Tokens* (JWT). The token is generated on login, it contains as a claim the id of the user and expires after one month. Tokens are generated using the following packages installed in **ApplicationCore**:

- Microsoft.IdentityModel.Token
- System.IdentityModel.Token.Jwt

Data security is achieved in three steps: password hashing, authentication and authorization.

4.3.1 Password Hashing

The password is not stored as a string in the database, instead, it is being hashed using the **Hmac SHA 512** algorithm with a private key available only locally on the server, and stored as a hash that is never being sent to the client.

```
1     public static byte[] CreatePasswordHash(string password)
2     {
3         using (var hmac = new HMACSHA512(CoreConfiguration.
              HashKeyBytes))
4         {
5             return hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
6         }
7     }
```

Listing 4.4: Computing the password hash from the raw password string.

When probing user credentials, the password string received from the client is hashed and compared to the hashed password in the database.

4.3.2 Authentication

When logging in (Fig. 4.8), the user inputs the credentials: email and password. After probing, if the credentials match, the JWT is created using a local symmetric security key with the same **HmacSHA512** algorithm, the id of the user, issuer and an expiration date set one month into the future:

```
1     public static string CreateToken(User user)
2     {
3         var key = new SymmetricSecurityKey(CoreConfiguration.
              TokenKeyBytes);
4         var creds = new SigningCredentials(key, SecurityAlgorithms.
              HmacSha512Signature);
5         var token = new JwtSecurityToken(
6             claims: new List<Claim> {
7                 new Claim("ID", user.Id.ToString())},
8             expires: DateTime.Now.AddMonths(1),
9             issuer: "RL",
10            signingCredentials: creds
11            );
12        return new JwtSecurityTokenHandler().WriteToken(token);
13    }
```

The claim containing the **userId** will be used every time in the future an action on a resource that is available only to the owner is executed. The **id** will be extracted from the claim and compared to the id of the resource owner.

4.3.3 Authorization

The authorization process (Fig. 4.5) uses the same JWT tokens created in the authentication of the user and is executed for the methods marked with the **[Authorize]** attribute.

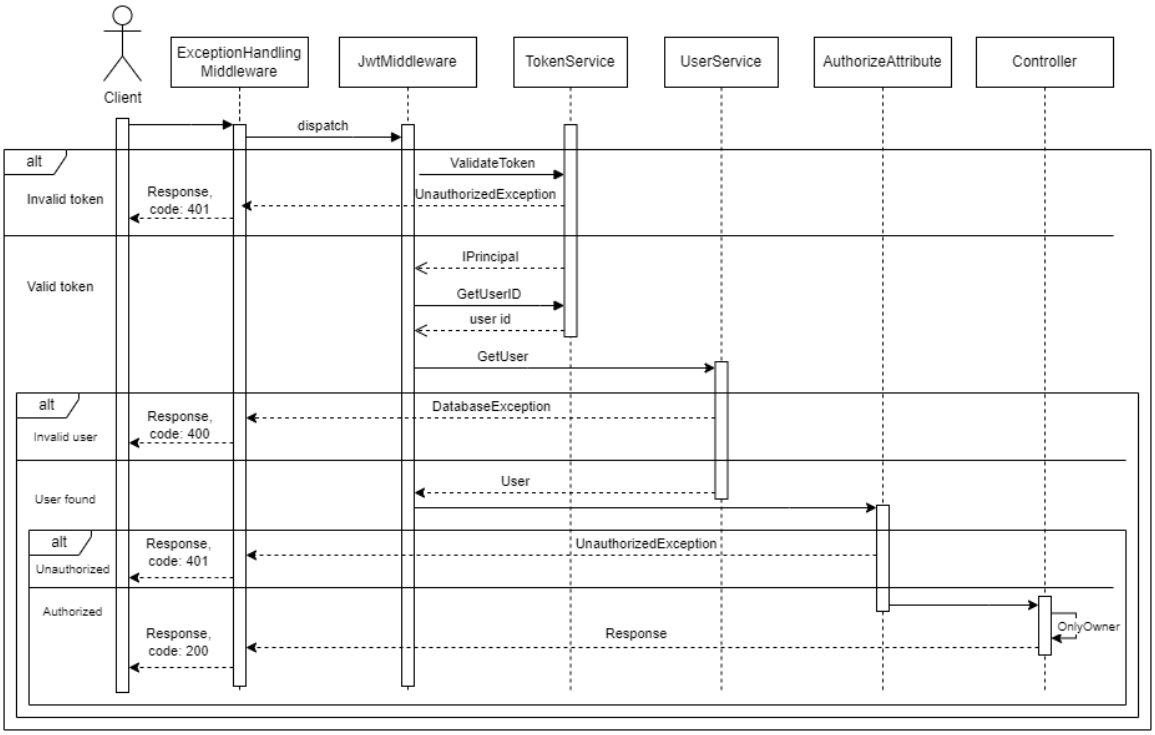


Figure 4.5: Sequence diagram for method authorization

The token can be found in the request header by the key **"authorization"**. It is extracted and validated in the **JwtMiddleware** class using **TokenService**. The userId is then decoded from the claims of the token, the user is searched in the database and, if existent, is added to the HTTPContext as a **"User"** item.

In the next step, we use the **Authorize** and **AllowAnonymous** attributes.

```
1     [HttpGet]
2     [Route("landmarks/user")]
3     [Authorize]
4     public List<NoModelLandmarkDTO> GetUserLandmarks()
```

Listing 4.5: Example on the use of "Authorize" attribute

If a method or a class is marked with **[Authorize]**, it can only be executed if a valid JW Token is supplied in the request header in the form:

```
1     authorization = Bearer <token>
```

If a controller class is marked, all the methods in the class are subject to the same behaviour. This can be overridden using **[AllowAnonymous]** attribute on the desired method.

Ultimately, when modifying resources that belong to one use, we check if the **id** of the sender (the user from the JwtMiddleware) is the same as the **id** of the owner of the resource (route, landmark).

4.4 Error Handling

Exceptions in the server provide important cues to the intended behaviour in the client application. For example, an unauthorized exception might redirect the user to a completely different flow than an authorized user. WanderAR server provides three types of custom exceptions, that all inherit from the class **BaseException**:

- **DatabaseException**: WEB error code **400**, caused by errors in the repository functions when accessing the database;
- **UnauthorizedException**: WEB error code **401**, caused by actions by unauthorized users or by authorized users on resources that do not belong to them;
- **ValidationException**: WEB error code **400**, caused by invalid data received as part of the request.

The *BaseException* class inherits from *Exception* and incorporates the WEB Error code associated with an exception type.

All the custom exceptions alongside all the other system exceptions are being caught in the *ExceptionHandlingMiddleware* and then converted into an *ExceptionDTO* that gets sent as an unsuccessful response to the client. The system exceptions will have a return code of **500** (Server error).

```
1     public class ExceptionDTO {
2         public string Type => "";
3         public string Title => "Standard error";
4         public int Status { get; set; }
5         public string TraceId => "";
6         public Dictionary<string, string[]> Errors { get; set; } }
```

Listing 4.6: Error response structure sent to client.

The client receives the type of error with a title, the WEB status code, trace id, and a dictionary containing the error messages. Multiple errors can be transmitted through a singular response.

4.5 WEB API

The API provides three controllers (Fig. 4.6): *Authentication*, *Landmark*, and *Route*. It communicates using **data transfer objects** (DTOs), serialized in JSON form with the help of the **Newtonsoft.Json** NuGet package. The base class *BaseDTO* provides infrastructure for easy conversion between the DTO and the model that it represents. The internal structure of the controllers and what functionalities they offer will be discussed in what follows.

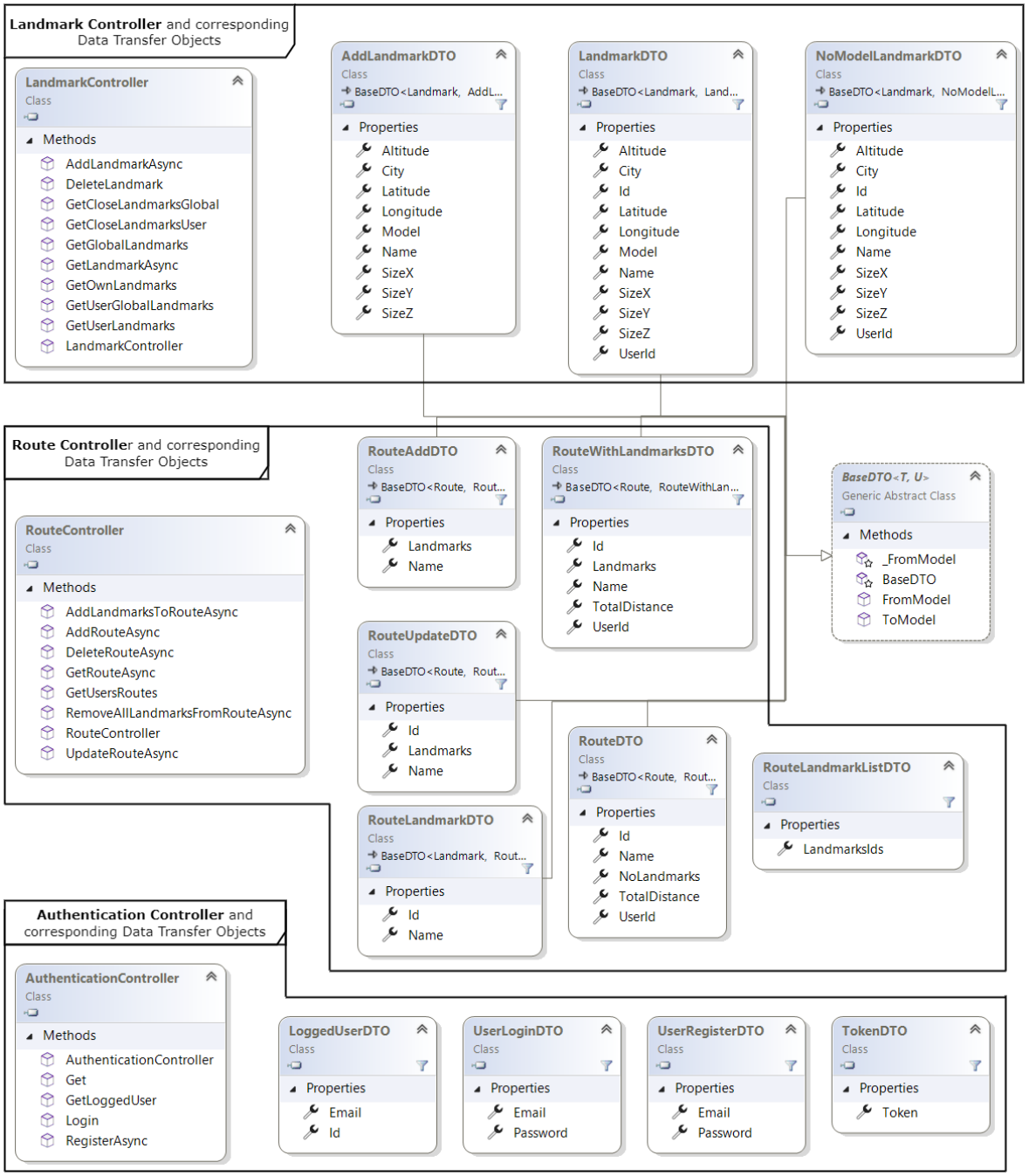


Figure 4.6: API controllers and their corresponding data transfer objects

4.5.1 Authentication Controller

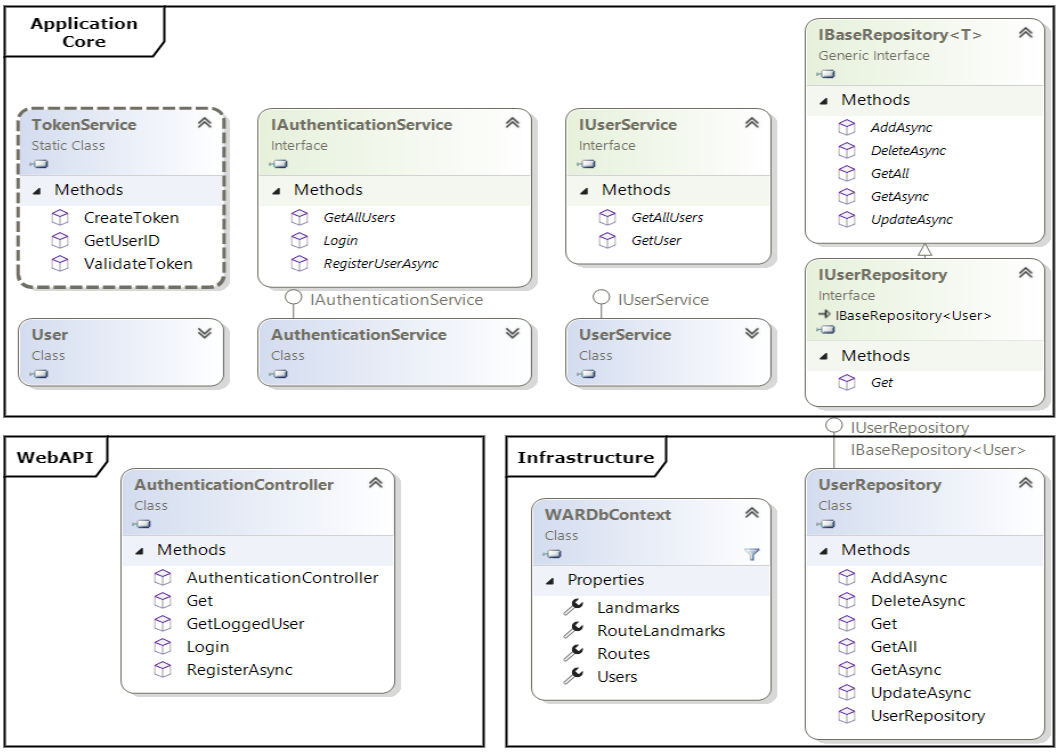


Figure 4.7: Internal structure of the Authentication Controller

The authentication controller (Fig. 4.7) handles the requests concerning the user accounts and authentication. It offers endpoints for **login** (Fig. 4.8), **register** and **account access**, the last of which is accessible only for an authenticated user that provides a valid *JWT*.

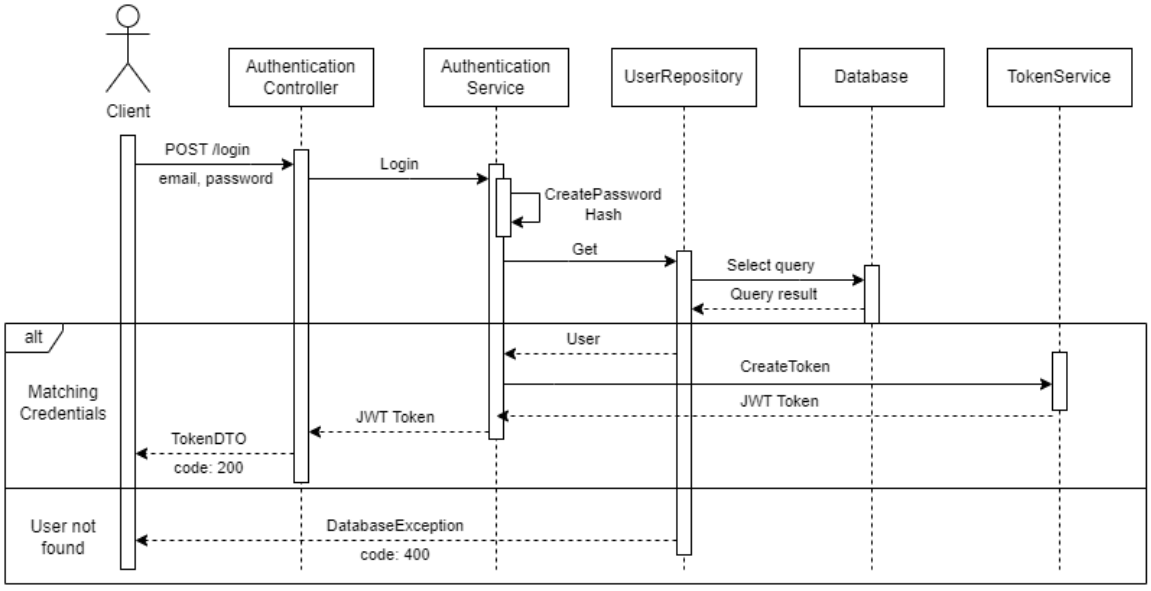


Figure 4.8: User login workflow

4.5.2 Landmark Controller

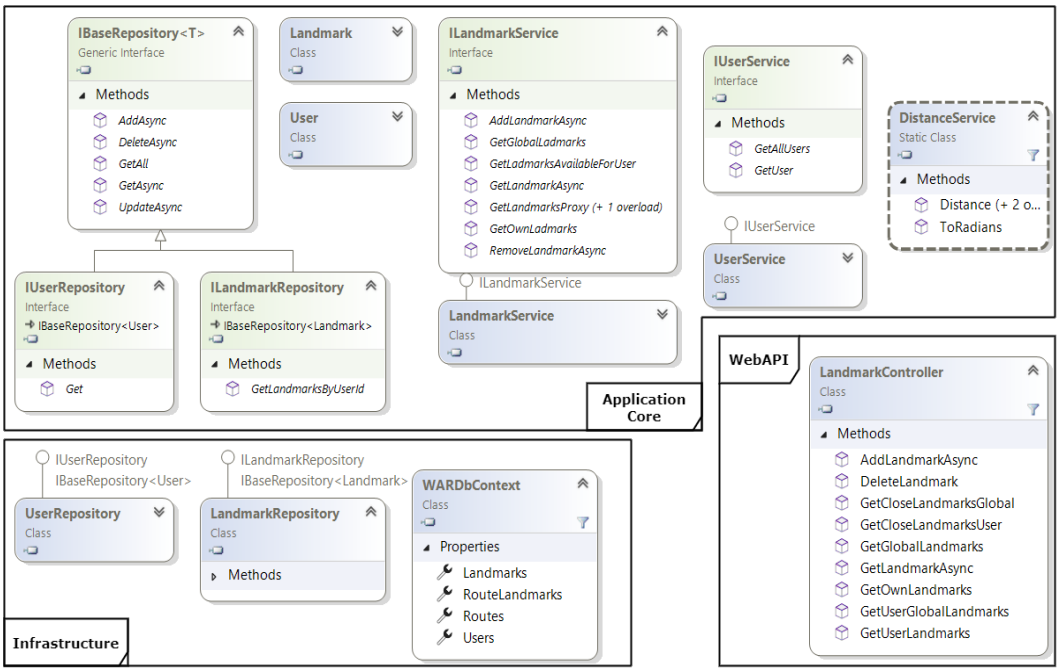


Figure 4.9: Internal structure of the Landmark Controller

The landmark controller (Fig. 4.9) provides the functionality regarding the landmark dataset. The client can access, without authentication, the landmarks provided globally, without having permission to modify them. In addition, a client that sends a valid authentication token can **visualise** (Fig. 4.10), **add** and **delete** its own landmarks.

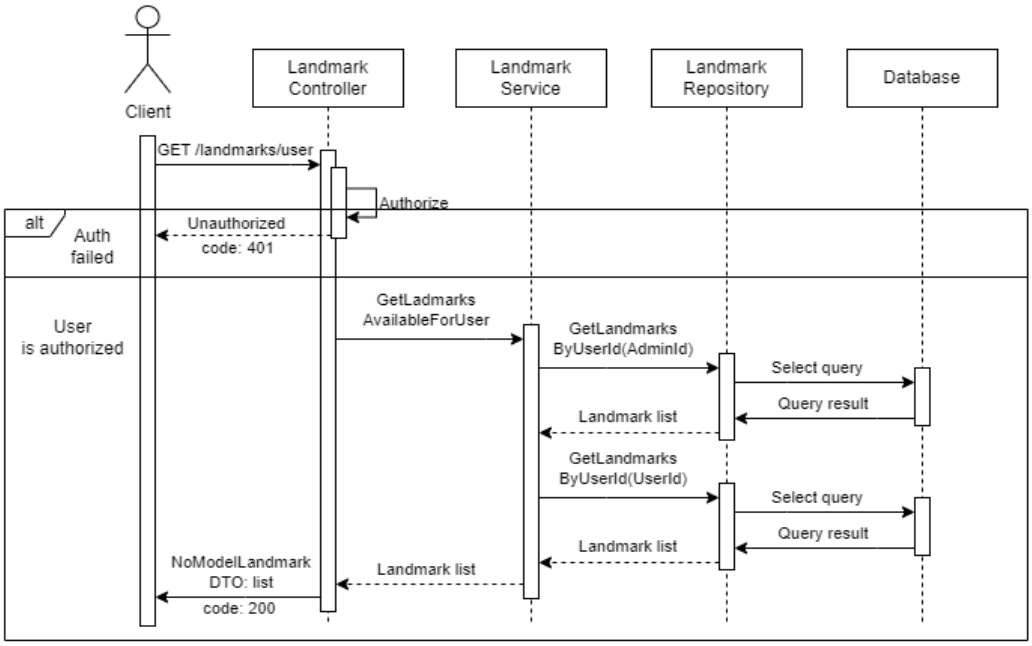


Figure 4.10: Get available landmarks workflow

4.5.3 Route Controller

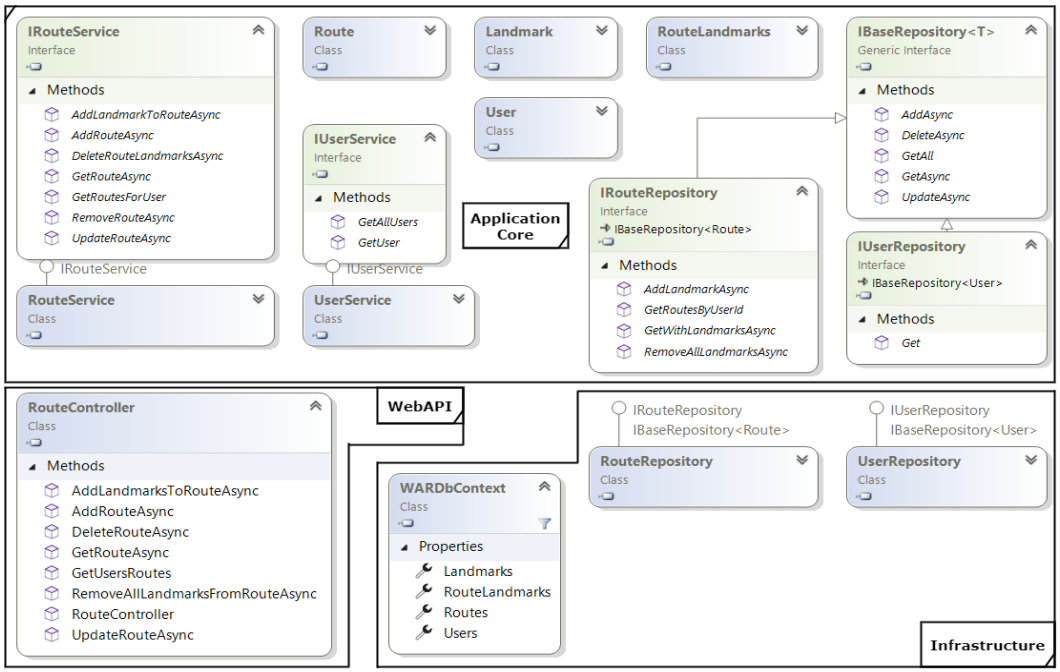


Figure 4.11: Internal structure of the Route Controller

The route controller (Fig. 4.11) is only accessible for logged users and is marked with the attribute **Authorize**. This controller offers the client the possibility to engage with its own custom routes composed of multiple landmarks.

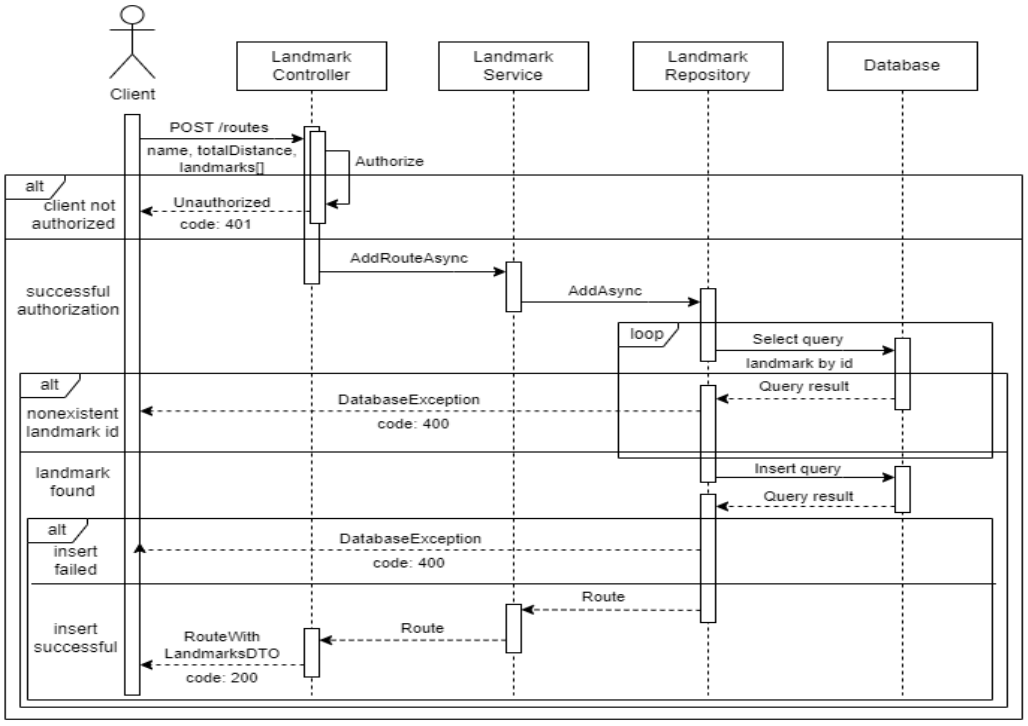


Figure 4.12: Add route workflow

It exposes endpoints for the basic CRUD operations: **add** a new route (Fig. 4.12), **read** routes, and **update** or **delete** a route that the user owns. In addition, it offers the possibility to **modify** only the list of landmarks from a route by deleting all its contents or appending another list to it. When retrieving the routes in the form of a list we have a simplified form for the landmarks collection, by using *RouteDTO* (Fig. 4.6), that only indicates their count and the total distance.

Chapter 5

WanderAR: Client Application

This chapter presents the concepts, design and implementation details of the client application, which represents the main component of this thesis. It centres around the **AR landmark navigation** functionality discussed in the previous chapters and provides the user with the means to interact with data about landmarks and routes stored on the server.

The following sections will underline the functionalities of the client application, its architecture and implementation details for the three main components:

- *Authentication* - details about the access to the user accounts and how different actions are being authorized;
- *Menu* - what does the main GUI offer, how it works and how can users access and manipulate landmarks and routes;
- *Navigation* - the different types of navigation and their particularities, how are the parameters of the navigation transmitted, details about device location and displayed landmark models.

5.1 Functionalities

5.1.1 User accounts

Each user must be authenticated in order to interact with the application. They can either create and manage their own **private** account or use the application as a **guest** with reduced functionality. An account is composed of a unique email and a password. The users can:

- *Log In*: Input the email and password and, if correct, gain access to the application, or receive an error message otherwise;

- *Log In as Guest:* Access the application as a **guest**;
- *Register*: Create a new account by providing an email and a password. If the email is already linked with another account, the user will be prompted an error message;
- *Log Out:* Logs the user out of their account.

5.1.2 Landmarks manipulation

Throughout the application, the user will be able to interact with landmarks, which can be either **global** - provided by default to all users, or **personal** - points of interest created by an authenticated user and visible only to them. The features regarding landmarks are (only users authenticated with personal accounts have access to the functionalities marked with *):

Visualise landmarks

Users should be able to visualise a list with all landmarks available to them. The system will fetch the items from the server and display them in a list with the according: *name*, *location* and *distance* from the user.

Sort and filter

The list of landmarks will be **sortable** by:

- *name*: alphabetically, ascending or descending;
- *distance*: ascending or descending.

and can be **filtered** by:

- *name*: search by partial string of the name;
- *type*: display only global, personal or all landmarks.

Add landmark*

The user can add a new personal landmark with a chosen name and the coordinates of their current, physical location.

Delete landmark*

The user can delete a landmark but **only** if it is **personal** and they own it.

5.1.3 Routes manipulation

The features in this category are **not** accessible by users authenticated as **guests**. The application provides basic CRUD operations for users to manage their **personal** routes:

Visualise routes

The user is presented a list with his personal routes, displayed by: *name*, *number of landmarks* and *total distance* (sum of the distances between consecutive landmarks in the route).

Add route

The user can add a new route. If the route is invalid, an error message will be displayed and the action will be cancelled.

Update route

The user can modify an existing route, **only** if it belongs to them. If the new route is invalid, an error message will be displayed and the action will be cancelled.

Delete route

The user can delete an existing route, **only** if they own it. In case of a failure, an error message will be displayed.

In addition, while **modifying** (add/update) a route, the system will provide the following functionalities that allow the user to manipulate the list of landmarks that compose the route:

Add landmarks to route

Choose landmarks from a list and, when ready, add all the selected landmarks to the route. The landmark can be added **only** if it is **global**, or the user owns it.

Delete landmark from route

Delete a landmark from the list corresponding to the route that is being modified.

5.1.4 Navigation

Localisation

Get the geodetic coordinates of the device (latitude, longitude, altitude) of the device and the orientation of the camera.

Load landmark model

If the selected landmark has a model, load it as a game object and apply textures, otherwise use the placeholder instead. If the model is not null but transforming it into a 3D object yields an error, display a message to the user and replace it with the placeholder.

Position landmark

Transform the known geodetic coordinates of the host device and landmark into a positioning vector for the landmark 3D model. This vector will correspond with the Cartesian coordinates of the model in a system where the camera is positioned in the origin.

Display landmark information

Display the *landmark name* and the *distance to the landmark* in an information panel. This functionality **is available only** if a single landmark is the subject of the navigation.

Toggle one/all display for route navigation

Toggle between *OneByOne* and *AllAtOnce* display modes for route navigation. When *OneByOne* is active, display only the selected landmark from the route. If *AllAtOnce* is selected, display all the landmarks in the route.
This functionality **is available only** if a route is the subject of the navigation.

Go to previous/next landmark

Display the previous/next landmark in the route.
This functionality **is available only** if a route is the subject of the navigation, and *OneByOne* is active.

Cancel navigation

End the navigation: clear the virtual objects and display only the camera view.

5.2 Architecture

The client application was developed using Unity with the afferent XR and ARCore packages. It was designed to build and run on Android and to interact with the smartphone camera and sensors to provide the augmented reality experience.

5.2.1 Layered architecture

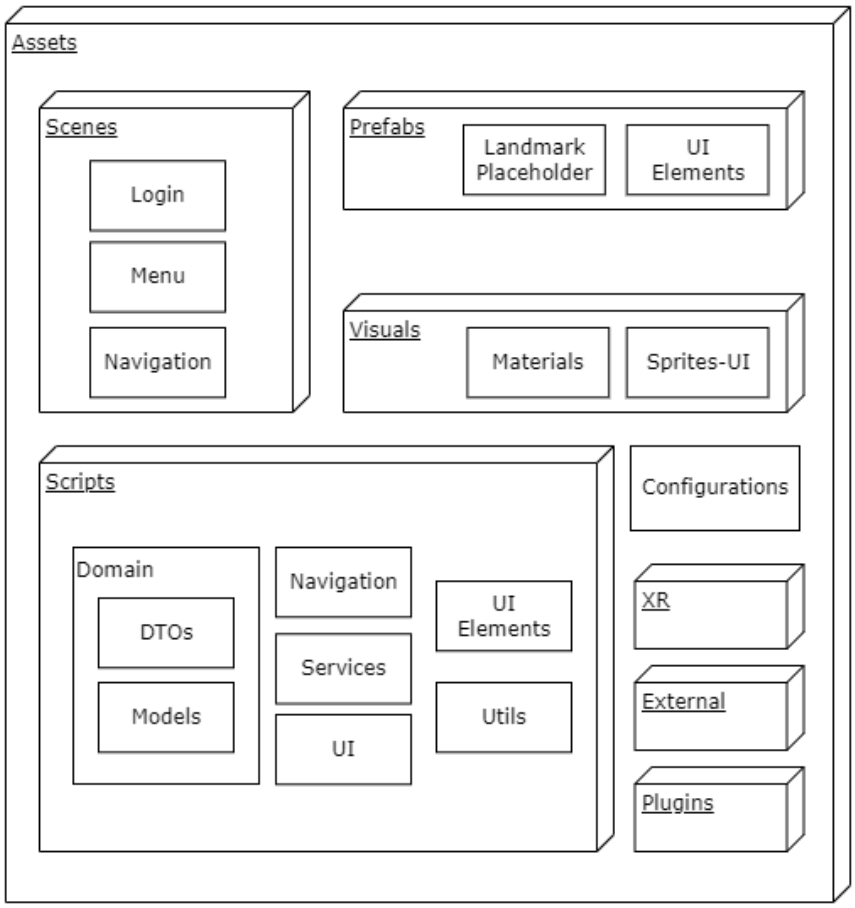


Figure 5.1: The architecture of the Unity Assets

In a Unity application, everything that models the virtual world, from scripts to high-complexity objects, is considered an *asset*. The architecture of this application (Fig. 5.1) defines a clear structure based on the major components of an AR application. It contains the *configuration* files containing session variables, token files, paths and other application global constants and configurations, and separates the rest of the assets into seven packages:

- *External* - External assets, from the asset store, used in diverse situations;
- *Plugins* - Plugins for Android and augmented reality;

- *XR* - Extended Reality elements: settings and loaders;
- *Prefabs* - Game objects used as templates: the landmark placeholder and different UI elements (list items, buttons, dropdowns, etc.);
- *Visuals* - Elements that affect the rendering of game objects: materials or sprites for the UI elements;
- *Scenes* - The scenes / layouts of the application;
- *Scripts* - Everything code related: object behaviours, UI behaviours or standard classes that form the application.

The **scenes** follow the major states of the application (Fig. 5.2): login scene - unauthorized state, menu scene - menu state, navigation scene - navigation state. The navigation between scenes happens under the same condition as the state transitions and is done using the *UnityEngine.SceneManager* class:

```
1     UnityEngine.SceneManager.LoadScene(AppScenes.NAVIGATION)
```

These scenes will each be further explored in their own section since they represent the main building blocks of the application.

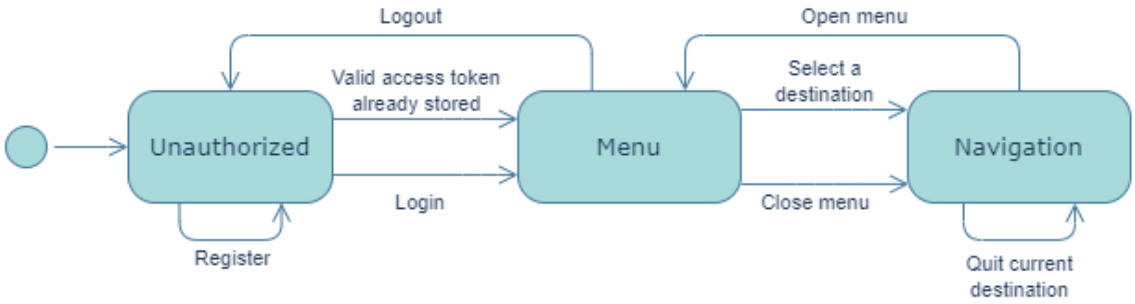


Figure 5.2: The application state diagram

Scripts are of particular importance because they define the behaviour of the application as a whole and of each component in part. They are organised in a layered architecture style with the majority controlling the GUI of the application (which we will discuss in more detail later), with two exceptions:

Domain entities

Domain entities are the base elements of the application. They are used to carry, store and present data about landmarks, routes and user accounts. The data transfer objects (DTOs) are used for communication with the server and mirror the DTOs used there.

Services

The services (Fig. 5.3) take care of the *API data access*, with one notable exception: *NotificationService*. They are responsible for making API calls using Unity's coroutine system [Tec23] and callbacks in order to gather data about users, landmarks and routes from the server while retaining the responsiveness of the application.

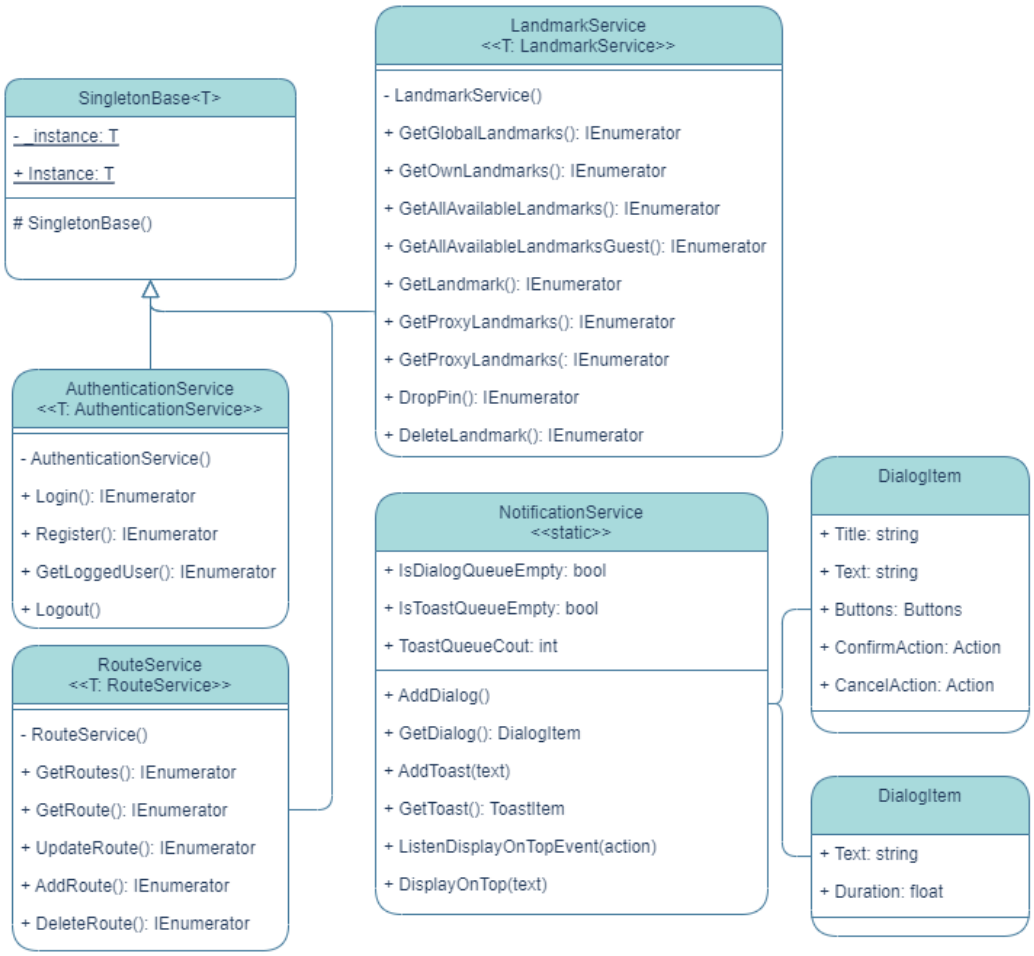


Figure 5.3: Services class diagram

The notification service for pop-ups and notifications

The *NotificationService* is used to notify the user of any changes or errors in the application. It allows for the introduction of toast messages and dialogues in queues and provides an interface for the UI elements to display them on screen (Fig. 5.4). This, paired with the *ErrorUtils* class, provides a simple framework to create and display error or information messages from anywhere within the application, without breaking the principles of layered architecture.

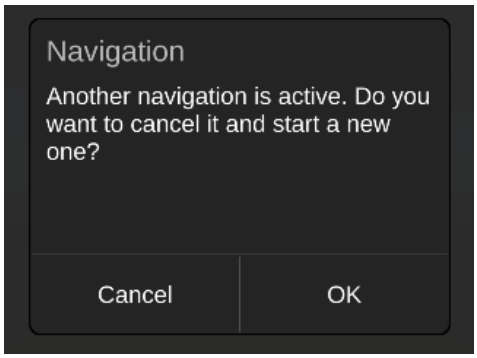


Figure 5.4: Dialog when a new navigation is started while another one is active

Testing

To ensure the reliability of the application, exploratory testing was performed in different stages of the development. This included:

- creating a new account;
- logging with a private account;
- checking all the views in the main menu and if the information is correctly loaded from the database;
- testing sorting and filtering;
- creating landmarks from the navigation view and deleting them;
- creating modifying and deleting routes;
- reordering landmarks within the route and saving or cancelling the action;
- navigating in all the possible modes: landmark, proximity, route all at once, and route one by one;
- repeatedly switching between menu and navigation scenes;
- exiting the application and checking if on reopen the user is still authorized;
- log out and repeat the step above;
- logging as a guest and repeating all the steps above testing for the limitations in functionality provided by the app in this mode.

The results were noted on a sheet of paper and the bugs were marked accordingly and fixed after the session.

5.2.2 Deployment architecture

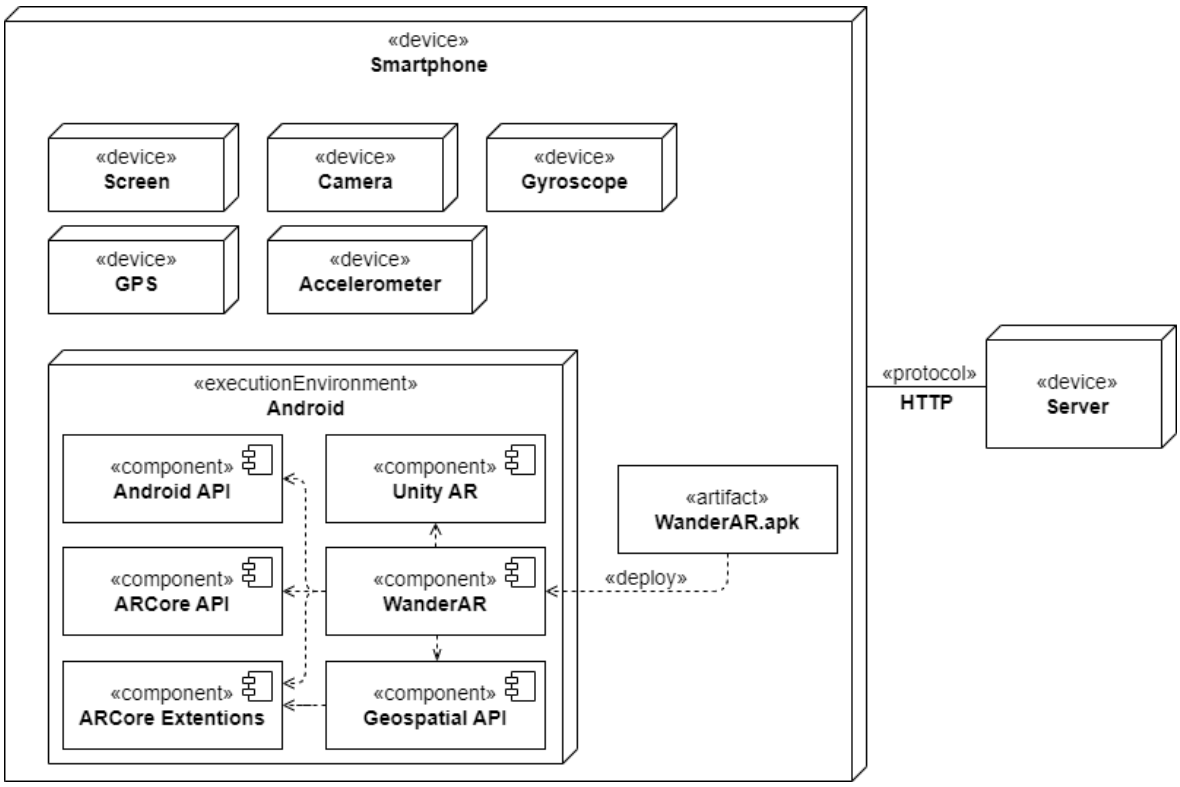


Figure 5.5: Deployment from the perspective of the **WanderAR** client application

The application is purposely built and will be supported only on Android devices. For the app to work properly, there are the following **system requirements:**

- The minimum supported Android version is **Android 9.0 'Pie'**;
- The device needs to be **ARCore compatible**[1] and have **Google Play Services for AR**[2] installed;
- The device needs to support the **Google's Geospatial API**.

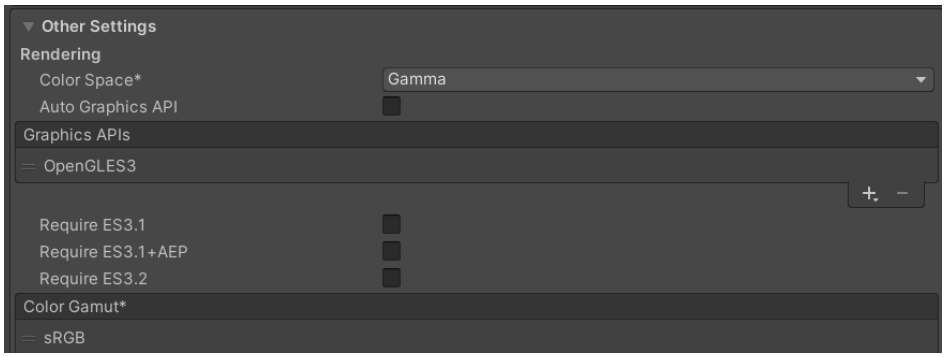
Furthermore, it requires the **"Camera access"** and **"Fine location"** permissions, and a connection to the internet is mandatory to access the server (Fig. 5.5).

Deployment steps

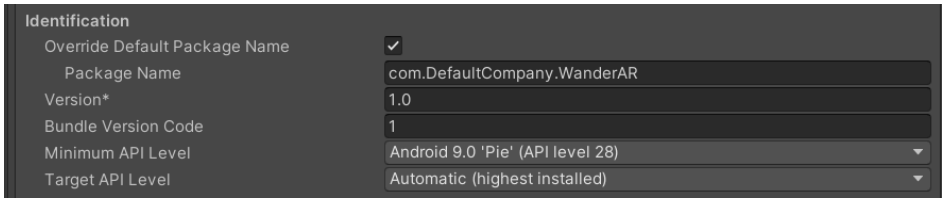
To build the application and deploy it to the mobile phone, we need to configure the following build settings:

- Set the **player settings** (Project Settings > Player) to Android and modify **rendering** to use only *OpenGLES3* Graphics API;

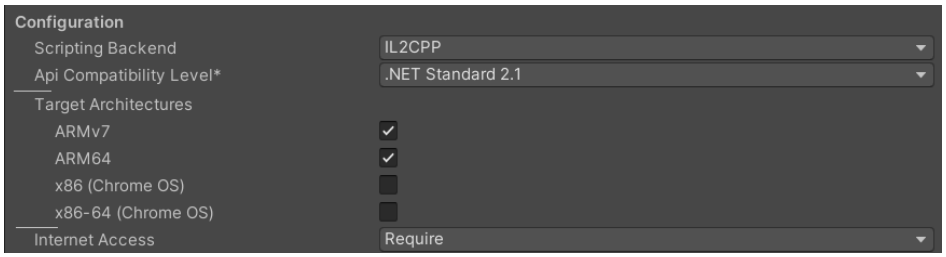
[1]https://developers.google.com/ar/devices#google_play
[2]https://play.google.com/store/apps/details?id=com.google.ar.core



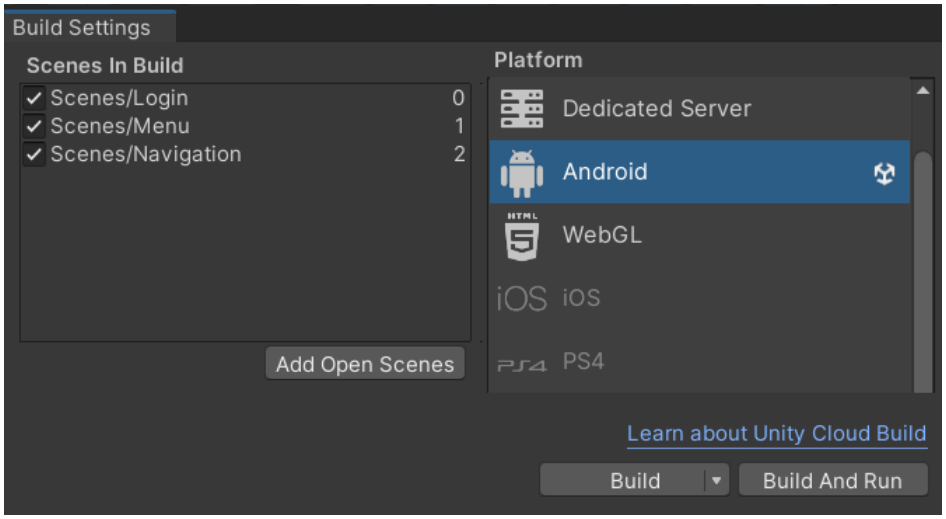
- Set the package name, version and the minimum required Android version;



- Select *IL2CPP* backend scripting, allow support for *ARMv7* and *ARM64* architectures and require *internet access*.



- Finally, in **Build Settings** switch the platform to Android, select all three scenes to be included in the build, and start the building process.



As a result, the deployment will be complete by loading the generated *APK* file to the mobile device.

5.3 Authentication

Authentication is a crucial part of the application as it defines what functionalities does the user have access to. In *WanderAR* there are three states of authentication (Fig. 5.6):

- *None*: The user does not have access to the application and is stuck in the login scene;
- *Logged*: The user has access to all the features of the application;
- *Guest*: The user has access only to a limited amount of functionalities offered by the app.

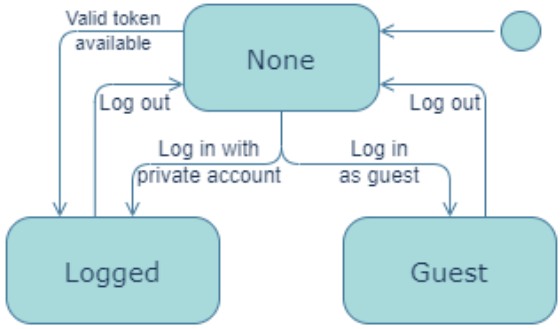


Figure 5.6: State diagram for user authentication

The authorization and authentication are done using **Json Web Tokens (JWTs)** that are being generated by the server on login, stored locally and sent as a method of authorization on the API calls that require it.

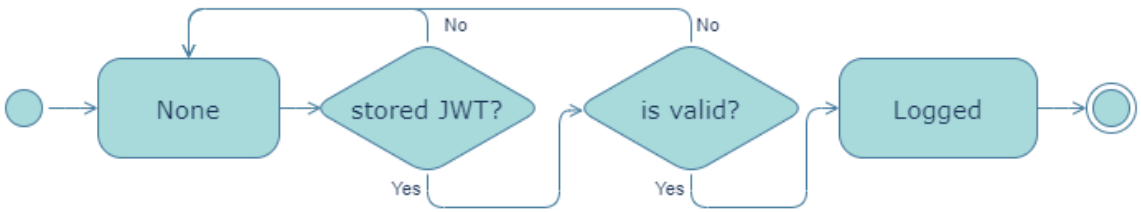


Figure 5.7: Check for a stored JWT opening the app

When the app is opened, it checks if there is a stored token (Fig. 5.7). If so, the application navigates to the menu scene with the logged user being the account

corresponding to the token. Otherwise, the user can create a new account, or log in with their credentials or as a guest. In the latter option, the application will go to the menu scene only if the credentials match an existing user.

If the authorization with a private account is successful, the API returns a **JWT** that will be stored locally. On subsequent launches of the application, the token will be used to authorize the session until it expires, or until the user explicitly logs out, action that will delete the token from memory.

5.4 Menu

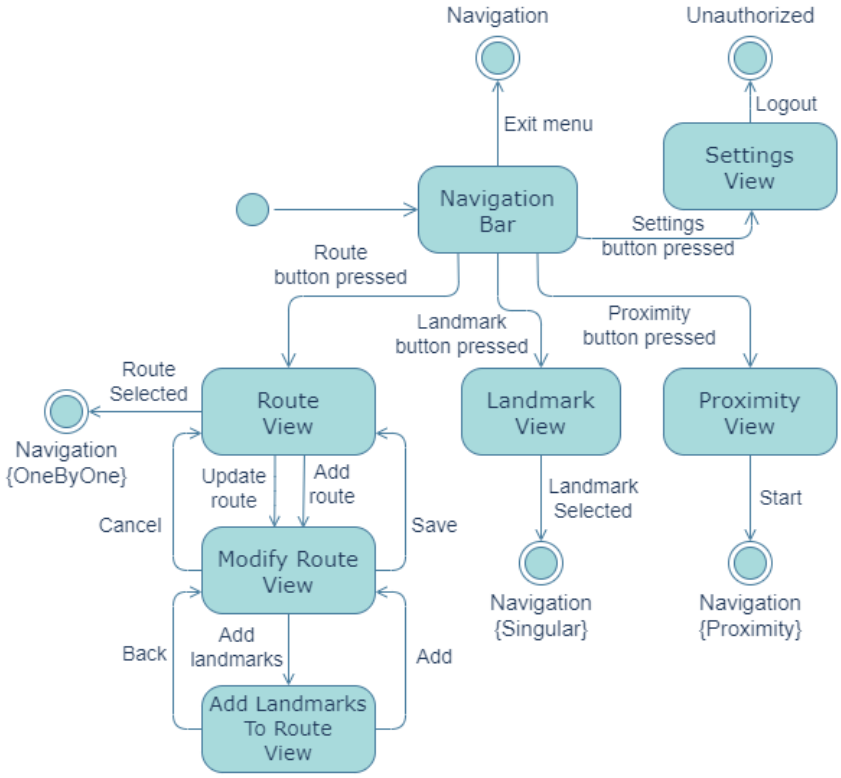


Figure 5.8: View navigation flow of the main menu

The main menu is the interface that allows the users to interact with the landmarks and routes stored on the server. There are four views that compose the main menu: settings, landmark, route and proximity, and the navigation logic between these views and their children can be observed in figure 5.8.

The *settings view* can be accessed from the top-left of the screen and displays the logged user account and provides the **logout** functionality.

The other three are grouped in a navigation bar at the bottom of the screen (5.9) and contain the majority of the functionalities regarding the UI. We will now take each of these and discuss it in more detail.

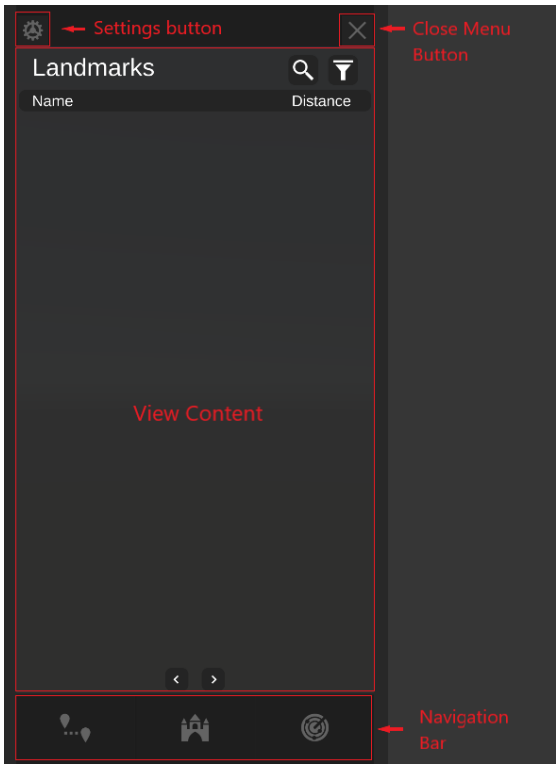


Figure 5.9: The GUI layout of the main menu

5.4.1 Landmark View

This is the landing view of the menu. It contains a list of all the available landmarks for the logged user. This list is paginated (in pages of 8 elements) and can be filtered and sorted using the options in the dropdown button (Fig. 5.10). Furthermore, landmarks can be searched by name using the search bar, and private landmarks can be deleted using a swipe-left motion.

Figure 5.10: **Dropdown** and **search bar** elements and their interaction

Selecting a landmark from this view will attempt to start a *Singular* navigation with the destination being the selected landmark. If another navigation is active, the user will be prompted to either cancel the action or stop the active navigation in order to complete it.

5.4.2 Route View

Similarly to the landmark view, this view contains functionalities for route manipulation. These functionality are available only for private accounts, thus, if the authentication state of the application is *Guest*, the user will be prompted to create an account to access them. Apart from viewing and deleting routes, there is also the possibility to update or add new ones. This action will redirect the user to a child view (Fig 5.8) that provides fields to modify an existing route or create it from scratch: setting the name and adding or removing existing landmarks from a list linked to it. The update and delete route functionalities are accessible, similar to landmark delete, by doing a swipe motion on the route object: right for update, left for delete. These functionalities are indicated by coloured markings on the corresponding edge of the tile (Fig. 5.11).

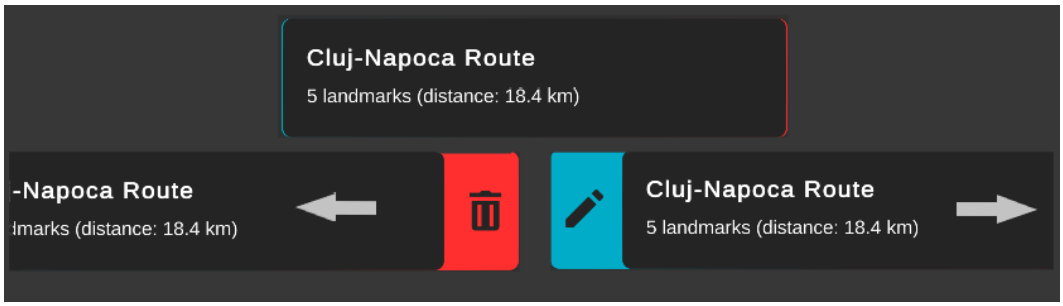


Figure 5.11: Swipe action to update or delete a route

Selecting a landmark from this view will attempt to start a *OneByOne* navigation after pulling all the available models of the landmarks in the route from the server. If another navigation is active, the user will receive the same message described at the landmark selection.

5.4.3 Proximity View

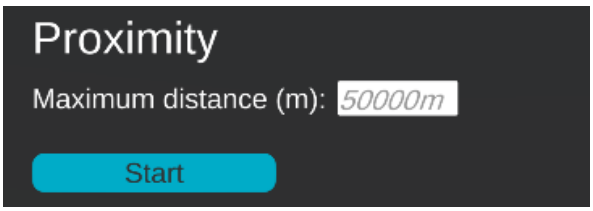


Figure 5.12: Proximity view

The final view of the main menu (Fig. 5.12) requires only a numeric input to start the navigation in the *Proximity* mode, that is the maximum range at which the landmarks will be visible on screen. The user will input the distance measured in metres, with the value in the interval $[500, 50000]$. If the number is outside of

the interval it will be automatically kept within bounds. When the start button is pressed, the navigation will start, with the same mention described above when another navigation is already active.

5.5 Navigation

The navigation component is the centre of the application and the core of the navigation approach presented in this thesis. Its focus is presenting users with the location of the landmark that is the subject of the navigation by projecting a 3D model (Fig. 5.13), such that the users will have a clear spatial view and can use their own skills to navigate to the selected target. When the model is unavailable, or the landmark is too far away to be seen due to the angular size limitation, an arbitrary sphere will be used as a marker of direction.

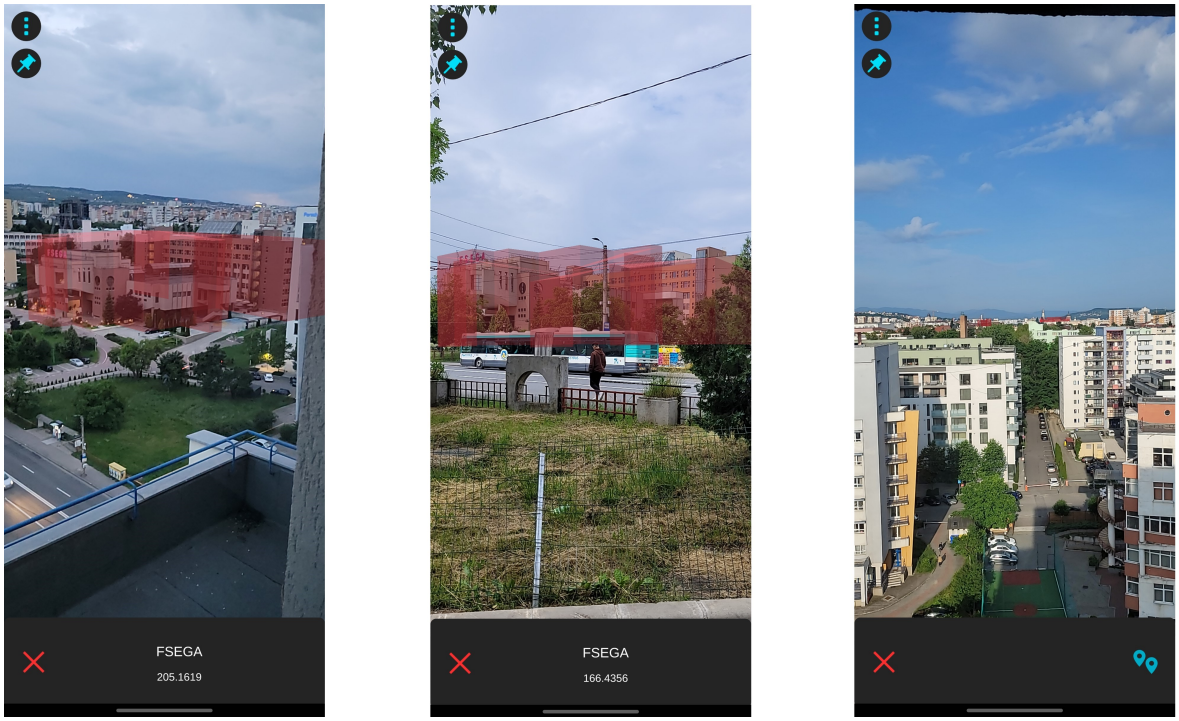


Figure 5.13: Examples of active navigation: *Singular*(left, and center) - destination **FSEGA** and *AllAtOnce* (right) - the **BBU** building and **Cluj Arena** can be observed in the distant plane

5.5.1 Navigation states

To exemplify this concept, *WanderAR* presents three innovative navigation modes, based on a single landmark, a route or the surrounding area. Each mode has a unique implementation and is linked to a navigation state (Fig. 5.14). The naviga-

tion states define the behaviour of the app: how many and which landmarks are displayed.

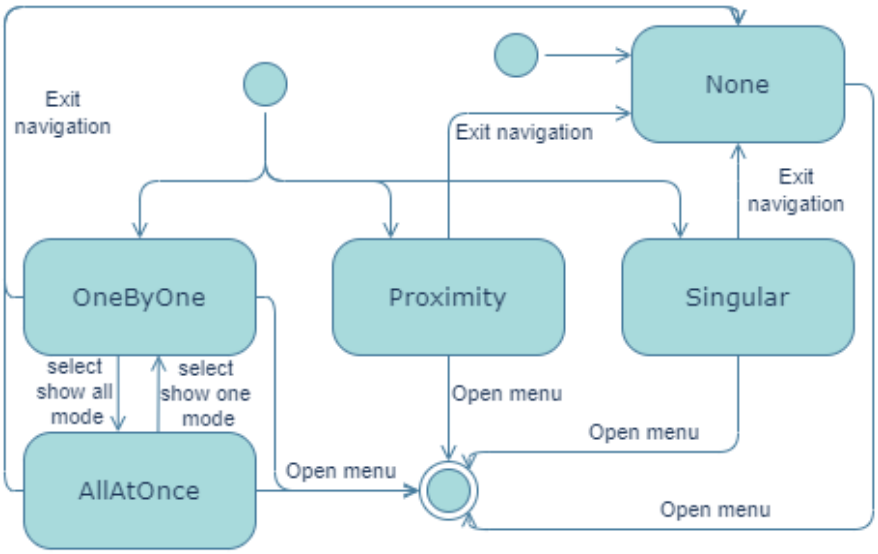


Figure 5.14: Navigation state diagram.

This state is stored in the global *AppState* class, it is changed by the actions in the menu, by cancelling navigation, or, in the case of route navigation states, toggling between states. When the navigation scene is accessed, the *NavManager* is responsible for setting up the current navigation mode and delegating the behaviour to the corresponding class (Fig. 5.16), which then handles the navigation process.

Every navigation class is modelled to fit the behaviour of a navigation state. It has a start and an end navigation method, and each one of them defines its own behaviour when: starting the navigation (*NavigationSetup*), navigating (*MoveAction*, *MoveCleanup*) and stopping the navigation (*StopCleanup*).

The states interact with the application as follows:

- *None*: This is the default state, it displays only the camera perspective with GUI;
- *Singular*: Is accessed when a landmark l is selected in the menu. The destination and sole model displayed is l. The information panel displays the name and distance to the destination (Fig. 5.13);
- *Proximity*: Is entered when a proximity navigation is started with a radius r. It displays all the landmarks available in the radius, which get updated on every iteration of the *Navigate* function;
- Route navigation states:

– *OneByOne*: When starting a navigation by selecting a route this is the state that the application enters. The target is the *current landmark* of the route. The user can choose to go to the next/previous landmark in the route or change to the *AllAtOnce* state (Fig. 5.15). The name and distance to the target are also displayed in this state;

Figure 5.15: Switching between *OneByOne* and *AllAtOnce* states

– *AllAtOnce*: In this state, the user can see all the landmarks in the route (Fig. 5.13) and can switch back to the *OneByOne* state (Fig. 5.15).

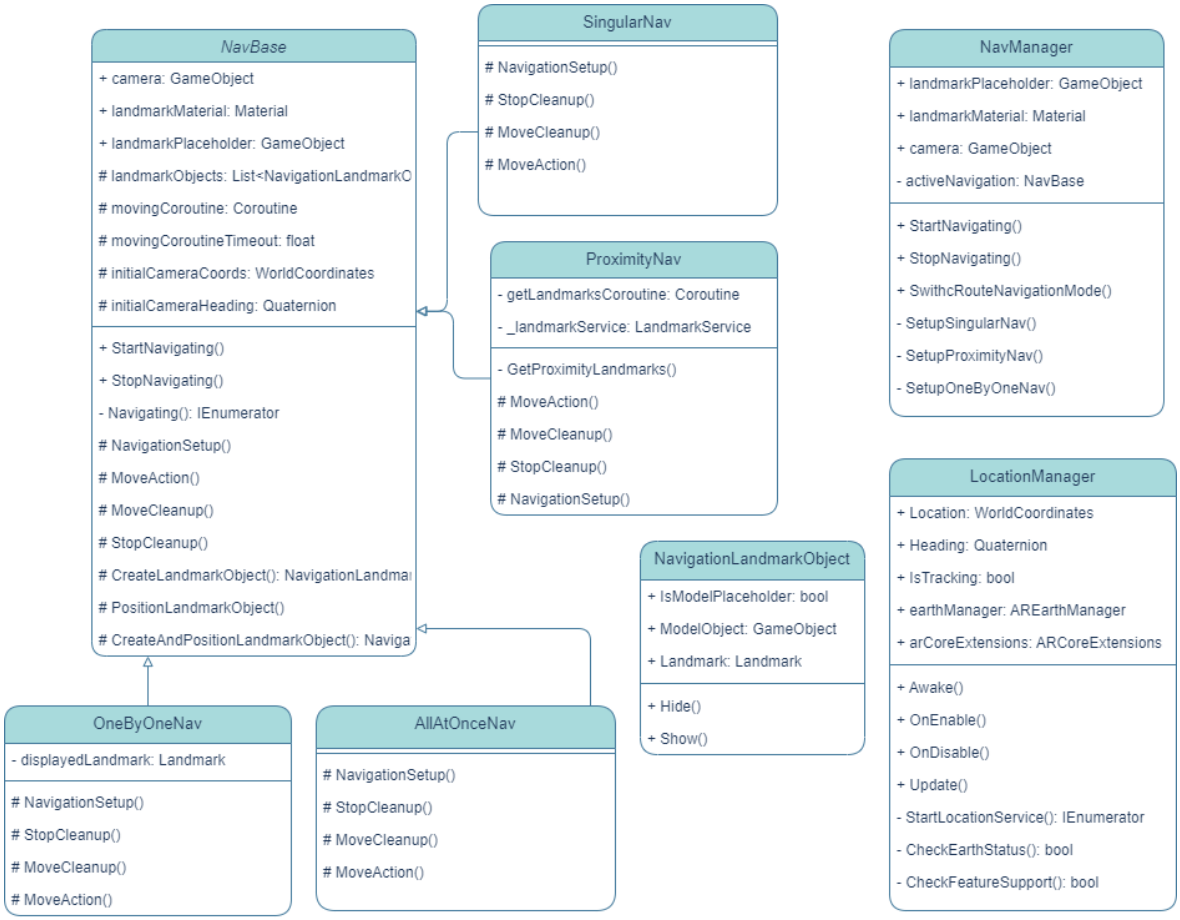


Figure 5.16: The navigation layer - classes that define the navigation behaviour

5.5.2 Localisation

Accurate localisation of the device is essential in displaying landmark models in accordance with reality and calculating the straight line distance between the user

and the desired landmark. In this implementation, we used Google's *Geospatial API* [LLC23a] for Unity ARCore applications. The Geospatial API uses a combination of smartphone GPS and sensor data with Google's *Visual Positioning System (VPS)* to obtain accurate data about the device's location and orientation.

VPS uses AI to compare camera feed with Google Maps Street View images. It works by extracting recognisable parts of the environment and matching them with Street View data to approximate the location and orientation of the mobile device, which then merges with the data from GPS and sensors to obtain accurate coordinates. However, this reliability on the camera often makes location and rotation freeze when the lightning is too dim or when pointing the camera towards a textureless object (like a white wall), in turn causing the virtual objects to also freeze.

To use the Geospatial API, the device on which WanderAR is run must support it. Furthermore, it requires "*fine-location*" permission.

```
#if UNITY_ANDROID
if (!Permission.HasUserAuthorizedPermission(Permission.FineLocation))
{
    Debug.Log("Requesting fine location permission.");
    Permission.RequestUserPermission(Permission.FineLocation);
    yield return new WaitForSeconds(3.0f);
}
#endif
```

Listing 5.1: Request fine location permission on android phone

When integrating this API in the application, we need to use the **ARCore Extensions** package [LLC23c], use the *ARCoreExtensions* script in the scene, and configure it to use an API key and accept **Geospatial** (Fig. 5.17). We will also use an *EarthManager* script that will give us all the information about the location that we need.

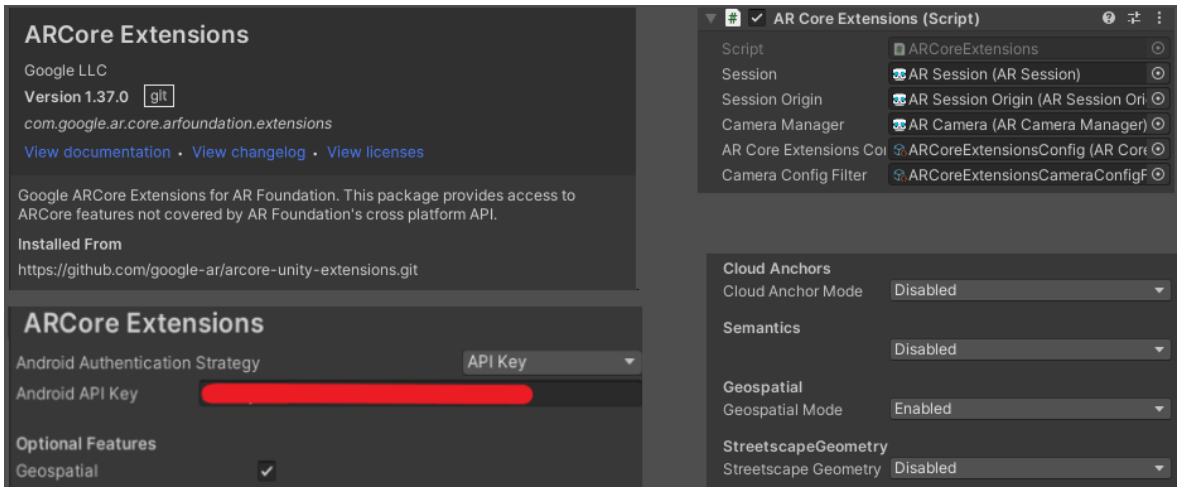


Figure 5.17: AR Core Extensions configuration for Geospatial API

To access the data returned by the Geospatial API we always need to check if the location is being tracked and then use the geospatial pose to get the coordinates and orientation of the device:

```
if (earthManager.EarthTrackingState == TrackingState.Tracking) {
    var pose = earthManager.CameraGeospatialPose;
    Location = new WorldCoordinates((float)pose.Latitude, (float)pose.
        Longitude, (float)pose.Altitude);
    Heading = pose.EunRotation;
}
```

This data is exposed by the *LocationManager* (Fig. 5.16) as a public, static properties: *Location* and *Heading* to the rest of the application.

5.5.3 Landmark models

Landmark models are the only visual cues that WanderAR offers as an aid for navigation. They are virtual representations of landmarks and must be as close to the real geometries, such that, when positioned and rotated correctly, they offer a clear indication of where the goal of the navigation is. A model does not need to be detailed but has to provide a somewhat accurate depiction, that approximately frames the object no matter the angle from which the landmark is observed.

In fact, a model is ideally as simple as possible, while retaining the key characteristics of the landmark, due to storage limitations in the database, and transfer speeds from the server to the client when a navigation begins. The models are loaded on the client device when the user starts navigating, and are rendered either instantaneously or when selected (in navigation *OneByOne* state).

There are multiple methods to get 3D models of landmarks, even creating them by hand. However, for the purpose of this thesis, they were obtained using the *Blender-OSM: OpenStreetMap and Terrain for Blender*[3] plug-in, that uses *OpenStreetMap* data to automatically generate building and terrain 3D models.

The models were saved as *.obj* and then stored and transported as byte strings and ultimately rendered using *"Runtime OBJ Importer"* asset package:

```
public static GameObject Import(byte[] modelBytes) {
    MemoryStream stream = new(modelBytes);
    return new OBJLoader().Load(stream);
}
```

After creating the GameObject corresponding to a landmark, a custom material will be applied to all its nodes to obtain the desired transparency and colour.

[3]https://prochitecture.gumroad.com/l/blender-osm

5.5.4 Landmark positioning

After creating the model for the landmark the last step that is left is positioning it in the virtual world such that it mirrors its location in reality (Fig. 5.18). When doing so, it is important to recall that the axis system used in Unity is based on the model of a screen, where X is the left-right axis, Y is down-up and Z is the depth.

To do so, we will use the coordinates of the device $D = (\varphi_d, \lambda_d, h_d)$ and the ones of the landmark $L = (\varphi_l, \lambda_l, h_l)$. So far, we are aware of the geodetic coordinates of both points, we can translate them to Cartesian using the ECEF system and the system (3.7). Now we can compute the straight-line distance between the two points:

$$dist = \sqrt{(X_l - X_d)^2 + (Y_l - Y_d)^2 + (Z_l - Z_d)^2} \tag{5.1}$$

Using this distance, we split the positioning into two methods:

Short range ($dist < 50km$): This method is based on the fact that, at such small distances, we can ignore the Earth's curvature, and uses an additional point at the same latitude as the landmark and the same longitude as the device: $P = (\varphi_l, \lambda_d, 0)$ to create a right-angled triangle. After computing the ECEF coordinates of P, it is enough to compute the distance between D and P to obtain the module of the Z value. If $\lambda_d > \lambda_l$ then the value is negative, otherwise it is positive. Similarly, we compute the value for the *X-axis*.

```
var Cecef = CoordinatesUtils.RadGeodeticToECEF(C.LatitudeRad,
    C.LongitudeRad);
var Pecef = CoordinatesUtils.RadGeodeticToECEF(P.LatitudeRad,
    P.LongitudeRad);
var dist = CoordinatesUtils.DistanceBetweenPoints(Cecef, Pecef);
return Vector3.forward * dist*(C.Latitude > P.Latitude ? -1 : 1);
```

Listing 5.2: Getting the direction vector from the camera to the point P

All that is left is computing the Y value, which will simply be the difference of altitudes: $y = h_l - h_c$.

```
Vector3 ShortRange(WorldCoordinates D, WorldCoordinates L)
{
    var P = new WorldCoordinates(L.Latitude, D.Longitude);
    var DP = GetDirectionVector(D, P);
    var PL = GetDirectionVector(P, L);
    var DL = DP + PL + Vector3.up * (L.Altitude - D.Altitude);
    return DL;
}
```

Listing 5.3: The method computing the pointing vector for small distances

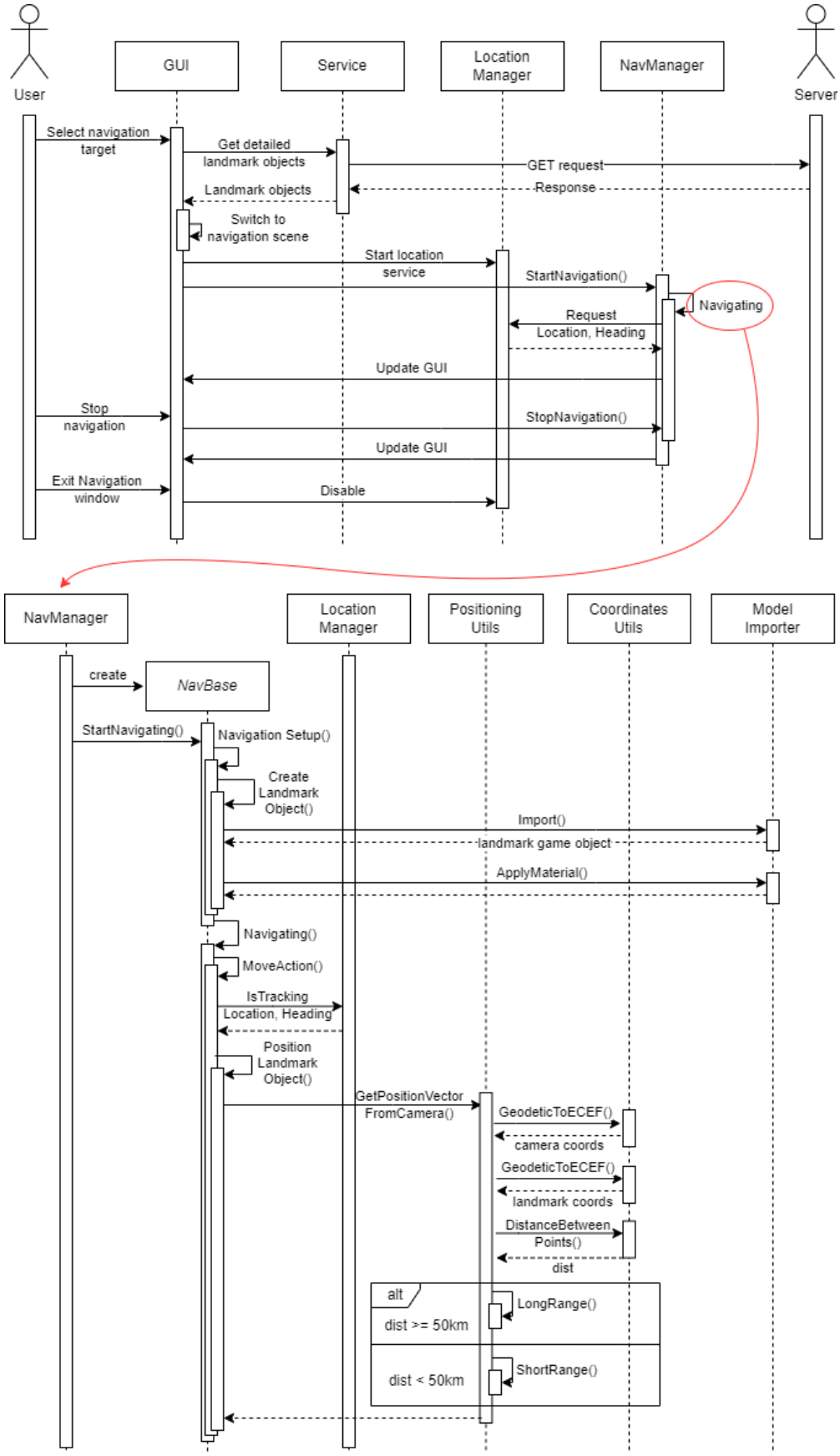


Figure 5.18: Creating the 3D landmark model, from menu to navigation.

Long range ($dist >= 50km$): This method uses the ENU system in the device point (which becomes East-Up-North because of the Unity axis system). Having the ECEF coordinates, it uses the transformation (3.7) to compute the vector from the device to the landmark. It then scales it down to 50km to keep the camera back-plane at a reasonable distance.

```
1 Vector3 LongRange(WorldCoordinates D, WorldCoordinates L)
2 {
3     var Lecef = CoordinatesUtils.GeodeticToECEF(L);
4     return CoordinatesUtils.ECEFtoEUN(Lecef, D).normalized *
          CRIT_DISTANCE;
5 }
```

Listing 5.4: The method computing the pointing vector for long distances

These calculations are made for each landmark in the navigation. The resulting positioning vectors are used as relative coordinates to the camera object, thus creating the virtual world.

The objects are positioned once the location tracking once the coordinates returned by the Geospatial API have the *OrientationYawAccurracy* and *HorizontalAccuracy* parameters smaller than a threshold. These parameters indicate the possible error in the rotation, respectively location measurements. If the errors are too high, the user is prompted to point the phone towards environmental elements that will help with the VPS measurements such as buildings, shops or signs. In this situation, the only tracking system for rotation and movement is the ARCore's built-in system, leading to a significant drop in performance. This behaviour can be very present in indoor environments or low-visibility conditions.

When the additional location data is available, the positioning of the landmarks can be performed - the model is loaded at the start of the navigation but the positioning is done only when accurate localisation is available. The object's virtual location is also adjusted during the navigation process if better yaw accuracy is achieved or a significant error can be detected: loss and regain of the *IsTracking* state due to drops in yaw or horizontal accuracy or camera freeze.

Chapter 6

Conclusions and future work

In this thesis, we have presented a new approach to navigation, a fusion between the classical concept of landmark navigation and the modern technology of augmented reality. We have seen how, in recent times, the overuse of GPS direction-based navigation raises concerns about the deterioration of the spatial orientation trait and that people are becoming too reliant on such systems. We have also explored how AR landmark navigation could represent a novel solution for these issues, by allowing users the freedom of choice, while still aiding them with achieving the end goal of the journey and doing so in an immersive fashion that fuels the curiosity and drive to explore and discover new environments.

Further on, we laid the foundations and explored a possible implementation of such technology and a few ideas on how AR landmark navigation could be used. Built using Unity's XR platform in conjunction with the AR Core framework, the *WanderAR* Android application exemplifies the theoretical aspects of this approach, an application that successfully locates and indicates the desired landmarks based on camera and sensor data. We presented in detail the general architecture and implementation but also specific features like localisation, landmark representation and how we can position landmarks such that the user sees them at an appropriate location and distance.

This implementation proved to outline some limitations, which might be improved by future research and improvements in the available technology. Areas such as localisation and positioning will be improved by developments in AI-assisted localisation algorithms and physical sensors for mobile devices.

WanderAR is supplied by a web server and an API that provides access and storage for information about landmarks, routes and users. This method of storing data keeps the memory cost of the application low, to deal with the storage limitations of smartphones. It also allows users to create their own materials and experiences, and carry them around no matter the used device.

Future work

In the future, this concept could be developed to be an integral part of everyday navigation applications. It can be easily paired with conventional AR navigation and has the advantage of providing users with a clear visual reference while navigating. The server-client architecture that was exemplified in this thesis facilitates scalability and avoids some of the limitations regarding the client device. This way, we might see this technology paired with AR glasses to offer a seamless, immersive navigation experience.

Following the developments in indoor localisation technology, the concept presented in this thesis can be adapted to aid small-scale interactive maps for museums, zoos or other enclosed tourist attractions, following the model of route navigation presented in this thesis.

Furthermore, the methods described in this thesis to position 3D models in a virtual space in relation to the user are not limited to landmarks. An example relevant to the current trends in navigation is using an adaptation of this system to display road representations through the car windshield. Thus, drivers will have a clear understanding of the road they have to follow or how sharp a turn will be, and adjust their speed, even without having a clear line of sight of the road. This can be used in harsh weather conditions with reduced visibility to improve safety on the roads. Moreover, in the future, we could see this technology display even other cars or hazards on the road, thus eliminating blind spots caused by other cars or environmental items.

With the demands in the industries based on localisation and navigation, and the involvement of important actors in the scene like Google or Apple, imagination is the only limit to the possibilities that augmented reality presents. We believe that the technology presented in this thesis and exemplified through WanderAR can be a building block for many future innovations.

Bibliography

[Age14] National Geospatial-Intelligence Agency. *Department of Defense World Geodetic System 1984: Its Definition and Relationships with Local Geodetic Systems*. 2014.

[BGB20] Răzvan Gabriel Boboc, Florin Gîrbacia, and Eugen Valentin Butilă. The Application of Augmented Reality in the Automotive Industry: A Systematic Literature Review. *Applied Sciences*, 10(12), 2020.

[CHYJ18] Kuang-Hsuan Chen, Yu-Wei Hsu, Jing-Jung Yang, and Fu-Shan Jaw. Evaluating the specifications of built-in accelerometers in smartphones on fall detection performance. *Instrumentation Science & Technology*, 46(2):194–206, 2018.

[Cor23] Microsoft Corporation. ASP.NET Core MVC with EF Core - tutorial series. `https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/?view=aspnetcore-6.0`, 2023. accessed 17 May 2023.

[CZL+20] Changhao Chen, Peijun Zhao, Chris Xiaoxuan Lu, Wei Wang, Andrew Markham, and Niki Trigoni. Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference. *IEEE Internet of Things Journal*, 7(5):4431–4441, 2020.

[DWQ+21] Weihua Dong, Yulin Wu, Tong Qin, Xinran Bian, Yan Zhao, Yanrou He, Yawei Xu, and Cheng Yu. What is the difference between augmented reality and 2D navigation electronic maps in pedestrian wayfinding? *Cartography and Geographic Information Science*, 48(3):225–240, 2021.

[Fin22] Mark Finch. Understanding the SessionOrigin. `https://learn.microsoft.com/en-us/mixed-reality/world-locking-tools/documentation/concepts/advanced/sessionorigin`, 2022. accessed 03 May 2023.

[GAB20] Mohinder S Grewal, Angus P Andrews, and Chris G Bartone. *Global navigation satellite systems, inertial navigation, and integration*. John Wiley & Sons, 2020.

[GSI19] Eg Su Goh, Mohd Shahrizal Sunar, and Ajune Wanis Ismail. 3D object manipulation techniques in handheld mobile augmented reality interface: A review. *IEEE Access*, 7:40581–40601, 2019.

[GZL+23] George Grouios, Efthymios Ziagkas, Andreas Loukovitis, Konstantinos Chatzinikolaou, and Eirini Koidou. Accelerometers in Our Pocket: Does Smartphone Accelerometer Technology Provide Accurate Data? *Sensors*, 23(1), 2023.

[Hei20] Guenter W Hein. Status, perspectives and trends of satellite navigation. *Satellite Navigation*, 1(1):22, 2020.

[HOM+18] Lukáš Hejtmánek, Ivana Oravcová, Jiří Motýl, Jiří Horáček, and Iveta Fajnerová. Spatial knowledge impairment after GPS guided navigation: Eye-tracking study in a virtual town. *International Journal of Human-Computer Studies*, 116:15–24, 2018.

[Hua19] Guoquan Huang. Visual-inertial navigation: A concise review. In *2019 international conference on robotics and automation (ICRA)*, pages 9572–9582. IEEE, 2019.

[KGR21] Tim Kuhlmann, Pablo Garaizar, and Ulf-Dietrich Reips. Smartphone sensor accuracy varies from device to device in mobile research: The case of spatial orientation. *Behavior research methods*, 53:22–33, 2021.

[Kum20] Venkatesh Kumar. Using Entity Framework Core Code First Approach. `https://www.c-sharpcorner.com/article/using-entity-framework-core/`, 2020. accessed 17 May 2023.

[LLC22] Google LLC. Overview of ARCore and supported development environments. `https://developers.google.com/ar/develop`, 2022. accessed 13 June 2023.

[LLC23a] Google LLC. Build global-scale, immersive, location-based AR experiences with the ARCore Geospatial API. `https://developers.google.com/ar/develop/geospatial`, 2023. accessed 11 June 2023.

[LLC23b] Google LLC. Geospatial Creator. `https://developers.google.com/ar/geospatialcreator/intro`, 2023. accessed 13 June 2023.

[LLC23c] Google LLC. Getting started with ARCore Extensions for AR Foundation. `https://developers.google.com/ar/develop/unity-arf/getting-started-extensions`, 2023. accessed 11 June 2023.

[LSL22] Jia Liu, Avinash Kumar Singh, and Chin-Teng Lin. Using virtual global landmark to improve incidental spatial learning. *Scientific Reports*, 12(1):6744, 2022.

[MGJ+00] Eleanor A. Maguire, David G. Gadian, Ingrid S. Johnsrude, Catriona D. Good, John Ashburner, Richard S. J. Frackowiak, and Christopher D. Frith. Navigation-related structural change in the hippocampi of taxi drivers. *Proceedings of the National Academy of Sciences*, 97(8):4398–4403, 2000.

[MR07] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3565–3572. IEEE, 2007.

[MvDSJ+21] Y Jade Morton, Frank van Diggelen, James J Spilker Jr, Bradford W Parkinson, Sherman Lo, and Grace Gao. *Position, navigation, and timing technologies in the 21st century: Integrated satellite navigation, sensor systems, and civil applications, volume 1*. John Wiley & Sons, 2021.

[MZS+06] Stefan Münzer, Hubert D. Zimmer, Maximilian Schwalm, Jörg Baus, and Ilhan Aslan. Computer-assisted navigation and the acquisition of route and survey knowledge. *Journal of Environmental Psychology*, 26(4):300–308, 2006.

[Ran20] Mirko Ranieri. A new sense of direction with Live View. `https://blog.google/products/maps/new-sense-direction-live-view/`, 2020. accessed 07 June 2023.

[RCRSC19] Ian T. Ruginski, Sarah H. Creem-Regehr, Jeanine K. Stefanucci, and Elizabeth Cashdan. GPS use negatively affects environmental learning through spatial transformation abilities. *Journal of Environmental Psychology*, 64:12–20, 2019.

[Shi05] Eun-Hwan Shin. Estimation techniques for low-cost inertial navigation. 2005.

[SZG+20] Chong Shen, Yu Zhang, Xiaoting Guo, Xiyuan Chen, Huiliang Cao, Jun Tang, Jie Li, and Jun Liu. Seamless GPS/inertial navigation system based on self-learning square-root cubature Kalman filter. *IEEE Transactions on Industrial Electronics*, 68(1):499–508, 2020.

[SZHP11a] J. Sanz Subirana, J.M. Juan Zornoza, and M. Hernández-Pajare. Conventional Terrestrial Reference System. `https://gssc.esa.int/navipedia/index.php/Conventional_Terrestrial_Reference_System`, 2011. accessed 09 June 2023.

[SZHP11b] J. Sanz Subirana, J.M. Juan Zornoza, and M. Hernández-Pajare. Ellipsoidal and Cartesian Coordinates Conversion. `https://gssc.esa.int/navipedia/index.php/Ellipsoidal_and_Cartesian_Coordinates_Conversion`, 2011. accessed 09 June 2023.

[SZHP11c] J. Sanz Subirana, J.M. Juan Zornoza, and M. Hernández-Pajare. Transformations between ECEF and ENU coordinates. `https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates`, 2011. accessed 09 June 2023.

[Tec23] Unity Technologies. Coroutines. `https://docs.unity3d.com/Manual/Coroutines.html`, 2023. accessed 10 June 2023.

[TS05] Philippe Terrier and Yves Schutz. How useful is satellite positioning system (GPS) to track gait parameters? A review. *Journal of neuroengineering and rehabilitation*, 2:28, 10 2005.

[YKA22] Zachary F. Yount, Steven J. Kass, and James E. Arruda. Route learning with augmented reality navigation aids. *Transportation Research Part F: Traffic Psychology and Behaviour*, 88:132–140, 2022.