Technische Universität Berlin

Development of an interactive digital campus map with navigation functionality for mobile devices

by

Patrick Christoph Zdanowski

Matriculation Number 0410378

A thesis submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Service-centric Networking

Bachelor's Thesis

March 7, 2024

Supervised by:
Prof. Dr. Axel Küpper

Assistant supervisor:
Dr. Dr. Chuck Norris

SNET
SERVICE-CENTRIC NETWORKING

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, March 7, 2024 Patrick Christoph Zdanowski

Abstract

In this thesis, we show that lorem ipsum dolor sit amet.

Zusammenfassung

Hier kommt das deutsche Abstract hin. Wie das geht, kann man wie immer auf Wikipedia nachlesen `http://de.wikipedia.org/wiki/Abstract`...

Contents

1 Introduction

1.1 Context and background

With over 60 different buildings, more than 35000 students and an area of over 600,000 square meters, TU Berlin's campus in Charlottenburg is one of the biggest continuous campuses in Europe. While studying is the main reason for most people to be on campus, learning and visiting courses are not the only tasks that take place on it. Campus Charlottenburg gives people the opportunity to learn and research, but also to connect to other like-minded people, eat meals in one of the cafeterias or bakeries, as well as to participate in one of the many social events that take place on it. Campus Charlottenburg is also the working place of over 7000 persons, including professors, students and researchers.

With its enormous amount of important facilities for students, families and other members of the university, campus Charlottenburg plays a central role in the lives of many people.

1.2 Motivation

TU Berlin's enormous size and complexity come with several difficulties for people working and studying on campus Charlottenburg, including an overwhelming amount of information about and around the campus as well as problems with navigation across it. New members of TU Berlin have problems localizing certain buildings and are overwhelmed by the amount of information the campus provides.

A working, manageable and modern digital information culture is one of the most significant key factors for overcoming those difficulties and for successful study, work and life on campus Charlottenburg. Such a system should offer an easy and intuitive way of accessing and managing information about the university to all of TU Berlin's members.

Some online resources and systems attempt to enhance the digital information culture including websites provided by TU Berlin and Studierendenwerk Berlin as well as a campus map and digital navigation systems for mobile devices. Nevertheless, there are several problems with the current landscape of web resources and platforms provided by TU Berlin and Studierendenwerk Berlin:

One of them is the fact that there is currently no fitting digital solution for navigation on the campus. Students currently have two options when choosing a tool to navigate across it: On the one hand, TU Berlin provides a downloadable image of campus Charlottenburg, which gives a basic aerial overview of the whole campus. Although this solution is straightforward and understandable, it only provides the possibility to navigate manually over the campus. It also does not take full advantage of the digital tools mobile devices and personal computers offer us.

On the other hand, there are several external digital navigation system providers such as Google or Apple Maps. Those systems take advantage of the digital tooling we have and provide a modern and mobile way of navigation. The main problem with these systems is the fact that they do not contain a detailed mapping of campus Charlottenburg. Searching for several buildings is difficult/impossible and since these systems are not profound for most of the paths on campus, proposed routes are often based on public routes around the campus rather than the faster ones inside of it.

Another problem is the fact that the amount and size of provided platforms is, in itself, overwhelming and complicated. The information about campus Charlottenburg is spread across the internet, instead of bundled. There is no central entity, that contains all information. The fact that most of the information is provided in the form of websites has another consequence for members of TU Berlin: Looking up information fast and intuitive on mobile devices, one of the most used digital tools in the present time, is often slow, unintuitive and unnecessarily complicated.

1.3 Proposed solution

This bachelor's thesis tries to investigate the current digital information culture at TU Berlin and wants to solve information retrieval and navigation difficulties by providing an implementation of a digital information and navigation system for mobile devices.

Both of these features use an underlying offline map of campus Charlottenburg, which provides a base layer for the main screen of the app. All other features and interface elements are therefore stacked in separate layers on top of it.

1.3.1 Campus map

A custom map of campus Charlottenburg represents the main element of the app. It comes completely bundled with the application for offline usage and displays the most relevant information about the structure of campus Charlottenburg, e.g., buildings, cafeterias, green areas and pathways across the campus.

1.3.2 Navigation system

A fast, reliable and offline-usable navigation system overlays the campus map and helps users find their way across the campus. It contains all the geodata and points of interest (POI) of the campus and can reliably calculate the fastest route between them. It also integrates the GPS functionality of the mobile device, to consider the current user location while navigating.

1.3.3 Information layer

The second layer on the map displays information about the campus. It fetches and parses them automatically from publicly available web resources and maps the information onto their respective POI.

2 Related Work

2.1 GPS

The global positioning system (GPS) as described in [1] and [2] is the modern standard for determining the position of a user on Earth. It is widely used across several different domains including navigation, tracking functionalities, military usage and other location-based applications. At least since the rise of mobile devices equipped with GPS capabilities the majority of people regularly use GPS within several mobile apps and services.

The basis of GPS technology consists of a synchronized satellite network, which constantly broadcasts status information about the respective position, orbit and time of its members. GPS devices calculate their latitude, longitude and altitude by receiving data from at least four satellites. After data is transmitted, the signal propagation time is used for lateriation, from which the position of the device can be determined. [3] presents the mathematical background for such position determination: Based on the signal runtime, the distance R_i between each satellite "i" and the receiver is calculated. With the unknown receiver position (x, y, z) and the known satellite position (x_i, y_i, z_i) the corresponding distance can be expressed as follows: $(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2 = R_i$. This set of (at least 3) equations can be then solved for the receiver position (x, y, z).

Since a correct measurement of signal runtime is essential (errors of 1 ms result in uncertainties of around 300 km [2]) and consumer-friendly GPS devices usually only contain a simple quartz-based clock, error estimation and correction systems are needed to estimate the exact time.

Depending on the environment and scenario in which GPS is used, several optimizations and improvements can be made. Since this thesis implements a navigation system for mobile devices, smartphone-focused implementation and usage techniques are particularly interesting.

One important example of such an improvement is the Google fused location provider API. This API is specifically designed for usage on mobile devices and improves the efficiency and accuracy of location retrieval by combining several internal smartphone sensors such as GPS and WiFi [4].

2.2 OSM mapping service

OpenStreetMap (OSM) is a community-driven open-source mapping service that provides annotated digital maps for most of the countries in the world. It was initiated in 2004 and consists of a web application [5] that hosts a static map as well as the Overpass Turbo API [6] for data retrieval.

One problem arising from the fact that OSM is community-based is the fact that it lacks quality assurance [7]. OSM quality varies depending on the country and region [7]: While rural areas often lack information, urban locations are often precisely represented and contain information comparable to data provided by the respective government of the selected area or commercial companies.

[7], with its implementation of the web app 'Is OSM up-to-date?´ provides a systematic approach for measuring the OSM quality of an area. It presents how recently certain areas have been edited by the community and infers from that the need for revision on the map. In the case of TU Berlin's campus in Charlottenburg, the available data is well-maintained [8] and can be used as a starting point for this thesis.

2.3 Techniques used in navigation systems for mobile devices

Navigation systems for mobile devices are nowadays a crucial part of the landscape of available navigation solutions. Popular mobile apps, such as Google [9] or Apple Maps [10], often provide users the ability to navigate, locate and explore the world around them. The core features of those systems and important aspects for this thesis are:

2.3.1 Routing

Routing through a network of streets and places is an important key functionality of every digital navigation system. The most popular techniques for routing all use an underlying graph representation of the street network [11]. With this technique, streets are represented as weighted nodes, which model the underlying cost of moving between different locations.

One of the most popular algorithms to determine the fastest way between two nodes is Dijkstra's Algorithm. It greedily computes the shortest paths in the whole network and always returns the optimal solution (the fastest path). Despite its popularity and effectiveness, the algorithm's runtime scales poorly for huge graphs, e.g., the underlying data of Google Maps [11]. This is the reason why most of the algorithms used for routing in a navigation environment use different heuristics to speed up calculation. These algorithms usually trade off optimal route calculation for increased execution speed [12]. The following list presents an overview of the basic Dijkstra implementation and other commonly used routing algorithms [12].

Dijkstra: This algorithm starts at the start node and calculates the movement cost to all directly reachable nodes. It enqueues all visited nodes and repeats this process for the first node in the queue. Processed nodes get dequeued and costs for movement are updated when a shorter path is found. By calculating all possible paths in the graph, the algorithm is guaranteed to find the optimal solution. The main disadvantage of this approach is its runtime for huge graphs. Use cases are small graphs, several internet routing protocols and educational purposes.

A*: A* is a generalized version of the Dijkstra algorithm. The difference can be found in the cost function for movement between two nodes: Instead of only using the edge weight, A* applies a heuristic to the visited nodes. This heuristic usually consists of the airline between the

current node and the destination and improves calculation speed by providing the algorithm an indication of nodes that can be prioritized in the search. A* also always finds the optimal solution, while being faster than Dijkstra in most cases. Use cases are routing in bigger networks and educational purposes.

Genetic algorithms: Genetic algorithms describe a class of techniques for routing based on evolutionary processes. These algorithms start by choosing a random path and treating it like a gene. Principles like genetic crossover and random mutations are applied to it and the paths with the least movement cost are used for creating new populations of genes. This evolutionary process often converges to an acceptable solution in an adequate time. Its biggest disadvantage is the fact that it does not necessarily provide an optimal solution. Use cases are large graphs and scenarios that do not need an optimal solution.

2.3.2 Location detection

The main system used for outdoor localization in Google Maps and other similar services is GPS [11]. During usage of the navigation app, the user gets the opportunity to activate it and give permission for its usage. While Google's fused location API is the default location provider for Android devices [4], IOS devices use Apple's Core Location framework [13].

Although GPS is the standard system for localization, several other techniques are often used to improve pedestrian location detection on mobile devices. One relevant for this thesis is pedestrian dead reckoning.

Pedestrian dead reckoning localizes the user by tracking its steps and the movement direction. In a mobile scenario, this can be accomplished with the built-in device sensors. Steps can be detected with a distinct pattern in the retrieved acceleration, while the heading angle is read from the magnetometer [14]. Additionally, the system needs a reference point for localization: Since only the distance of movement in a certain direction is tracked, the starting point has to be known. Another crucial factor for correct PDR is a precise estimation of step length [14]. The following formula presents a standard way to calculate the current location in a 2-dimensional environment with step-count i, position (x, y), estimated step length L and heading ϕ [15].

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} + L * \begin{pmatrix} sin(\phi_{i-1}) \\ cos(\phi_{i-1}) \end{pmatrix} \tag{2.1}$$

One of the biggest disadvantages of PDR is the fact that errors accumulate during usage. Imprecisions in magnetometer, acceleration and step length values can result in incorrect PDR localization during long distance usage.

[16] describes an approach for merging data of the smartphone's built-in sensors for PDR with GPS functionality using a Kalman filter. It tries to use pedestrian dead reckoning as a base for localization and corrects its error accumulation with additional position lookup through GPS. The work states that such a technique can improve localization accuracy as well as energy-saving in a pedestrian position detection scenario. This improvement is accomplished by using classical pedestrian dead reckoning techniques (PDR) and GPS capabilities for correction.

2.3.3 Estimated time of arrival (ETA)

Providing an estimation of the time it takes to overcome the distance from start to destination is an important task for modern navigation systems. Google Maps calculates the estimated time of arrival (ETA) based on several different factors [11], including:

- Distance from start point to destination
- Average speed of the user and its selected form of transportation
- Current traffic situation on the chosen route
- Official and recommended speed limits
- Road types
- Historical average speed data
- Collected data from users who traveled similar routes

By collecting and merging this information, Google Maps manages it to provide its users with a precise ETA [11].

2.3.4 Visual design

The next step after retrieving the user's location and calculating the best route as well as its ETA is the presentation of that information within the app. [17] provides an overview of different solutions for information presentation within a mobile navigation system. It compares auditory instructions with route presentation of a map, simple route presentation on a map, display of current location and direction as well as a list of textual descriptions.

After evaluating all four different methods, [17] suggests the following aspects for a successful presentation of navigation data:

- A map oriented in the current walking direction should show the user's current location as well as the taken route and information about the map (landmarks, street names)
- The map should provide a zoom function
- Textual instructions are the most inefficient way of communicating navigation information
- Audio instruction can improve the map functionality, although GPS accuracy has to be taken into account when providing information about distance

2.4 Location-based services

Location-based services are mobile applications/services, which utilize the ability to localize the user to provide customized information [18]. By offering users information about nearby restaurants, supermarkets, shops, hotels and other POI, mobile navigation systems such as Google [9] or Apple Maps [10] also fall into the category of such services. Other popular apps using location-based systems are emergency services, tour guides, delivery tracking systems, social-media platforms with location-sharing functionalities, weather apps and mobility or taxi apps [19] [20].

Location-based mobile apps nowadays usually use the in-build GPS capabilities of mobile devices to provide their services in outdoor environments. Other techniques used for indoor environments are WiFI, RFID, ZigBee and Bluetooth [20].

Since the core functionality of the software developed in this thesis is comparable to mobile navigation systems such as Google or Apple Maps, the goal of this thesis itself can be described as a location-based service.

2.4.1 Geofencing

Geofencing describes a class of techniques used to automatically send notifications based on entering or leaving a specified geographic location [21]. These geographic areas are usually specified geometrically, by placing circles, polygons or polylines into specific locations, or symbolically by specifying a location by name such as a state, a country or an address [21].

There are several use cases for geofencing including marketing systems that send promotional notifications when users enter defined areas, logistic systems, that track and detect when certain deliveries and vehicles enter destination locations, mobile apps for navigation and social networking, as well as safety applications for tracking and detecting dangerous areas. One of the most prominent examples of geofence usage in a mobile application for marketing purposes is Burger King's "Whopper detour" campaign launched in 2019 [22]. The company sold a Whopper for one cent to every user who placed an order via the Burger King app within a McDonald's restaurant.

One key enabler for geofencing is an extensive background tracking functionality [19]. In the case of modern mobile devices, smartphone apps with a geofencing functionality need the ability to constantly query the user's location, even if the app is in the background.

3 Concept and Design

The goal of this bachelor's thesis is the development of an app for mobile devices, which provides students at TU Berlin the possibility to navigate and inform themselves about their campus in Charlottenburg. The main concept of the mobile app is inspired by popular smartphone navigation systems such as Google or Apple Maps and focuses on mapping TU Berlin's main campus and the most important web resources connected to it onto a single digital map.

3.1 Key features and technologies

The following sections provide a detailed overview of the app's key features and the underlying technologies used in this thesis.

3.1.1 Digital map of campus Charlottenburg

The main element of the mobile app consists of a locally implemented map of TU Berlin's central campus in Charlottenburg. It provides the user a manageable overview of TU Berlin's buildings, pathways, green areas as well as its surrounding environment. The following list displays possible map features with a description of their relevance for the mobile app:

- All buildings of TU Berlin: To provide the user the ability to easily locate the buildings of TU Berlin, all facilities connected to the university need to be specially highlighted on the map. The buildings of TU Berlin are therefore the most important map feature.
- All pathways on campus: Footwalks and cycleways that lie on the campus are important for the navigation system of the mobile app. To prevent the map from being cluttered, a hierarchy must be established between important pathways and smaller routes.
- All green areas: Parks, trees and other green areas of TU Berlin and its surroundings need to be specially marked on the map. Combined with the buildings and pathways of TU Berlin, they provide the user reference points for manual localization.
- External buildings: Buildings not connected to the TU Berlin do not contribute to the localization and navigation on campus. They can be nevertheless used as weak informative reference points. A toned-down and subliminal representation on the map can be used in this case.
- External pathways: All footwalks and cycleways that are outside of the campus do not provide any relevant information for the mobile app. They furthermore make the map appear more cluttered and are therefore not present on it.
- Main roads: Main roads surrounding TU Berlin's campus (e.g., Straße des 17. Juli) sim-

plify the exploration and search process while interacting with the map. They provide an important source of guidance and must be prominently presented on the map.

- Small roads: Small roads also support an organized map concept. Since they are less important than main roads, a more restrained manner of display is appropriate.
- External POI: The POI surrounding the campus (such as Ernst-Reuter-Platz) are heavily recognizable landmarks and support the user's orientation and localization on the map. They are therefore completely displayed on it.

All relevant features are retrieved via the Overpass-Turbo API from publicly available OpenStreetMap data. The data is then fed into the geographic information system QGIS, which is used for the creation and export of the campus map as well as manual annotation and correction of the downloaded data. Finally, the exported data is converted into a 3d model of the campus which is displayed in a respective rendering environment inside the app.

3.1.2 Navigation across the campus

The mobile app provides the user the ability to easily navigate across TU Berlin's main campus. The most important technological aspects to successfully achieve this task are routing, localization, geocoding, visual presentation of the current navigation state and calculation of estimated time of arrival (ETA).

The underlying data structure used for the whole navigation process is a weighted graph. Its vertices represent collected geodata points and the respective weighted edges are the distances between the vertices. To account for the fact that, in some cases, the fastest route between two points leads through other buildings, the entrances of all facilities are included in the graph. An edge connecting every pair of different entrance nodes of the same building is further included. The following section provides an overview of the procedure for the complete navigation process:

1. Geocoding is used to decode the user's human-readable destination input, e.g., "MAR-Gebäude" into its respective geo-coordinates and node in the graph.
2. To determine the starting point for the navigation, either another manually inserted location is geocoded or the current location of the user is determined using the built-in GPS module of the mobile phone. If the user selects the latter, the current location and heading are further tracked during the whole navigation process.
3. Considering the limited size of the graph, Dijkstra's algorithm is chosen to calculate the fastest route between the start and destination points. If the fastest route leads through other buildings, a live indoor navigation approach cannot be achieved due to imprecise GPS results. In this case, geofences are placed on the entrance and exit of the respective building. They detect when the user enters and leaves it and update the state of the navigation system accordingly.
4. The ETA is calculated based on the weights/distance of the fastest route and other factors (e.g., time of day, stoplights on the route, ...).
5. The navigation is presented visually on the campus map by drawing a polyline of the calculated route. If the user selects its location as a starting point, the current location as well as the heading retrieved from the device's magnetometer is displayed. When the

user is located within a certain range of the polyline, its position and heading are mapped onto it.

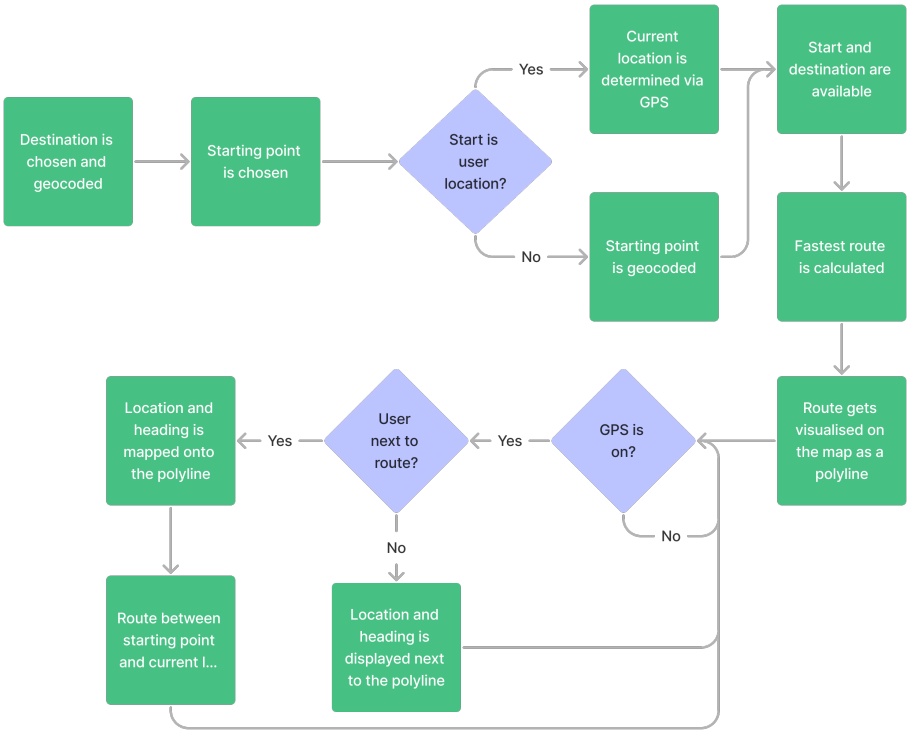

Figure 3.1: Complete navigation system procedure

3.1.3 Information layer

An additional layer on top of the campus map consisting of location-based information is a natural addition to the feature set of the mobile app. It enhances the user experience, the relevance of the whole product and the overall digital information culture of TU Berlin by providing overlays, markers and labels filled with data about the campus.

This section provides a brief overview of the underlying data and technology used to provide the information on the layer.

There are three main sources, that are publicly available and relevant to the information layer. On the one hand, there is the official website of TU Berlin [1], with sections for current events, the latest news, deadlines, etc. On the other hand, there is the MOSES system [2] containing data

[1] https://www.tu.berlin
[2] https://moseskonto.tu-berlin.de/moses/index.html

about all the different courses, rooms and studies. A third relevant source of information is the website of Studierendenwerk Berlin [3] with meal plans for its different canteens and a timetable for events.

Information is collected by downloading and parsing the content of respective websites. The data is further categorized and -if possible- mapped onto different POIs on the campus map (e.g., a canteen gets its meal plan assigned). This helps to provide an intuitive and organized presentation and access to the user.

The collected data can be categorized according to its timeliness and the intervals in which it has to be updated: General information about different fields of study, courses and rooms only changes by semester. This particular data can be retrieved once at the start of every semester and does not need daily live updates. It is also possible to supply it via app updates, instead of providing a direct in-app functionality for data retrieval. Data that gets updated regularly on the other hand, e.g., meal plans, events, news, etc. has to be always retrievable from the app.

The proposed solution for timely data (meals, events, ...) collection and provision runs on a web server and consists of a web crawler for information retrieval, parsing and POI mapping, a database to store the crawled information in a standardized and simple-to-use format and a REST API, that provides the client/mobile device the ability to retrieve the information. The web crawler is triggered periodically by several CRON jobs, whose timings are dependent on the timeliness of the crawled data.

Web crawlers dealing with data that needs rare updates (rooms, buildings, courses, ...) on the other hand, are not deployed on a web server. These programs are designed for manual usage at the beginning of a semester and provide their scraped data in JSON format, which can be directly included in the mobile app. This results in the fact that all data falling under this category can be used on the client side without the need for a network connection. It also means that some sort of deployment system needs to be included in the app for this data: Either the information is stored on a web server and gets automatically downloaded and stored on the client's device at the beginning of a semester or a complete app update from the respective app stores is rolled out.

By splitting the logic for crawling and provision, the workload that arises on TU Berlin's and Studierendenwerk's websites from retrieving data can be limited to a minimum: The server only crawls data when an update is needed (e.g., weekly for meal plans), instead of loading and parsing the web resources on client request. Further advantages are the fact that loading times for requests from clients are independent of TU Berlin's and Studierendenwerk's infrastructure, that the language for web crawling can be chosen independently from the programming framework (in this case Python with its Selenium [4] and BeautifulSoup [5] libraries are used) and that the web crawling logic can be changed without updating the app.

[3] https://www.stw.berlin
[4] https://www.selenium.dev
[5] https://pypi.org/project/beautifulsoup4

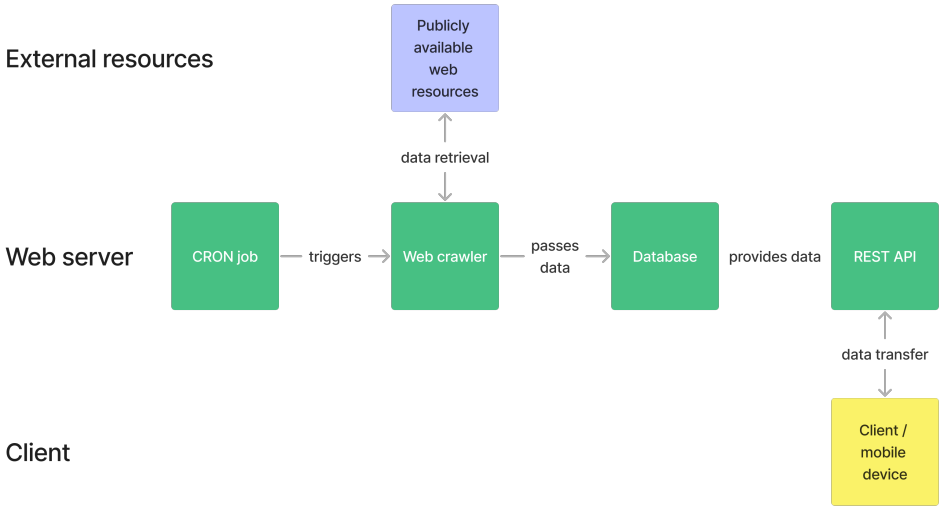

Figure 3.2: Data retrieval process for the information layer

3.1.4 Offline-first design

One important aspect of the app concept is an extensive focus on offline usability. The whole system is designed to allow the user (as far as possible) a network-connection-free app usage. This applies in particular to the navigation system, the campus map and the information layer of the app:

Campus map: The campus map is locally implemented on the device. All necessary information can be therefore loaded and used without a network connection.

Navigation system: Due to the comparatively small scope of the navigation system, the weighted graph for navigation as well as the system for routing can both be implemented and executed on-device. Only the navigation feature working with live user location updates relies on stable GPS functionality.

Information layer: Loading and presenting information on TU Berlin's main campus requires access to the respected online resources and cannot be realized without a network connection. The need for network connectivity can be nevertheless reduced: Since no data requires real-time updates (the most critical data being meal plans that require weekly updates), the data for the information layer can be stored locally after download and loaded from memory for the remaining week. This approach reduces the need for internet connectivity to a minimum.

3.2 App development framework

The choice of app development framework is an important step while conceptualizing and designing the product. It determines the programming language of the project, the built-in device- and system-specific capabilities that can be accessed during development as well as the performance of the app. It has to be selected in consideration of the project's requirements. The following enumeration presents the most important demands for this thesis:

- Used technologies: Working with low-level hardware and operating system APIs is an important part of development. Technologies and systems used in the key features of the product (e.g., GPS capabilities) have to be accessible or implemented in the selected framework.
- Cross-platform capabilities: Native app development is complex and time-consuming. Learning and especially working with two different codebases (Java / Kotlin for Android, Swift / Obj. C for IOS) is difficult and not possible within the scope of this thesis. Cross-platform frameworks solve this problem by providing the ability to work with a single codebase for different operating systems.
- Focus on high-quality user interfaces: One of the most important requirements of the app development framework is the availability to easily build high-quality and modern user interfaces with it.
- Execution speed: Regarding the fact that the whole navigation functionality has to take place locally on the phone, the speed with which the code of the mobile app gets executed is crucial to a responsive and fast user experience. Particularly the algorithms used for routing have to be reliably computable within seconds.
- 3d rendering capabilities: The framework needs to provide a way to render and interact with the created 3d model of campus Charlottenburg. Further built-in systems such as an extensive linear algebra toolkit, a lighting engine and other real-time rendering capabilities also need to be taken into account.

Based on the previously defined requirements, the Flutter framework [6] is chosen for this work. In addition to its cross-platform capabilities, it also offers the ability to compile source code into platform-specific machine code for near-native execution performance. It further comes with access to thousands of packages through the dart package manager pub [7], an extensive set of pre-built components for user interfaces and the ability to write platform-specific code for low-level API access. Since Flutter lacks 3d rendering capabilities, the Unity engine [8] is additionally used to create and display the campus map. It gets included in the Flutter build and is responsible for rendering and interacting with the 3d campus map.

3.3 User experience design

The following sections provide an overview of the whole user experience (UX) design for the campus app. In the scope of this thesis, the complete process is broken down into UX research and wireframe design.

[6] https://flutter.dev
[7] https://pub.dev
[8] https://unity.com/de

UX research is a process where the main goals and tasks associated with the app are identified and broken down into its most relevant UX aspects. This helps to find out the key usability requirements for the visual user interface design, consisting of wireframes and high-fidelity mockups (user interface design).

Wireframes, on the other hand, form the first visual version of the app design. They determine the basic structure of the app, its layout, the placement of basic user interface elements as well as their hierarchy. Wireframes are based on the previously defined usability requirements in the UX research process.

It is important to notice that all these design choices are not chosen arbitrarily, but rather based on previous research in the studies of user experience design. Important work was contributed over the years by psychologists and researchers like Dr. Jakob Nielsen [23], Dr. Paul Fitts [24], Jon Postel and research conducted at IBM and Xerox Corporation. An overview of the most important research and its consequences for user experience design can be found in Jon Yablonski's book "Laws of UX" [25] (web-version [26]).

The next paragraphs describe a set of important usability requirements that are then further incorporated into the wireframe design.

3.3.1 Search versus exploration

One of the most important usability requirements for the app results from the clash of two important user needs, namely search and exploration. These needs result from the fact that users may use the app for different purposes and with different expectations.

Search, on the one hand, describes a scenario, in which the user tries to find or locate a specific piece of information within the app. This search for information can be expressed with concrete questions, e.g.:

- What is the fastest way between MAR and TEL buildings?
- What are the opening times of the TU library?
- What food can I get tomorrow at the main cafeteria?
- Where does course App-Entwicklung take place?
- In which building is the SNET department located?

One main requirement for the user experience of the app is therefore the possibility for fast, easy and structured access to important key information about TU Berlin's campus. Based on [27] and [28], all search-task-related user interface elements should be easily identifiable as such, prominently positioned and accessible to the user. Additionally, to reduce the search time for the seeking person, the design should also provide multiple ways to access the most important information. Important user interface elements in this case are search bars, descriptive icons and textual hints as well as an adequately chunked information hierarchy in the whole app. This usage of commonly used user interface elements also supports [28].

Exploration, on the other hand, is an app use case, in which the user "just browses around" in search of nothing particular. The seeking person either wants to learn, find out or experience

something new or uses the app with an open question in mind. In this scenario, the user does not seek a specific piece of information but rather expects the app to provide the possibility, to easily navigate and view through structured content without the need for a concrete search.

Examples of goals and questions for exploration use cases can be the following:

- Are there any interesting places on TU Berlin's campus that I could visit?
- Which cafeteria has the best food today?
- Are there any appealing courses, outside of the scope of my studies, that I can attend this semester?
- Search for upcoming events by TU Berlin or Studierendenwerk
- Search for learning spaces at TU Berlin

In these use cases, the outcome of the exploration action is often determined by the subjective preferences of the users. The app therefore cannot give an optimal final solution to the search but rather present a set of information, from which the user can select or find the most suitable one.

This results in two important requirements for the app design. Firstly, visual elements for the structured presentation of information need to be used. These elements help the user by giving hints about existing information in the app and by structuring related information, which then can be easily overviewed and potentially compared while exploring. Further motivation for this can be found in principles [28], [29], [30] and [31]. Examples of such elements can be tables, scrollable lists, clearly designed information cards and different kinds of markers on the campus map.

Secondly, the choice of which information is portrayed in the app is important. Since too much information provision risks a bloating of the app (and therefore leads to a reduced user experience) [30], the final campus data needs to be carefully selected and filtered for in-app usage. The establishment of a clear hierarchy between important, frequently used data and less important information also contributes to this.

3.3.2 Designing a user-friendly campus map

A precise and clear map of TU Berlin's main campus in Charlottenburg is the most important user interface element in the app. Accordingly, special emphasis should be placed on the design of it. This section describes possible requirements for the map element and the resulting implications for its appearance and implementation in the app.

There are two intended ways how the users should interact with the map: Interaction for exploration/search tasks and interaction for navigation purposes. Both these tasks come with separate design requirements.

Exploration and search tasks, mainly interacting with POI and buildings on TU Berlin's campus, have a strong focus on its entities. A straightforward map design with a focus on the recognizability of buildings, pathways, natural areas and POI is important. This supports the user's need for orientation and simplifies the process of matching real-life buildings with their counterparts on the digital map, which is proven to enhance the user experience [28].

This mostly differs from popular available mobile navigation services since their primary

tasks are navigation and orientation of the user. They are also laid out for presenting huge areas, e.g., districts or even complete cities. A detailed portrayal of a specific area is therefore not feasible, perfomant or necessary.

The campus map only deals with a pre-defined and small area. This provides the possibility for the creation of a detailed and complex map, with a pleasing aesthetic (based on [32]) and a strong focus on TU Berlin's entities on campus Charlottenburg. To achieve maximum recognition value for the buildings, parks and pathways on the campus, a 3D overview, similar to TU Berlin's campus plan [33], is used for portrayal. This is especially beneficial for users who are familiar with the official campus plan of TU Berlin, who can transfer their learning from the manual map onto the digital one, which plays well with [28]. The 3D perspective also lays out a clear emphasis on buildings, which are the most important entities on the campus and prominently stick out on the map. It is nevertheless also beneficial for other entities, which can be more easily localized with the buildings.

To direct the focus even more onto TU Berlin's campus, all of its surroundings are not used in the 3D view (based on [29]). Furthermore, all important buildings are labeled with their respective abbreviations through markers on the map. Lastly, the user also gets the possibility to alter the map view between 2D and 3D modes, so that the preferred style of view can be selected.

Another important aspect is the zoom range the user can utilize while interacting. Considering that the campus area is small and that the app wants to focus on its details, a close zoom, with a little room maneuver should be appropriate.

Navigation tasks, namely interacting with / following a polyline route and working with a marker that displays the current user location, have a strong focus on pathways and routes on TU Berlin's main campus. It is therefore important to highlight these routes in the design accordingly. Other entities like buildings, natural areas (green areas, water) and POI have a lower priority in this case but are nevertheless important for orientation purposes. To design for this use case one can take inspiration from already established mobile map services like Google Maps [9] and Apple Maps [10]. These apps mainly display the road network, important markers and POI while in navigation mode and only present low-priority entities (e.g., outlines of buildings) on a close zoom level.

[9] https://developers.google.com/maps?hl=de
[10] https://www.apple.com/de/maps

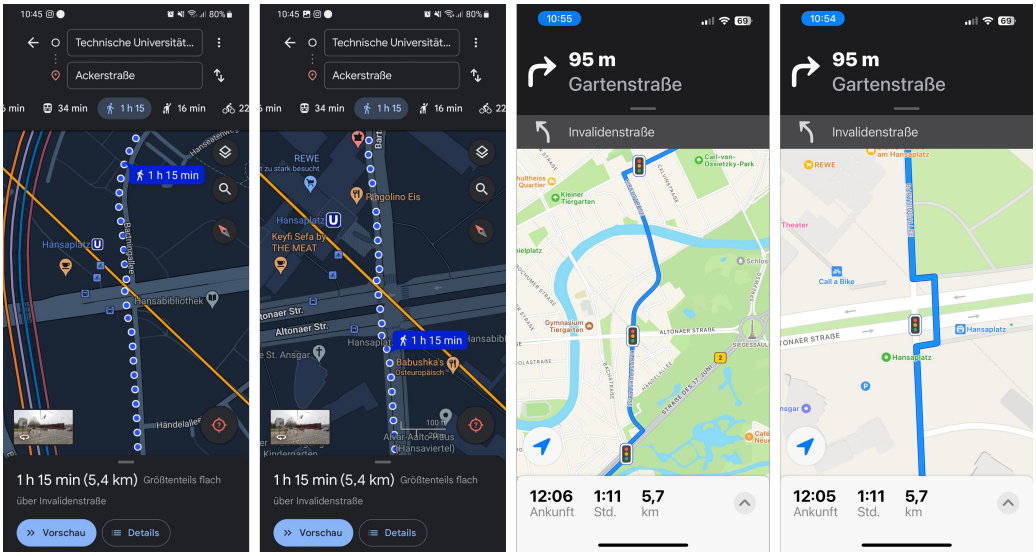

Figure 3.3: Different styles of map presentation in Google Maps and Apple Maps

Based on the fact, that common usability patterns are already established by the user base of such popular mobile navigation services, the approach is adopted in this thesis (enhancement of user experience based on [28]): The user's focus is directed to the road network during the navigation phase, unimportant markers are discarded, roads are highlighted and the map's camera is adjusted to display a 2D perspective from the top.

3.3.3 Designing a good user experience for navigation

Creating a robust and easy-to-use digital navigation system is a challenging task since it consists of digital instructions resulting in real-life actions.

Based on [28], it is beneficial for the user experience to orientate the design to industry standards. In this case, a polyline is usually used for the display of the calculated route, a round marker stands for the user's position and a change in color of the polyline tracks the already traveled path segment. The location marker is further mapped onto the polyline during navigation, as it marks the point between the traveled and preceding route.

Nevertheless, one important difference between established navigation systems and campus navigation is the fact that the latter can also lead through buildings of TU Berlin. Since the app only detects whether (and where) the user enters or leaves a building (the traditional polyline approach is therefore not feasible) and the user base is likely not familiar with a switch from outdoor to indoor navigation, a robust and clear user experience has to be provided during this phase.

To specify the design requirements for this section of the navigation process, the outdoor indoor switch can be broken down into a set of phases, each with its demands:

1. The user is outdoors during the navigation phase or examines a precalculated route and sees an (upcoming) segment that leads through a building: In this case, the user must understand that a segment leads through a building. It should be furthermore clear which

entrance has to be used. The latter can be achieved by marking the entrance position on the map, either with a model of a door (3D perspective) [28] or a marker containing an appropriate icon (2D perspective). To clarify the indoor section even more, the polyline of the route should be connected to the entrance and a descriptive marker (e.g., "upcoming indoor section") or other indication should be next to the indoor segment. This repetitive "announcing" of an upcoming indoor section is strongly connected to [34] and [35] and an enhancement of user experience can be therefore expected. It is important to notice that due to [36], the user should not be punished in a scenario, where a wrong entrance is taken into the correct building. The app should simply continue the navigation in this case.

2. The user walks into the marked entrance of the building and is now indoors, the app registers this: In this phase, the biggest explanation part is fulfilled, the user has understood that the next segment is indoors and has entered the building successfully. It is now important to provide the user a visual feedback, that strengthens his correct choice of entering the building ("Great, you have found your way into the building!"), updates the progress on the route accordingly (due to [37]) and explains that there is no indoor localization available. The use of animation is a great tool to indicate a change in the state of a user interface. In this case, the map moves and zooms to the entered building and a non-moving marker appears, indicating that the user is inside of the building. A pop-up appears, explaining the location of the user and its goal to find a specific exit of the building.
3. The user is inside of the building: No specific changes to the user interface need to be made, the app waits till the user has found the exit. In an optimized scenario, an additional pop-up could be triggered if the user spends too long inside the building and searching for the exit. This pop-up would display additional information for finding the desired exit (enhances the UX according to [34]).
4. The user walks out of the building and is now outdoors, the app registers this: This phase should reverse the changes made when entering the building. The navigation continues on the polyline with a reappearing location marker, the map unlocks the focus on the exited building, and all markers and pop-ups for indoor navigation disappear. The user gets the feeling that he has successfully made it through the indoor section and can now continue his journey outdoors. Just like in the entering phase and according to [36] the user should also not be punished when taking the wrong exit in the building. In this case, the app should simply detect the wrong exit, recalculate the route from the new user position and display this in the user interface.

3.3.4 Chunking of information within the app

According to [31], an organized visual information structure is key to providing a good user experience. Especially concerning the fact that a huge amount of campus data is available and exploration tasks are key activities in the app. Chunking is the task of grouping related information visually together and is a key enabler for an organized layout [31]. The following figure provides a clustered overview of the available data connected to TU Berlin's main campus:

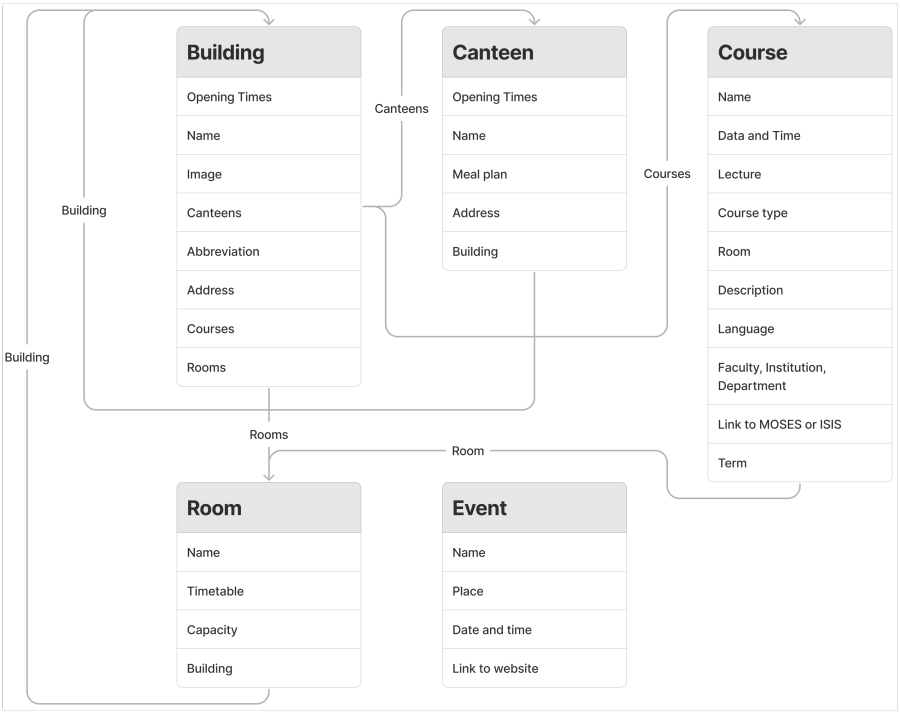

Figure 3.4: Clustered data of TU Berlin with arrows between related groups

Based on connection count and data, the "Building" and "Course" clusters can be identified as key entities in the system. It is therefore advisable, to attach importance to the display of these groups in the user interface of the app. Other groups are therefore placed less prominently but also get referenced (according to the relations in the figure) in places where "Building" and "Course" are portrayed.

According to [29] and [38], it is advisable to display information related to one common cluster next to each other and similarly. This improves the user experience by emphasizing associated information. The comprehensibility of the visual design is also important: The user should directly understand the meaning of the portrayed data, which can be mainly achieved with descriptive texts and iconography.

3.3.5 Wireframes

Based on the previously defined requirements, wireframes outlining the basic design concept of the app are created. This section displays the most important pages and provides a brief enumerated overview of their features. (Note that not all wireframes are presented and that certain design and layout changes occur between UX and UI design.)

Figure 3.5: Most important wireframes with annotations

1. Search bars are prominently placed on the main and search pages. The positioning is inspired by already established apps and enhances the UX due to [28].

2. To speed up the search process for certain entities, filter options are placed beneath the search bars.
3. A button, that toggles whether the information cards are visible or not. This gives the user the decision whether to explore TU Belin's campus with the help of cards or directly by moving the map. This element also breaks down the number of choices by grouping entities by category, which, due to [30], also improves usability.
4. Markers are placed on buildings with their respective abbreviations, the main communication symbol for buildings on TU Berlin's campus. A detailed information page is opened when clicked. They provide a sense of familiarity from other navigation services [28].
5. Information cards are displayed for every entity on the campus. They present the most important data of a landmark and, when in focus, move the map to the selected place and highlight it. The cards are positioned prominently on the button, to strengthen the interactivity [27].
6. A button that toggles between 3d and 2d map perspective.
7. A button that detects the GPS position of the user and moves the map to it.
8. Result cards that contain different base information about the searched entities. They are sorted by category for easier search and chunking [30]. A detailed information page is opened when clicked on a card.
9. A doubled search bar, inspired by common navigation apps [28] for start and destination input.
10. A special marker, which represents the current GPS position of the user.
11. An overview of the calculated route, which pops up after inputting the start and the destination. It presents the most important information for the route as well as an opportunity to start the navigation. Placed on the bottom for easier access [27].
12. Tags with the most important key facts on the detail pages for certain entities. The detail pages as well as the tags are all designed in a similar way to provide the user a structured, well-organized experience.
13. A small pager displaying the courses, that currently take place, on the detail page for a building. They are linked to the respective course detail pages.
14. A primary action button, prominently placed to guide the user to the navigation functionality of the app. When clicked, the app switches into navigation mode with the current building selected as the destination.
15. A list of rooms, that can be found inside of the selected building. All cards are linked to the respective room detail pages.
16. A timetable displaying all the courses, that take place in the selected room. It is sorted by dates. All cards are linked to the course information pages.
17. Description and other key information on the detail page for a selected course.
18. Further details about the course.
19. Button options, that link to the ISIS and MOSES pages of the respective course. They are prominently placed on the bottom to allow for easy access [27].

3.4 User interface design

The next step in the design process of the app is the user interface (UI) design. This procedure takes the already elaborated wireframes from the UX design phase and applies a consistent design language to it. One main outcome of this design step should be an identifiable corporate design of the digital product. Due to [32], which states that a beautifully designed user interface is perceived as a more useful one, user interface design can be seen as a subset of UX design.

Concepts to which particular attention is paid in UI design are choice of color, typography, iconography, shape-language, animation as well as usage of media, e.g., photos, videos or text. The following section and figure provide a brief specification of those parameters (often called "Styleguide") for the app design.

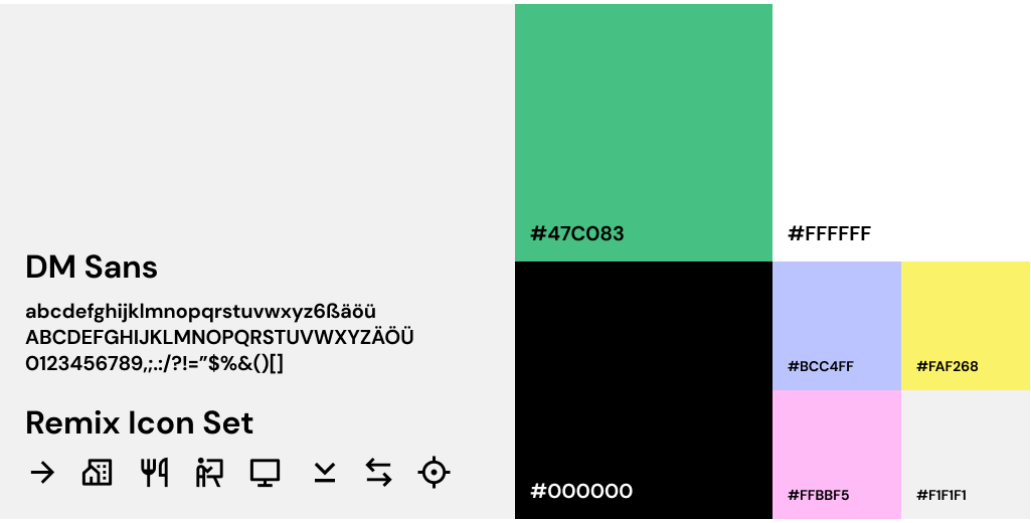

Figure 3.6: Styleguide with colors, typography and iconography

- DM Sans [11] is chosen as the main font for the app, it is usually written in real black (HEX #000000) and is used in its regular, medium and bold variations throughout the app. It occurs in font sizes 12px, 16px and 32px.
- The open-source Remix-Icon [12] icon set with its respective Flutter package. The icons are used in their not-filled versions and set a basis for a slightly rounded design language for the entire app. Icons are consistently 20px * 20px sized and colored in HEX #000000 black.
- The main color of the app is the green tone HEX #47C083, which is supported by real black #000000 for icons and text and real white #FFFFFF for backgrounds and areas. Additionally, several colors representing the individual campus entities are selected: HEX #BCC4FF (blue) for buildings, HEX #FFBBF5 (pink) for courses, #FAF268 (yellow) for canteens as well as HEX #E8D4C7 (beige) for events. Furthermore, to color bigger areas inside of the white layout, two grey tones are introduced: HEX #F1F1F1 (light grey) and HEX #BBBBBB (dark grey) are used e.g., in a search bar, where light grey colors the

[11] https://fonts.google.com/specimen/DM+Sans?query=DM+Sans
[12] https://remixicon.com

element and dark grey the overlaying hint text.

- Animations are all 250ms long and consist of two different curves: While all map-related camera animations in Unity use a linear curve to present animation progress (camera movement over the map), the animations in the remaining user interface are based on Flutter's "Ease" curve [13], which consists of a cubic curve mapping from [0, 1] to [0, 1] parametrized by two control points (0.25, 0.1) and (0.25, 1.0). By mimicking physical movement (which is often based on constant acceleration, e.g., gravity), the latter animation curve provides a sense of naturalness to the user while the first linear animation style makes it easier to follow the camera across the map.
- Usage of media: Images of TU Berlin's buildings are presented on the respective pages to provide the user with another possibility for visual reference while browsing through the app.

The following figure provides an overview of the most important screens of the final user interface design of the app.

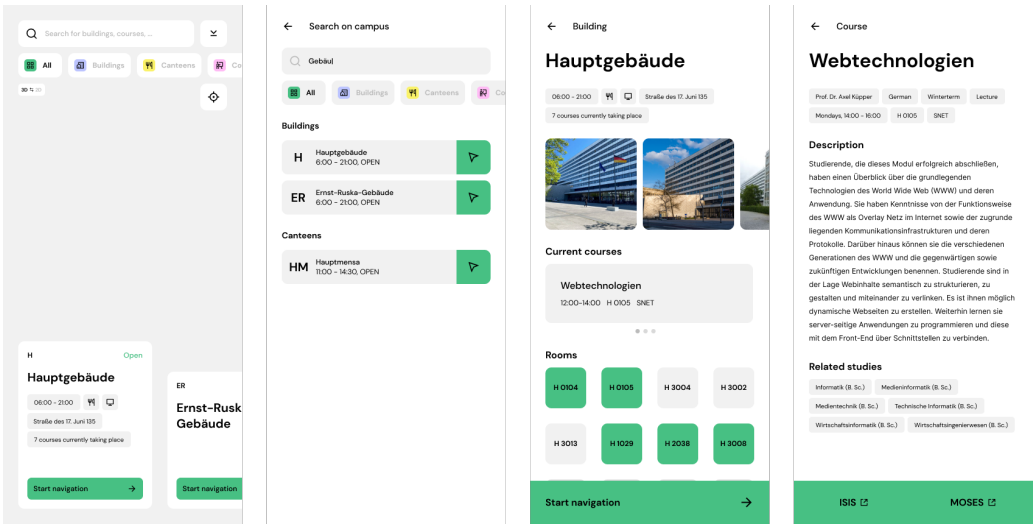

Figure 3.7: UI-Design

[13] https://api.flutter.dev/flutter/animation/Curves/ease-constant.html

4 Implementation

This chapter deals with the implementation details of the campus app. It starts by describing the process for the collection and processing of geodata and explains how the campus map and the navigation system are derived from it. It further showcases the procedure for gathering campus-relevant data from publicly available web sources and concludes with a description of the implementation process for the user interface, which finally merges the campus map, navigation and information layers.

4.1 Collection of geodata and POI

Geodata refers to all data about geographic information. In the case of this thesis, it mainly consists of coordinate points, which describe certain campus-relevant entities, such as outlines of buildings, the street network, pathways, entrances as well as boundaries of green areas and water. Except for single-point usage, multiple coordinate points can be grouped into polygons, representing the closed outline of an entity and polylines, mainly used to describe lines, streets and paths. In addition to its coordinates, geodata also describes other attributes of an entity e.g., its name, altitude, width or height. This can be used for tasks like geocoding/reverse geocoding, display of information, 3d terrain generation or general data analysis.

4.1.1 Overview of needed geodata

Geodata is the building block for map generation and routing through a street network. The following list presents an overview of the geodata needed for this implementation:

- Buildings: The outlines of buildings are polygons consisting of multiple coordinate points. Additionally to that, the height of each building is needed for appropriate 3D representation.
- Streets: Streets are represented by polylines and are (based on their importance, size and purpose) divided into three categories, namely main roads, small roads and pathways. This separation allows the different road types to be designed and displayed independently from each other. They can therefore vary in size, style and coloring when used in the campus map.
- Green areas: The outlines of green areas are retrieved in the polygon format.
- Water: The outlines of water (especially rivers) are also retrieved in the polygon format.
- Entrances to TU Berlin's buildings: To successfully connect TU Berlin's buildings to its underlying street network, entrances need to be defined. A single-point representation paired with an identifier for the buildings is used.

4.1.2 Collection data from OSM via Overpass Turbo API

OpenStreetMap (OSM) is a free mapping service, that allows its users to access, edit and download its available geodata. Its main querying language is Overpass Turbo which comes with a web interface for data mining and a respective web API [6]. This querying system is used to provide a starting point for the complete set of geodata needed for this thesis. The used bounding box for retrieval is (52.49993096650543, 13.307022255971093) for the south-west and (52.52269751545147, 13.341918533901309) for the north-east corner. The following table presents the queried data over all categories together with the node count (one node usually consists of a coordinate point), the type of retrieved data and the respective query.

Entities	Type of geodata	Number of queried nodes	Overpass Turbo query
Buildings	Polygons	1949	1
Main roads	Polylines	2036	2
Small roads	Polylines	2578	3
Pathways and sidewalks	Polylines	11757	4
Green areas	Polygons	5057	5
Water	Polygons	1651	6
Entrances	Single points	1758	7

Table 4.1: Overview of queried geodata

This results in a total of 26786 queried nodes. Each category is further exported as a separate geoJSON file, containing the relations between individual points (polygon, polyline, single node) as well as other OSM attributes (e.g., the names of buildings and streets, the height of buildings, etc.).

4.1.3 Correction of OSM data for map generation in QGIS

Since the queried data contains several imperfections that are not desirable for map generation, all geoJSON files are imported into QGIS for cleanup. QGIS is an open-source geographic information system, that provides a convenient visual user interface for editing geodata. The following images present a visual reference for the changes performed to the data (note, that all street data is represented in one image for simplicity and that water data does not need cleanup and is therefore not present).

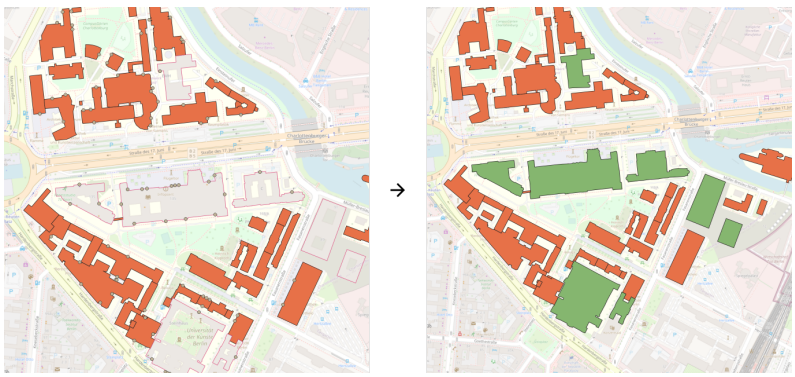

Figure 4.1: Correction of TU Berlin's building data

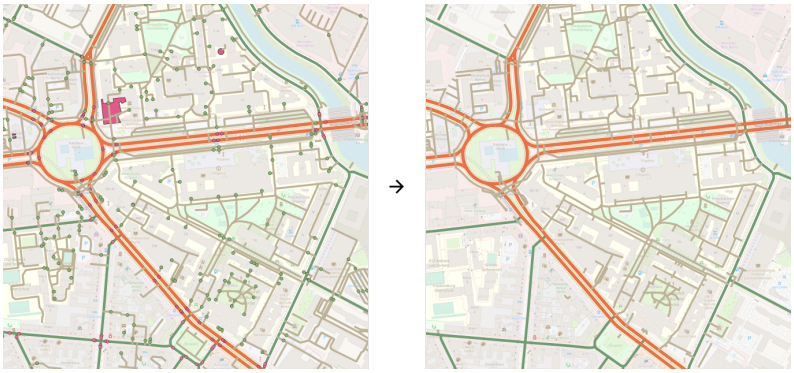

Figure 4.2: Correction of TU Berlin's street network data

Figure 4.3: Correction of TU Berlin's green area data

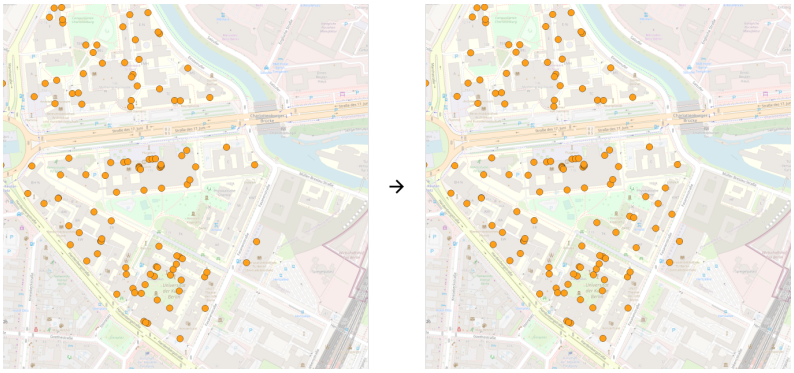

Figure 4.4: Correction of TU Berlin's entrance data

The main problem that gets corrected consists of wrong geometry types in the data. Since is desirable that every dataset only contains one kind of geometry (e.g., the data for the buildings should only contain polygon outlines, the street data should only contain polylines, etc.), all wrong geometry types in a dataset are either deleted (e.g., single points in the street or building data) or reshaped to match the expected geometry type (e.g., polylines representing the outline of a building/green area are converted into polygons).

Missing entities (e.g., certain entrances to buildings) are furthermore added and information not related to the campus (mainly data from the street network) is removed.

4.2 Generation of digital campus map

This section provides a detailed overview of the procedure for generating the digital campus map in the Unity3d real-time rendering engine. It starts by describing how the already collected data is used and imported into Unity and further explains how 3d entities are created from it. The section concludes with a brief overview of the map design and the implementation details for common map gestures such as pinch-to-zoom or slide-to-move.

4.2.1 Data import and conversion in Unity

The first step in the map generation process is the import of the manually corrected OSM data from QGIS into Unity3d. This is done in geoJSON format [39], which specifies a certain JSON structure for geodata and contains all coordinate points and their relations to each other as well as additional attributes.

After import, a C# class named GeoNode is introduced, which represents a single node in the OSM data and contains its position. Furthermore, a class for every category of entities on the campus map (buildings, green areas, streets, ...) is implemented, which contains an array of position vectors, derived from the GeoNodes, describing the respective (out)line of its entity. Lastly, methods for parsing geoJSON into their C# object representations are created.

One main problem during the parsing process is the fact that OSM and the campus map are constructed differently: While OpenStreetMap consists of a map projected onto a sphere, the campus map is generated on a 2-dimensional plane. This results in the usage of different coordinate systems. OpenStreetMap in this case uses the WGS 84/Pseudo-Mercator system [40] [41], which describes points on Earth as pairs of latitude and longitude. This is ideal for spherical map projection but is not suitable for a 2d plane base, where general length units (e.g., meters) are advantageous. A conversion from WGS 84/Pseudo-Mercator to 2d position vectors (x, y) in meters from the map origin is therefore chosen and integrated into the parsing process. It is further defined that x represents the longitude and y is the original latitude component of WGS 84.

The unit conversion process is based on OpenStreetMap's reference implementation of the WGS 84/Pseudo-Mercator projection [42] in C and computes the following formulas:

$$x(lon) = lon * r$$

$$y(lat) = \ln(\tan(\frac{lat}{2} + \frac{\pi}{4})) * r$$

Where the WGS 84/Pseudo-Mercator coordinates (lat, lon) in radians are converted to (x, y) in approximate meters, with r being the radius of the Earth (note that the WGS 84 coordinates are usually defined in degrees and need to be converted into radians beforehand).

The resulting coordinates are furthermore stored inside the GeoNode class as floats and are also converted into Unity's 3-dimensional Vector3 position system, where a position (x, y) is

mapped to Vector3(x, 0, y). To center the campus section of TU Berlin around Unity's origin at Vector3(0, 0, 0), the central map point of campus Charlottenburg (52.5126624, 13.3231489) is subtracted from all OSM data points before conversion.

4.2.2 Mesh generation for streets, green areas, water and 3d buildings

Meshes are the building blocks for every 3-dimensional object rendered on a digital screen. They define the structure of an object and provide parameters for lighting calculations and material mapping and are generated by defining a set of attributes, the most relevant ones being:

1. Vertices: Vertices are points in space, that define the corners of an object.
2. Triangles: To generate a surface area, 3 different previously defined vertices are connected to form a triangle. A net of triangles finally determines the final shape and surface of an object. One important concept to keep in mind is the backface culling step of the render pipeline: For optimization purposes, the rendering engine decides whether surfaces created by triangles are drawn or not by using the order of vertices that define a certain triangle. Unity3d only renders triangles with a clockwise ordering of vertices when projected onto the screen. The whole process of generating a triangle net to fill a predefined surface area is called surface triangulation.
3. Surface normals: Surface normals are vectors that are perpendicular to their related surface. They are used to differentiate between the "front" and "back" sides of their respective surface and are an important component for calculating the lighting of an object. Generally, each triangle is assigned a surface normal, which can be calculated by taking the cross product of two different edges. (Note: Unity differs from that and uses a surface normal per vertex approach. This normal can be calculated by averaging all normals of triangles containing the respective vertex. For better understanding, the normal per triangle model is nevertheless displayed in the figures.)
4. UV coordinates: UVs are 2-dimensional coordinates that specify the texture mapping of an object. There are as many UV coordinates as vertices in a mesh. Based on the shape of an object, the UV coordinate system can be mapped differently onto the vertices.

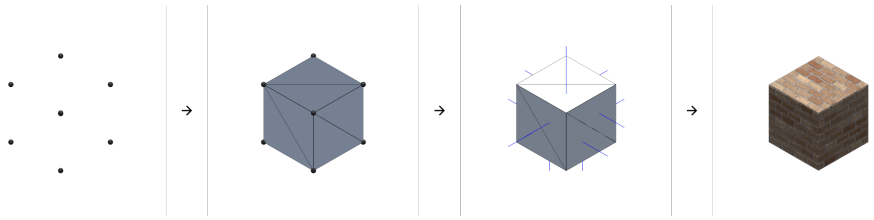

Figure 4.5: Mesh generation process for a cube

In the case of the campus map, several simplifications can be made: Surface normals can be calculated from the predefined vertices and triangles. Since the entities on the campus are only unicolored (no distortable textures are applied) the 2-dimensional UV coordinates can be chosen arbitrarily (or not at all) for every vertex. This breaks down the mesh generation process

for the campus map into two separate tasks: Defining the outlining vertices of an entity from its GeoNodes and triangulating its corresponding surface.

Mesh generation for streets: To define the outline of a street, the corresponding GeoNodes, which only represent a polyline, need to be replaced by two separate polylines defining the edges of the street. The width w of the street is then defined by the distance between both outlining polylines. To generate them, every GeoNode at index i with coordinate $\vec{g_i}$ of the original polyline can be shifted with a vector $\vec{v}$, which is perpendicular to the road direction at $\vec{g_i}$ and the up-facing vector (in the case of Unity Vector3(0, 1, 0)). A pair of outlining points can be therefore generated for every GeoNode coordinate $\vec{g_i}$:

$$\vec{eRight_i} = (\vec{g_{i+1}} - \vec{g_i}) \times (0,1,0) * \frac{w}{2}$$

$$\vec{eLeft_i} = (\vec{g_{i+1}} - \vec{g_i}) \times (0,1,0) * -\frac{w}{2}$$

Where $\vec{eRight_i}$ is the point on the right polyline and $\vec{eLeft_i}$ is the point on the left polyline corresponding to $\vec{g_i}$ (when looked into the direction $\vec{g_{i+1}} - \vec{g_i}$).

These vertices can be further triangulated by grouping 4 points. To fill the street surface between coordinates $\vec{g_i}$ and $\vec{g_{i+1}}$, the points $A = \vec{eLeft_i}$, $B = \vec{eRight_i}$, $C = \vec{eRight_{i+1}}$ and $D = \vec{eLeft_{i+1}}$ are selected and two triangles ACB and ADC are created.

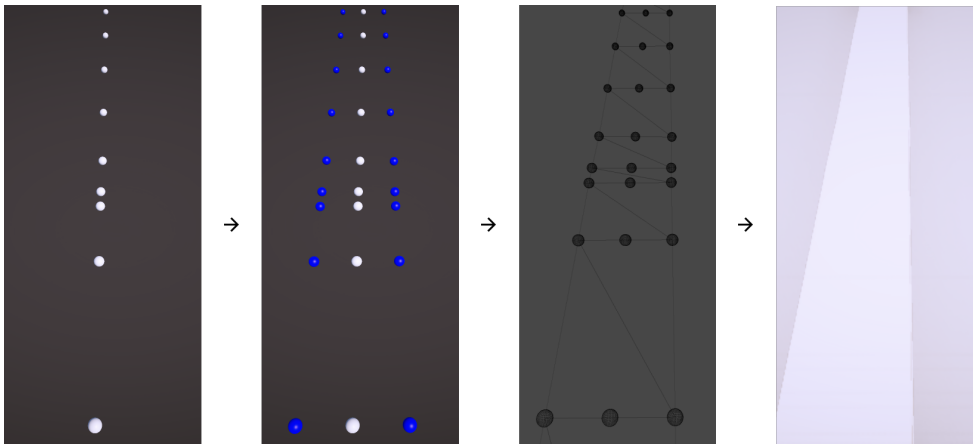

Figure 4.6: Mesh generation process for a street

Mesh generation for buildings: Buildings consist of a ceiling tile and a set of surrounding walls. To determine the outlining vertices, the predefined set of GeoNode coordinates can be extended by shifting it upwards. The amount of shift is determined by the height h of the respective building. For every GeoNode coordinate $\vec{g_i}$, a new coordinate $\vec{c_i}$ is therefore created and added to the final set of building vertices:

$$\vec{c_i} = \vec{g_i} + (0,1,0) * h$$

The triangulation process for the resulting set of vertices can be broken down into two sep-

arate procedures for the ceiling and the walls. On the one hand, a routine similar to the street triangulation process can be applied to create the walls. A single wall is here defined by two consecutive GeoNode coordinates $\vec{g_i}$ and $\vec{g_{i+1}}$ and their corresponding shifted coordinates $\vec{c_i}$ and $\vec{c_{i+1}}$. The surface area between these four points is then exactly triangulated like a piece of street defined by four points (see the previous section).

On the other hand is the triangulation process for the ceiling tile. In this case, a standard algorithm for the triangulation of concave polygons can be applied. The "Triangulation by Ear Clipping" [43] algorithm is used with a C# implementation for Unity derived from [1]. To avoid backface culling problems and for simplicity, all triangles are generated double-sided, with vertices specified in both orderings.

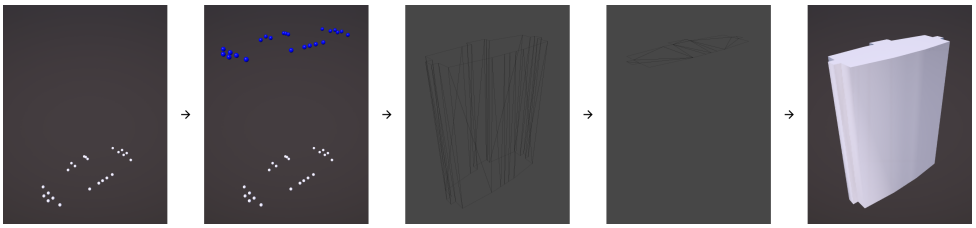

Figure 4.7: Mesh generation process for a building

Mesh generation for green areas and water: Since green areas and water both consist of flat and concave polygons, the same procedure can be used for generating both respective meshes. The outlining vertices are in this case simply the predefined GeoNode coordinates and can be triangulated in the same manner as the ceiling of a building.

4.2.3 Camera setup

One main component for the 3d rendering of the campus map is a virtual camera, that can be regarded as a viewport through which the user looks at the scene. Its transform (the set of values that determines the camera's position, rotation and scale) as well as several other parameters specifying how the scene is rendered can be manipulated to achieve a specific look of the scene.

To easily control the camera for movement, zoom and rotation, an empty and transparent object with position $\overrightarrow{p}$ is placed on the ground of the map. This focus point object is furthermore declared as the parent of the camera, which results in the fact that all camera transformations are now relative to the focus point's transform. This means, that $\overrightarrow{p}$ now serves as the camera's world origin and that the camera furthermore has two different transforms: The global transform GC, which reflects the actual position, rotation and scale of the camera in the scene and the local transform LC, which consists of the camera's position, rotation and scale relative to its parental focus point. To make it more comprehensible one can look at an example regarding the positioning of the camera: If the local position of the camera $\overrightarrow{lc}$ is at $(0, 0, 0)$ (and neither a rotation nor a scaling is anywhere applied), the camera's true global position in the scene $\overrightarrow{gc}$ is the focus point's position $\overrightarrow{p}$. In general (without rotation or scaling) $\overrightarrow{gc} = \overrightarrow{p} + \overrightarrow{lc}$, which

[1] https://luminaryapps.com/blog/triangulating-3d-polygons-in-unity/index.html

means that the camera now "follows" the focus point object with offset $\overrightarrow{lc}$ whenever its position changes.

The camera is now locally transformed such that the focus point object always lies in the center of its viewport. In principle, this can be done arbitrarily, but to mimic the perspective of TU Berlin's official campus map [33], a setup is chosen where the camera is rotated 45 degrees down (x-axis) and $\overrightarrow{lc} = (0, z, -z), z \geq 0$ (x-axis, y-axis, z-axis). An alternative setup for top-down view (called "2d" mode inside the app) is also provided, which positions the camera directly above the focus point with a rotation of 90 degrees downwards and $\overrightarrow{lc} = (0, z, 0)$. The following figure presents the setup for both cases.

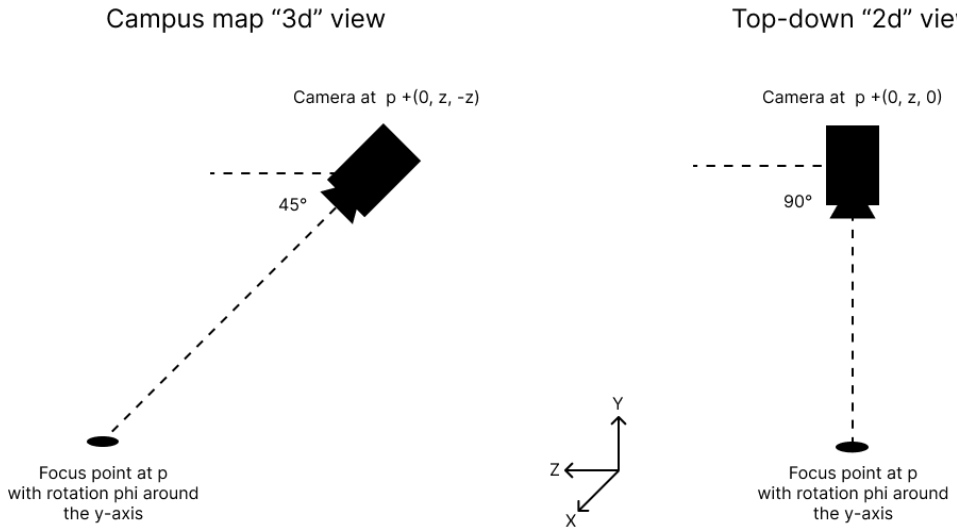

Figure 4.8: Camera setups for "3d" and top-down "2d" view

With this setup as a prerequisite, all map-relevant camera transformations can be defined with the parameters $\overrightarrow{p}$ (position of the focus point object), which is responsible for the camera movement, φ (rotation of the focus point object around the y-axis), which represents the camera rotation and z (parameter of the local position of the camera), which is the camera's zoom amount.

Besides the transform parameters of the camera, two other important render settings need to be considered for the campus map: On the one hand, there is the projection mode, which determines how the 3d environment is projected onto the 2d screen. Here the perspective projection is chosen since it resembles the real-world projection of the human eye and therefore creates an intuitive feeling of depth inside the scene. On the other hand, the field-of-view (FOV) value, which determines the opening angle of the camera viewport is set to 55 degrees.

4.2.4 Implementing common interaction gestures

Touch gestures are an established way of controlling the previously defined parameters for movement, rotation and camera zoom. This section explains which gestures are chosen and how they are implemented for this thesis.

One prerequisite for a smooth and robust interaction experience is the use of animation when moving, rotating and zooming on the map. A parameter is therefore not directly set, but rather linearly interpolated over time between its current value and a desired target value. From a programming perspective, the system constantly tries to perform such an interpolation and only stops, when the actual parameter is equal to the target one. Therefore the only parameters that need to be set while interacting with the map are the target values, which are hereafter denoted as $\overrightarrow{p_t}$, φ_t and z_t.

Map movement: The user can move the camera on the campus map by dragging a finger over the touchscreen. The internal evaluation of this process can be broken down into multiple steps:

1. The user's finger touches the screen for the first time (touch start): The initial touch position $\overrightarrow{t_{start}}$ is detected.
2. The user slides the finger over the touchscreen (touch moved): The current touch position $\overrightarrow{t_{cur}}$ is detected and the desired camera position is calculated as $\overrightarrow{p_t} = \overrightarrow{p} + (\overrightarrow{t_{start}} - \overrightarrow{t_{cur}}) * \alpha$, where α determines the drag speed of the campus map and is chosen as 5.

Pinch-to-zoom: Pinching is a type of gesture where the user positions two fingers on the touchscreen and an action is triggered when the distance between both touchpoints changes. In this case, the camera should zoom in when the user brings the fingers closer together and vice versa. The following steps are necessary to implement this functionality:

1. Two fingers touch the screen simultaneously for the first time (touch start): Both touch start positions are retrieved and their distance d_{start} is calculated.
2. The user moves one or more fingers (touch move): The current distance d_{cur} between both fingers is calculated. The target zoom is set as $z_t = z_t + (d_{cur} - d_{start}) * s$, where s is the strength of the zoom and is selected as 0,1.

Rotation of the map: The gesture for map rotation is similar to the pinch gesture from the zoom functionality, but rather takes the rotation angle between the start- and end position of both fingers into account, instead of their distance. This can be also realized in two different steps:

1. Two fingers touch the screen simultaneously for the first time (touch start): The difference between both touch positions is stored as a 2-dimensional vector $\overrightarrow{t_{sdiff}}$.
2. The user moves one or more fingers (touch move): The current difference of touch positions is also stored as a 2-dimensional vector $\overrightarrow{t_{cdiff}}$. The camera rotation is updated with $\varphi_t = \varphi + singedAngle(\overrightarrow{t_{sdiff}}, \overrightarrow{t_{cdiff}}) * \beta$, where $singleAngle(a, b)$ is a function that returns the signed angle between two 2-dimensional vectors and β is a parameter for the rotational strength and practically set to 0,5.

4.2.5 Map design

The last step in the campus map generation process is the elaboration of a conceptually fitting design. The requirements for such a design mainly consist of the fact that all important land-

marks should be easily identifiable and explorable by the user as well as that the map should fit into the already created UI design.

For the color scheme of the map, inspiration can be taken from already established map services [9] [10] and the real-world counterparts of the presented entities. Green areas are displayed in a green tone (HEX #71C79D), water is displayed in blue (HEX #90CEFF), the street network is white (HEX #FFFFFF) and buildings are presented in a neutral grey tone (HEX #E6E6E6).

In Unity3d, colors are applied through so-called material objects, which further also contain parameters for lighting calculations. One material is chosen per entity. The only rendering parameters, that are modified for every material, are the ones that define the light calculation model as well as shadow casting options. The latter one is turned on, which results in the fact that in contrast to [9] and [10], all entities on the map cast and receive shadows. The light rendering mode, on the other hand, is set to the metallic workflow mode, which offers specular reflections based on predefined "metallic" and "smoothness" values. These values are set individually for every material type.

To emphasize the 3d environment as well as to form additional reference points for orientation, tree models are placed in the green areas on the north- and south sections of the campus. Furthermore, additional building structures are manually applied onto certain buildings (e.g., Hauptgebäude and Mathematikgebäude), to more closely resemble their respective real-world shapes.

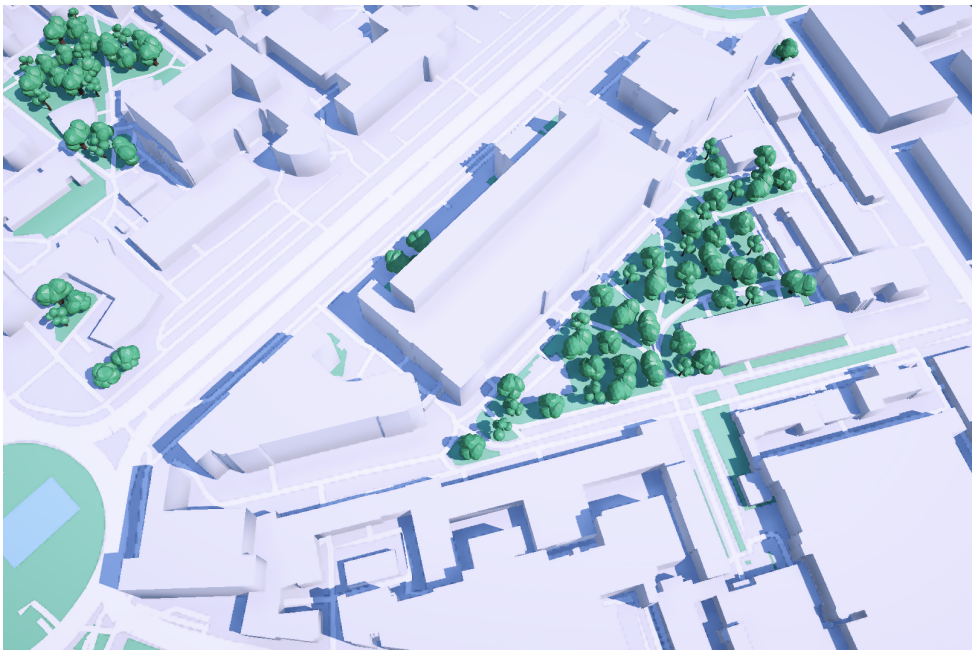

Figure 4.9: Overview of the generated campus map

4.3 Navigation system development

The mobile navigation system for TU Berlin's main campus forms the most important key feature of this thesis. It provides its users the ability to navigate and localize themselves on campus Charlottenburg and utilizes the already developed campus map and its underlying street network for routing and guidance.

The following sections provide a chronological overview of the necessary steps performed to develop such a system.

4.3.1 Representation of geodata for navigation

The main building block of the navigation system is the underlying data of TU Berlin's street network and its relation to the entities on the campus. The street network is stored as a weighted graph, with individual nodes representing building entrances and control points on a path as well as edges (connections) between those nodes, describing the actual underlying street sections.

For this thesis, the graph is constructed with the so-called NavigationNode class. This class represents, when instantiated, a single node in the weighted graph, that is used in the latter routing system. It furthermore defines the connections between individual nodes as well as their relation to TU Berlin's buildings. The following table presents an overview of the attributes of the NavigationNode:

Name	Datatype	Is required?	Default value
id	int	yes	-
latitude	double	yes	-
longitude	double	yes	-
associatedBuildingId	int	no	null
connections	int[]	no	[]

Table 4.2: NavigationNode class diagram

4.3.2 Manual generation of TU Berlin's street network

One main problem with the street network provided by OSM is the fact that the queried data is not completely suitable for navigation. The underlying issue is that certain connections between streets are missing in the dataset, which results in problems when trying to calculate the fastest way during routing. This problem especially applies to transitions between different street types (e.g., a footway merging into a small street) and small pathways on the campus, the latter making up the biggest part of the street network used in this thesis. Another important point is the fact that there are no connections between different, sometimes completely missing, entrances of a building. This disables the possibility of including indoor segments when routing.

To improve the quality of the street network for routing, a small computer app is developed that presents the queried nodes from OSM as small dots on a map (green dots are basic nodes, red dots are nodes related to an entrance of a building). The tool provides the possibility to add

additional nodes to the network as well as to create and edit connections (edges) between them (represented in blue). The manually annotated street network can be furthermore exported into a JSON file, which is stored locally in the app and deserialized into NavigationNodes during runtime. The following figure provides an overview of the manually annotated street network for navigation.

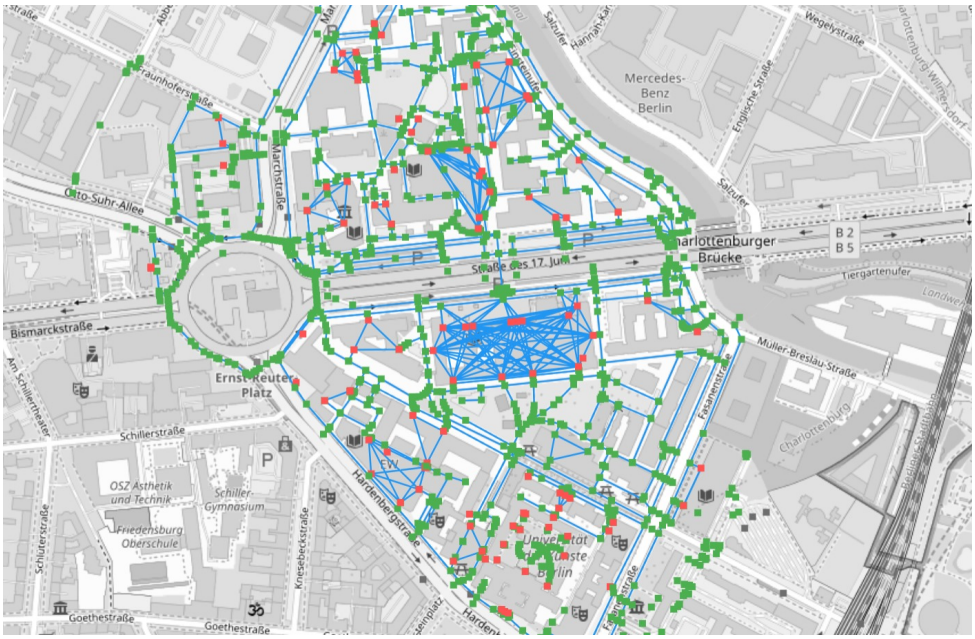

Figure 4.10: Manually annotated street network of campus Charlottenburg

4.3.3 Defining weights and calculating ETA

Weights are values assigned to edges in a graph, that represent a certain cost metric. When used in a graph for navigation purposes, they usually define how efficiently and fast a route section, defined as the edge between two nodes, can be traveled. Well-chosen weights are therefore a fundamental component for routing through a road network. This section presents the function used for determining the weight of an edge for the previously defined graph.

A weight in this thesis is defined as the (estimated) seconds needed to walk the route section defined by an edge. The following parameters are considered when choosing a weight for an edge:

- Distance d between both nodes that define the edge
- Walking speed v of a human
- Whether traffic lights need to be passed when walking the route section
- Whether the edge leads through a building
- In case of an indoor section: Size and complexity of the respective building

The base for the weight calculation is formed by the time it takes to walk the distance of the respective route section. This can be calculated with d/v. Since the largest part of the users

consists of younger students, the average walking speed for this age group is selected from [44] as $v = 1,4m/s$.

Due to the way the nodes and edges were chosen when the street network was constructed, an edge can maximally contain one pair of traffic lights on its route section. For each edge to which this applies, an additional traffic light bonus time c_{tl} is added to the weight. Study [45] portrays that $c_{tl} = 40s$ can be chosen as an upper bound for the waiting time at a traffic light.

The last important factor that needs to be considered is the adjustment of weights for indoor route sections: Since every edge in the graph either entirely lies outdoors or completely leads through a building, the length of an indoor edge can be used as a metric to indicate the complexity and size of its respective building (longer indoor sections indicate more complex buildings, which in turn, are regarded as responsible for longer walking times). To adjust for this, the distance d of the route section of an edge is multiplied by an environment factor f, which is either set to $f_o = 1$ for outdoor sections or $f_i = 1.05$ for indoor sections. This approximates that an indoor section takes 5% percent longer than its equivalent outdoor section.

This results in the following equation for a weight w as a function of an edge e and its corresponding distance d_e:

$$w_e = \begin{cases} (v * f_o)/d_e & \text{for an outdoor edge without traffic light} \\ (v * f_o)/d_e + c_{tl} & \text{for an outdoor edge with traffic light} \\ (v * f_i)/d_e & \text{for an indoor edge} \end{cases}$$

To further calculate the estimated total walking time t of a route consisting of edges 0 ... n, their respective weights need to be summed:

$$t = \sum_{i=0}^{n} w_{e_i}$$

4.3.4 Routing across the campus

After defining the weighted graph for TU Berlin's street network, the fastest routes between two nodes can be calculated using a routing algorithm. For this thesis, a simple implementation of Dijkstra's algorithm for finding the shortest paths in a graph [46] is used.

4.3.5 Embedding the current user location via GPS

There are two different use case scenarios for navigation system usage: On the one hand, users can search for two different POIs on the campus and display the fastest route between them. On the other hand, a user searches for a specific destination and expects the system to directly calculate and display the route between its current location and the selected destination point. The latter scenario needs the capability to detect the user's current location, which is realized with the help of the built-in GPS module of the mobile device.

To achieve this, two different methods for resolving the GPS position are used: Firstly, to determine the starting point of the route calculation, the current position is detected with a one-shot query. After the respective route is then calculated and the live navigation functionality is

initiated by the user, the GPS API is queried with a stream, continuously providing the current device position by firing a callback. This behavior can be modeled as a loop which serves as the foundation for the procedure described by 3.1.

4.4 Interactive information layer development

The second component of the app consists of an interactive information layer that acts as a location-based service and provides the user with detailed information about TU Berlin's campus. The following sections describe the implementation details, dealing with the data model, its partition into offline and online categories and the related acquisition process from TU Berlin's and Studierendenwerk's web resources.

4.4.1 Underlying data model

To create a foundation for the development process of the information layer, the needed data must be specified. This is done in the form of a relational database schema, consisting of all entities, their attributes and the relations between them. The fact whether a specific dataset is stored locally on the device (offline) or a web server (online) as well as the publicly available web resources, from which the data is queried are also specified in this process. The following table provides an overview of the relevant actors in this dataset:

Entity	**Storage type**	**Update interval**	**Type of update**	**Source**
Building	offline	Infrastructural changes	App update	TU Berlin's website
NavigationNode	offline	Infrastructural changes	App update	OpenStreetMap
Canteen	offline	Infrastructural changes	App update	Studierendenwerk's website
Room	offline	Infrastructural changes	App update	MOSES
Course	offline	By semester	App update	MOSES and ISIS
CourseEvent	offline	By semester	App update	MOSES
Event	online	Daily	via REST-API	TU Berlin's website
Meal	online	Daily	via REST-API	Studierendenwerk's website

Table 4.3: Overview of information layer entities

As displayed in the table, all data with low timeliness regarding the update interval is stored locally on the client device. This provides the user the possibility to access it without the need for a network connection. It nevertheless comes with the drawback that an update of the data can only take place with a complete app update, which would need to be downloaded from the respective app stores in a production use case. Online storage of the respective data with a semester-wise in-app synchronization system should be therefore taken into consideration in a real-world usage scenario. The latter point mainly applies to the course timetables ("Course" and "CourseEvent" entities), which changed by semester.

Since daily changes can occur in the meal and event datasets, both information endpoints are constantly provided via REST-API from a web server and therefore only available when a network connection can be established. To obtain a more precise overview of the individual attributes and relations of the entities, the complete database schema is provided in the following figure. Green tables represent locally stored (offline) entities.

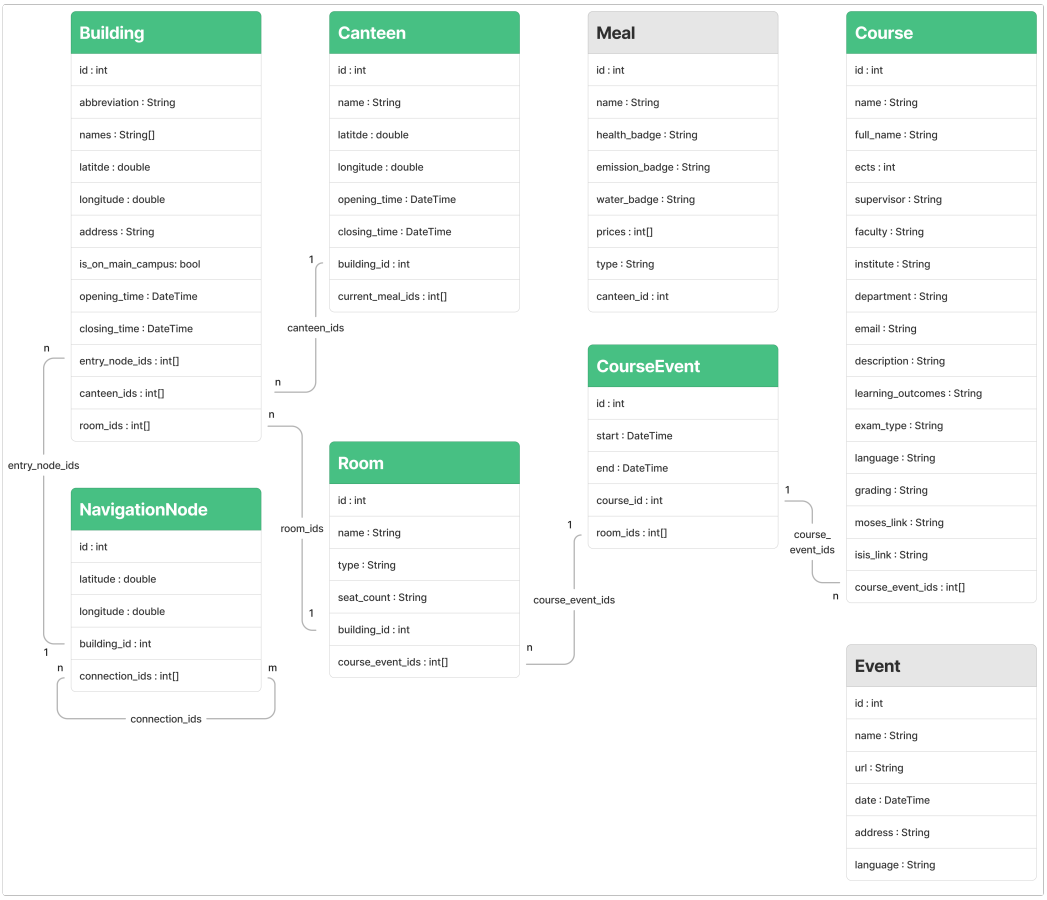

Figure 4.11: Database schema of the campus app

4.4.2 Collection of campus relevant information from the web

The foundation for every data collection operation consists of a Python script, mainly dependent on the Beatfulsoup [2] package, which provides the possibility to parse HTML files and search for specific elements and their attributes in them. These scripts are then, according to the category of their dataset (offline vs. online), deployed either on a web server or used locally for data collection. This section provides a short overview of the different scripts, their deployment and the resulting data formats they produce.

[2] https://pypi.org/project/beautifulsoup4/

NavigationNode: Data for NavigatonNodes creates the building block for the navigation system and is therefore implemented locally on the device. It is furthermore only updated when a breaking infrastructural change occurs on TU Berlin's campus. The respective acquisition script for the data is intended for manual usage and downloads the relevant information via Overpass Turbo API from OpenStreetMap. Since the downloaded data needs heavy manual annotation, the rest of the collection process is done manually with the street network tool described in the subsection for street network genetation.

Building: Buildings are also part of TU Berlin's campus infrastructure and therefore rarely need updates. Since no central online resource contains all the data to fully describe the set of them and the amount of buildings is small compared to the size of the other datasets, the scraping process is done manually. Similar to the NavigationNode dataset, all data is locally stored on the client's device in a JSON file.

Room: Rooms are downloaded from TU Berlin's MOSES system [3] where all rooms are provided and categorized by their respective building. The script starts by loading the provided website and then further proceeds to retrieve the room information by building from all the linked subsites. After completion, the dataset is exported in JSON format for local embedding. Room data is the third dataset connected to TU Berlin's infrastructure and therefore also only needs occasional updates.

Course and CourseEvent: The obtaining process for the course and CourseEvent datasets consists of several different steps: Firstly, all timetables containing the relevant information for the upcoming semester are downloaded from MOSES' room occupancy website [4] in CSV format. After completion, the script categorizes the different timetable data by course. This step results in the course name and its room occupancies, which are assigned to the course in the form of the CourseEvent data type. To find out all further required attributes of a course, its name is used to query the MOSES course database [5]. If a fitting course is found on that page, the course's MOSES description is opened and all relevant attributes are extracted. It can be assumed that every course without a respective MOSES description is not relevant to the dataset and therefore can rejected from the dataset. To complete the data collection process, the course name is also queried against the ISIS course search [6], from where, if possible, a link leading to the course's ISIS page is taken. The data is finally also exported in JSON format for offline usage and is updated by semester.

Canteen: Canteens are the fourth and last entity which is usually not subject to change. This point together with the fact that TU Berlin's campus only consists of 7 different canteen leads to a manual data collection approach for this datatype. This data is also exported into a JSON file for local offline storage.

Meal: Meal plans are updated weekly (sometimes daily) on the websites of Studierendenwerk Berlin [7]. This means, that the data needs to be collected daily and cannot be stored locally on the client's mobile device. For the meal collection process to work, the meal table presented in the database schema is modeled into a database system running on a web server. The Python

[3] https://moseskonto.tu-berlin.de/moses/verzeichnis/raeume/gebaeude.html
[4] https://moseskonto.tu-berlin.de/moses/verzeichnis/veranstaltungen/raum.html?raumgruppe=31&search=true
[5] https://moseskonto.tu-berlin.de/moses/modultransfersystem/bolognamodule/suchen.html
[6] https://isis.tu-berlin.de/
[7] https://www.stw.berlin/mensen/einrichtungen/technische-universit%C3%A4t-berlin/mensa-tu-hardenbergstra%C3%9Fe.html

script for meal data collection is deployed as a cloud function and a CRON job, adjusted to fire daily at 00:00 o'clock, is set up to trigger the system. The cloud function then removes all the meal data stored in the database, downloads the new meals from Studierendenwerk's website and inserts them back into the online storage. A REST API connected to the database is used to provide the data to the mobile client.

Event: Events make up the second dataset that needs daily updates. They are queried from TU Berlin's online event calendar [8] and are collected in a similar way to the meal plans.

Data	Number of queried objects	Size on disk
NavigationNode	1820	279 KB
Building	63	29 KB
Room	559	53 KB
Course and CourseEvent	1942, 31583	13,9 MB
Canteen	7	4 KB
Meal	avg. 70	approx. 30 KB
Event	avg. 50	approx. 25 KB

Table 4.4: NavigationNode class diagram

The upper table presents the object count as well as the disk size of the queried data. One thing that stands out is the enormous size of the "Course and CourseEvent" data when compared to the other datasets. This aspect together with an efficient loading and processing system for this dataset, could be a possible task for future work.

4.5 User interface development

4.5.1 Navigation system

4.5.2 Information layer

4.5.3 Enhancing the user experience with additional screens and features

[8] https://www.tu.berlin/veranstaltungsdetails

5 Evaluation

This section provides an overview of a set of metrics established to compare the developed campus app to already established routing solutions for mobile devices (Google Maps and Apple Maps), highlight the benefits of the campus app over the manual campus map provided by TU Berlin and measure the improvement of usability when compared to TU Berlin's online web resources. Furthermore, all steps regarding implementation, data acquisition and measurement as well as the resulting consequences for the respective metric are presented.

5.1 Campus map verfication

The digital campus map of TU Berlin forms the app's most important element. It is therefore essential to evaluate and compare its implementation to other digital maps. This section presents an overview of different metrics that can be evaluated quantitatively for map verification and compares the actual measurements to the geolocation services of Android and IOS.

To verify the campus map, an important emphasis must be laid on the correctness and quality of the underlying geodata. To achieve this, two different benchmarks, one for measuring the geographical resolution of a map and one for measuring its geocoding capabilities, are introduced in the following sections.

5.1.1 Comparison of map resolution

The geographical resolution of a map describes the level of detail that the underlying geodata has. A more detailed set of geodata can result in better map visualization for the user interface (improving usability as well as localization capabilities) and a more complex underlying navigation graph that can calculate individual routes more precisely.

To measure a comparable indicator for the geographical resolution of a particular map, the underlying geocoding API of the respective mobile app provider is used. This API provides the possibility to query the system's most fitting geodata points for a particular geographical input (pair of latitude and longitude).

It can be assumed that the distances of the candidate points returned by a geocoding API to the original point can be seen as a metric that indicates the geodata coverage of the specified region, with low distances suggesting that the system contains an adequate representation of the area around the geographic campus point. The following list describes the set of steps that are performed to measure this map resolution metric for the whole campus:

1. A random set of N geographical points $P = \{p_0, p_1, p_2, \ldots\}$, each lying on campus Charlottenburg, is generated
2. The set of the system's most fitting candidate points $C_i = \{c_{i_0}, c_{i_1}, c_{i_2}, \ldots\}$ is queried for each random point p_i from the respective geocoding API
3. The distances $D_i = \{dist(p_i, c_{i_0}), dist(p_i, c_{i_1}), dist(p_i, c_{i_2}), \ldots\}$ are calculated
4. The minimal distance $d_i = \min(D_i)$ for each point p_i is retrieved and the average of all minimal distances $\frac{d_0+d_1+d_2+\cdots+d_N}{N}$ for a specific geocoding provider is calculated
5. These averaged values are compared between geocoding engines and provide an indicator for the geographical node coverage of each system's campus representation

For practical implementation, the campus is split into 3 bounding boxes 1. A total of 250 random geographic points are generated for each bounding box and the average closest point distances are calculated for each campus region and each geocoding provider API (campus app, Android and IOS). The following figure presents the results:

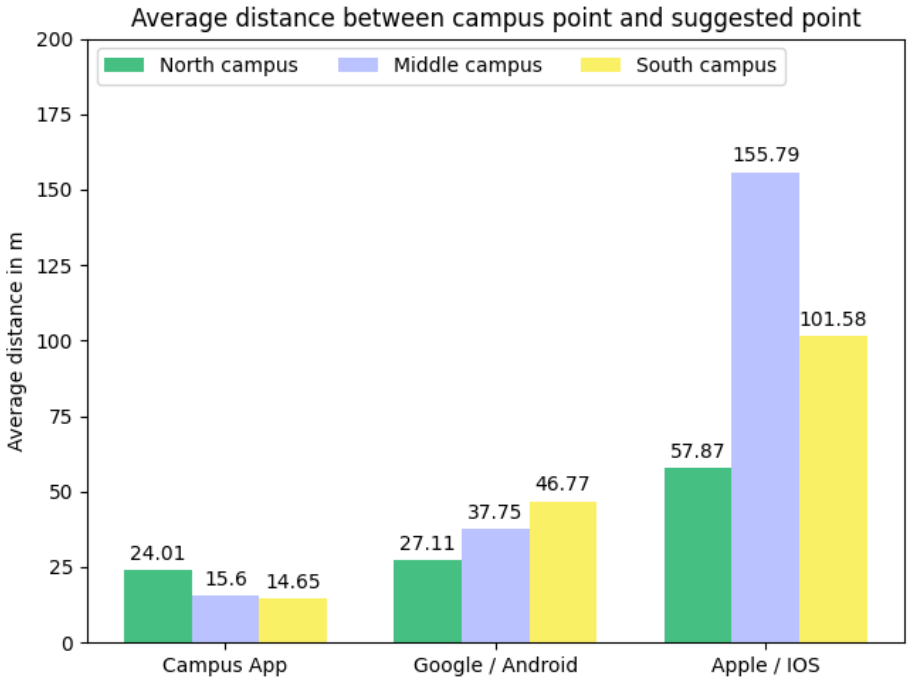

Figure 5.1: Average distance between campus point and suggested point

The figure shows that the campus app outperforms the geocoding capabilities of the Android and IOS operating system geocoding services on randomly sampled point datasets for every region of campus Charlottenburg by factors 1 to 3 for Google Maps and 2 to 10 for Apple Maps.

The values for the entire campus are 18,08 m for the campus app, 37,21 m (approx. factor 2) for Google Maps and 105,08 m (approx. factor 5) for Apple Maps. When extrapolated for the entirety of all geographical points on the campus, it can be concluded that the overall resolution of campus Charlottenburg differs by presented factors between those geocoding APIs.

To provide a more differentiated evaluation of the campus coverage, a set of heatmaps is created, each displaying the map of campus Charlottenburg with the geodata coverage of the respective geocoding API.

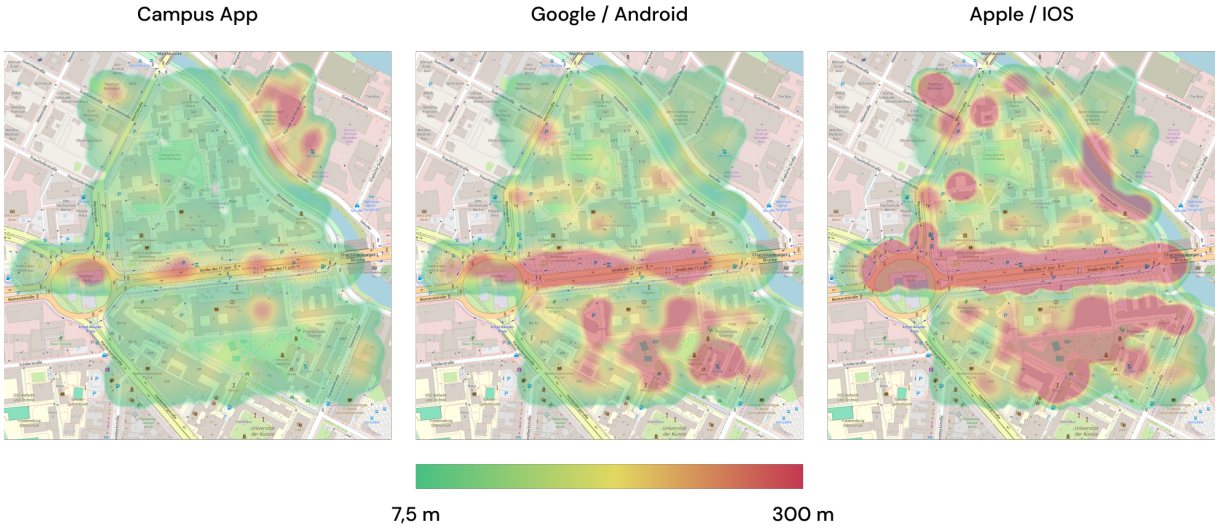

Figure 5.2: Heatmaps for campus map coverage by average point distance

One important factor that can be drawn from both data representations is the fact that the north side of campus Charlottenburg has a better geodata coverage than the south side and the middle section throughout all different providers. The heatmaps also present the fact that the campus app has the best average campus resolution and that its resolution weaknesses can (since no important entity or pathway is located in these areas) be seen as negligible for the proper functionality of the system.

Generally can be said that the Android geolocation services are more precise than the IOS counterparts in the scope of campus Charlottenburg. Both systems nevertheless struggle when points are queried for the south side of campus Charlottenburg as well as for Straße des 17. Juli. The latter weakness mostly arises from the fact that every geolocation in the campus-splitting section of Straße des 17. Juli gets mapped onto the TU main building.

5.1.2 Comparison of geocoding capabilities for campus-specific names and identifiers

One important factor for the usability of a geocoding system is the resolution of human-readable entity names and identifiers into their respective geographic coordinates. Regarding the focus on TU Berlin's campus in Charlottenburg, a system should be able to correctly identify the building's abbreviations (e.g., TEL, MAR, H, ...) as well as their official names (e.g., TU-Hochhaus, Marchstraße-Gebäude, Hauptgebäude, ...). This section provides a benchmark to measure this ability across the geocoding APIs of the campus app, Google / Android and Apple / IOS.

To perform such a benchmark, a dataset is established, containing the entirety of the campus-specific identifiers (abbreviations) and official names of TU Berlin's entities. In exceptional cases, additional commonly used building names (e.g., "Telefunken Hochhaus" instead of "TU-

Hochhaus") are also added to the dataset. This dataset is then fed into the respective geocoding APIs and the sets of possible candidate points for each query are retrieved. If at least one of the resulting candidate points is located close enough to the respective center coordinates of the inputted entity (in the test case less than 100 meters away), the human-readable input is counted as recognized by the system. To compensate for the fact that the Android and IOS geocoding APIs are not focussed on TU Berlin's campus entities, the suffix "TU Berlin" is added to each query, mimicking a common user behavior when searching for campus entities in both respective mobile apps. Furthermore, the geocoding of the actual address for each building is also analyzed for reference. The following figure presents the results.

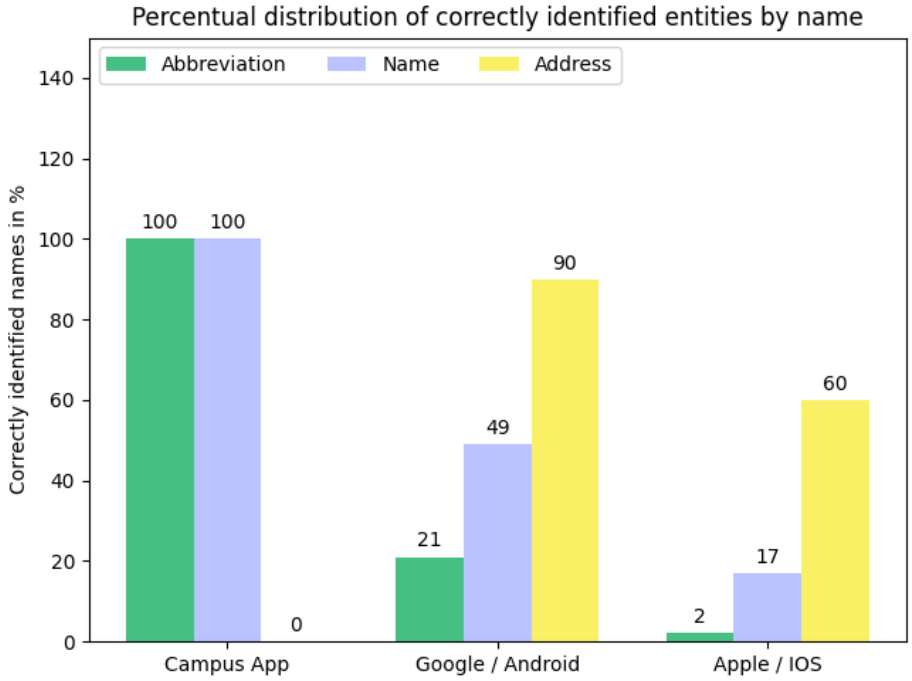

Figure 5.3: Average distance between campus point and suggested point

The campus app surpasses the geocoding capabilities of Google and Apple in terms of knowledge of entity abbreviations and names. Furthermore, Google outperforms Apple throughout all categories, but both systems struggle to detect an adequate amount of abbreviations. The name metrics are in the midfield, while the systems tend to perform better when resolving direct building addresses.

5.2 Navigation system verification

One important criterion for navigation system verification lies in the quality of the routes suggested by it. A route is thereby defined as the suggested path between a specified start and destination point, which, in the case of this thesis, are both part of TU Berlin's main campus in Charlottenburg. To analyze named quality for a specific route, the estimated time it takes to walk as well as its length are both measured. Lower walking times, as well as shorter route lengths, are thereby both regarded as preferable.

To perform a comparison between different navigation systems, a set of predefined start and destination points is established. The route calculation process of each navigation system is then applied to the defined reference points and the route metrics of the suggested routes are collected respectively. The navigation systems can be then compared for a specific pair of start and destination points as well as for the average metrics of the whole dataset.

The pool of reference points for this comparison is chosen from the underlying building data of TU Berlin. Each center point of a building is added to the set and can be used as a starting and destination point for route calculation. The navigation systems therefore calculate all possible routes between TU Berlin's buildings. This comes with two advantages: On the one hand, the chosen pool of possible routes is large enough to be quantitatively meaningful. On the other hand, the navigation between different buildings portrays a common task for the target audience of the campus app and therefore mimics the expected user behavior. The chosen pool of reference points is furthermore representative of the geographical structure of campus Charlottenburg.

The following table presents the average results for the entire reference point dataset measured respectively on the campus app, Google Maps and Apple Maps.

Navigation system	Average walking time	Average route length
Campus App	6 min 44 s	approx. 510 m
Google Maps	9 min 30 s	approx. 690 m
Apple Maps	9 min 38 s	approx. 670 m

Table 5.1: Comparison of walking time and route length

The measured key figures suggest average time savings of 30% (Google Maps) and 30% (Apple Maps) as well as average distance savings of 23% (Google Maps) and 20% (Apple Maps) when the route recommended by the campus app is chosen over its counterparts. To further analyze the underlying factors for this speedup, the percentual route metrics are split according to the underlying route length suggested by the campus app:

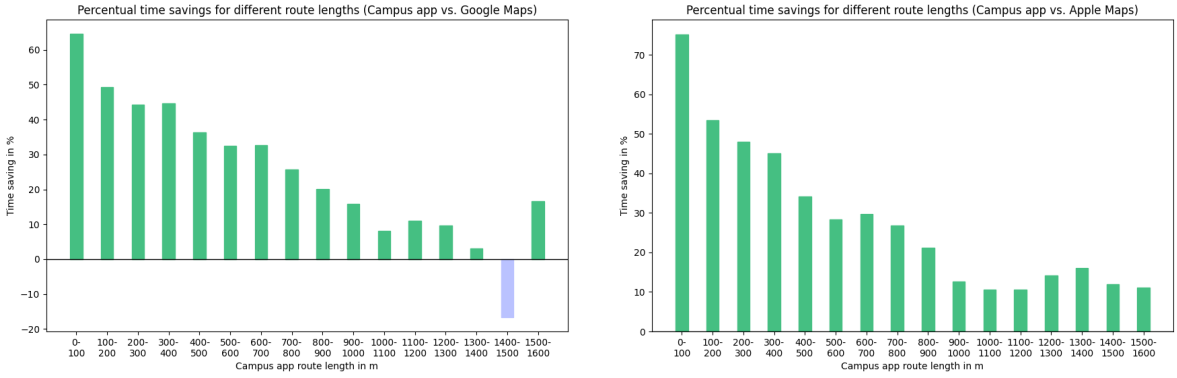

Figure 5.4: Average percentual time savings for different route lengths

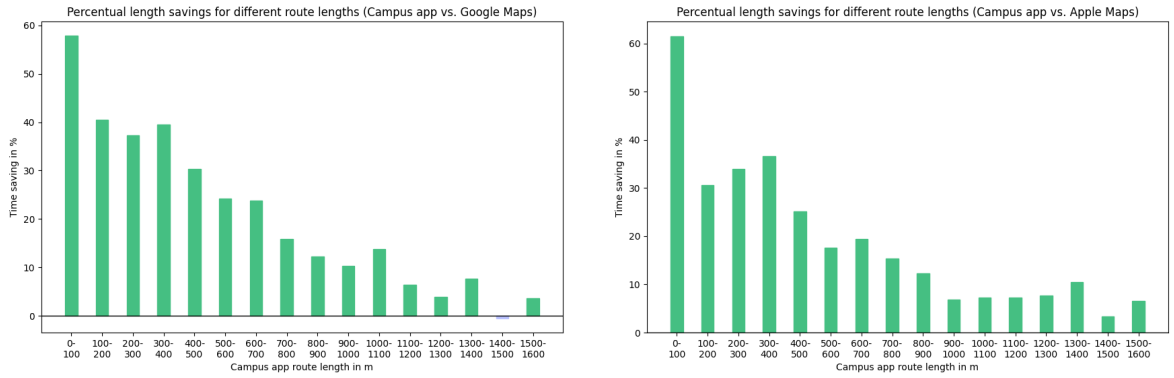

Figure 5.5: Average percentual distance savings for different route lengths

One main evaluation point that can be derived from the plots is the fact that through all route length categories (except for the distance interval of 1.4 km to 1.5 km in the Google Maps comparison), the suggested routes from the campus app always result in faster and shorter paths. This speedup is especially high for shorter routes (< 700 m) and declines with increasing route length. The overall biggest speedup is achieved in the category for routes under 100 meters. In this section, start and destination points often lay in different but connected buildings. The routing for this circumstance can be solved very efficiently by the campus app as an indoor route between both buildings can be proposed.

5.3 Information layer verification

6 Conclusion

List of Tables

List of Figures

Bibliography

[1] I. Getting, "Perspective/navigation-the global positioning system," *IEEE Spectrum*, vol. 30, no. 12, pp. 36–38, 1993.

[2] S. Kumar and K. B. Moore, "The evolution of global positioning system gps technology," *Journal of Science Education and Technology*, vol. 11, pp. 59–80, 2002.

[3] S. Farahani, "Chapter 7 - location estimation methods," pp. 225–246, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780750683937000078

[4] Google, "Fused location provider api | google for developers," 2023. [Online]. Available: https://developers.google.com/location-context/fused-location-provider?hl=de

[5] "OpenStreetMap," Oct. 2023, [Online; accessed 29. Oct. 2023]. [Online]. Available: https://www.openstreetmap.org/#map=14/53.3797/9.7587

[6] "Overpass turbo – OpenStreetMap Wiki," Sep. 2023, [Online; accessed 29. Oct. 2023]. [Online]. Available: https://wiki.openstreetmap.org/wiki/Overpass_turbo

[7] M. Minghini and F. Frassinelli, "Openstreetmap history for intrinsic quality assessment: Is osm up-to-date?" *Open Geospatial Data, Software and Standards*, vol. 4, 2019. [Online]. Available: https://doi.org/10.1186/s40965-019-0067-x

[8] "Is OSM up-to-date?" May 2022, [Online; accessed 29. Oct. 2023]. [Online]. Available: https://is-osm-uptodate.frafra.eu/#17/52.51274/13.32600

[9] "Google Maps Platform," Jun. 2023, [Online; accessed 30. Oct. 2023]. [Online]. Available: https://developers.google.com/maps?hl=de

[10] "Karten," Oct. 2023, [Online; accessed 30. Oct. 2023]. [Online]. Available: https://www.apple.com/de/maps

[11] H. Mehta, P. Kanani, and P. Lande, "Google maps," *International Journal of Computer Applications*, vol. 178, pp. 41–46, 05 2019.

[12] M. Noto and H. Sato, "A method for the shortest path search by extended dijkstra algorithm," vol. 3, pp. 2316–2320 vol.3, 2000.

[13] "Core Location | Apple Developer Documentation," Oct. 2023, [Online; accessed 29. Oct. 2023]. [Online]. Available: https://developer.apple.com/documentation/corelocation

[14] W. Kang and Y. Han, "Smartpdr: Smartphone-based pedestrian dead reckoning for indoor localization," *IEEE Sensors Journal*, vol. 15, no. 5, pp. 2906–2916, 2015.

[15] B. Wang, X. Liu, B. Yu, R. Jia, and X. Gan, "Pedestrian dead reckoning based on motion mode recognition using a smartphone," *Sensors*, vol. 18, no. 6, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/6/1811

[16] M. Basso, A. Martinelli, S. Morosi, and F. Sera, "A real-time gnss/pdr navigation

system for mobile devices," *Remote Sensing*, vol. 13, no. 8, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/8/1567

[17] A. Stark, M. Riebeck, and J. Kawalek, "How to design an advanced pedestrian navigation system: Field trial results," pp. 690–694, Sep. 2007.

[18] A. Küpper, *Location-Based Services: Fundamentals and Operation*. John Wiley and Sons Ltd, 2005.

[19] A. Küpper, U. Bareth, and B. Freese, "Geofencing and background tracking – the next features in lbss," *INFORMATIK 2011 - Informatik schafft Communities*, 2011.

[20] P. Sadhukhan, N. Mukherjee, and P. K. Das, *Location-Based Services for Smart Living in Urban Areas*. Cham: Springer International Publishing, 2021, pp. 53–69. [Online]. Available: https://doi.org/10.1007/978-3-030-71288-4_3

[21] S. Rodriguez Garzon and B. Deva, "Geofencing 2.0: Taking location-based notifications to the next level," p. 921–932, 2014. [Online]. Available: https://doi.org/10.1145/2632048.2636093

[22] "Campaign of the Year: Burger King's 'Whopper Detour'," *Marketing Dive*, Dec. 2019, [Online; accessed 28. Nov. 2023]. [Online]. Available: https://www.marketingdive.com/news/burger-king-whopper-detour-mobile-marketer-awards/566224

[23] "Jakob Nielsen, Ph.D. and Principal at Nielsen Norman Group," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://www.nngroup.com/people/jakob-nielsen

[24] "Paul M. Fitts," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://research.osu.edu/paul-m-fitts

[25] J. Yablonski, *Laws of UX: Using Psychology to Design Better Products & Services*. O'Reilly Media, Incorporated, 2020. [Online]. Available: https://books.google.de/books?id=fve3yQEACAAJ

[26] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com

[27] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/fittss-law

[28] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/jakobs-law

[29] ——, "Law of Common Region | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/law-of-common-region

[30] ——, "Hick's Law | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/hicks-law

[31] ——, "Miller's Law | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/millers-law

[32] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/aesthetic-usability-effect

[33] "Campusplan: Standorte der Technischen Universität Berlin," Jan. 2024, [Online; accessed 11. Jan. 2024]. [Online]. Available: https://www.tu.berlin/studieren/uni-leben/campusplan

[34] J. Yablonski, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/teslers-law

[35] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available:

https://lawsofux.com/law-of-proximity

[36] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/postels-law

[37] ——, "Home | Laws of UX," Jan. 2024, [Online; accessed 13. Jan. 2024]. [Online]. Available: https://lawsofux.com/goal-gradient-effect

[38] ——, "Law of Similarity | Laws of UX," Jan. 2024, [Online; accessed 14. Jan. 2024]. [Online]. Available: https://lawsofux.com/law-of-similarity

[39] "RFC 7946: The GeoJSON Format," Jan. 2024, [Online; accessed 21. Jan. 2024]. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7946

[40] "Wayback Machine," Jan. 2024, [Online; accessed 21. Jan. 2024]. [Online]. Available: https://web.archive.org/web/20141009142830/http://earth-info.nga.mil/GandG/wgs84/web_mercator/(U)%20NGA_SIG_0011_1.0.0_WEBMERC.pdf

[41] "Wayback Machine," Jan. 2024, [Online; accessed 21. Jan. 2024]. [Online]. Available: https://web.archive.org/web/20150215113220/http://cegis.usgs.gov/projection/pdf/Battersby_Implications%20of%20Web%20Mecator%20and%20Its%20Use%20in%20Online%20Mapping.pdf

[42] "Mercator – OpenStreetMap Wiki," Nov. 2023, [Online; accessed 21. Jan. 2024]. [Online]. Available: https://wiki.openstreetmap.org/wiki/Mercator

[43] M. Held, "Efficient and reliable triangulation of polygons," pp. 633–643, 1998.

[44] "Establishing Pedestrian Walking Speeds," Feb. 2024, [Online; accessed 17. Feb. 2024]. [Online]. Available: https://www.westernite.org/datacollectionfund/2005/psu_ped_summary.pdf

[45] Zafri, Niaz and Rony, Atikul and Adri,, "Analysis of pedestrian crossing speed and waiting time at intersections in dhaka," *Infrastructures*, vol. 4, p. 39, 07 2019.

[46] Javaid, Muhammad Adeel, "Understanding dijkstra's algorithm," 2013. [Online]. Available: http://dx.doi.org/10.2139/ssrn.2340905

Appendices

Appendix 1

Listing 1: Overpass Turbo query for all of TU Berlin's buildings

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
        way["building"="university"];>;
        way(456211792);>;
        way(358373103);>;
        way(104507902);>;
        way(290166587);>;
        way(172802144);>;
        way(262367508);>;
        way(48793247);>;
        way(50557514);>;
        way(993968604);>;
        way(993968602);>;
);
out meta;
```

Listing 2: Overpass Turbo query for all main roads next to the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
    way["highway"="primary"];>;
    way["highway"="secondary"];>;
);
out meta;
```

Listing 3: Overpass Turbo query for all small roads next to the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
    way["highway"="residential"];>;
    way["highway"="tertiary"];>;
);
out meta;
```

Listing 4: Overpass Turbo query for all pathways and sidewalks next to the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
```

```
    way["highway"]["highway"!="primary"]["highway"!="secondary"]["highway"!="
        residential"]["highway"!="tertiary"][!"footway"];>;
);
out meta;
```

Listing 5: Overpass Turbo query for all water green next to the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
    way["landuse"="grass"];>;
    way["leisure"="park"];>;
    way["leisure"="garden"];>;
    way["natural"="scrub"];>;
    way(314528756);>;
    way(104712588);>;
    way(531886237);>;
    way(531886252);>;
);
out meta;
```

Listing 6: Overpass Turbo query for water areas next to the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
(
    way["natural"="water"];>;
);
out meta;
```

Listing 7: Overpass Turbo query for all entrances to buildings on the campus

```
[bbox:52.49993096650543, 13.307022255971093, 52.52269751545147,
    13.341918533901309];
node["entrance"];
out meta;
```

Region name	North-West coordinates	South-East coordinates
North Campus	(52.51667973620, 13.32222127479)	(52.5133200241, 13.329653624217)
Middle Campus	(52.513220529803, 13.319945039190)	(52.5127460156, 13.330647119317)
South Campus	(52.512386299901, 13.323101586912)	(52.51048052265, 13.33127591368)

Table 1: Different campus regions