



Lehrstuhl für Informatik  
insb. Mobile Software Systeme/Mobilität

Projektbericht

# **Mobilität für Menschen mit Mobilitätseinschränkungen (MoMM)**

vorgelegt von

**Jan Günter Adelhardt, Claudia Esch, Katharina Koppermann, Nils Philipp  
Thiergarten, Adrian-Berthold Völker**

Gutachter:

**Prof. Dr. Daniela Nicklas  
Michael Sünkel**

Bamberg, 27.03.2022



---

## Zusammenfassung

Mobilität für Menschen mit Mobilitätseinschränkungen ist keineswegs eine Selbstverständlichkeit und es kommen viele Unwägbarkeiten, wie beispielsweise unpassende Wege- und Straßenzustände oder unüberwindbare Treppen, hinzu. Mithilfe von konstruktiven Algorithmen und bereits existierenden Softwarelösungen können auf Basis der frei verfügbaren OpenStreetMap-Daten Routen- und Navigationsanweisungen erzeugt werden. Gerade in der Stadt Bamberg soll jedoch eine technologische Lösung entwickelt werden, die auch Parameter für barrierefreies Routing mit in die algorithmische Berechnung einbezieht. Genau dies soll in der folgenden Arbeit analysiert, umgesetzt und evaluiert werden. Dabei wird von einer hohen Diversität der Zielgruppen ausgegangen, da sich nicht nur die Formen der Mobilitätseinschränkungen (z.B. Personen im Rollstuhl, ältere Menschen mit Geh-Einschränkungen oder Personen mit Kinderwagen), sondern auch die persönlichen Präferenzen und Möglichkeiten jeweils stark unterscheiden. Daher kommt der individuellen Parametrisierung der Suche eine große Bedeutung zu. Die Anforderungen an das Routing ergeben sich aus ersten vorläufigen Ergebnissen einer Stakeholder-Befragung von Cogita sowie den im Projektteam erarbeiteten Ideen.

Im weiteren Verlauf der Arbeit werden dann verschiedene bereits existierende Routing-Algorithmen und Routing-Systeme genauer betrachtet und auf Tauglichkeit im aktuellen Kontext untersucht. Dabei konnten drei Systeme (OpenStreetMap, BRouter und GraphHopper) in die nähere Betrachtung aufgenommen werden. In einem speziell für diese Arbeit entwickelten und dargestellten Prototypen werden dann die drei ausgewählten Routing-Systeme zur Routenerzeugung herangezogen. Dieser Prototyp erlaubt es die deklarierten Parameter individuell einzustellen und diese (sofern es beim jeweiligen System möglich ist) in die Routenerzeugung miteinzubeziehen. Über eine graphische Schnittstelle wird es dem Benutzer ermöglicht die errechneten Routen in einer interaktiven Karte darzustellen, zu evaluieren und dadurch auch zu nutzen. Der vorläufige Prototyp, dessen Backend und die gewählten Entwicklungsparadigmen werden in der vorliegenden Arbeit ebenfalls genauer analysiert und erklärt.

Folgend werden dann die aktuellen Unzulänglichkeiten der untersuchten Systeme herausgestellt und Implementationsanweisungen zur zukünftigen Umsetzung gegeben. Ebenfalls werden Handlungsaufforderungen dargelegt, was, welche Daten und welche zusätzlichen Programmieraufgaben benötigt werden, um ein zukünftiges barrierefreies Routing für Menschen mit Mobilitätseinschränkungen in der Stadt Bamberg durch eine entsprechende Applikation, beziehungsweise eine Webseite, zu ermöglichen.

## Abstract

Mobility for people with mobility impairments is by no means a matter of course and there are many imponderables, such as unsuitable path and road conditions or insurmountable stairs. With the help of constructive algorithms and already existing software solutions, routing and navigation instructions can be generated on the basis of the freely available OpenStreetMap data. However, especially in the city of Bamberg, a technological solution is to be developed that also includes parameters for barrier-free routing in the algorithmic calculation. This is precisely what will be analysed, implemented and evaluated in the following work. Thereby, a high diversity of target groups is assumed, since not only the forms of mobility restrictions (e.g. people in wheelchairs, elderly people with walking restrictions or people with prams), but also the personal preferences and possibilities differ significantly in each case. Therefore, the individual parameterisation of the navigation is of great importance. The requirements for the routing result from the first preliminary results of a stakeholder survey of Cogita and the ideas developed by the project team.

In the subsequent course of the work, various already existing routing algorithms and systems will be examined more closely and assessed for their suitability in the specific context. Three systems (OpenStreetMap, BRouter and GraphHopper) were included in the closer examination. In the prototype developed and presented especially for this work, the three selected routing systems are used for route generation. The defined parameters can be set individually and thus (if possible for the respective system) included in the route generation. A graphical interface enables the user to display, evaluate and thus also use the computed routes in an interactive map. The preliminary prototype, its backend and the chosen development paradigms will be analysed and implemented in more detail in this thesis.

In the remainder of this paper, the current shortcomings of the systems examined are highlighted and implementation instructions for future implementation are given. Calls for action are also outlined as to what, which data and which additional programming tasks are required to enable future barrier-free routing for people with mobility impairments in the city of Bamberg through a corresponding application or website.

# Inhalt

<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung und Ziele . . . . .	1
1.2 Spezifikationen und Parameter . . . . .	1
1.3 Gliederung . . . . .	1
1.4 Hinweise zur Autorenschaft und zum Urheberrecht . . . . .	2
<b>2 Vorgehensweise</b>	<b>3</b>
<b>3 Grundlagen</b>	<b>4</b>
3.1 OpenStreetMap . . . . .	4
3.2 Funktionsweise von Routing-Algorithmen . . . . .	4
3.3 Existierende Arbeiten . . . . .	5
3.4 Einordnung zu MoMM . . . . .	7
<b>4 Untersuchte Algorithmen</b>	<b>8</b>
<b>5 Ausgewählte Algorithmen</b>	<b>10</b>
5.1 Openrouteservice . . . . .	10
5.2 BRouter . . . . .	10
5.3 GraphHopper . . . . .	11
<b>6 Geo-Daten in Bamberg</b>	<b>12</b>
6.1 Relevante Geo-Daten . . . . .	12
6.2 Definition des Testgebiets . . . . .	12
6.3 Qualität der Geo-Daten im Testgebiet . . . . .	13
<b>7 Prototyp</b>	<b>15</b>
7.1 Grundsätzliche Anforderungen an den Prototypen . . . . .	15
7.2 Umsetzungsmöglichkeiten des Prototyps . . . . .	15
7.3 Programmierung des Prototyps . . . . .	16
<b>8 Backend</b>	<b>17</b>
8.1 Programmierung des Backends . . . . .	17
8.2 Funktionalität des Backends . . . . .	17
8.3 Bereitstellung des Backends . . . . .	19
<b>9 Frontend</b>	<b>22</b>
9.1 Gestaltung der Benutzeroberfläche . . . . .	22
9.2 Technische Umsetzung der Benutzeroberfläche . . . . .	23

<b>10 Evaluation</b>	<b>27</b>
10.1 ORS . . . . .	27
10.2 BRouter . . . . .	30
10.3 GraphHopper . . . . .	32
<b>11 Fazit und Handlungsempfehlungen</b>	<b>34</b>
11.1 Vergleichende Betrachtung der Algorithmen . . . . .	34
11.2 Konkrete Handlungsempfehlungen . . . . .	34
11.3 Abschließende Zusammenfassung . . . . .	35
<b>Abkürzungen</b>	<b>37</b>
<b>Abbildungen</b>	<b>37</b>
<b>Literatur</b>	<b>39</b>

# 1 Einleitung

Das Projekt, das im Wintersemester 2021/22 im Rahmen einer Lehrveranstaltung des Lehrstuhls für Mobile Software Systeme an der Universität Bamberg betreut wurde, widmet sich der Entwicklung eines Routing-Angebots für Bamberg, das auf sehr spezifische, aber dennoch flexible Bedürfnisse ausgerichtet ist. Im Folgenden soll kurz in die Problemstellung, die Projektziele, die Spezifikationen sowie den Ablauf des Projekts eingeführt werden.

## 1.1 Problemstellung und Ziele

Kernidee des MoMM-Projekt ist es, ein für das Stadtgebiet Bamberg ausgelegtes Routing für Menschen mit temporären oder dauerhaften Mobilitätseinschränkungen zu entwickeln. Es wird von einer hohen Diversität der Zielgruppe ausgegangen, da sich nicht nur die Formen der Mobilitätseinschränkungen (z.B. Personen im Rollstuhl, ältere Menschen mit Geh-Einschränkungen oder Personen mit Kinderwagen), sondern auch die persönlichen Präferenzen und Möglichkeiten jeweils stark unterscheiden. Daher kommt der individuellen Parametrisierung der Suche eine große Bedeutung zu.

Ziel des Projekts war es, Voruntersuchungen für die Entwicklung anzustellen, geeignete Routing-Algorithmen zu identifizieren und zu testen sowie eine erste Abschätzung des Aufwands für eine Implementierung sowie für eventuell notwendige Vorarbeiten, wie etwa eine Ergänzung von OSM-Daten in Bamberg, abzugeben.

## 1.2 Spezifikationen und Parameter

Die Anforderungen an das Routing ergaben sich aus den vorläufigen Ergebnissen einer Stakeholder-Befragung von Cogita sowie den im Projektteam erarbeiteten Ideen. Als Parameter für die Testphase wurden zunächst die Beschaffenheit des Straßenbelags, die maximale Höhe der Bordsteinkanten, die maximale Steigung, die Vermeidung von Treppen sowie das Vorhandensein eines Handlaufs bei Treppen festgelegt.

Eine von Cogita organisierte Ortsbegehung mit einem Stakeholder gegen Ende der Projektlaufzeit (01. Februar 2022) hat zum einen die Priorisierung der Parameter weitgehend bestätigt, zum anderen Ergänzungen hinsichtlich des geplanten Front-Ends des Prototypen angeregt. Im Gespräch mit dem Stakeholder wurde der Wunsch nach einer Anzeige- und Auswahlmöglichkeit für verschiedene alternative Routen geäußert, um so den Nutzer\*innen eine individuelle Kosten-Nutzen-Abwägung zu ermöglichen. Dafür ist die Anzeige möglichst detaillierter Informationen über den Streckenverlauf, etwa das Höhenprofil oder der Anteil bestimmter Arten von Straßenbelägen, sinnvoll. Daraufhin wurde vom Projektteam die Implementierung einer Höhenprofil-Visualisierung für den Prototyp geprüft.

## 1.3 Gliederung

Der vorliegende Bericht gibt neben einer kurzen Einführung in die Vorgehensweise des Projektteams (Kapitel 2) und den zentralen Grundlagen für das Routing im Allgemeinen sowie das Projekt im Besonderen (Kapitel 3) vor allem den Arbeitsablauf sowie die wichtigsten Ergebnisse wieder. Zunächst wurden vom Projektteam mehrere Algorithmen untersucht (Kapitel 4) und aus diesen drei geeignet

erscheinende Kandidaten ausgewählt (Kapitel 5). Um die Funktionsweise der Algorithmen und die Möglichkeiten zur ihrer Anpassung auf die spezifischen Anforderungen des MoMM-Projekts zu testen, wurden die Geo-Daten in Bamberg analysiert und ein Testgebiet definiert (Kapitel 6). Anschließend entstand ein Prototyp (Kapitel 7, 8 und 9), der die Einstellung der individuellen Parameter sowie den direkten Vergleich der Routen, die die drei Algorithmen ausgeben, erlaubt. Mit Hilfe des Prototypen konnten die drei Algorithmen hinsichtlich ihrer Eignung für ein MoMM-Routing in Bamberg evaluiert (Kapitel 10) und abschließend Handlungsempfehlungen hinsichtlich des weiteren Vorgehens bei der Entwicklung eines zukünftigen Produkts (Kapitel 11) formuliert werden.

#### **1.4 Hinweise zur Autorenschaft und zum Urheberrecht**

Sowohl die vorliegende schriftliche Arbeit, als auch die zu erledigenden Aufgaben wurden gemeinschaftlich von den fünf Studierenden zusammengestellt. Dabei wurden jedoch teilweise Probleme und Teilaufgaben individuell gelöst. Im ähnlichen Maß verhält es sich mit den jeweiligen schriftlichen Abschnitten in der vorliegenden Arbeit, dabei haben unterschiedlichen Studierende die Hauptverantwortung für die Verschriftlichung getragen. Sollten detaillierte Urheberschaften der Texte benötigt werden, kann dies über eine separate Anfrage beantwortet werden.

---

## 2 Vorgehensweise

Im Rahmen des MoMM-Projektes fanden regelmäßige Treffen (alle 2 Wochen) der Studierenden mit Projektbetreuern vom Mobi-Lehrstuhl statt, in welchen die Idee des Projekts vorgestellt wurde und den Studierenden jeweils Aufgaben gestellt wurden, welche bis zum nächsten Termin bearbeitet und dort vorgestellt werden sollten. Eine der Aufgaben umfasste eine Grundlagenrecherche über bestehende Routing-Algorithmen und Routing-Anwendungen, deren Funktionsweise und genutzte Datenbasis, sowie die Frage, wie die OSM-Daten für Bamberg aussehen. Zudem sollte evaluiert werden, welche Routing-Algorithmen für Bamberg geeignet seien. Nach Schaffen der Grundlagen, wurde sich mit dem Design eines Prototyps beschäftigt und hierbei vor allem mit der Auswahl der Routing-Algorithmen, die vom Prototyp genutzt werden sollen, sowie mit den Anforderungen die einzelne Routing-Algorithmen anzupassen, sodass die für das MoMM-Projekt relevanten Parameter unterstützt werden können. Im weiteren Vorgehen wurde der Prototyp in Teamarbeit durch die Studierenden implementiert und es wurde festgehalten, inwiefern es Einschränkungen der Anpassbarkeit der einzelnen Algorithmen gab. Auch wurde eine Teststrecke definiert, um den Prototypen zu evaluieren. Im letzten Schritt wurden die generierten Routen der einzelnen Algorithmen für die Teststrecke unter Heranziehung der OSM-Daten für Bamberg evaluiert und Empfehlungen und Ansätze für weitergehende Projekte formuliert.

Zusätzlich zu den Treffen in der gesamten Gruppe gab es studentische Treffen, um sich über den aktuellen Stand der Aufgabenbearbeitung zu informieren und neu gestellte Aufgaben zu verteilen. Der genaue Inhalt dieser Treffen wurde protokolliert. Im Rahmen der Treffen fand auch gemeinschaftliches Programmieren/Debuggen über Bildschirmübertragungen statt. Ebenso wurde sich über die technischen Grundlagen des zu implementierenden Prototyps ausgetauscht, wobei insbesondere die Studierenden mit mehr Programmiererfahrung ihr Wissen mit den anderen teilten. Sowohl die studentischen als auch die Treffen in der großen Gruppe fanden online als Videokonferenz statt.

Um einen möglichst guten Einblick über die Anforderungen der Stakeholder (Menschen mit Mobilitätseinschränkungen) an eine Routing-Implementation zu erhalten, hat diesbezüglich die studentische Unternehmensberatung Cogita, welche mit dem MoMM-Projekt zusammenarbeitet, die Kommunikation mit den Stakeholdern geführt und die gesammelten Erkenntnisse an die Mitgliedern des MoMM-Projekt weitergetragen. Ebenso organisierte Cogita eine gemeinsame Ortbegehung eines Teils der gewählten Teststrecke in Bamberg mit den Projektteilnehmern und einem Stakeholder, welcher in einem elektrischen Rollstuhl sitzt.

Die gemeinsame Verwaltung der Projektmaterialien erfolgte über das Software-Verwaltungs-Tool GitLab. Diese Materialien umfassen Präsentationsunterlagen der erarbeiteten Grundlagen, Protokolle der studentischen Treffen, geteilte Dokumente, um Zwischenergebnisse zu dokumentieren und den Quellcode des Prototyps. Für die Auswertung der OSM-Daten wurde das Tool Java OpenStreetMap Editor verwendet, zum Erstellen des Prototypen die integrierte Entwicklungsumgebung Visual Studio Code. Um HTTP-Requests zu identifizieren, testen und zu evaluieren wurden die Entwickler Tools des jeweiligen Internet-Browsers genutzt.

## 3 Grundlagen

In diesem Abschnitt wird auf die Funktionsweise von Routing-Algorithmen, für das Routing benötigten OSM-Daten und existierende Arbeiten bezüglich der Navigation von Menschen mit Mobilitätseinschränkungen eingegangen.

### 3.1 OpenStreetMap

OSM ist ein 2004 gegründetes Open-Source-Projekt mit dem Vorhaben, eine freie Weltkarte zu schaffen, die Informationen zu Straßen, Flüssen, Läden und weiteren landschaftlichen und innerstädtischen Merkmalen enthält. Die Karte basiert auf Daten, die von der OSM-Community gesammelt werden. Diese Daten können entweder in Rohform oder anhand vorberechneter Kartenbilder verwendet werden [1]. Zu jedem Element (Laden, Straße etc.) der Karte können Tags hinzugefügt werden, die dieses Element beschreiben. Ein Tag ist ein Key-Value-Paar, wobei sowohl Key als auch Value im Textformat gespeichert werden. Durch die Tags einer Straße kann bspw. die maximal zulässige Geschwindigkeit dieser oder deren Steigung beschrieben werden [2].

### 3.2 Funktionsweise von Routing-Algorithmen

Routing-Algorithmen basieren auf der Datenstruktur von kantengewichteten Graphen. Graphen bestehen aus Knotenpunkten und Kanten, welche die Knotenpunkte miteinander verbinden. Jede Kante hat Gewichte, welche bei Routenberechnungen bspw. die Streckenlänge zwischen zwei Knotenpunkten bedeutet. Im Kontext der Problemstellung dieses Projektes ist es notwendig mehrere Parameter (z.B Steigung) statt nur die Streckenlänge zu berücksichtigen. Hierzu verwendet man Kostenfunktionen, die die unterschiedlichen Parameter miteinander berechnen und als Rückgabewert eine reelle Zahl als Gewicht zurückgeben. Im Verlauf des Projektes haben sich die Studierenden mit drei Routing-Algorithmen auseinandergesetzt, die auf zwei Algorithmen basieren. Diese sind der A\*-Algorithmus und der Kontraktionshierarchien-Algorithmus (Contraction Hierarchies). Beide basieren auf dem Dijkstra-Algorithmus, welcher nun kurz beschrieben wird. Der Dijkstra-Algorithmus berechnet für einen beliebigen Startknotenpunkt den kürzesten Weg jeden anderen Knotenpunkt im Graphen [3]. Da dies jedoch in einem Straßennetz mit mehreren Millionen Knoten zu rechenintensiv ist, verwenden viele Routing-Services A\*. Im Gegensatz zu Dijkstra erweitert A\* einen Knoten nur dann, wenn er vielversprechend erscheint. Sein einziges Ziel ist es, den Zielknoten so schnell wie möglich vom aktuellen Knoten aus zu erreichen, und nicht zu versuchen, jeden anderen Knoten zu erreichen [4]. Die Entscheidung, ob ein Knoten vielversprechend ist, basiert auf Heuristiken [5]. Der Kontraktionshierarchien-Algorithmus nutzt Eigenschaften von Graphen von Straßennetzen aus, um die Berechnungszeit von Routen deutlich zu beschleunigt. Die Beschleunigung wird erreicht, indem in einer Vorverarbeitungsphase Abkürzungen geschaffen werden, die dann bei einer Abfrage des kürzesten Weges verwendet werden, um „unwichtige“ Knoten zu überspringen. Dies beruht auf der Beobachtung, dass Straßennetze stark hierarchisch aufgebaut sind. Einige Kreuzungen, beispielsweise Autobahnkreuze, sind „wichtiger“ und in der Hierarchie höher als beispielsweise eine Kreuzung, die in eine Sackgasse führt[6]. Diese Hierarchie ist hierbei kontextabhängig und kann vom Benutzer konfiguriert werden.

### 3.3 Existierende Arbeiten

Es bestehen bereits einige andere Projekte von anderen Forschungseinrichtungen, die sich mit der Navigation für Menschen mit Mobilitätseinschränkungen auseinandergesetzt haben. Im Folgenden werden fünf Arbeiten mit Bezug zum MoMM-Projekt vorgestellt.

#### 3.3.1 WheelScout - Barrier-Free Navigation

Das Projekt WheelScout - Barrier-Free Navigation wird seit 2013/14 an der Hochschule Darmstadt im Department of Computer Science entwickelt [7]. Abbildung ?? zeigt das Konzept dieses Projekts. Die in dem Projekt entwickelte Applikation soll Nutzer\*innen, die im Rollstuhl sitzen, bei Indoor- und Outdoor-Navigation unterstützen. Das Routing und die Navigation erfolgt auf Basis der Routing-Engine GraphHopper mit Nutzung von Daten aus OSM. Hinzufügen von Barrieren und persönlichen Nutzerprofilen ist möglich (siehe Abbildung 3.1). Zusätzlich werden Sensordaten des Smartphones genutzt [8].

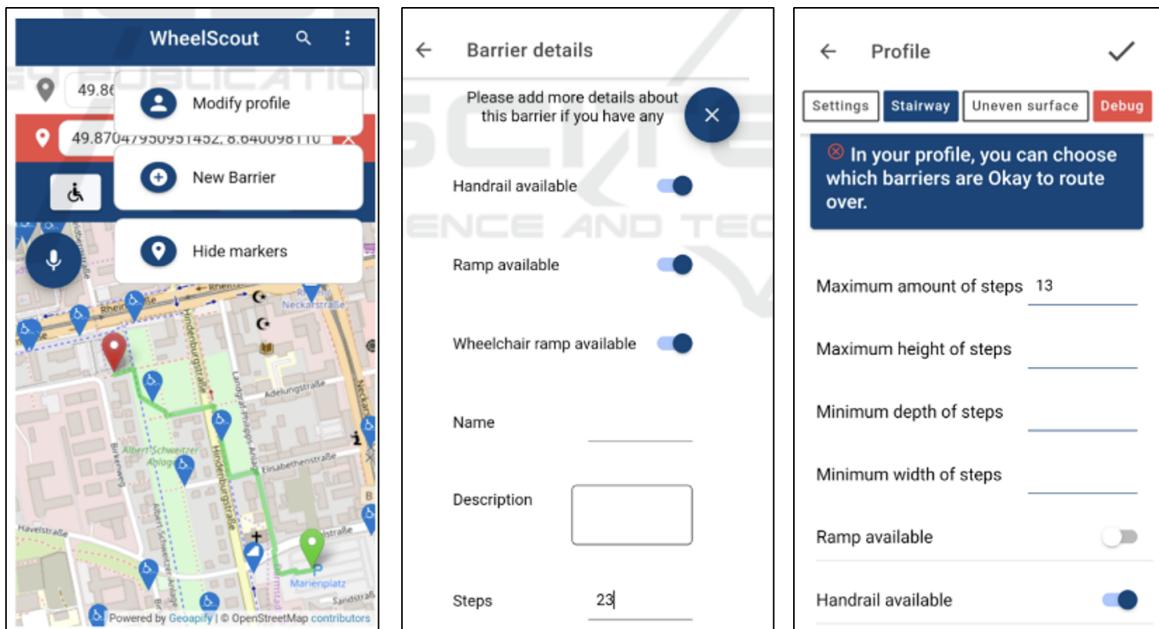


Abbildung 3.1: WheelScout App Screenshot [9]

#### 3.3.2 Accessible Maps

Das Projekt Accessible Maps wurde von der Technischen Universität Dresden in Kooperation mit dem Karlsruher Institut für Technologie entwickelt. Ziel dieses Projekts war die Entwicklung von Indoor-Karten mit individuell benötigten Informationen zur Barrierefreiheit im Beruf und am Arbeitsplatz [10]. Indoor-Karten wurden automatisch mit Computer-Vision und Deep-Learning-Verfahren generiert. Daten wurden beispielsweise aus Fluchtplänen gewonnen. In dem Projekt fand eine große Kontextanalyse potentieller Nutzer\*innen statt [11].

### 3.3.3 Maps without Barriers

Maps without Barriers ist ein tschechisches Projekt zur Kartierung von barrierefrei zugänglichen Einrichtungen und Objekten in Tschechien (siehe Abbildung 3.2). Besonderes Augenmerk wird dabei auf nationale Kulturdenkmäler und andere Objekte von touristischem Interesse gelegt. Das Projekt greift das Problem von Menschen mit Behinderungen auf, für die es derzeit schwierig ist Informationen zu der Barrierefreiheit von Reisezielen zu finden [12].

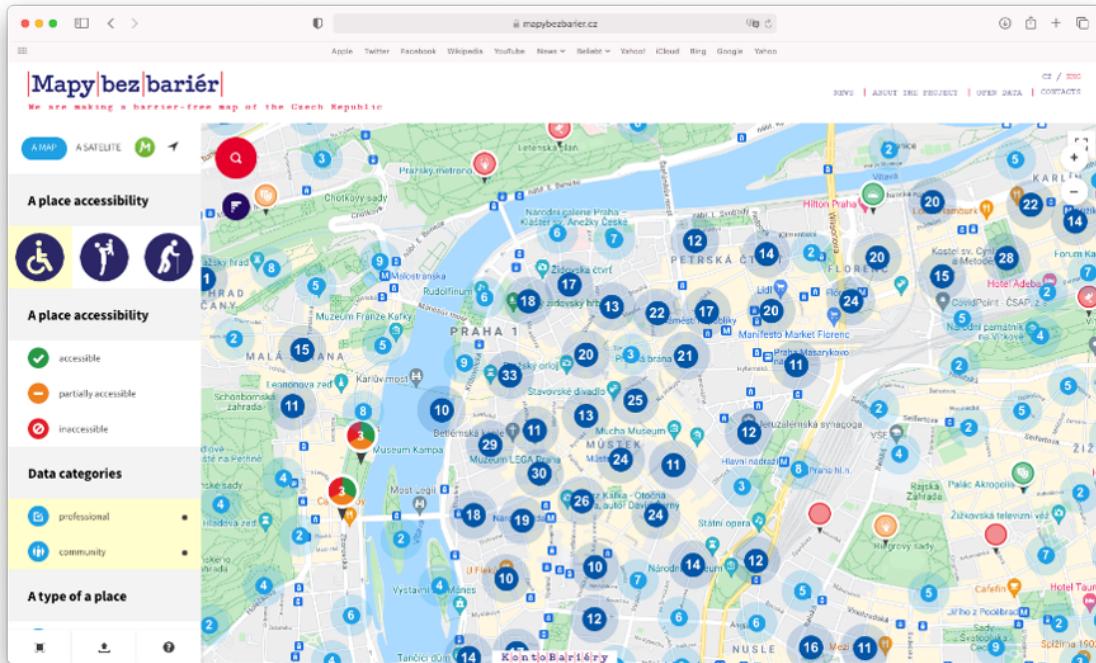


Abbildung 3.2: Maps without Barriers Screenshot [13]

### 3.3.4 Wheelmap

Wheelmap ist ein Online-Kartendienst bei dem Orte nach ihrer „Rollstuhlgerechtigkeit“ auf einem Ampelsystem eingestuft werden (siehe Abbildung 3.3). Wheelmap basiert auf Geodaten von OSM und ist ein Open-Source-Projekt. Das Kartieren erfolgt durch Freiwillige [14]. Der Schwerpunkt von Wheelmap liegt auf der Barrierefreiheit von Orten.

### 3.3.5 Ways4all

Ways4all ist ein Projekt um den öffentlichen Nahverkehr in Wien zugänglicher zu machen. Es adressiert insbesondere blinde Personen. Das Projekt soll die Wegfindung zwischen den einzelnen Verkehrsmitteln erleichtern [15].

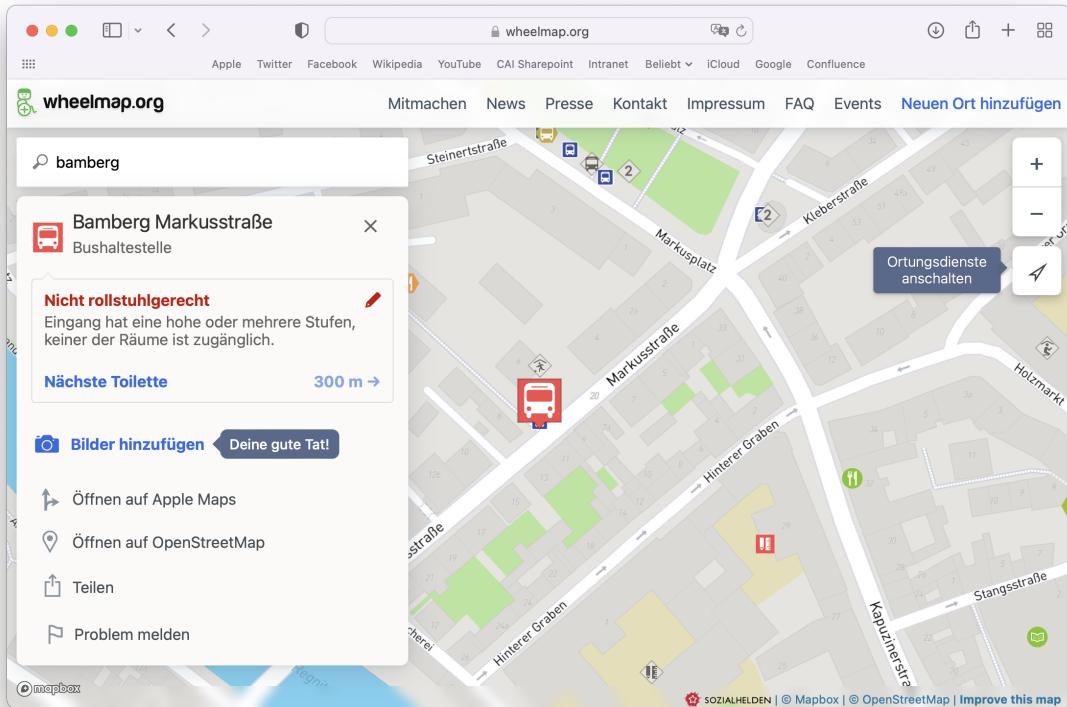


Abbildung 3.3: Wheelmap Screenshot

### 3.4 Einordnung zu MoMM

Die vorgestellten Projekte weisen Aspekte und Ideen auf, die in dem MoMM-Projekt nützlich sind. Vor allem das Projekt WheelScout greift Aspekte auf, die nützlich für das MoMM-Projekt sind, da das Projekt sich direkt mit der Navigation für Rollstuhlfahrer\*innen auseinandersetzt. Das Projekt Accessible Maps ist aufgrund der großen Kontextanalyse für das MoMM-Projekt interessant. In dem Projekt werden die Stakeholder genau betrachtet und Personas entwickelt.

## 4 Untersuchte Algorithmen

In einem ersten Schritt wurden die sechs Routing-Algorithmen BRouter, GraphHopper, ORS, OSRM und Valhalla ausgewählt, die alle zumindest überwiegend dem Open-Source-Ansatz folgen. Diese wurden hinsichtlich ihrer Funktionsweise, der zu Grunde liegenden Daten und der Parametrisierung untersucht.

*Tabelle 4.1: Untersuchte Algorithmen*

	[HTML]C0C0C0BRouter	[HTML]C0C0C0Gosmore	[HTML]C0C0C0GraphHopper	[HTML]C0C0C0ORS	[HTML]C0C0C0OSRM	[HTML]C0C0C0Valhalla
Sprache	Java	C++	Java	Java	C++	C++
Algorithmus	A*	A* CH	A* CH	A*	CH	A*
Basisdaten	OSM	OSM	OSM	OSM	OSM	OSM
Steigungsdaten	SRTM	-	SRTM	SRTM, GMTED	-	Terrain Tiles
Sonstige Daten	-	-	GTFS	GHSI	Einspeisung möglich	Transitland
Lizenz	MIT	BSD	Apache2	LGPL	BSD	MIT
Rollstuhlprofil	ja	nein	ja	ja	nein	nein
Parametrisierung	Profile (Runtime)	Restriktionen	Profile (Runtime)	Profile (Runtime)	Profile (Compile Time)	Kostenmodelle
Sonstiges		keine Weiterentwicklung	teilweise kommerziell			

Gosmore nutzt OSM-Daten, die in einem eigenem Binärformat gespeichert werden, um Routen als 2D- und 3D-Karte anzuzeigen. Die Modifikation der Routen erfolgt durch die Definition von Restriktionen, eine Gewichtung von Parametern ist nicht möglich. Auf Grund dieser Beschränkung sowie der Tatsache, dass der Algorithmus seit 2011 nicht mehr weiterentwickelt wird [16], wird er als wenig geeignet für das MoMM-Projekt bewertet.

Valhalla, ursprünglich von der Firma Mapzen entwickelt, wird seit 2020 als Open-Source-Projekt der Linux Foundation geführt und von einer aktiven Community unterstützt. Ein Demo-Server mit HTTP-Schnittstellen und sowie ein Karten-Frontend wird aktuell vom Verein FOSSGIS gehostet [17]. Vallhalla verwendet neben OSM-Daten auch Steigungsdaten von Terrain Tiles und Informationen zum öffentlichen Nahverkehr von Transitland, beides von Mapzen ins Leben gerufene Services [18]. Durch die Verwendung eines gekachelten, hierarchisch gegliederten Routing-Graphen, der auf jeder Ebene einen unterschiedlichen Detailgrad aufweist, wird nur wenig RAM in Anspruch genommen, so dass ein Offline-Betrieb auch auf Geräten mit geringen Speicher möglich ist. Die Routen können unter Einbezug verschiedener Kostenmodelle berechnet werden, die jeweils über mehrere Parameter beeinflussbar sind [19]. Auch ein multimodales Routing ist möglich, allerdings aktuell nicht auf dem FOSSGIS-Server. Der Hauptnachteil von Valhalla im Rahmen des MoMM-Projekts ist das Fehlen eines geeigneten Rollstuhl-Kostenmodells bei gleichzeitig hohem Aufwand für das Erstellen eigener Kostenmodelle. Hierfür müsste ein separater Server aufgesetzt und der C++-Code modifiziert werden. Zudem ist eine flexible Parametrisierung durch die Nutzer\*innen schwierig zu implementieren. Daher wurde der Algorithmus, der prinzipiell Potenzial hat, nicht in die engere Auswahl aufgenommen.

OSRM wird seit 2018 ebenfalls durch FOSSGIS gehostet, die Weiterentwicklung liegt in den Händen einer aktiven Community. OSRM arbeitet auf OSM-Daten, ermöglicht aber eine Integration weiterer Daten als ASCII-Raster-Daten. Der Algorithmus basiert im Gegensatz zu vielen anderen auf Contraction Hierachies, wodurch eine sehr schnelle Berechnung von Routen möglich ist. Da der Graph allerdings im RAM gehalten wird, ist ein hoher Speicherbedarf gegeben [17]. Individuelle Routen-Einstellungen sind über Profile möglich, die in Lua geschrieben sind und eine hohe Flexibilität bieten [20]. Ein Rollstuhlprofil existiert aktuell nicht, die Erstellung und Einspeisung eigener Profile ist aber möglich. Diese werden allerdings während der Kompilierung des Graphen (Compile Time) angewandt, eine flexible Parametrisierung bei der Suche in Runtime ist nicht möglich. Damit wäre OSRM zwar ein geeigneter Kandidat für die Entwicklung eines festen Rollstuhlprofils, die im

---

MoMM-Projekt angestrebte flexible Parametrisierung nach individuellen Bedürfnissen lässt sich aber nicht oder nur schwer implementieren.

Das Projektteam entschied sich daher, mit den anderen drei Algorithmen weiterzuarbeiten. Alle drei stellen eine HTTP-API zur Verfügung, so dass zumindest in der Test-Phase kein eigenes Server-Backend bereitgestellt werden muss. Darüber hinaus bieten sie auch jeweils ein webbasiertes Karten-Frontend, das ein eigenes MoMM-Frontend nicht ersetzt, aber ggf. für Debugging und Analyse genutzt werden kann. Ebenso verfügt jeder der Kandidaten über ein Rollstuhl-Profil in unterschiedlichen Entwicklungsstadien und berücksichtigt Steigungsdaten. Die Algorithmen sind jeweils in Java geschrieben, was hinsichtlich der Programmiererfahrung des Projektteams vorteilhaft ist.

Der große Vorteil von BRouter ist eine hohe Flexibilität bezüglich der Profile, die nicht nur die Definition sehr komplexer Bedingungen erlauben, sondern auch mit geringem Aufwand erstellt und jederzeit in Runtime modifiziert werden können. Das ermöglicht die von MoMM angestrebte Individualisierung der Anfragen. GraphHopper bietet ebenfalls die Möglichkeit zur Erstellung eigener Profile unter Auslesen beliebiger OSM-Attribute und stellt somit die gewünschte Flexibilität sicher [21]. Allerdings handelt es sich um ein kommerzielles Projekt, für das nicht alle Bestandteile kostenfrei zugänglich sind. Ob der als Open-Source bereitgestellte Teil für MoMM genügt, wird die weitere Analyse ergeben. Für ORS spricht die Existenz eines sehr weit entwickelten, in Runtime parametrierbaren Rollstuhl-Profils, das als Ausgangsbasis dienen kann. Zudem bietet das an der Universität Heidelberg angesiedelte Projekt einen hervorragenden Support, so dass hier bei Problemen Ansprechpartner zur Verfügung stehen.

## 5 Ausgewählte Algorithmen

Im folgenden Abschnitt werden die Eigenschaften und Funktionsweisen der ausgewählten Routing-Algorithmen ORS, BRouter und GraphHopper erklärt.

### 5.1 Openrouteservice

ORS ist ein Open-Source-Projekt der Universität Heidelberg, das seit 2008 vom HeiGIT (Heidelberg Institute for Geoinformation and Technology) entwickelt und betreut wird [22]. Grundlage der ORS-API ist die GraphHopper-API (Version 0.13), welche weiterentwickelt und angepasst wurde. Geschrieben ist ORS in der Programmiersprache Java, die Applikation basiert auf dem A\*-Algorithmus. ORS bietet seine Dienste über ein HTTP-Interface (Tomcat) an, hierbei können die Requests um Profilinformationen und detaillierte Settings in den Formaten GPS, JSON und GeoJSON angereichert werden. Neben der Navigationsfunktion gibt es in ORS auch Möglichkeiten für Isochrones-Suche (welche Orte sind unter Beachtung einer maximalen, eingegebenen Wegzeit erreichbar) und Point-of-Interest-Suche [23]. Zum Test von personalisierten HTTP-Requests existiert ein frei verfügbarer API-Playground. Hier kann Art und Datenformat des Requests angepasst werden, sowie der Request durch Setzen von Headern und Request-Body spezifiziert werden. Beim Testen der Navigationsfunktion werden die Koordinaten der gefundenen Strecke als JSON zurückgegeben, außerdem wird die Strecke graphisch dargestellt [24]. Datengrundlage von ORS sind die Basisdaten von OSM, Steigungen von SRTM und GMTED, Bevölkerungsdaten von Global Human Settlement Layer, Grenzdaten aus Wikipedia und Projektdaten zu grünen/ruhigen Gegenden von der Universität Heidelberg [25]. Es existiert bereits ein Rollstuhl-Profil, welches für Europa genutzt werden kann und weiterhin aktualisiert und gepflegt wird [26]. Während des Graphenaufbaus (der Berechnung der Strecke zwischen angegebenen Start und Ziel) werden je nach gewähltem Profil bestimmte Wege anhand von OSM-Tags ausgeschlossen. Für die jeweiligen Profile ist hinterlegt, welche OSM-Tags akzeptiert (accept), ausgeschlossen (reject) oder nur unter bestimmten Bedingungen zulässig (conditional) sind [27].

### 5.2 BRouter

Da BRouter 2013 als privates Projekt für Fahrradrouting von Arndt Brenschede entstand, existiert kein offizieller Support, aber eine sehr aktive Nutzer-Community. Da bei der Entwicklung die individuell sehr unterschiedlichen Präferenzen von Fahrradfahrern im Vordergrund standen, ist der Algorithmus auf eine einfache und flexible Anpassung durch die Nutzer\*innen optimiert. BRouter kann sowohl über eine Web-API als auch offline betrieben werden. Der Algorithmus basiert auf dem A\*-Algorithmus mit zwei Durchläufen und arbeitet auf Segmenten, d.h. die OSM-Daten werden in Binärdateien (Endung .rd5) umgewandelt und gespeichert, wobei jede Binärdatei jeweils ein Gebiet von 5\*5 Längen- bzw. Breitengraden abbildet. Die Binärdateien können, wöchentlich aktualisiert, von BRouter heruntergeladen oder mit Hilfe eines Skripts aus OSM-Daten selbst erstellt werden. Die Route wird auf dieser Basis unter Anwendung individuell konfigurierbarer, in einer quasiprogrammiersprache abgefasster Profile (Endung .brf) erstellt, wobei die Codierung der verfügbaren OSM-Tags über eine Lookup Table (lookup.dat) erfolgt [28]. Das separat verfügbare Web-Frontend eignet sich sehr gut für das Debugging selbst erstellter Profile, da hier der Profil-Code direkt angezeigt, modifiziert und erneut angewendet werden kann [29]. Falls Anpassung an der Lookup Table,

wie etwa das Hinzufügen weiterer OSM-Tags, erfolgen sollen, kann die Web-API nicht verwendet werden. In diesem Fall muss BRouter als eigener Server mit selbst erstellen Segmenten betrieben werden, da die Versionsnummer der Lookup Table bei der Erstellung der rd5-Dateien codiert wird.

### 5.3 GraphHopper

GraphHopper ist eine Open-Source-Routing-Software von der GraphHopper GmbH sesshaft in München die seit 2014 entwickelt und betreut wird [30]. Geschrieben ist GraphHopper in Java und unterstützt mehrere Routing-Algorithmen wie Dijkstra oder A\*. Standardmäßig wird der Kontraktionshierarchien-Algorithmus verwendet, welcher die Berechnungszeit von Routen zusätzlich reduziert [31]. Als Datengrundlage bedient sich GraphHopper an den Daten von OSM und kann mit beliebigen Datenquellen erweitert werden [31]. Die Routing-Engine kann sowohl online als auch offline verwendet werden und unterstützt sowohl Android als auch iOS. Ebenso kann GraphHopper serverseitig verwendet werden [32]. Im Rahmen dieses Projektes wurde die nicht kommerzielle Web-API verwendet. GraphHopper bietet unterschiedliche Nutzerprofile, die zusätzlich noch konfiguriert werden können. Die Konfigurationen basieren auf den prozentualen Priorisierungen von Straßenattributen wie der Straßenart, dem Straßenbelag und der Qualität der Straßenoberfläche. Zum Berechnen der Strecke wird ein HTTP-Request erstellt, der die Priorisierungen der Straßenattribute beinhaltet sowie die Start und Zielkoordinaten. Als Rückgabewert werden die Koordinaten der Strecke als JSON zurückgegeben.

## 6 Geo-Daten in Bamberg

Neben den geeigneten Algorithmen sind die Geo-Daten, die als Grundlage für die Berechnung zur Verfügung stehen, die wichtigste Säule für ein erfolgreiches Routing. Daher sollen im Folgenden kurz die für das MoMM-Projekt besonders relevanten Geo-Daten, insbesondere die für die Parametrisierung zentralen OSM-Tags, erläutert sowie deren Qualität im ausgewählten Testgebiet beleuchtet werden.

### 6.1 Relevante Geo-Daten

Für die fünf im Prototypen implementierten Parameter (Beschaffenheit des Straßenbelags, maximale Bordsteinhöhe, maximale Steigung, Vermeidung von Treppen und Vorhandensein eines Handlaufs) wurden unterschiedliche OSM-Tags betrachtet.

Die Beschaffenheit des Straßenbelags wurde über den Tag *surface* eingelesen, der primär bei Objekten wie Straßen, Wegen und Pfaden (d.h. bei Objekten mit dem Hauptattribut *highway=\**) vergeben wird. Meist ist er direkt als Tag mit dem Objekt verknüpft, kann aber auch als Ergänzung zu anderen Tags hinzugefügt werden (z.B. kann bei einer Straße ein Gehweg als Attribut gemappt sein, der wiederum als Attribut den Straßenbelag Asphalt erhält).

Für die maximale Bordsteinhöhe ist der OSM-Tag *kerb* relevant, der als Wert unter anderem *raised* (mehr als 3 cm Höhe), *lowered* (etwa 3 cm) oder *flush* (auf Straßenebene) annehmen kann. Alternativ kann auch die genaue Höhe (*kerb:height*) gemappt sein. Der Tag kann als Attribut für Straßen und Wege verwendet werden oder auch als Wert für den Tag *barrier*, d.h. als *barrier=kerb*, gemappt sein.

Die Steigung floss bei den untersuchten Algorithmen zum Teil unter Berücksichtigung von Steigungsdaten wie SRTM, die von OSM unabhängig sind, in die Routenberechnung ein. Daneben war aber für den MoMM-Parameter maximale Steigung auch der OSM-Tag *incline* relevant, der als Attribut für Straßen, Wege und Pfade verwendet wird. Dieser Tag ist bei Straßen nur selten gemappt, häufiger dagegen bei Treppen. Treppen werden in OSM über den Wert des Hauptattributs *highway* (*highway=steps*) erfasst. Wenn ein Objekt in OSM als Treppe gemappt ist, kann zudem über den Tag *handrail* das Vorhandensein eines Handlaufs angegeben werden.

### 6.2 Definition des Testgebiets

Als Testgebiet wurde der Bereich zwischen der Unteren Königstraße 1 und dem Domplatz 1 gewählt (Koordinaten: Startpunkt 49.896906, 10.891636; Zielpunkt 49.891231, 10.883798). Zwischen diesen Punkten sind Routen mit verschiedenen Annotationen und physischen Gegebenheiten vorhanden, unter anderem Brücken, Steigungen, Fußgängerzonen, Treppen sowie unterschiedliche Oberflächen einschließlich Kopfsteinpflaster. Damit können alle im Prototypen implementierten Parameter getestet werden. Auf Wunsch der Stadt Bamberg wurde zudem darauf geachtet, dass das Testgebiet auch die Karolinenstraße berücksichtigt, die als Show Case im Projekt Digitaler Zwilling fungiert. Dieses Gebiet stand bei der Analyse der Algorithmen im Fokus, der Prototyp und die Algorithmen sind jedoch nicht darauf beschränkt. Sie können für jedes andere in OSM gemappte Gebiete ebenso eingesetzt werden.

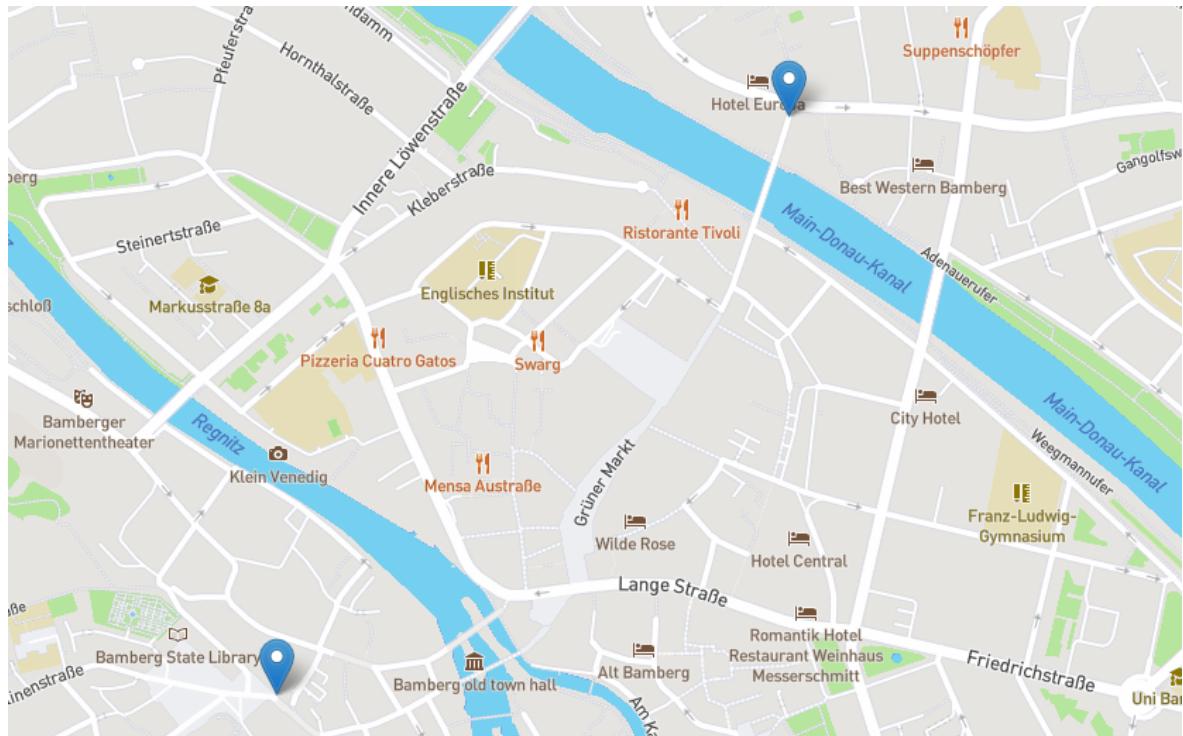


Abbildung 6.1: Testgebiet

### 6.3 Qualität der Geo-Daten im Testgebiet

Einer der Hauptprobleme im MoMM-Projekt ist der Mangel an relevanten Daten. Alle ausgewählten Routing-Algorithmen beziehen zum Berechnen ihrer Routen die Daten von OSM. OSM verspricht eine Vielzahl an nützlichen Daten bezüglich des Projektes. Jedoch ist die Dichte an relevanten Daten in OSM sehr dünn. So ist das Attribut *smoothness* nach dem OSM Wiki-Artikel [33] dafür da, um die Qualität und somit die physische Benutzbarkeit von Straßenbelägen zu klassifizieren.

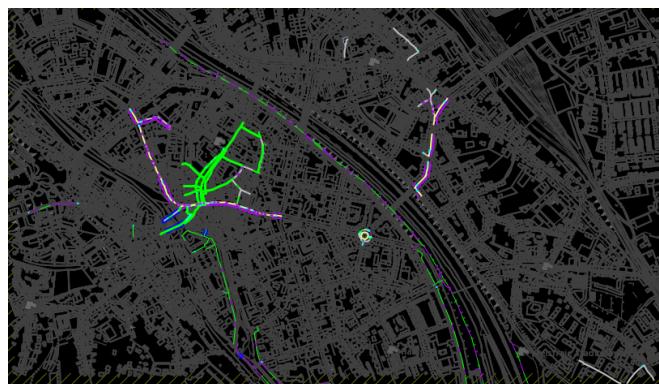


Abbildung 6.2: Annotationen von smoothness im Stadtzentrum

Abbildung 6.2 zeigt, dass nur wenige Straßen im Stadtzentrum mit *smoothness* annotiert sind, was einige Probleme mit sich führt. So kann man zu einem nicht garantieren, dass die berechneten Routen

tatsächliche für die Zielgruppen des Projektes komfortabel sind, da diese tendenzielle über baufällige Straßen navigiert werden können. Zum anderen mussten wir nun zwischen guten und schlechten Straßenoberflächen differenzieren, um implizit die physische Benutzbarkeit zu ermitteln. Hierbei ist jedoch zu sagen, dass der Straßenbelag (*surface*) in Bamberg vollständig annotiert ist. Das Testgebiet besteht aus Asphalt, behauene Steinpflaster mit geglätteter Oberfläche, Kopfsteinpflaster, Beton und Verbundsteinen. Die zugehörigen OSM-Tags lauten *asphalt*, *sett*, *cobblestone* *concrete* und *paving stone*. Zudem sind die Attribute für Treppen und Handläufe gut kartiert in OSM.



Abbildung 6.3: Annotationen der Straßenbeläge im Stadtzentrum

Ein weiteres Attribut mit geringer Datendichte in Bamberg ist die Bordsteinhöhe. So sind die blauen Punkte in Abbildung 6.4 die 66 Annotationen mit dem Attribut *kerb* im Stadtzentrum. Der Mangel an relevanten Daten hat die Folge, dass es zu ineffizienten Routenverschlägen kommen kann. So kann es bspw. dazu kommen, dass größere Umwege berechnet werden, um einen annotierten abgesenkten Bordstein zu nutzen, obwohl es auf der kürzesten Strecke mehrere abgesenkte Bordsteine gibt, die jedoch bisher noch nicht annotiert wurden.

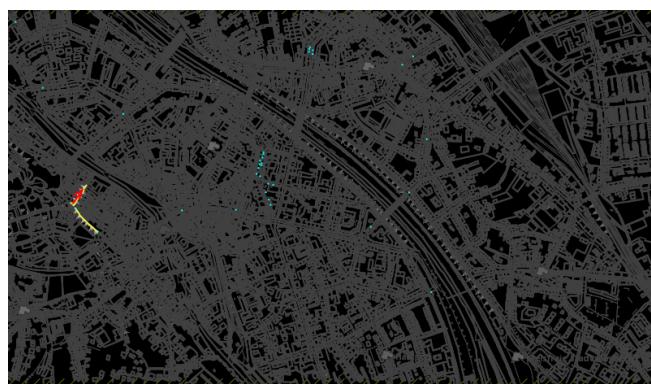


Abbildung 6.4: Annotationen der Bordsteinhöhe im Stadtzentrum

## 7 Prototyp

Ein weiteres Ziel des MoMM-Projektes ist es einen Prototyp zu erstellen, der es einer Nutzer\*in ermöglicht, sich Routen und somit Navigationsanweisungen auf Basis bestimmter Parameter erzeugen und anzeigen zu lassen. Dabei sollen die ausgewählten Algorithmen und Routing-Systeme für die Routenerzeugung herangezogen werden. Der angezeigte Routenvorschlag des jeweiligen Algorithmus soll dann mit anderen Routenvorschlägen verglichen werden können, um die jeweiligen Systeme zu evaluieren.

Das zu entwickelnde System soll stets als ein in Entwicklung befindlicher Prototyp gesehen werden und muss noch keine vollständige Kompatibilität mit (gesetzlichen) Regularien und eine vollumfängliche Problemlösung bieten. Vielmehr soll es als eine Plattform zum Testen, zum Evaluieren und zur weiteren Nutzung, beziehungsweise Erweiterung dienen.

Der folgende Abschnitt der Arbeit begründet und beschreibt die Entwicklung der Software. Um eine allgemeine, somit auch von Fachpersonal ohne besonderen Hintergrund in der Informatik, verständliche Dokumentation und Erklärung zu bieten, werden bestimmte Konzepte, Entwurfsmuster und Implementierungen breiter und vereinfacht erläutert.

### 7.1 Grundsätzliche Anforderungen an den Prototypen

Die Benutzeroberfläche soll einfach, übersichtlich und intuitiv sein, um eine schnelle Nutzung des Prototyps zu ermöglichen. Um in Zukunft auch auf mobilen Endgeräten das System nutzen zu können, soll es zudem eine mobile Applikation geben. Ein wichtiger Punkt, wenn nicht sogar der essentielle, ist die Eingabe der Start- und Zielpunkt für die Routenerzeugung. Diese soll im ersten Prototyp zunächst in der Form von Koordinaten stattfinden. Die Möglichkeit des automatischen Ausfüllens mit der aktuellen Position der Nutzer\*in als Start-Punkt soll ebenfalls gegeben werden. Nachdem die Nutzer\*in die Start- und Zielpunkte eingegeben hat, sollen bestimmte weitere Merkmale parametrisiert werden können. Diese ausgewählten Parameter werden im Laufe der Arbeit noch genauer spezifiziert. Um die verschiedenen ausgewählten Algorithmen gegenüber zu stellen und somit evaluieren zu können, soll es zudem die Möglichkeit geben auszuwählen, welche Routing-Algorithmen und Systeme an-, beziehungsweise abgefragt werden sollen. Die resultierenden Routenanweisungen (sofern vorhanden) sollen dann in einer interaktiven Karte gleichzeitig angezeigt werden. Dies ermöglicht eine direkte Umsetzung der Navigationsvorschläge und vereinfacht die Evaluation der Algorithmen, da direkt verglichen werden können.

### 7.2 Umsetzungsmöglichkeiten des Prototyps

Der folgende Abschnitt soll verschiedene Umsetzungs- und Implementierungsmöglichkeiten des Prototyps kurz genauer beleuchten und das gewählte Verfahren begründen. Da es in jedem Falle Quellcode gibt, wird zudem eine Lösung zur Synchronisation und Dokumentation zwischen den verschiedenen Computern und Implementierungsvorhaben der beteiligten Studierenden benötigt. Für dieses Projekt wird dafür die Open-Source-Lösung git in Verbindungen mit dem universitätseigenem Dienst ([gitlab.rz.uni-bamberg.de](http://gitlab.rz.uni-bamberg.de)) genutzt [34].

### 7.2.1 Umsetzung als (native) Applikation

Eine Möglichkeit den Prototypen zu programmieren ist es diesen als native Anwendung zu erstellen und dann auf die betreffenden Betriebssysteme auszuliefern und dort zu installieren. Konkret bedeutet das somit, dass der Quellcode vor der Nutzung zunächst in maschinenlesbaren Code kompiliert werden muss. Die Kompilierung muss für jedes Ziel-Gerät und Ziel-Betriebssystem einzeln durchgeführt werden. Ein Vorteil von nativen Applikationen ist, dass diese aufgrund der direkten Integration in die Zielpfaltform eine bessere Performance im Kontext der Geschwindigkeit haben können.

Eine solche Programmierung kann jedoch auch zu einem erheblichen Mehraufwand führen, da es mitunter zu Problemen kommen kann, wenn sich bestimmte Programmiermöglichkeiten des Betriebssystems und Endgeräts, wie zum Beispiel die nutzbare Programmiersprache, unterscheiden. Ebenfalls kann es sein, dass es zu Problemen bei der Installation der Applikationen auf entwicklungsforeignen Testgeräten kommt, da beispielsweise eine Applikationsentwicklung für die Betriebssystemplattform iOS und die Auslieferung an betreffende Geräte einen kostenpflichtigen Entwicklungs-Account benötigt. Für jede Änderung des Prototyps muss dann die Applikation erneut kompiliert und wieder installiert werden. Dies kann gerade bei der Arbeit in kleinschrittigen und iterativen Zyklen für erheblichen organisatorischen Aufwand und entwicklungsverlangsamenden Aufwand sorgen.

Alles in allem lässt sich somit festhalten, dass die Programmierung als native „App“ für verschiedene Zielpfaltformen besondere Probleme, beziehungsweise einen erheblichen Mehraufwand bietet und somit nicht als ideale Lösung angesehen werden kann.

### 7.2.2 Programmierung als Webseite

Eine Möglichkeit, die bereits angesprochenen Probleme der nativen Applikation zu umgeht, ist die Umsetzung des Programms als Webseite. In diesem Falle wird die graphische Oberfläche und die Funktionalität des Prototyps durch die Markup-Sprache HTML und die Darstellungsbeschreibung CSS3, beziehungsweise über die Programmiersprache JavaScript umgesetzt. Die Prototypendateien werden dann über einen Webserver verfügbar gemacht und können über einen Webbrower angezeigt, parametrisiert und genutzt werden. Da bei der aktuellen Architektur des Prototyps die Berechnung der Routen über jeweilige separate Systeme der Algorithmen stattfindet, wird eine erhöhte Rechenleistung und Datenspeicherung auf dem Nutzersystem nicht benötigt. Durch das aktive Abrufen der Applikation von einem Server jedes Mal bei der Benutzung kann die aktuelle Version einfach verteilt und bereitgestellt werden. Auf den Zielpfaltformen /-geräten muss lediglich ein aktueller Webbrower installiert sein, um die Webseite aufrufen zu können. Somit kann effizient plattformübergreifend programmiert und die Anwendung genutzt werden.

Die vorangegangenen Punkte zeigen, dass die Programmierung als Webseite, beziehungsweise als Web-Applikation sinnvoll ist und daher auch so umgesetzt wird.

## 7.3 Programmierung des Prototyps

In den folgenden Kapiteln soll die genaue Umsetzung des Prototyps als Webseite beschrieben werden. Dabei wird sowohl auf das Backend eingegangen, welches die Webseite zu Verfügung stellt und ausliefert, als auch des Frontend, welches die eigentliche graphische und funktionale Umsetzung der Webseite ist.

## 8 Backend

Um im Prototypen die verschiedenen Routingsysteme und Algorithmen parametrisieren, anfragen und darstellen zu können wird eine Infrastruktur und Software benötigt, welche die betreffenden Website-Daten zusammenstellt und dem Nutzer zugänglich ausliefert. Diese Entität wird in dieser Arbeit als Backend bezeichnet und soll im Folgenden genauer betrachtet werden.

### 8.1 Programmierung des Backends

Da der Prototyp als Ganzes in Form einer Webseite programmiert wird und dadurch besonders die algorithmische Logik in der Programmiersprache JavaScript umgesetzt wird, bietet es sich an, auch das Backend in JavaScript zu erstellen, um eine zusätzliche Komplexitätsebene einzusparen und den Code einfacher von verschiedenen Entwickelnden erweiterbar zu machen. Dafür eignet sich die Plattform Node.js (Version: 16.14.0) [35] und deren Paketmanager NPM (Version: 8.3.1) [36].

Um die Webseite zu verteilen wird das Protokoll HTTP genutzt. Für eine entsprechende Funktionalität im Node.js-Programm wird dafür die Open-Source-Softwarebibliothek expressJS (Version: 4.16.4) [37] verwendet. Um eventuell die HTML-Dateien für die graphische Oberfläche der Webseite serverseitig mit Informationen dynamisch beim Aufruf zu ergänzen, wird die Open-Source-Softwarebibliothek, beziehungsweise Template-Engine, Handlebars (Version: 4.7.7) [38] integriert. Aufgrund von bestimmten Abhängigkeiten und Funktionalitäten der zuvor genannten externen Module werden weitere Bibliotheken benötigt und automatisch geladen. Diese sind alle frei verfügbar und können anhand einer Datei (package-lock.json) in den betreffenden Versionen nachverfolgt werden.

### 8.2 Funktionalität des Backends

Am effizientesten kann die Funktionsweise und der Aufbau des Backends anhand der integrierten Ordner- und Dateistruktur (Abbildung 8.1) erläutert werden:

Der Prototyp (benannt „MoMM-WayFinder“) wird zunächst über den Befehl „npm start“ in der Kommandozeile des Betriebssystems ausgeführt. Dabei wird von NPM das Skript „www“ im Ordner für die ausführbaren Dateien (Binärdistributionen) der Software „bin“ ausgeführt. In diesem Kontext werden die benötigten Module geladen, übergebene Parameter, wie zum Beispiel die IP-Adresse und die Ports auf denen der Server antworten soll, geprüft und die Hauptdatei „app.js“ gestartet. In den jeweiligen Submodulen können auch zu einem späteren Zeitpunkt noch Bibliotheken bei Bedarf nachgeladen werden.

Der Server ist standardmäßig unter der IP-Adresse „127.0.0.1“ und den Ports „3000“ (für unverschlüsseltes HTTP) und „3443“ für mit selbsterzeugten Zertifikaten (im Ordner „certificates“) verschlüsseltes HTTPS erreichbar. Diese und weitere Einstellung können über sogenannte Environment-Variablen angepasst werden. Das ermöglicht eine einfachere spätere Parametrisierung und Ausführung in einem Container- und Virtualisierungssystem, wie zum Beispiel Docker.

Die Dateien „Dockerfile“, „docker-compose.yml“ und „dockerignore“ sind benötigte Definitionen und Dateien, die zur Ausführung des Prototyps und des Backends in Form eines Softwarecontainers auf der Plattform Docker notwendig sind.

```
▽ MOMM-WAYFINDER
  ▽ bin
    └ www
  ▽ config\tls\certificates
    └ server.cert
    └ server.key
  ▽ public
    ▽ images\landingPage
      gps_icon.png
      OSM_Card-Bamberg-For_Backround.png
    ▽ javascripts\landingPage
      brouter.js
      graphhopper.js
      helpers.js
      main.js
      map.js
      mapDisplay.js
      ors.js
    ▽ stylesheets
      ▽ landingPage
        main.css
        style.css
    ▽ routes
      landingPage.js
    ▽ views
      ▽ landingPage
        landingPage.hbs
        error.hbs
        layout.hbs
      .dockerignore
      .gitignore
      app.js
      docker-compose.yml
      Dockerfile
      package-lock.json
      package.json
      README.md
```

Abbildung 8.1: Ordner- und Programmstruktur des MoMM-WayFinder Prototyps.

Die bereits erwähnten Dateien „package.json“ und „package-lock.json“ geben entwicklungstechnische Informationen über das gesamte Node.js-Programm. Die Datei „README.md“ gibt in konzentrierter Form Anweisungen zum Starten des Prototyps. Die Datei „.gitignore“ dient lediglich der verwendeten Versionierungssoftware „git“, um zu definieren, welche Ordner oder Dateien von der Synchronisation ausgeschlossen werden sollen. Als Beispiel kann der Ordner „node\_modules“, der den Quelltext der extern verwendeten Bibliotheken von Node.js beinhaltet, angeführt werden.

Der Prototyp im Backend ist nach dem Entwurfsschema Model-View-Controller aufgebaut, wobei die Entität des Models nicht direkt vorhanden ist, da keine integrierte Datenhaltung zum Beispiel in Form einer Datenbank stattfindet. Trotzdem ist die Speicherung und Ausführung in theoretisch unterschiedliche Views und Routen (Controller) im Backend vorgesehen. Der Quellcode für die jeweiligen Module befindet sich in den korrespondierenden Ordnern. Konkret bedeutet das, dass der Webserver, der in der „app.js“-Datei gestartet wird zunächst alle möglichen Routen einlädt. Diese Routen stellen die möglichen HTTP-Endpunkte für die betreffende Funktionalität, beziehungsweise View dar. Aktuell gibt es nur eine Unterseite, die auch gleichzeitig als erste Seite des Prototyps dient. Diese ist die Landing-Page. Der betreffende Endpunkt für diese Funktionalität ist „/“ und wird über die „landingPage.js“ im Order „routes“ in die Instanz des Webservers bei Programmstart eingeladen.

Wird somit die URL „<http://127.0.0.1:3000/>“ in einem lokalen Webbrower aufgerufen, wird der Controller der Landing-Page ausgeführt. Dieser liefert die korrespondierende HTML-View-Datei „landingPage.hbs“ im korrespondierenden Unterordner der Views aus. In diesem Zusammenhang könnte auch dynamisch noch (Quell-)Text über die Template-Engine vom Controller eingefügt werden, dies wird jedoch in der aktuellen Version weder benötigt noch benutzt. Es ist aber bereits im Voraus in die Struktur und das Entwurfsmuster eingebaut. Sollten nicht programmierte oder fehlerhafte Endpunkte/Routen angefragt werden, wird automatisch die View „error.hbs“ vom Webserver zurückgegeben. Ebenfalls gibt die globale View „layout.hbs“ einen Rahmen, in dessen die Unterseiten und anderen Views angezeigt werden. Dieser ist aktuell leer, könnte aber mit globalem Quelltext, wie einer Navigationsleiste, gefüllt werden.

Die zugänglichen und jeweiligen Webseitendaten (z.B. HTML-, CSS- und JavaScript-Dateien) der unterschiedlichen Views sind im Ordner „public“ und den betreffenden Unterordnern nach Dateityp und dem Namen der View zu finden. Diese Dateien werden vom Webserver statisch (ohne spezielles Rendering) ausgeliefert. Dateien, die außerhalb eines konkreten View-spezifischen Ordners sind, wie zum Beispiel „style.css“, gelten für alle Views gemeinsam, beziehungsweise für den Rahmen der Views.

Die genaue Struktur der Dateien wird im Kapitel 9 der Arbeit nochmals betrachtet. Es gilt jedoch, dass jede einzelne Komponente, Funktionalität und logischer Abschnitt in eine eigene Datei ausgelagert werden sollte, um die Erweiterbarkeit, Versionierung und das Finden von Fehlern durch den modularen Aufbau dynamisch, einfacher und von verschiedenen Entwickelnden durchführbar zu machen.

## 8.3 Bereitstellung des Backends

In den vorherigen Kapiteln wurde bereits der Aufbau des MoMM-WayFinders und des dazugehörigen Backends beschrieben. In diesem Abschnitt soll nun kurz dargestellt werden, wie das Backend und somit die Software an sich bereitgestellt werden kann und auf welche Besonderheiten, wie eine Absicherung gegen Denial-of-Service-Attacken, geachtet werden muss. Anhand der folgenden Abbildung

(fig. 8.2) kann dieses Konzept am besten erläutert werden: (Hinweis des Autors: Einen ähnlichen Aufbau eines Backends hat der Autor auch in anderen Arbeiten gewählt und beschrieben. Daher können Ähnlichkeiten in den Diagrammen und Strukturen aufgrund der gleichen Autorenschaft zu finden sein.)

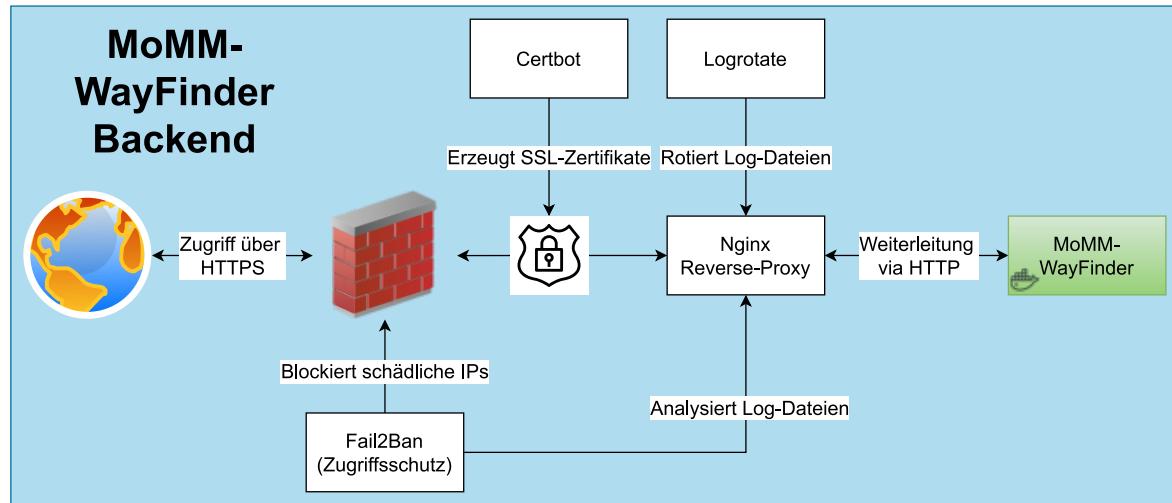


Abbildung 8.2: Exemplarisches Bereitstellungsdiagramm des MoMM-WayFinder-Backends.  
(Erstellt mittels "<https://app.diagrams.net>"[39])

Bevor Nutzer\*innen auf die MoMM-WayFinder-Software und somit die Webseite zugreifen können, sollte der Datenverkehr durch eine Reihe von anderen Softwareanwendungen geroutet werden. Zunächst kann der Datenverkehr nur über den HTTPS gesichert an die Software gesendet werden. Über den HTTP-Port übertragener unverschlüsselter Netzwerkverkehr sollte automatisch in HTTPS überführt und alle anderen Ports durch eine Firewall blockiert werden. Das für HTTPS benötigte digitale Zertifikat kann kostenlose von einer Zertifizierungsstelle (z.B. Let's Encrypt) [40] für die jeweilige Domain der Webseite mittels des Softwaretools Certbot erzeugt werden. Über ein automatisiertes Skript muss dieses Zertifikat stets bei Bedarf vom System erneuert werden. Hierzu eignet sich beispielsweise die Software Cron.

Bevor der einkommende HTTPS-Netzwerkverkehr dann die eigentliche Software erreicht, durchläuft dieser noch einen Reverse-Proxy. Dafür eignet sich beispielsweise das Tool nginx [41]. Der Reverse-Proxy empfängt die einkommenden Anfragen und determiniert den HTTPS-Traffic. Die Anfragen werden von diesem Tool protokolliert und anschließend mittels unverschlüsselter Datenpakete an die WayFinder-Software weitergeleitet. Dieser Aufbau bietet im wesentlichen drei Vorteile:

Zum ersten ermöglicht es eine weitere Sicherheitsschicht, die für Verschlüsselung in außerhalb liegende Netzwerke sorgt. Die digitalen Zertifikate liegen somit nicht direkt in der MoMM-WayFinder-Software und sollten nicht bei einer Fehlfunktion ausgelesen werden können.

Zum anderen ermöglicht es eine Skalierung, da einkommende Netzwerkpakete an verschiedene Instanzen der MoMM-WayFinder-Software weitergeleitet werden können. Diese eventuell zusätzlichen Instanzen könnten dann auf anderen internen Maschinen betrieben werden und als eigenes Modul nur ihre Aufgaben (das Ausliefern der Webseite und Untersuchen von eingegebenen Texten) ausführen. Somit wäre eine beliebige Skalierung auf viele Instanzen mühelos möglich.

Ein weiterer Punkt ist die Überwachung und das Monitoring. Da alle Anfragen nur über den Reverse-Proxy gesendet werden, kann durch diesen genau protokolliert werden, wie oft Anfragen von einer bestimmten IP-Adresse gesendet werden. Eventuelle Denial-of-Service (DoS) Angriffe durch Anfragenüberflutung könnten somit in den Log-Dateien protokolliert werden. Eine solche Protokollierung könnten über das nginx-Modul stattfinden, durch das Maximalwerte für Anfragen pro Zeit definiert werden können.

Genau an dieser Stelle setzt auch das weitere Programm Fail2ban [42] an. Dieses überprüft kontinuierlich die Log-Datei des Reverse-Proxys. Über eine für dieses Tool definiert Überprüfungsregel wird geprüft, ob sich eine bestimmte Meldung in den Logs befindet, die angibt, ob eine bestimmte IP das Zugriffslimit pro Zeitintervall überschreitet. Sollte dies der Fall sein, wird diese IP über die Firewall temporär blockiert. Damit die Log-Dateien nicht beliebig groß werden und dann die Speicherkapazität des Servers minimieren, sollten die Log-Dateien beispielsweise mit dem Tool Logrotate [43] regelmäßig rotiert, also komprimiert und überschrieben werden. Nach der Prüfung durch bereits genannte Module, kann der ordentliche, unverschlüsselte und funktionale Datenverkehr an die MoMM-WayFinder-Software weitergeleitet werden. Die MoMM-WayFinder-Software an sich könnte, wie bereits beschrieben, in einer containerisierten Umgebung, wie zum Beispiel Docker [44], ausgeführt werden. Die benötigten Dateien, Skripte und Konfigurationen sind vorhanden und ermöglichen somit eine einfache Bereitstellung auf einer passenden Softwareplattform.

## 9 Frontend

Zur Darstellung der unterschiedlichen Routing-Algorithmen und Einstellung individueller Profile wird eine Benutzeroberfläche benötigt. In diesem Abschnitt wird die Gestaltung und der technische Hintergrund des Frontends vorgestellt.

### 9.1 Gestaltung der Benutzeroberfläche

Zur Darstellung und zum Vergleich der unterschiedlichen Routing-Algorithmen steht die zentrale Abbildung der Karte im Vordergrund. Im oberen Bereich wird eine Landkarte mit den drei verschiedenen Routing-Ergebnissen dargestellt (siehe Abbildung 10.4). Die berechnete Route des Algorithmus BRouter wird in der Farbe blau, die des Algorithmus ORS in rot und GraphHopper in grün dargestellt.

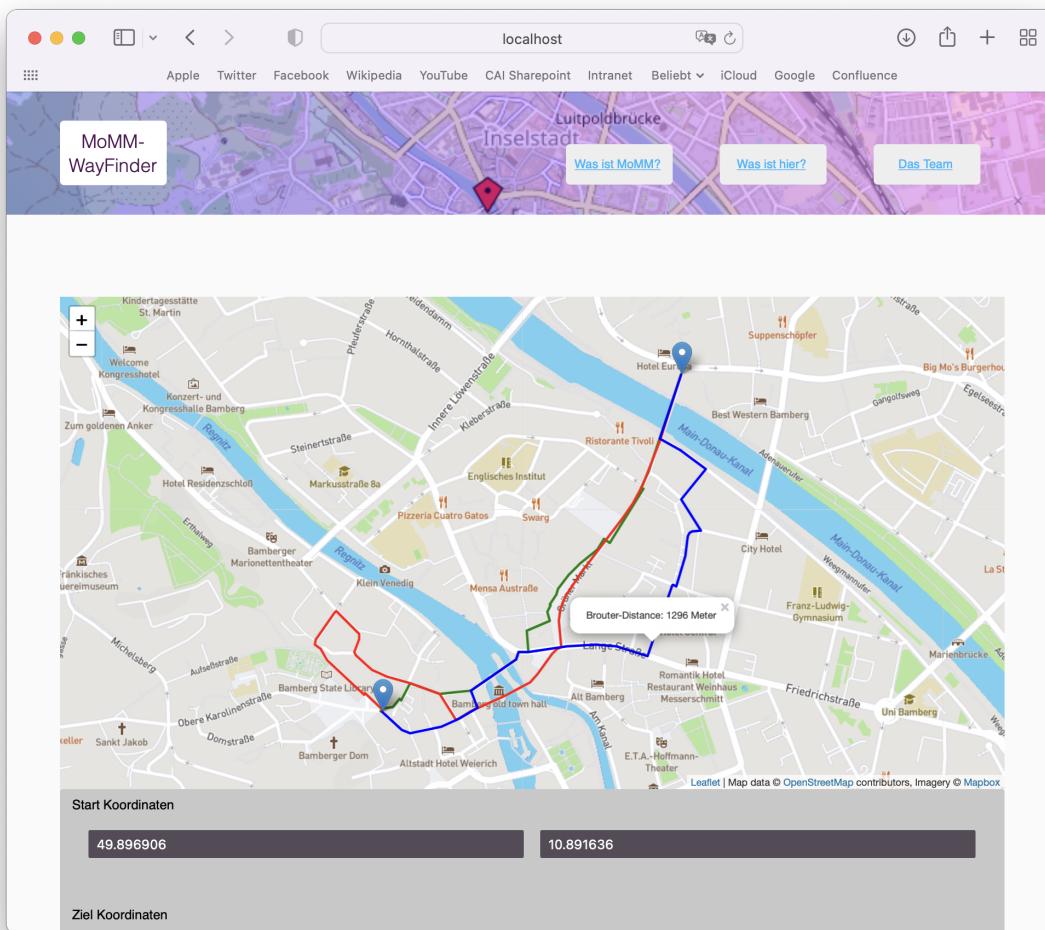


Abbildung 9.1: Benutzeroberfläche Karte

Die Länge und der Name der drei unterschiedlichen Routing-Algorithmen können durch Klicken auf die jeweilige Route angezeigt werden (siehe Abbildung 9.2 und 9.3). Start- und Zielkoordinaten können durch Ziehen des jeweiligen Icons verändert werden. Zusätzlich können diese auch in dem unter der Karte in Textfeldern eingegeben werden. Weitere Einstellungen verschiedener Parameter können unter der Karte ebenfalls eingegeben werden.

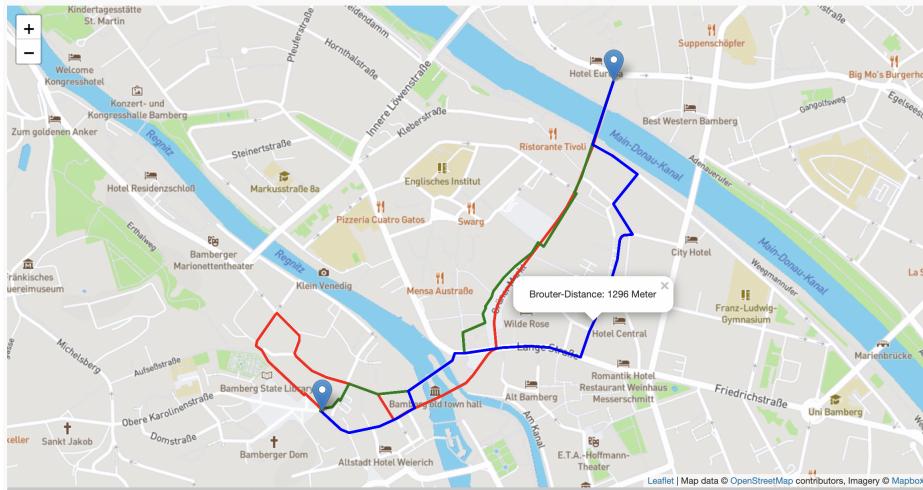


Abbildung 9.2: Anzeige von BRouter auf der Karte

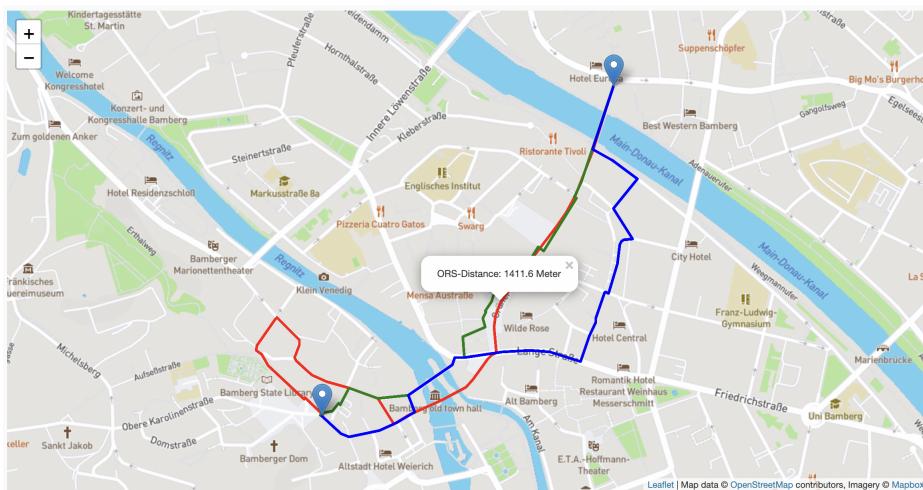


Abbildung 9.3: Anzeige von ORS auf der Karte

## 9.2 Technische Umsetzung der Benutzeroberfläche

Die Benutzeroberfläche lässt sich grob in drei Teile aufteilen: die Darstellung der Karte, die Anfragen und Antworten der Routing-Algorithmen und die personalisierten Benutzerprofile. In diesem Abschnitt werden die drei Teile vorgestellt.

### 9.2.1 Routen

Die Anfragen der Routen erfolgen unabhängig voneinander. In den jeweiligen Dateien *brouter.js*, *graphhopper.js* und *ors.js* erfolgen die API-Anfragen für eine Abfrage der Routen. Die in der Benutzeroberfläche eingestellten Parameter werden an die Routing-Algorithmen übermittelt. Der in Abbildung 9.4 dargestellte Code-Ausschnitt für eine Anfrage für ORS zeigt dies: Es werden die eingestellten Parameter im *Body* (Zeile 21-26) einer Post-Anfrage über die Schnittstelle an die URL <https://api.openrouteservice.org/v2/directions/wheelchair/geojson> gesendet. Die Abbildung 9.5 zeigt die Antwort auf eine solche Anfrage. Die Nutzung der in der Antwort enthaltenen Daten wird im folgenden Abschnitt beschrieben.

ORS unterstützt alle in dem Profil angegeben Parameter, GraphHopper hingegen nur, ob Stufen in der Route enthalten sein sollen. BRouter unterstützt ebenfalls alle Parameter.

```

12     let apiKey = "5b3ce359785110001cf6248aa5c9455dcf844ccb3c778ba19e86b88";
13     let contentType = "application/geo+json; charset=UTF-8";
14     let profile = "wheelchair";
15
16     let url = "https://api.openrouteservice.org/v2/directions/" + profile + "/geojson";
17
18     let no_steps = "";
19     if(no_steps_on_route){
20         no_steps = "\\"avoid_features\\":[\"steps\"],";
21     }
22
23     let body = "{\"coordinates\":[[" + start_lat + "," + start_long + "],[[" + dest_lat + "," + dest_long + "]]," +
24     "\"options\":{" + no_steps + "\"profile_params\":{\"restrictions\":\"" +
25     "\"maximum_incline\":\"" + maximum_incline + "\",\"surface_type\":\"" + surface_type + "\",\"" +
26     "\"maximum_sloped_kerb\":\"" + maximum_kerb_height + "}}}";

```

Abbildung 9.4: ORS-Anfrage Code-Ausschnitt

### 9.2.2 Karte

Der zentrale Bestandteil der Benutzeroberfläche – die Karte – wird mit dem Javascript-Framework Leaflet [45] realisiert. Zur grafischen Darstellung der Karte, den sogenannten Tiles, wird auf Bildmaterial von Mapbox [46] zugegriffen. Der Aufruf der Karte erfolgt in der Datei *map.js*.

Vereinfacht gesagt werden die Koordinaten der Eckpunkte der vorab angefragten Routing-Algorithmen auf die Karte gezeichnet, miteinander verbunden und somit als Route dargestellt. Die Länge der Wegstrecke wird ebenfalls aus der Antwort der Routing-Algorithmen übernommen und an die jeweiligen Algorithmen annotiert. Zur unterschiedlichen farblichen Darstellung wird in einem Switch-Case der Name des Routing-Algorithmus abgefragt. Nach dem Ziehen (Dragging) der Start- und Ziel-Icons wird jeweils eine neue Anfrage an die Routing-Algorithmen gesendet und die Karte neu dargestellt. Anhand des in Abbildung 9.6 dargestellten Code-Ausschnitts wird der Ablauf deutlich: In dem Switch-Case wird der Fall 'ors' ausgewählt. Es wird geprüft, ob der Nutzer in dem Formular ausgewählt hat, dass er diesen Algorithmus anzeigen lassen möchte. Ist dies der Fall, wird eine Linie ("polyline") erstellt. Dieser Linie wird mit der Länge annotiert ("bindPopup"). Die Linie wird in rot dargestellt. Die Funktion *map.fitBounds(polylineOrs.getBounds())*; zentriert die Karte auf die dargestellte Route.

Abbildung 9.5: Beispiel Antwort API-Anfrage ORS

```
case 'ors':
  console.log('ORS');
  if (document.getElementById(elementId: "checkboxORS").checked == true) {
    var polylineOrs = L.polyline(points).bindPopup('ORS-Distance: ' + length + ' Meter').addTo(map).openPopup();
    polylineOrs.setStyle({
      color: 'red'
    });
    map.fitBounds(polylineOrs.getBounds());
  }
}
```

Abbildung 9.6: Code-Ausschnitt Switch-Case ORS in maps.js

### 9.2.3 Profile und Parametrisierung

Unterschiedliche Parameter können in der Benutzeroberfläche eingestellt werden. Die eingegebenen Profile können abgespeichert werden. Die Speicherung der Profile findet lokal auf dem Endgerät der Nutzer:innen statt. In dem Formular (siehe Abbildung 9.7) können die Start- und Zielkoordinaten (1) eingegeben werden. Im Feld 2 wird die maximale Bordsteinhöhe, also die Höhe des Bordsteins zur Straße in der Einheit Meter eingegeben. Im Feld 3 wird die Steigung der Strecke in Prozent eingegeben. Hierfür wird der OSM-Tag *kerb* angesprochen. Im Feld 4 können Nutzer:innen auswählen, ob Treppen vermieden werden sollen. Im Feld 5 wählt der Nutzer aus, ob ein Handlauf an den Treppen benötigt wird. Hierfür wird der OSM-Tag *handrail* angesprochen. Im Feld 6 wird der benötigte Straßenbelag ausgewählt. In den Feldern 7 können die dargestellten Algorithmen ausgewählt werden. Im darunter folgenden Abschnitt können die Ziele benannt werden und die ausgewählten Parameter gespeichert werden. Die Abfrage der Parameter erfolgt in der Datei *landingPage.hbs* (ab Zeile 59) und

die Verarbeitung in der Datei *main.js*. Die Speicherung und der Abruf der Profile erfolgt ebenfalls in der Datei *main.js*.

The screenshot shows a search interface with the following annotated fields:

- Start Koordinaten:** Two input fields for latitude and longitude. The left field contains "49.896906" and the right field contains "10.891636". A large number **1** is placed between them.
- Ziel Koordinaten:** Two input fields for latitude and longitude. The left field contains "49.891231" and the right field contains "10.883798".
- Maximale Bordsteinkantenhöhe in Meter:** An input field containing "0,2". A large number **2** is placed next to it.
- Maximale Steigung in Prozent:** An input field containing "6". A large number **3** is placed next to it.
- Treppen meiden?**: A checkbox input field. A small number **4** is placed next to it.
- Handlauf benötigt?**: A checkbox input field. A small number **5** is placed next to it.
- Straßenbelag von mindestens dieser Art:** A dropdown menu with "Gras" selected. A large number **6** is placed next to it.
- Algorithmen:** Three checked checkboxes: "Graphhopper", "BRouter", and "ORS". A large number **7** is placed next to the "BRouter" checkbox.
- Ziel-Name:** An input field for the goal name, containing "Name des Ziels (optional) zur Zwischenspeicherung." Below it is a purple button labeled **Berechnen!**
- Buttons at the bottom:** "Profil Speichern" (highlighted in blue), "Profil laden", and "Profil löschen".
- File Input Fields:** "Profilename" and "Speichern" (highlighted in purple).

Abbildung 9.7: Formular mit Anmerkungen

## 10 Evaluation

Der Prototyp erlaubt einen ausführliche Test der drei ausgewählten Algorithmen hinsichtlich eines MoMM-Routings in Bamberg. Die Ergebnisse dieser Evaluierung werden im Folgenden zusammengefasst.

### 10.1 ORS

Um die Koordinaten einer Route von der ORS-API anzufragen, benötigt der REST-Request die Start- und Ziel-Koordinaten der gewählten Route und das Routing-Profil, das genutzt werden soll. Je nachdem welche Parameter von dem jeweiligen Routing-Profil unterstützt werden, kann der Request zusätzlich Werte für diese Parameter enthalten. Für ORS existiert bereits ein Rollstuhlprofil, für welches die MoMM-Parameter maximale Bordsteinhöhe, maximale Steigung, Treppen vermeiden und mindestens benötigter Straßenbelag übergeben werden können. Zusätzliche Einstellungsparameter, die dieses Profil bietet, sind der mindestens benötigte Tracktype (Nutzbarkeit insbesondere in Bezug auf Oberflächenfestigkeit), die mindestens benötigte Smoothness (Nutzbarkeit insbesondere in Bezug auf Ebenmäßigkeit des Bodens) und Straßenbreite [26]. Der MoMM-Parameter Handlauf kann durch einen Rest-Request an die ORS-API nicht übergeben werden, um diesen dennoch zu unterstützen wäre es notwendig, das ORS-Backend selbst zu installieren und den Quellcode so anzupassen, dass dieser unterstützt werden kann. Eine Installationsanleitung befindet sich in der Dokumentation von ORS, hierfür wird empfohlen Docker zu nutzen. Der Service kann dann lokal im Browser abgerufen werden [47]. Das Rollstuhlprofil müsste dann um den Parameter ergänzt werden. Da es sich bei dem bereits vorhandenen Parameter Treppen meiden ebenfalls um einen Boolean handelt, kann die Einbindung von diesem als Vorlage genutzt werden und analog dazu vorgegangen werden. Um einen neuen Parameter in ORS einzubinden, muss jedoch der Quellcode an vielen Stellen und in großem Umfang modifiziert werden, was sich als sehr aufwendig gestalten würde. Auch nachteilhaft daran ORS selbst zu hosten ist, dass mögliche Features neuerer Versionen des Routing-Algorithmus, die für das MoMM-Projekt ebenfalls nützlich sein könnten, dann nicht zur Verfügung stünden. Da ORS aktiv weiterentwickelt wird und auf seiner GitHub-Seite deutlich macht, offen für Vorschläge bezüglich neuer Funktionalitäten zu sein, wäre eine weitere Option, sich an die Entwickler zu wenden und vorzuschlagen den Parameter Handlauf zum Rollstuhlprofil hinzuzufügen.

Um das Routing von ORS für das gewählte Testgebiet durch den MoMM-Prototypen zu evaluieren, wurden die Einstellungsparameter, die vom Rollstuhlprofil unterstützt werden, schrittweise erhöht und festgehalten inwiefern dies Auswirkungen auf die angezeigte Route hat.

Für die maximale Steigung (in Prozent) wird bei ORS mit den Werte 3, 6, 10, 15 und *beliebig* gearbeitet. Hierbei ist zu beachten, dass die Eingabe abweichender Werte zwar möglich ist und keinen Fehler erzeugt, diese jedoch auf den nächsthöheren Wert geglättet werden. Wird beispielsweise eine maximale Steigung von 4 über den Prototypen an die ORS-API übermittelt, arbeitet ORS mit dem Wert 6. Bei der Variation der Steigung verändert sich die Route in dem Moment, in dem ein Wert für die maximale Steigung von 6 oder weniger eingegeben wird. Ein Blick in der OSM-Daten zeigt, dass der obere Teil der Karolinenstraße mit einer Steigung von 8 Prozent annotiert ist, was die einzige für das Testgebiet gemappte Steigung ist. Der Teil der Karolinenstraße mit 8 prozentiger Steigung wird dann umgangen und eine alternative Route für das letzte Stück bis zum Domplatz ausgegeben. Wird der Punkt der Zielkoordinate ein wenig verschoben, wird eine zusätzliche zweite alternative Route

angezeigt. Durch die Vermeidung der Karolinenstraße, welches der direktere Weg wäre, führen beide alternative Routen über einen Umweg, der die Gesamtlänge der Strecke deutlich erhöht.

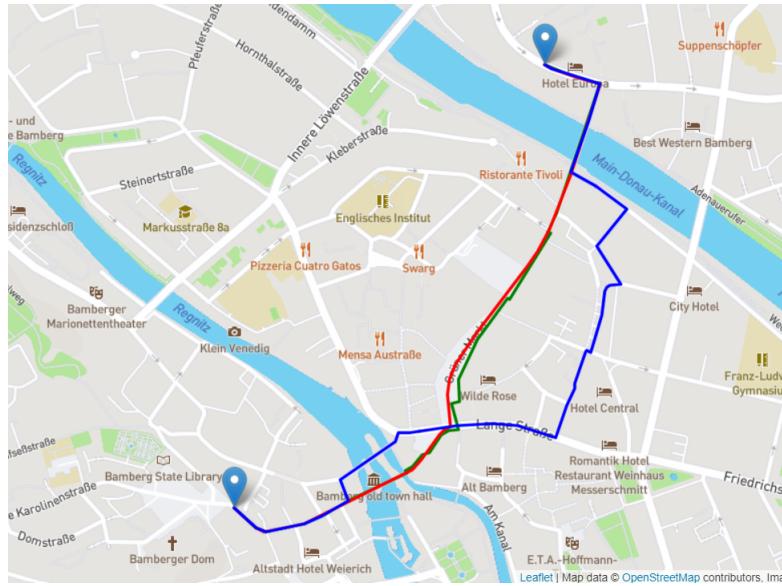


Abbildung 10.1: Ausgangsrouting ohne Einschränkungen (ORS in rot)

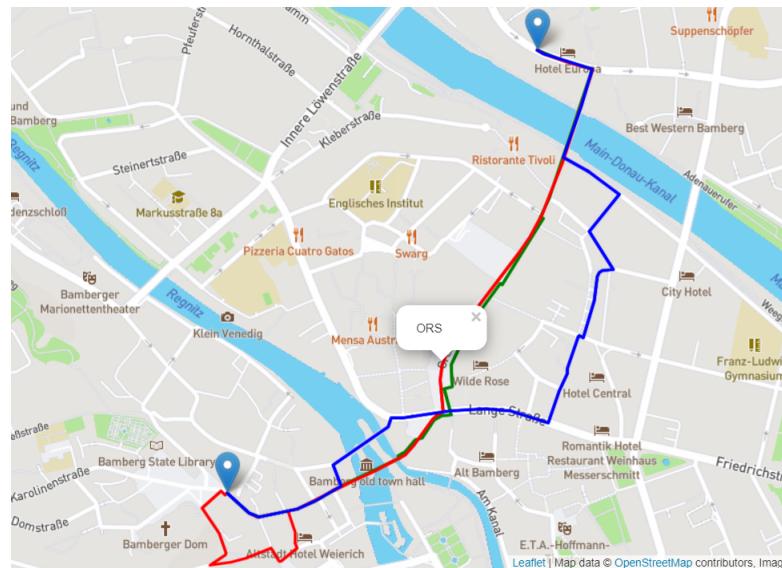


Abbildung 10.2: Routing mit 6 Prozent Steigung (ORS in rot)

Wird der Parameter mindestens benötigter Straßenbelag variiert, so wird ab dem Mindestbelag ebenes Kopfsteinpflaster der Fehler zurückgegeben, dass keine Route zwischen Start- und Zielpunkt gefunden werden konnte. Das OSM-Mapping ist an dieser Stelle inkonsistent, denn theoretisch ist das Ziel auch ohne Kopfsteinpflaster erreichbar. Wird der Zielpunkt statt auf den Domplatz auf eine Straße gesetzt, welche nicht als Kopfsteinpflaster annotiert ist, findet ORS eine Route, welche gegebenenfalls zwar deutlich länger ist, aber nur Straßen mit zulässigen Straßenbelägen mit in die Berechnung einbezieht.

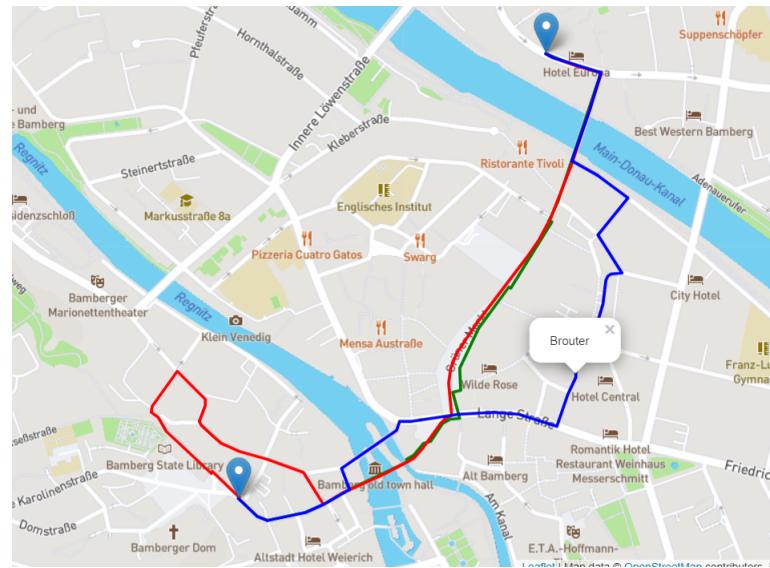


Abbildung 10.3: Alternativrouting mit 6 Prozent Steigung (ORS in rot)

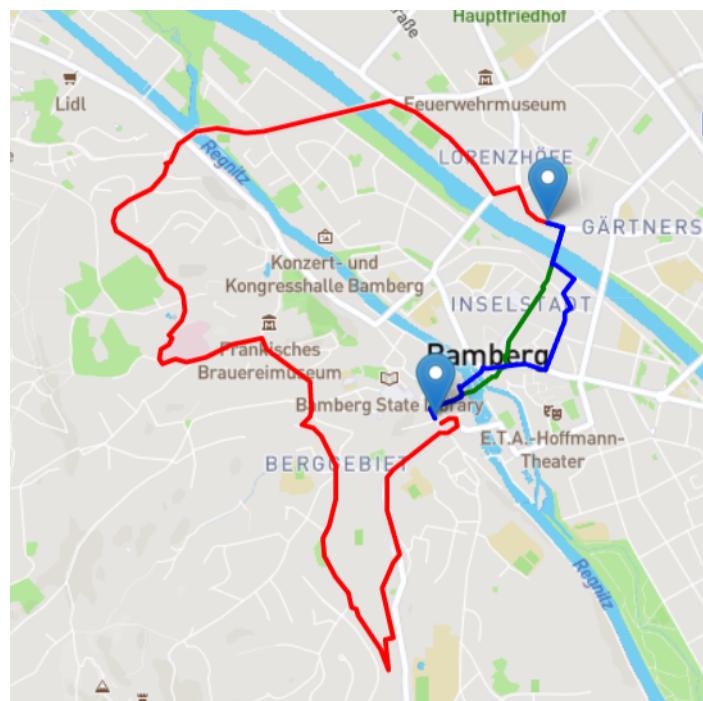


Abbildung 10.4: Routing für den Mindestbelag Kopfsteinplaster (ORS in rot)

Beim Testen verschiedener Eingabewerte für den Parameter maximale Bordsteinhöhe, verändert sich die angezeigte Route nicht. Grund hierfür ist, dass im Testgebiet keine Tags existieren, welche die Bordsteinhöhe mappen. Analog zu der maximalen Steigung, gibt es auch für die maximale Bordsteinhöhe verschiedene Werte, mit denen ORS arbeiten kann, 0.03, 0.06, 0.1 und *beliebig*.

Da auf der vom Algorithmus gewählten Route keine Treppe liegt, ändert sich auch die gewählte Route nicht, wenn der Parameter Treppen meiden auf *true* gesetzt wird. Werden Start- und Zielpunkt nun so gewählt, dass sie am Kopf und Fuß einer Treppe liegen, lässt sich beobachten, dass selbst hier die Treppe umgangen und ein Umweg gewählt wird. Beim Ändern des Profils auf Fußgänger oder Radfahrer kann hingegen beobachtet werden, dass die Route über die Treppe führt, wenn diese nicht vermieden werden soll und anderenfalls auch den alternativen Weg ohne Treppen vorschlägt. Diese Fallunterscheidung, für einen Rollstuhlfahrer in jedem Fall Treppen zu umgehen, außer es ist nicht anders möglich eine Strecke zu überwinden, kann sich als sinnvoll erweisen, weil angenommen werden kann, dass Treppen für die meisten Rollstuhlfahrer mindestens ein erschwerendes Hindernis darstellen, muss jedoch bei der Evaluation der Algorithmen berücksichtigt werden.

## 10.2 BRouter

BRouter bietet durch seine konfigurierbaren Profile große Flexibilität, die allerdings durch die Anzahl und das Format der in der Lookup Table verfügbaren OSM-Tags eingeschränkt wird. Dort verzeichnete Tags können beliebig kombiniert, jedoch nicht durch andere Tags ergänzt werden [48]. Dafür wäre eine Modifikation der Lookup Table nötig, die den Betrieb einer eigenen BRouter-Instanz erfordert (vgl. Kapitel 5.2). Dies ist dank verfügbarer Skripte mit mittlerem Aufwand umsetzbar, zieht aber langfristig einen gewissen Wartungsaufwand nach sich, da die regelmäßige Aktualisierung des Kartenmaterials sichergestellt werden sollte. Alternativ könnte auch die Aufnahme neuer Tags in die Lookup Table durch die BRouter-Community angefragt werden.

Für das MoMM-Projekt wurde ein zuletzt 2015 bearbeitetes Rollstuhl-Profil als Basis genommen [49] und modifiziert. Die Wahl der minimalen Beschaffenheit des Straßenbelags sowie die Vermeidung von Treppen konnten als Parameter mit den vorhandenen Tags implementiert werden. Die maximale Bordsteinhöhe sowie das Vorhandensein eines Handlaufs bei Treppen erfordert dagegen den OSM-Tag *handrail* sowie für den Tag *kerb* verschiedene Werte für die Höhe (bislang ist nur *kerb=barrier* codiert), die in der Lookup Table fehlen. Diese Parameter konnten daher nicht berücksichtigt werden [48].

Eine Herausforderung stellt der MoMM-Parameter maximale Steigung bei der Routenberechnung dar. BRouter berücksichtigt automatisch die Steigung, die mit Hilfe von SRTM-Daten berechnet wird. Steigungen können vermieden werden, indem sie bei der Routenberechnung mit erhöhten Kosten bestraft werden. Die Angabe eines exakten Maximal- oder Minimalwerts für das Gefälle ist aber auf diesem Wege nicht möglich, da der Kostenfaktor dynamisch anhand der Länge der Strecke und der Steigung mit verschiedenen Buffern berechnet wird [50]. Um dennoch einen präzisen Grenzwert für das Gefälle anzugeben, kann zusätzlich mit dem OSM-Tag *incline* gearbeitet werden, was im Prototypen auch implementiert wurde. Durch die Restriktionen der Lookup Table können nur Stufen (maximal 3, 5, 10 oder 15 Prozent Steigung) gewählt werden, die aber für den MoMM-Anwendungsfall ausreichend sind. Problematischer ist, dass der Tag in Bamberg nur selten gemappt ist.

Bei der Analyse der Route im Testgebiet fällt auf, dass BRouter fälschlich Steigungen in der flachen Fußgängerzone in Bamberg erkennt (Abbildung 10.5, Analyse des Höhenprofils mit BRouter Web Client). Das lässt sich auf ein Grundproblem der SRTM-Daten in urbaner Umgebung zurückführen. Das Geländeprofil der Erde wurde mit Hilfe von Radar aus dem Weltraum vermessen, wobei zum Teil statt der Erdoberfläche die Bebauung kartiert wurde [51]. Da BRouter jedoch mit Buffern arbeitet, die Steigungen erst ab einer gewissen Länge und Grad berücksichtigen, beeinflussen diese Fehler

die Routenplanung im Testgebiet nicht wesentlich. Auch bei Deaktivierung der Steigungsberechnung (im Profil durch Setzen eines Bits möglich) wird dieselbe Route durch die Fußgängerzone genommen. Um im Testgebiet überhaupt Auswirkungen der Steigungs-Kostenfunktion auf die Route zu erreichen, müssten die uphill/downhill-Kosten drastisch erhöht (von 7 auf 100) und die Elevation-Buffer auf null gesetzt werden. In diesem Fall wird allerdings vor allem die eigentlich flache Fußgängerzone vermieden. Beim Setzen dieser Parameter (Kosten und Buffer) ist also ein Tradeoff zwischen Effektivität und Fehleranfälligkeit zu beachten. Insgesamt ist diese Funktionalität daher nur bedingt für Routing im urbanen Setting geeignet.

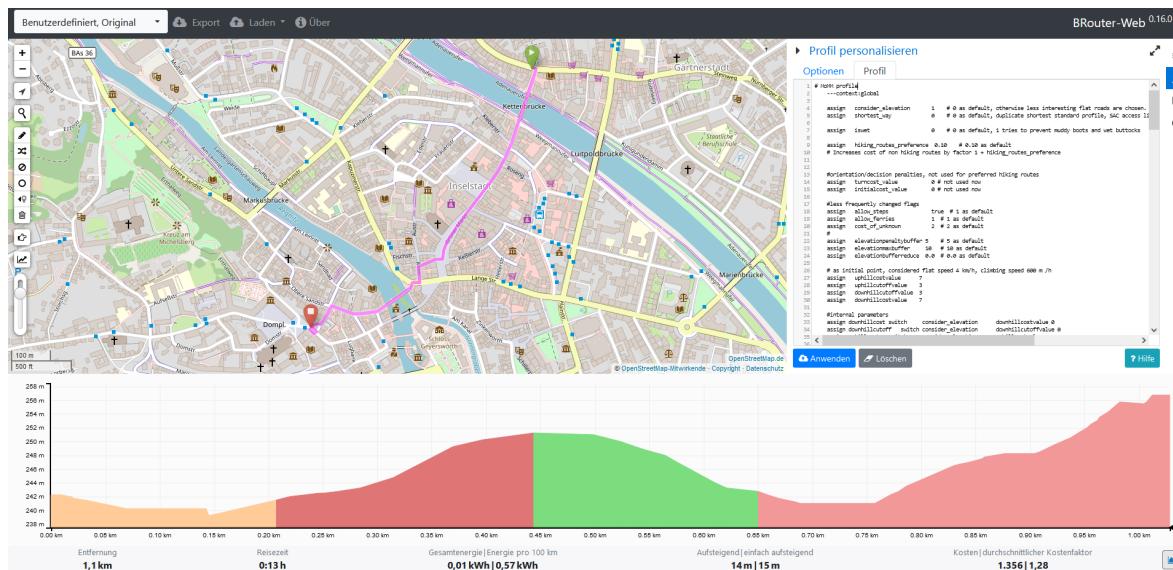


Abbildung 10.5: BRouter-Höhenprofil für Teststrecke

Die Vermeidung von Treppen sowie von Steigungen, die über *incline* gemappt sind, funktioniert erwartungsgemäß. Bei einer maximalen Steigung von 10 oder 15 wird nach der unteren Brücke der Weg über die Karolinenstraße genommen, die abschnittsweise mit *incline*=8% gemappt ist (Abbildung 10.6). Bei Maximalwerten darunter wird alternativ der Weg über die Dominikanerstraße und die Treppe am Katzenberg gewählt (Abbildung 10.8). Wenn zusätzlich Treppen vermieden werden sollen, wird eine Alternativroute über die Markusbrücke vorgeschlagen (Abbildung 10.7). Für eine Vermeidung besonders steiler Wegabschnitte ist dieses Verfahren im Vergleich zur Berücksichtigung der SRTM-Daten besser geeignet, setzt aber ein ausreichendes Mapping voraus.

Bezüglich des Parameters für die Beschaffenheit des Bodenbelags wird bei Asphalt als Mindestwert die Fußgängerzone sowie die Altstadt großräumig umgangen und ein Weg über die Kleberstraße, Markusstraße und Untere Sandstraße gewählt. Bei ebenem Kopfsteinpflaster als Mindestwert wird im Vergleich zur Route ohne Einschränkungen lediglich die Fußgängerzone über den Grünen Markt vermieden. BRouter ist durch seine Gewichtung über Kostenfunktionen insgesamt gut geeignet, das Vermeiden verschiedener Straßenbeläge zu erreichen, ohne das Routing durch zu starke Einschränkungen unmöglich zu machen (es wird stets ein Weg gefunden).

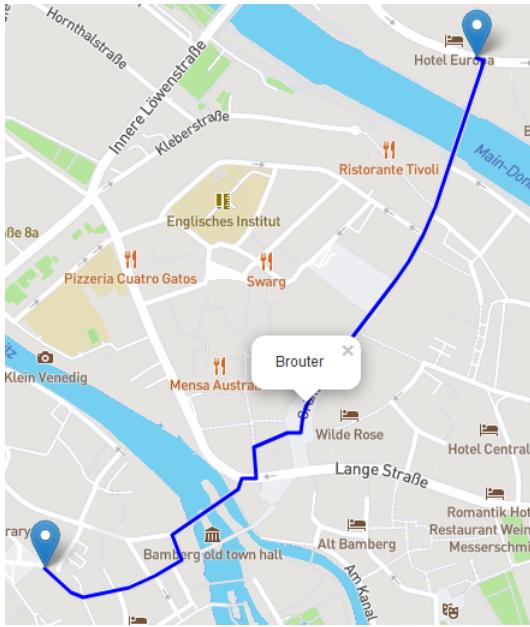


Abbildung 10.6: Route mit max. 10% Steigung

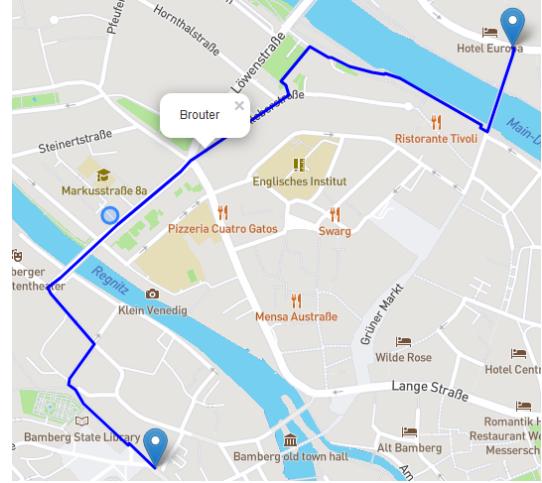


Abbildung 10.7: Route mit max. 5% Prozent Steigung ohne Treppen

### 10.3 GraphHopper

Da im Vergleich zu BRouter und ORS sich die GraphHopper API im Verlauf des Projektes in der Parametrisierung zu statisch erwiesen hat, handelt dieser Abschnitt um die allgemeine Mächtigkeit von GraphHopper abseits der API und dem Aufwand um diesen für das Projekt zu implementieren. Grund für das statische Verhalten von GraphHopper ist, dass die API zurzeit die Berücksichtigung von Bordsteinhöhen, Handläufen oder Steigungen nicht unterstützt. Die Parameter für Treppen und Straßenoberflächen konnten eingeschränkt implementiert werden. Zudem steht in der API-Dokumentation [52], dass sich die Konfigurierung von Profilen nur auf Standardprofile wie bspw. dem Fußgängerprofil einsetzen lässt. Des Weiteren wird zwar eine frühe Alphaversion für die Routen-Optimierung-API erwähnt, jedoch hat sich beim Austausch mit dem Entwicklerteam herausgestellt, dass diese nicht die MoMM-relevanten Attribute unterstützt. Weiter hat sich bei diesem E-Mail-Verkehr herausgestellt, dass das Entwicklerteam zurzeit an einem Rollstuhlprofil [53] arbeitet, dieses aber bisher noch nicht in der API integriert ist, sondern nur auf ihren Open-Source-Projekt vorzufinden ist. So hat sich beim Einlesen in den Quellcode des Rollstuhlprofils gezeigt, das dieses alle Kriterien unterstützt welches wir für dieses Projekt gewählt haben. Die Mächtigkeit der GraphHopper-Engine an sich besteht aus der Anpassbarkeit an individuellen Daten, der schnellen Berechnungszeit von Routen und den zahlreichen Priorisierungsmöglichkeit für individuelle Benutzerprofile. So ist es möglich sämtliche OSM-Daten zu integrieren wie bspw. barrierefreie Treppen. Zudem ist es möglich für die Routenberechnungen den öffentlichen Nahverkehr mit einzubeziehen. Hierzu werden sogenannte GFTS-Datensätze (General Transit Feed Specification) benötigt, welche alle Verkehrslinien, Haltepunkte und Zeitpläne enthalten. Diese sind für Deutschland kostenfrei erhältlich und werden wöchentlich aktualisiert [54]. Generell ist jedoch zu sagen, dass die Implementierung der GraphHopper Open-Source-Engine aufwendig ist, da man sich im Vergleich mit der API wesentlich mehr mit den Quellcode auseinandersetzen muss, um diesen in unser bisheriges System erfolgreich zu integrieren und zu

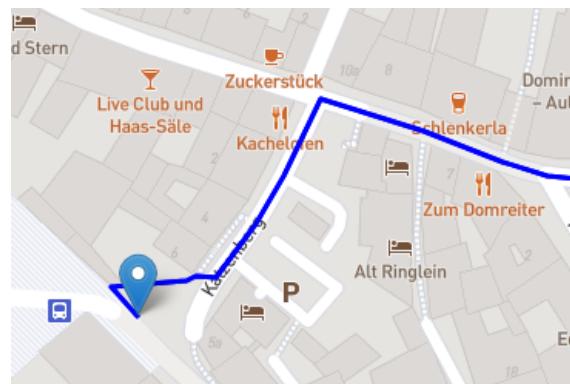


Abbildung 10.8: Route mit max. 5% Steigung mit Treppen (Ausschnitt)

modifizieren. Zudem ist zu erwarten, dass in nächster Zeit das Rollstuhlprofil ebenfalls in GraphHopper API integriert wird, was die aktuellen Anforderungen für das Projektes vollkommen erfüllt.

## 11 Fazit und Handlungsempfehlungen

Im folgenden Abschnitt der Arbeit sollen nun die in der Arbeit gewonnenen Erkenntnisse zusammengefasst und in konkrete Handlungsempfehlungen überführt werden.

### 11.1 Vergleichende Betrachtung der Algorithmen

BRouter zeichnet sich vor allem durch seine Flexibilität und seine einfache Implementierung aus. Eine Anpassung von Parametern zur Laufzeit ist durch das Versenden des Profil-Inhalts mit jeder HTTP-Anfrage problemlos möglich. Die Berücksichtigung der Parameter Beschaffenheit des Bodenbelags und Vermeidung von Treppen ist bereits gegeben. Um aber alle für ein MoMM-Routing relevanten OSM-Tags auslesen zu können, müsste entweder die Aufnahme neuer Tags in die Lookup Table bei der Community veranlasst oder eine eigene Instanz betrieben werden. Letzteres ist auf Grund gut dokumentierter Skripte mit mittlerem Aufwand zu bewältigen, erfordert aber eine eigene Infrastruktur und Ressourcen für die regelmäßige Aktualisierung des Kartenmaterials. Die größte Herausforderung stellt die Berücksichtigung von Steigungen dar. Die von BRouter ausgewerteten SRMT-Daten sind in einem urbanen Setting nur bedingt nützlich, so dass eine effektive Vermeidung von bestimmten Gefällen nur über den OSM-Tag *incline* erreicht werden kann. Das ist aus algorithmischer Sicht leicht zu implementieren, setzt jedoch ein noch zu leistendes umfangreiches Mapping voraus.

Vorteilhaft an ORS als Routing-Algorithmus ist die vorhandene Unterstützung vieler der für das MoMM-Projekt als wichtig erachteten Parameter, ohne Anpassungen am Quellcode vornehmen zu müssen. Um ORS für Bamberg im vollständig möglichen Umfang nutzen zu können, muss allerdings ein umfassenderes Mapping vor allem von Straßensteigungen und Bordsteinhöhen erfolgen. Eine Anpassung des Backends von ORS um weitere Parameter zu unterstützen, wäre durchführbar jedoch mit wesentlich mehr Aufwand verbunden als bei BRouter und GraphHopper.

Die Web API von GraphHopper ist zum Vergleich zu ORS und BRouter schlechter geeignet, da nicht alle MoMM-relevanten Parameter unterstützt werden können. So können die Werte für Straßensteigungen, Bordsteinhöhen und Handläufe nicht berücksichtigt werden. Generell und abseits der Web API ist jedoch zu erwähnen, dass der GraphHopper Routing-Algorithmus durch seine schnelle Routenberechnung profitiert, mit weiteren Daten erweiterbar ist und vom Entwicklerteam regelmäßig aktualisiert und erweitert wird.

### 11.2 Konkrete Handlungsempfehlungen

Im Folgenden sind konkrete Handlungsempfehlungen an weiterführende Projekte mit dem Ziel, in Bamberg Menschen mit Mobilitätseinschränkungen eine geeignete Routing-Anwendung zur Verfügung zu stellen, zusammengefasst. Diese Empfehlungen beziehen sich auf die Gestaltung des Frontends, das Hinzufügen fehlender OSM-Tags und die Auswahl des Routing-Algorithmus.

Während der Ortsbegehung mit einem Stakeholder, welcher selbst im Rollstuhl sitzt, haben sich einige Wünsche für Funktionen einer Routing-Applikation für Menschen mit Mobilitätseinschränkungen herausgestellt. Zum einen variiert es von Betroffenen zu Betroffenen, welche Hindernisse recht einfach überwunden werden können, welche mehr Anstrengung erfordern und welche unmöglich zu

überwinden sind. Gerade unter dem Gesichtspunkt, eine Anwendung für mehrere Mobilitätseinschränkungen schaffen zu wollen, und nicht auf Rollstuhlfahrer\*innen limitiert zu sein, hat sich das Konzept des MoMM-Prototypen, als Nutzer\*in selbst individuelle Parameter setzen zu können, als sinnvoll erwiesen. Ebenfalls wurde darauf hingewiesen, dass viele Hindernisse (Steigungen, Bordsteine etc.) zwar überwindbar wären, jedoch viel Unannehmlichkeit mit sich bringen. Dies führt zu einem Tradeoff zwischen kürzeren Strecken auf der einen Seite und komfortablen Stecken auf der anderen oder zu dem Konflikt, zwei mögliche Strecke zu haben, welche je unterschiedliche Parameterausprägungen der MoMM-Parameter besitzen. Der Wunsch wurde geäußert, ein User-Interface zu erstellen, welches mehrere mögliche Routen anzeigt und zu diesen Informationen über Länge der Strecke, Bodenbeläge, Steigung und ähnliches ausgibt. Die Nutzer\*in hat so die Möglichkeit selbst eine der Strecken wählen.

Um eine Routing-Applikation zu schaffen, die an die Bedürfnisse von Menschen mit Mobilitäts einschränkungen angepasst ist, müssen die für diese relevanten Parameter für die Stadt Bamberg besser gemappt werden, damit der jeweilige Algorithmus auch die tatsächlich am besten geeignete Route (oder Routen) errechnen kann. In den OSM-Daten für Bamberg fehlen insbesondere Angaben über Bordsteinhöhen, Handläufe und Straßensteigungen.

Bei der Wahl des Routing-Algorithmus muss zunächst entschieden werden, ob das vorhandene Maß der Parametrisierbarkeit einer der betrachteten Algorithmen für die eigenen Ziele ausreichend ist oder zwingend Anpassungen im Quellcode vorgenommen werden müssten. Fällt die Entscheidung darauf, den Quellcode nicht zu verändern und lediglich die API des jeweiligen Algorithmus anzusprechen, wie es auch der MoMM-Prototyp macht, so eignet sich ORS von den drei Algorithmen am besten, da hier aktuell die meisten als relevant eingestuften Parameter unterstützt werden. Wird hingegen entschieden, dass Abänderungen des Backends unvermeidbar sind, würde sich die Modifikation von ORS aufgrund vieler Änderungen an vielen Stellen im Quellcode als am aufwendigsten gestalten. Die benötigten Anpassungen BRouters, um zusätzliche Parameter zu unterstützen, ist deutlich weniger komplex und auch gut dokumentiert. Auch GraphHopper könnte durch verhältnismäßig wenig Aufwand entsprechend angepasst werden.

### 11.3 Abschließende Zusammenfassung

Die vorangegangene Arbeit setzt sich intensiv mit der Problemstellung eines barrierefreien und angepassten Routings für Menschen mit Mobilitäts einschränkungen auseinander. Dabei ist die Kernidee des MoMM-Projektes, ein für das Stadtgebiet Bamberg ausgelegtes Routing für Menschen mit temporären oder dauerhaften Mobilitäts einschränkungen zu entwickeln. Es wird von einer hohen Diversität der Zielgruppen ausgegangen, da sich nicht nur die Formen der Mobilitäts einschränkungen (z.B. Personen im Rollstuhl, ältere Menschen mit Geh-Einschränkungen oder Personen mit Kinderwagen), sondern auch die persönlichen Präferenzen und Möglichkeiten jeweils stark unterscheiden. Daher kommt der individuellen Parametrisierung der Suche eine große Bedeutung zu. Die Anforderungen an das Routing ergaben sich aus ersten vorläufigen Ergebnissen einer Stakeholder-Befragung von Cogita sowie den im Projektteam erarbeiteten Ideen.

Im weiteren Verlauf der Arbeit werden dann verschiedene bereits existierende Routingalgorithmen genauer betrachtet und auf Tauglichkeit im aktuellen Kontext untersucht. Dabei konnten drei Systeme in die nähere Betrachtung aufgenommen werden.

In einem speziell für diese Arbeit entwickelten und dargestellten Prototypen konnten die drei ausgewählten Routing-Systeme zur Routenerzeugung herangezogen werden. Dabei können die deklarierten Parameter individuell eingestellt und somit (sofern es beim jeweiligen System möglich ist) in die Routenerzeugung mit einbezogen werden. Über eine graphische Schnittstelle ist es den Benutzer\*innen möglich, die errechneten Routen in einer interaktiven Karte darzustellen und somit auch zu nutzen. Es ist ebenfalls möglich, die Route dynamisch durch verschieben von Start- und Zielpunkten einfach und in Echtzeit zu ändern. Der Prototyp erlaubt es zudem die verschiedenen erzeugten Routen gegenüberzustellen und somit die unterschiedlichen Systeme zu evaluieren. Ein analytisch gewähltes Paar aus Start- und Zielpunkten dient hierbei als Muster- und Vergleichsstrecke in der Bamberger Innenstadt. Der MoMM-WayFinder-Prototyp ist eine funktionale Software, die jedoch noch nicht produktions- und distributionsreif ist. Es fehlen beispielsweise noch die Möglichkeit in verschiedenen Sprachen Informationen zu erhalten oder eine barrierefreie Umsetzung der Webseite. Dennoch kann der Prototyp zur Erstellung von Routen herangezogen werden. Durch die ebenfalls erklärte Struktur kann dieser und dessen Backend mithilfe der vorliegenden Arbeit implementiert werden.

Im weiteren Verlauf der Arbeit wurden dann die aktuellen Unzulänglichkeiten der untersuchten Systeme herausgestellt und Implementationsanweisungen zur zukünftigen Umsetzung gegeben. Ebenfalls werden Handlungsaufforderungen dargelegt, was, welche Daten und welche zusätzlichen Programmieraufgaben benötigt werden, um ein zukünftiges barrierefreies Routing für Menschen mit Mobilitätseinschränkungen in der Stadt Bamberg durch eine entsprechende Applikation, beziehungsweise Webseite zu ermöglichen.

Alles in allem lässt sich resümieren, dass es auch mit bereits existierenden technologischen Lösungen möglich und zeitnah umsetzbar ist, ein zukünftiges barrierefreies Routing für Menschen mit Mobilitätseinschränkungen in der Stadt Bamberg zu ermöglichen. Um dies umzusetzen werden unter anderem jedoch gut orchestrierte OpenStreetMap-Daten und etwaige Anpassungen im Quellcode verschiedener Routing-Systeme benötigt. Die in dieser Arbeit herausgestellten Parameter bei der Routenerzeugung sollen jedoch keineswegs als finit angesehen werden. Vielmehr können und sollten diese durch ausgiebige Analysen oder Studien mit Nutzer\*innen der Lösung gemeinsam besprochen, priorisiert und schließlich umgesetzt werden.

## Abkürzungen

App	Applikation
CMTED	Global Multi-resolution Terrain Data
CSS	Cascading Style Sheets
FOSSGIS	Freie und Open Source Software für Geoinformationssysteme e.V.
GHSL	Global Human Settlement Layer
GMTED	Global Multi-resolution Terrain Elevation Data
GPS	Global Positioning System
GTFS	General Transit Feed Specification
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
LGPL	GNU Lesser General Public License
MIT	Massachusetts Institute of Technology Lizenz
MoMM	Mobilität für Menschen mit Mobilitätseinschränkungen
NPM	Node Packet Manager
ORS	Openrouteservice
OSM	OpenStreetMap
OSRM	Open Source Routing Machine
SRTM	Shuttle Radar Topography Mission
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator

## Abbildungen

3.1	WheelScout App Screenshot [9] . . . . .	5
3.2	Maps without Barriers Screenshot [13] . . . . .	6
3.3	Wheelmap Screenshot . . . . .	7
6.1	Testgebiet . . . . .	13
6.2	Annotationen von <i>smoothness</i> . . . . .	13
6.3	Annotationen der Straßenbeläge . . . . .	14
6.4	Annotation der Bordsteinhöhe . . . . .	14
8.1	Struktur des Backends . . . . .	18
8.2	Bereitstellungsdiagramm Backend . . . . .	20
9.1	Benutzeroberfläche Karte . . . . .	22
9.2	Anzeige von BRouter auf der Karte . . . . .	23
9.3	Anzeige von ORS auf der Karte . . . . .	23
9.4	ORS-Anfrage Code-Ausschnitt . . . . .	24
9.5	Beispiel Antwort API-Anfrage ORS . . . . .	25
9.6	Code-Ausschnitt Switch-Case ORS in <i>maps.js</i> . . . . .	25
9.7	Formular mit Anmerkungen . . . . .	26
10.1	Ausgangsrouting ohne Einschränkungen (ORS in rot) . . . . .	28
10.2	Routing mit 6 Prozent Steigung (ORS in rot) . . . . .	28
10.3	Alternativrouting mit 6 Prozent Steigung (ORS in rot) . . . . .	29
10.4	Routing für den Mindestbelag Kopfsteinplaster (ORS in rot) . . . . .	29
10.5	BRouter-Höhenprofil . . . . .	31
10.6	BRouter-Route 10% Steigung . . . . .	32
10.7	BRouter-Route 5% Steigung ohne Treppen . . . . .	32
10.8	BRouter-Route 5% Steigung mit Treppen . . . . .	33

## Literatur

- [1] *OpenStreetMap Deutschland*. Adresse: <https://www.openstreetmap.de> (besucht am 30.03.2022).
- [2] *OpenStreetMap Wiki Tags*. Adresse: <https://wiki.openstreetmap.org/wiki/Tags> (besucht am 30.03.2022).
- [3] *Dijkstra Algorithmus*. Adresse: [https://en.wikipedia.org/wiki/Dijkstra's\\_algorithm#Practical\\_optimizations\\_and\\_infinite\\_graphs](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Practical_optimizations_and_infinite_graphs) (besucht am 13.03.2022).
- [4] *A\* Vorteile*. Adresse: <https://www.slant.co/options/11585/~a-algorithm-review> (besucht am 13.03.2022).
- [5] *A\* Algorithmus*. Adresse: [https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm) (besucht am 13.03.2022).
- [6] *Contraction hierarchies*. Adresse: [https://de.abcdef.wiki/wiki/Contraction\\_hierarchies](https://de.abcdef.wiki/wiki/Contraction_hierarchies) (besucht am 13.03.2022).
- [7] *Wheel Scout-App: Für Rollstuhlfahrer*. Adresse: <https://www.digitalstadt-darmstadt.de/news/wheel-scout-app-fuer-rollstuhlfahrer/> (besucht am 13.03.2022).
- [8] B. Harriehausen-Mühlbauer, “Computation of Incline Barriers via Smartphone Sensors in the Mobile App WheelScout,” en, in *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 3*, K. Arai, S. Kapoor und R. Bhatia, Hrsg., Ser. Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, 2021, S. 121–139, ISBN: 978-3-030-63092-8. DOI: 10.1007/978-3-030-63092-8\_9.
- [9] B. Harriehausen-Mühlbauer und J. Roth, “WheelScout - Barrier-Free Navigation,” de, in *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 1*, Ser. Lecture Notes in Networks and Systems, Springer International Publishing : Imprint: Springer, 2018. DOI: DOI:10.1007/978-3-319-56994-9. Adresse: <https://www.springerprofessional.de/wheelscout-barrier-free-navigation/14227238> (besucht am 27.03.2022).
- [10] *Accessible maps – Barrier-Free Maps To Improve Occupational Mobility*. Adresse: [https://accessiblemaps.de/?page\\_id=121&lang=en](https://accessiblemaps.de/?page_id=121&lang=en) (besucht am 13.03.2022).
- [11] *Accessible maps - results*. Adresse: [https://accessiblemaps.de/?page\\_id=451&lang=en](https://accessiblemaps.de/?page_id=451&lang=en) (besucht am 13.03.2022).
- [12] P. Rozenkraft, *Maps without barriers - About project*. Adresse: <https://web.mapybezbarier.cz/en/about-project/> (besucht am 13.03.2022).
- [13] M. Pičman, *Konto Bariéry - Maps without barriers*. Adresse: <https://mapybezbarier.cz/en> (besucht am 27.03.2022).
- [14] *Mitmachen*, de-DE. Adresse: <https://news.wheelmap.org/mitmachen/> (besucht am 24.03.2022).
- [15] ways4all, *Ways4all*. Adresse: <http://www.ways4all.at/index.php/de/ways4all> (besucht am 24.03.2022).
- [16] *Gosmore*. Adresse: <https://wiki.openstreetmap.org/wiki/Gosmore> (besucht am 30.03.2022).

- [17] *Globaler Valhalla Server Online*. Adresse: [https://fossgis.de/news/2021-11-12\\_funding\\_valhalla/](https://fossgis.de/news/2021-11-12_funding_valhalla/) (besucht am 12.02.2022).
- [18] *Data Sources in Valhalla*. Adresse: [https://valhalla.readthedocs.io/en/latest/mjolnir/data\\_sources/](https://valhalla.readthedocs.io/en/latest/mjolnir/data_sources/) (besucht am 30.03.2022).
- [19] *Costing Models*. Adresse: <https://valhalla.readthedocs.io/en/latest/api/turn-by-turn/api-reference/> (besucht am 30.03.2022).
- [20] *OSRM profiles*. Adresse: <https://github.com/Project-OSRM/osrm-backend/blob/master/docs/profiles.md> (besucht am 30.03.2022).
- [21] *GraphHopper Profile*. Adresse: <https://github.com/graphhopper/graphhopper/blob/master/docs/core/profiles.md> (besucht am 30.03.2022).
- [22] *Openrouteservice Wiki*. Adresse: <https://wiki.openstreetmap.org/wiki/DE:Openrouteservice> (besucht am 30.03.2022).
- [23] *Openrouteservice Github*. Adresse: <https://github.com/GIScience/openrouteservice> (besucht am 30.03.2022).
- [24] *Openrouteservice API Playground*. Adresse: <https://openrouteservice.org/dev/#/api-docs> (besucht am 30.03.2022).
- [25] *Openrouteservice Github Data*. Adresse: <https://giscience.github.io/openrouteservice/Data.html> (besucht am 30.03.2022).
- [26] *Openrouteservice Github Routing-Options*. Adresse: <https://giscience.github.io/openrouteservice/documentation/routing-options/Routing-Options.html> (besucht am 30.03.2022).
- [27] *Openrouteservice Github Tag-Filtering*. Adresse: <https://giscience.github.io/openrouteservice/documentation/Tag-Filtering.html> (besucht am 30.03.2022).
- [28] *BRouter segment files*. Adresse: <https://github.com/abrensch/brouter/blob/master/misc/readmes/mapcreation.md> (besucht am 30.03.2022).
- [29] *BRouter Webclient*. Adresse: <https://brouter.de/brouter-web> (besucht am 30.03.2022).
- [30] *About us Graphhopper*. Adresse: <https://www.graphhopper.com/about-us/> (besucht am 13.03.2022).
- [31] *README Graphhopper*. Adresse: <https://github.com/graphhopper/graphhopper/blob/master/README.md> (besucht am 13.03.2022).
- [32] *raphhopper Open Source*. Adresse: <https://www.graphhopper.com/open-source/> (besucht am 13.03.2022).
- [33] *Key Smoothness*. Adresse: <https://wiki.openstreetmap.org/wiki/DE:Key:smoothness> (besucht am 13.03.2022).
- [34] Universität Bamberg, *gitlab.rz.uni-bamberg.de*, publisher: Universität Bamberg, März 2022. Adresse: <https://gitlab.rz.uni-bamberg.de/>.
- [35] OpenJS Foundation und Joyent, *Node.js*, publisher: OpenJS Foundation, März 2022. Adresse: <https://nodejs.org/>.
- [36] npm, Inc., *npm*, publisher: npm, Inc., März 2022. Adresse: <https://www.npmjs.com/>.

- [37] StrongLoop, IBM, and other expressjs.com contributors, *Express - Node.js web application framework*, publisher: OpenJS Foundation, 2017.
- [38] Y. Katz, *Handlebars*, publisher: Yehuda Katz, 2011. Adresse: <https://handlebarsjs.com/>.
- [39] JGraph Ltd., *Diagram Software and Flowchart Maker*, publisher: JGraph Ltd., 2021. Adresse: <https://www.diagrams.net/>.
- [40] Internet Security Research Group (ISRG), *Let's Encrypt - Freie SSL/TLS Zertifikate*, de-DE, 2022. Adresse: <https://letsencrypt.org/de/> (besucht am 07.02.2022).
- [41] nginx, *Advanced Load Balancer, Web Server, & Reverse Proxy*, en-US, 2022. Adresse: <https://www.nginx.com/> (besucht am 07.02.2022).
- [42] C. Jaquier, *fail2ban*, original-date: 2011-09-28T16:24:20Z, 2022. Adresse: <https://github.com/fail2ban/fail2ban> (besucht am 08.02.2022).
- [43] E. Troan, P. rRown und J. Kaluza, *logrotate(8) - Linux man page*. Adresse: <https://linux.die.net/man/8/logrotate>.
- [44] Docker Inc., *Empowering App Development for Developers — Docker*, 2022. Adresse: <https://www.docker.com/> (besucht am 08.02.2022).
- [45] *Leaflet — an open-source JavaScript library for interactive maps*. Adresse: <https://leafletjs.com/> (besucht am 13.03.2022).
- [46] *Maps, geocoding, and navigation APIs & SDKs — Mapbox*. Adresse: <https://www.mapbox.com/> (besucht am 13.03.2022).
- [47] *Openrouteservice Github Installation*. Adresse: <https://giscience.github.io/openrouteservice/installation/Installation-and-Usage.html> (besucht am 30.03.2022).
- [48] *BRouter lookup table*. Adresse: <https://brouter.de/brouter/profiles2/lookups.dat> (besucht am 30.03.2022).
- [49] *BRouter wheelchair profile*. Adresse: <https://github.com/poutnikl/Hiking-Poutnik/blob/master/Wheelchair.brf> (besucht am 30.03.2022).
- [50] *BRouter*. Adresse: [https://brouter.de/brouter/profile\\_developers\\_guide.txt](https://brouter.de/brouter/profile_developers_guide.txt) (besucht am 30.03.2022).
- [51] *SRTM Daten*. Adresse: <https://de.wikipedia.org/wiki/SRTM-Daten> (besucht am 30.03.2022).
- [52] *GraphHopper API Documentation*. Adresse: <https://docs.graphhopper.com/> (besucht am 10.03.2022).
- [53] *Custom Wheelchair Profil*. Adresse: <https://github.com/graphhopper/graphhopper/pull/2503/files> (besucht am 10.03.2022).
- [54] *GFTS Deutschland*. Adresse: <https://gtfs.de/de/> (besucht am 10.03.2022).