

StudMap

Indoor Navigation

Projektdokumentation

im Fach Fortgeschrittene Internetanwendungen



Westfälische Hochschule

Gelsenkirchen Bocholt Recklinghausen

vorgelegt von: Thomas Buning, Marcus Büscher,
Daniel Hardes, Christoph Inhestern,
Dennis Miller, Fabian Paus,
Christian Schlütter

Studienbereich: Informationstechnik

Gutachter: Prof. Dr. Martin Schulten

Abgabetermin: 21.01.2014

Inhaltsverzeichnis

1	Einleitung	1
1.1	Projektorganisation	1
2	Architektur	2
12	Datenbank	32
12.1	Maps	32
12.1.1	Maps	32
12.1.2	Floors	33
12.1.3	Graphs	33
12.1.4	Edges	33
12.1.5	Nodes	34
12.1.6	NodeInformation	34
12.1.7	PoIs	34
12.1.8	PoiTypes	35
12.2	Users	35
4	Service	8
4.1	Allgemeine Struktur	8
5	Collector	9
5.1	Recherche	9
5.1.1	QR-Codes	9
5.1.1.1	Serverseitig	9
5.1.1.2	Clientseitig	10
5.1.2	NFC-Tags	10
5.1.2.1	Umsetzung	10
5.1.3	OCR der Raumschilder	11
5.1.3.1	Bibliothek tesseract	11
5.1.3.2	App Google Goggles	11
5.1.4	WLAN Fingerprinting	13
5.1.4.1	Sammeln von WLAN Fingerprints	13
5.1.4.2	Kalibrierung	14
5.1.4.3	Positionierung mittels WLAN Fingerprints	14
5.2	Allgemeine Struktur	14
5.3	Benutzeroberfläche	14
5.4	Core Bibliothek	15



6 Admin	16
6.1 Allgemeine Struktur	16
6.2 Benutzeroberfläche	17
6.3 Admin Spezifisches ?!	17
7 Client	18
7.1 Allgemeine Struktur	18
7.1.1 Positionserkennung	18
7.1.2 Anzeige und Navigation auf einer Karte	18
7.1.3 Visuell ansprechende und intuitiv bedienbare Applikation	19
7.2 Benutzeroberfläche	19
8 Fazit	20
9 Ausblick	21
10 Domänenmodell	22
10.1 Architektur	22
10.1.1 Webservice (StudMap.Service)	22
10.1.2 Admin-Oberfläche (StudMap.Admin)	22
10.1.3 Navigations-Client (StudMap.Navigator)	22
10.1.4 Sammler-Client (StudMap.Collector)	22
10.2 Benutzerrollen	22
10.2.1 Anwender (User)	22
10.2.2 Administrator	23
10.3 Begriffe	23
10.3.1 Karte (Map)	23
10.3.2 Stockwerk (Floor)	23
10.3.3 Schicht (Layer)	23
10.3.4 Route	23
10.3.5 Graph	23
10.3.6 Knoten (Node)	24
10.3.7 Kante (Edge)	24
10.3.8 Point of Interest (POI)	24
10.4 Recherche	25
10.4.1 QR-Codes	25
10.4.1.1 Serverseitig	25
10.4.1.2 Clientseitig	25
10.4.2 NFC-Tags	26
10.4.2.1 Umsetzung	26



10.4.3	OCR der Raumschilder	26
10.4.3.1	Bibliothek tesseract	27
10.4.3.2	App Google Goggles	28
10.4.4	WLAN Fingerprinting	28
10.4.4.1	Sammeln von WLAN Fingerprints	29
10.4.4.2	Kalibrierung	29
10.4.4.3	Positionierung mittels WLAN Fingerprints	29
11	Positionsermittlung	31
11.1	QR-Tags	31
11.2	NFC-Tags	31
12	Datenbank	32
12.1	Maps	32
12.1.1	Maps	32
12.1.2	Floors	33
12.1.3	Graphs	33
12.1.4	Edges	33
12.1.5	Nodes	34
12.1.6	NodeInformation	34
12.1.7	PoIs	34
12.1.8	PoiTypes	35
12.2	Users	35
13	Benutzerverwaltung	37
14	Webservice	40
14.1	Allgemeine Objekte	40
14.1.1	Edge	40
14.1.2	Node	41
14.1.3	Graph	41
14.1.4	Pathplot	41
14.1.5	FloorPlanData	42
14.1.6	Room	42
14.1.7	PoiType	43
14.1.8	PoI	43
14.1.9	RoomAndPoI	43
14.1.10	NodeInformation	44
14.1.11	QRCode	44
14.1.12	FullNodeInformation	44
14.1.13	Floor	45



14.1.14 Map	45
14.1.15 User	46
14.1.16 SaveGraphRequest	46
14.2 Rückgabe Objekte	46
14.2.1 BaseResponse	46
14.2.1.1 ResponseStatus	47
14.2.1.2 ResponseError	47
14.2.2 ObjectResponse	48
14.2.3 ListResponse	49
14.3 MapsController	49
14.3.1 CreateMap	49
14.3.2 DeleteMap	49
14.3.3 GetMaps	49
14.3.4 CreateFloor	50
14.3.5 DeleteFloor	50
14.3.6 GetFloorsForMap	50
14.3.7 GetFloor	50
14.3.8 GetFloorplanImage	51
14.3.9 SaveGraphForFloor	51
14.3.10 DeleteGraphForFloor	51
14.3.11 GetGraphForFloor	51
14.3.12 GetFloorPlanData	52
14.3.13 GetRouteBetween	52
14.3.14 GetNodeInformationForNode	52
14.3.15 SaveNodeInformation	53
14.3.16 GetPoiTypes	53
14.3.17 GetPoIsForMap	53
14.3.18 GetRoomsForMap	53
14.3.19 GetNodeForNFC	54
14.3.20 GetNodeForQRCode	54
14.4 UsersController	54
14.4.1 Register	54
14.4.2 Login	55
14.4.3 Logout	55
14.4.4 GetActiveUsers	55
15 Verwendung des Webservices	56
15.1 Verwendung der Benutzerschnittstelle	56
15.1.1 Registrierung	56
15.1.2 Aktive und inaktive Benutzer	56



15.1.3 Aktive Benutzer abfragen	56
16 Maintenance Tool	57
16.1 QR-Codes generieren	57



1 Einleitung

Im Fach Fortgeschrittene Internetanwendungen haben wir uns im Rahmen einer studentischen Projektarbeit mit dem Thema Navigation und Lokalisierung innerhalb von Gebäuden beschäftigt. Dabei beschränkte sich das Ziel unseres Projekts auf die Navigation im Gebäude der Westfälischen Hochschule am Campus Bocholt. Dazu stellten wir uns zu Projektbeginn eine Karte des Gebäudes vor, auf der alle möglichen Navigationsziele eingezeichnet sind. Bei der Auswahl eines Navigationsziels sollte, nach unseren Vorstellungen, eine entsprechende Wegbeschreibung eingeblendet werden, die uns von unserem aktuellen Standpunkt zum gewünschten Ziel führt.

Um dieses Ziel zu erreichen, mussten wir uns mit verschiedenen Problemstellungen auseinandersetzen. Zunächst einmal war es nötig das Gebäude vollständig in einem (unserem) System zu erfassen und dieses auf der Karte unserer Vorstellung darzustellen. Zum anderen mussten wir die aktuelle Position (innerhalb des Gebäudes) ermitteln, um diese ebenfalls auf der Karte abbilden zu können.

1.1 Projektorganisation

Als Plattform für unser Projekt verwenden wir Google Code:

<https://code.google.com/p/studmap/>

Dort nutzen wir das SVN Repository zur Quellcode Ablage und den Issue Tracker zur Verwaltung von Benutzeranforderungen und Fehlern. Wir haben uns in unserem Projekt für eine agile Projektorganisation nach dem Vorbild von Scrum entschieden und den Issue Tracker entsprechend konfiguriert. So stehen uns die Issue Typen User Story, Task und Bug zur Verfügung. Zusätzlich haben wir noch vier Kategorien eingeführt: ProductBacklog, SprintBacklog, OpenBugs und OpenTasks. Mittels der Kategorien können wir die verschiedenen Issues besser strukturieren.

Kurz nach Beginn des Projektes haben wir die Benutzeranforderungen in Form von User Stories angelegt und dem ProductBacklog zugewiesen. Für dieses Projekt haben wir uns auf Sprints mit einer Dauer von jeweils zwei Wochen geeinigt. Zu Beginn eines jeden Sprints haben wir entsprechende User Stories in den SprintBacklog übertragen und abgearbeitet.



2 Architektur

3 Datenbank

Die Daten für StudMap werden in einer zentralen Datenbank gespeichert. In diesem Kapitel werden die einzelnen Tabellen thematisch gruppiert und Besonderheiten erläutert.

3.1 Maps

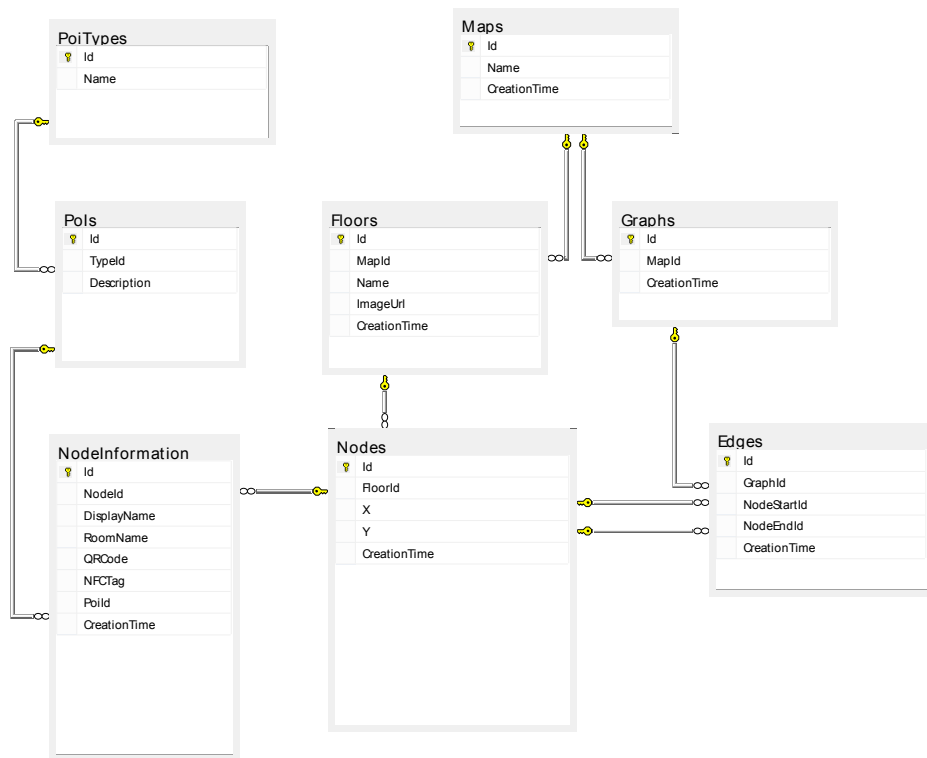


Abbildung 3.1: Datenbankmodell für die Kartenobjekte

3.1.1 Maps

Für jede Karte wird ein Eintrag in dieser Tabelle erzeugt. Zu jeder Karte wird ein frei vergebener Name und der Erstellungszeitpunkt gespeichert.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Karte
Name	NVARCHAR(255)	Name der Karte
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.2 Floors

Für jedes Stockwerk wird ein Eintrag in dieser Variable angelegt. Zu jedem Stockwerk wird ein frei vergebenen Name, eine URL auf ein Bild des Stockwerks und ein Erstellungszeitpunkt gespeichert. Ein Stockwerk ist genau einer Karte zugeordnet

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des Stockwerks
MapId	INTEGER (FK)	ID der zugeordneten Karte
Name	NVARCHAR(255)	Name des Stockwerks
ImageUrl	NVARCHAR(MAX)	URL des Bilds
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.3 Graphs

Ein Graph beschreibt die Knoten- und Kantenstruktur auf einer Karte. Dazu werden alle Kanten mit dem Graphen verknüpft. Über die Kanten sind auch die Knoten mit dem indirekt verknüpft.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des Graphen
MapId	INTEGER (FK)	ID der zugeordneten Karte
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.4 Edges

Eine Kante verknüpft zwei Knoten in einem Graphen.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Kante
GraphId	INTEGER (FK)	ID der zugeordneten Graphen
NodeStartId	INTEGER (FK)	ID des Startknotens
NodeEndId	INTEGER (FK)	ID des Endknotens
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.5 Nodes

Die Position eines Knoten wird durch die Zuordnung zu einem Stockwerk und seine X/Y-Koordinaten auf diesem Stockwerk bestimmt. Außerdem wird der Erstellungszeitpunkt eines Knotens gespeichert.

Die X- und Y-Koordinaten werden im Bereich 0.0 bis 1.0 gespeichert. Dabei bedeutet 0.0 ganz links (X) oder ganz oben (Y) und 1.0 ganz rechts (X) bzw. ganz unten (Y) auf dem Bild des Stockwerks.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knotens
FloorId	INTEGER (FK)	ID der zugeordneten Stockwerks
X	DECIMAL(18,17)	X-Koordinate auf dem Stockwerk
Y	DECIMAL(18,17)	Y-Koordinate auf dem Stockwerk
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.6 NodeInformation

Zu einem Knoten können noch weitere Informationen hinterlegt werden. Diese sind optional und werden nur zu wichtigen Knoten wie Seminarräumen, Büros und Toiletten. Über die Knoteninformationen kann auch ein PoI mit dem Knoten verknüpft werden.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knoteninformation
NodeId	INTEGER (FK)	ID der zugeordneten Knotens
DisplayName	NVARCHAR(50)	Name der angezeigt werden soll
RoomName	NVARCHAR(255)	Offizieller Raumname (z.B. B4.0.1.11)
QRCode	NVARCHAR(255)	Hinterlegter QR-Code
NFCTAG	NVARCHAR(50)	Hinterlegtes NFC-Tag
PoiId	INTEGER (FK)	Optionalen zugeordneter PoI
CreationTime	DATETIME	Erstellungszeitpunkt

3.1.7 Pols

Ein PoI (Point of Interest) kategorisiert für den Benutzer relevante Knoten. Hier kann z.B. nach Dozentenbüros, Mensa, Bibliothek und Toiletten gefiltert werden.



Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knotens
TypeId	INTEGER (FK)	ID des PoI-Typs
Description	NVARCHAR(MAX)	Zusätzliche Beschreibung des PoIs

3.1.8 PoiTypes

Diese Tabelle enthält die möglichen Typen von PoIs.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des PoI-Typs
Name	NVARCHAR(255)	Name des PoI-Typs

3.2 Users

ä

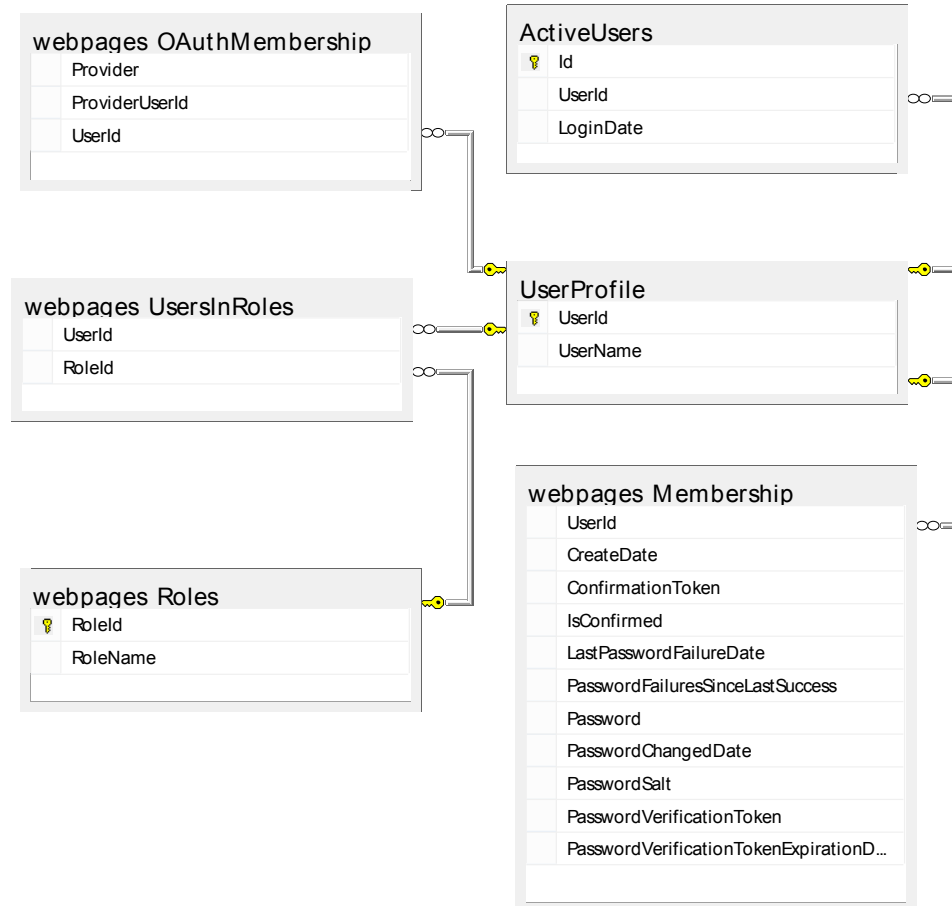


Abbildung 3.2: Datenbankmodell für Benutzerobjekte



4 Service

4.1 Allgemeine Struktur

5 Collector

Der Collector ist eine Android-Applikation, die dazu dient, Daten zu vorher vom Administrator definierten Knoten/Punkten zu sammeln und diese entsprechend in der Datenbank zu hinterlegen. Damit ist die Applikation ein Werkzeug der Administratoren und nicht der Endbenutzer.

Mit der Entwicklung des Collectors haben wir eine intensive Recherche betrieben, wie wir eine Indoor-Navigation ermöglichen können. Bevor wir das Endergebnis der Collector-Applikation näher erläutern, bieten wir nun einen Überblick über unsere Recherche-Ergebnisse.

5.1 Recherche

Für die Navigation innerhalb von Gebäuden konnten wir nicht auf die gängigen Standards zurückgreifen, sondern mussten uns andere Wege überlegen, wie wir die Nutzer innerhalb der Gebäude lokalisieren.

Außerhalb von Gebäuden ist die Lokalisierung mittels GPS sehr verbreitet und auch einfach und akkurat. Um eine ähnliche Lokalisierung unserer Nutzer zu ermöglichen haben wir uns die folgenden Möglichkeiten überlegt.

5.1.1 QR-Codes

QR-Codes sind weit verbreitet, einfach zu erstellen und mit vielen Geräten einzulesen. Dadurch bieten QR-Codes die Möglichkeit Informationen einfach an Orten anzubringen und von Maschinen einzulesen.

Wir haben uns zwei Konzepte überlegt, QR-Codes in unserem Projekt einzubauen. Dazu haben wir einmal die Möglichkeit betrachtet die Auswertung des QR-Codes am Server zu realisieren und mit der Möglichkeit verglichen die Auswertung direkt am Client des Nutzers zu implementieren.

5.1.1.1 Serverseitig

Für die serverseitige Umsetzung hat gesprochen, dass das Smartphone keine Rechenleistung benötigt um die QR-Codes zu dekodieren. Des Weiteren wird durch eine serverseitige Implementierung vermieden, dass der Nutzer weitere Apps auf seinem Smartphone installieren muss.



Für die Implementierung in den Webservice haben wir uns für die offene Bibliothek [ZXing.NET](#) benutzt. Allerdings ist uns bei der Implementierung und Testen der Bibliothek direkt aufgefallen, dass die Bilder zuvor am Smartphone verkleinert werden müssen um Bandbreite zu sparen und die Laufzeit der Bibliothek zu verringern. Dadurch wird, obwohl es ein Ziel der serverseitigen Umsetzung war, Rechenleistung benötigt. Darüber hinaus mussten wir feststellen, dass die Bibliothek keine zuverlässige Dekodierung der QR-Codes bietet.

5.1.1.2 Clientseitig

Im Gegensatz zur serverseitigen Umsetzung wird bei dieser Implementierung die gesamte Dekodierung am Client vorgenommen. Dadurch wird die Netzlast verringert, der Aufwand am Client aber erhöht.

Hier bot sich zum einen an eine Bibliothek in den Client aufzunehmen, oder eine externe Anwendung zum Dekodieren der QR-Codes zu benutzen. Wir haben uns schlussendlich dazu entschieden

Entscheidung
beim QR-
Code
Reader
aufschrei-
ben und
warum

5.1.2 NFC-Tags

NFC ist eine Technologie, auf die wir im Alltag immer häufiger stoßen, sei es bei Werbung, Bezahl- oder Ticketsystemen. NFC steht für Near Field Communication und besteht aus zwei Komponenten, der passiven, dem NFC-Tag, und der aktiven Komponente, beispielsweise dem Smartphone.

Die als Datenspeicher fungierenden NFC-Tags werden immer preiswerter und sind zudem relativ klein, was eine Anbringung an den gewünschten Orten problemlos ermöglicht. Auf der anderen, der aktiven Seite stehen immer mehr Smartphones bereit, die diese Technologie unterstützen.

Die steigende Beliebtheit der NFC-Technologie verdankt diese dem Komfort. Wie der Name bereits aussagt, reicht schon die Nähe der aktiven Komponente zur passiven um Daten zu kommunizieren. Diesen Komfort bieten wir dem Benutzer, um seine Position dem Navigator mitzuteilen.

Des Weiteren lassen sich auf dem NFC-Tag zusätzliche Informationen hinterlegen, die Unwissende auf die Navigationsmöglichkeit aufmerksam machen.

5.1.2.1 Umsetzung

Der gesamte Prozess der Positionsermittlung findet clientseitig statt. Zum einen ist das Client-Gerät unumgänglich für die Kommunikation mit dem NFC-Tag und zum anderen sind die Datenmengen und der Aufwand der Interpretation sehr gering.

Die Android NFC API ermöglicht uns die native Umsetzung der Positionsermittlung für Android-Geräte.

5.1.3 OCR der Raumschilder

Anstatt QR- oder NFC-Tags an den Räumen der FH anzubringen, besteht die Möglichkeit die bereits angebrachten Türschilder zu verwenden. Hierzu ist eine Erkennung der Raumnummer auf dem Türschild notwendig. Die OCR (Optical Character Recognition) kann über eine Bibliothek sowohl auf Client-, als auch auf Serverseite erfolgen. Erfolgt die Erkennung auf dem Client, dann könnten ebenfalls bestehende Apps zur Texterkennung verwendet werden.

5.1.3.1 Bibliothek tesseract

Tesseract¹ ist eine native Bibliothek für die Erkennung von Text in Bilddateien. Es gibt sowohl für .NET als auch für Java (Android) entsprechende Wrapper, die von uns verwendet werden können. Bei den Tests zu der OCR-Bibliothek haben sich allerdings einige Schwächen gezeigt. Tesseract ist für die Texterkennung von gescannten Dokumenten gedacht und arbeitet deshalb nur stabil, wenn sich auf dem Bild ausschließlich Text befindet. Dies wird an folgenden Beispielbildern deutlich.

Auf dem Bild mit Raumschild und Wand wird nur unzuverlässig Text erkannt (siehe 10.1). Die Ergebnisse variieren von "Kein Text erkannt" bis hin zu "Buchstabensalat mit Raumnummer". Schneidet man den relevanten Teil per Hand aus (siehe 10.2), wird der Text einwandfrei erkannt.

Damit die Raumschilder zuverlässig erkannt werden, ist es notwendig, dass aufgenommene Bilder zunächst auf den Bereich mit der Raumnummer zugeschnitten werden. Dies erfordert einen hohen Entwicklungsaufwand.

5.1.3.2 App Google Goggles

Google Goggles² ist eine Android-App, die zur Erkennung von Text, Symbolen und QR-Tags verwendet werden kann. Diese App lieferte in Tests auch bei suboptimalen Bildern gute Ergebnisse. Außerdem ist die App gut in das Android-Umfeld eingebettet und lässt sich leicht bedienen.

Allerdings gibt es für die Verwendung von Google Goggles noch keine öffentliche API³. Diese ist zwar von Google geplant, aber nie umgesetzt worden. Auch wenn

¹<https://code.google.com/p/tesseract-ocr/>

²<https://play.google.com/store/apps/details?id=com.google.android.apps.unveil>

³<http://stackoverflow.com/questions/2080731/google-goggles-api>



Abbildung 5.1: Gesamtes Bild Raumschild

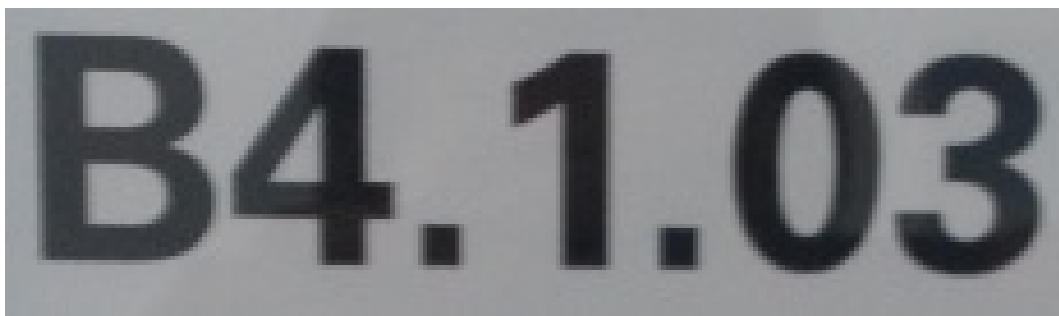


Abbildung 5.2: Ausschnitt Raumschild

die App eine komfortable Möglichkeit zur OCR-Erkennung bietet, kann diese ohne API nicht in unserem Projekt verwendet werden.

5.1.4 WLAN Fingerprinting

Bei der Positionierung des Nutzers mittels WLAN haben wir über eine für den Nutzer passive Positionierung recherchiert. Alle anderen Positionierungsmethoden benötigten eine Eingabe des Nutzers. Wir haben hier die Eingabe der Position auf einer Karte und das Einlesen von QR-Codes oder NFC-Tags behandelt. Die Positionierung ermöglicht es allerdings im Hintergrund zu laufen und ohne Eingabe des Nutzers die Position zu bestimmen.

WLAN ist zur Positionierung innerhalb von Gebäuden geeignet, da es zum einen eine weit verbreitete Infrastruktur ist, auf vielen mobilen Plattformen verfügbar ist, Wände durchdringt und Standard WLAN Access Points bereits eine Lokalisierung auf Raum-Genauigkeit ermöglicht.

Aus all diesen Gründen haben wir uns mit der Positionierung mittels WLAN beschäftigt.

5.1.4.1 Sammeln von WLAN Fingerprints

Um später Vergleiche im Client anstellen zu können mussten wir zuerst Daten des Netzwerkes sammeln. Ein Access Point wird dabei eindeutig durch eine **BSSID** gekennzeichnet und der Client gibt Auskunft über die empfangene Signalstärke (**RSS**⁴), welche beobachtet und aufgezeichnet werden kann.

Ziel dieser Phase war es an möglichststen vielen Punkten in der Hochschule die *RSS* zu messen und diese zu einem Punkt auf der Karte der Hochschule zu speichern.



⁴RSS: received signal strength wird in dBm gemessen.



5.1.4.2 Kalibrierung

Da das Sammeln der WLAN Fingerprints mit einem Smartphone realisiert wird und wir davon ausgehen mussten, dass nicht jeder Nutzer das gleiche Smartphone besitzt, mussten wir uns eine Möglichkeit der Kalibrierung überlegen. Dazu haben wir überlegt, dass der Nutzer zuerst in einer Kalibrierungsphase selbst einen Fingerprint erstellt von einem von uns festgelegten Ort und diesen mit dem von uns gemessenen Fingerprint vergleicht. Dadurch bekommen wir einen Faktor um den das Smartphone des Nutzer von unserem Gerät abweicht. Da wir vermuten, dass die WLAN Antennen der Smartphones auch in verschiedenen Bereichen, hohe, mittlere und niedrige Signalstärke, sich stark unterscheiden berechnen wir diesen Faktor für die gerade genannten Bereiche.

Dieser Teilbereich ist in der *StudMap-App* umgesetzt.

5.1.4.3 Positionierung mittels WLAN Fingerprints

Um die Position eines Nutzers ermitteln zu können muss dieser, wie der *Collector* einen Fingerprint des WLANs an seiner aktuellen Position erstellen. Diesen Fingerprint und seine Faktoren, welche während der Kalibrierung ermittelt wurden, schickt der Client zum Server, welcher durch Vergleiche den Standpunkt ermittelt und zurückgibt.

Dieses Feature ist auch in der *StudMap-App* umgesetzt.

Wie werden die FP verglichen.

5.2 Allgemeine Struktur

Die Collector-Applikation wurde als Android-Applikation umgesetzt. Es wird Android 3.0 auf einem Smartphone vorausgesetzt, um alle Funktionen nutzen können. Anhand der Recherche-Ergebnisse haben wir uns entschieden einen Positionserkennung mittels QR-Codes, NFC-Tags und Wlan-Fingerprinting zu implementieren. Dabei werden QR-Codes mittels eines Tools auf dem Server generiert. Es bedarf hierbei keiner weiteren Behandlung durch die Collector-Applikation. Dem gegenüber stehen das Zuordnen von NFC-Tags zu Knoten und das Sammeln von Wlan-Fingerprints durch die Collector-Applikation. Zur Erfüllung dieser beiden Aufgaben bietet die Applikation eine schlichte Benutzeroberfläche.

5.3 Benutzeroberfläche

Benutzeroberfläche und Funktionsweise sind mir fremd.



5.4 Core Bibliothek

Wir haben schon früh festgestellt, dass es eine Vielzahl an Strukturen und Funktionalitäten geben wird, die sowohl in der Collector-Applikation als auch in unserer Navigator-Applikation für den Endbenutzer Anwendung finden. Dem entsprechend haben wir diese in eine eigene Android-Bibliothek ausgelagert, die wir wiederum in unsere Android-Applikationen einbinden konnten. Zu den Kernfunktionalitäten und -Strukturen gehören unter Anderem folgende:

- Grundlegende Definitionen einer Map, eines Floors oder eines Knoten
- Konstanten für die Kommunikation mit dem Webservice
- Ein ErrorHandler für alle grundlegenden Fehler
- Snippets zur einfachen Kommunikation mit dem Benutzer mittels Dialogen o.ä.
- Abbildung des Webservices zur vereinfachten Nutzung
- Javascript-Schnittstellendefinitionen für die Interaktion auf der Karte
- Asynchrone Tasks für Webservicekommunikation inkl. entsprechender Listener



6 Admin

Wesentlicher Bestandteil unseres Projektes war die Administrationsumgebung. Genauer musste eine Anwendung geschaffen werden, die dem jeweiligen Endanwender Navigationsdaten zur Verfügung stellt. Also eine Umgebung, die einen Upload für Kartenmaterial bereitstellt. Des weiteren muss die Administrationsanwendung Features besitzen um Routen zu definieren und Meta-Informationen zu besonderen Orten festhalten zu können. Die Meta-Informationen können durch Points of Interest (PoI) Informationen erweitert werden.

Im folgenden Abschnitt wird zunächst die allgemeine Struktur erläutert und anschließend liegt das Hauptaugenmerk auf der Benutzeroberfläche. Eine ausführliche Bedienungsanleitung der Administrationsumgebung ist im Anhang beigefügt.

6.1 Allgemeine Struktur

ASP.NET MVC 4

Basis unserer Projektstruktur war das **ASP.NET MVC Framework**, welches ein Web Application Framework ist, und ein Model-View-Controller-Pattern implementiert.

Dies ermöglichte uns, eine Webanwendung zu entwickeln, bei der die Daten (*Model*) gekapselt von der Ausgabe (*View*) und dem *Controller* vorliegen. Die *View* repräsentiert unsere Daten und der *Controller* reagiert auf Zustandsänderungen und ist sozusagen das Bindeglied oder die Schnittstelle zwischen *View* und *Model*.

Controller

HomeController

Speziell für unser Projekt bedeutet es, dass wir drei Controller angelegt haben. Der Einstiegspunkt unserer Web-Anwendung ist der sogenannte HomeController. Dies ist der Controller, der zum Zuge kommt, sofern die anderen beiden Controller eine Interaktion oder Controller-Aufrufe mit gewissen Parametern nicht unterstützen.



AccountController

AdminController

6.2 Benutzeroberfläche

6.3 Admin Spezifisches ?!

7 Client

Der Client ist eine Android-Applikation für den Endbenutzer. Es handelt sich hierbei um einen Navigator. Wie jedes handelsübliche Navigationsgerät beinhaltet auch unsere Applikation eine Karte, in unserem Fall ein Gebäudeplan, da wir uns in unserem Projekt mit Indoor-Navigation beschäftigen. Darüber hinaus kann man einen Zielpunkt wählen und mit Hilfe eines QR-Codes, NFC-Tags oder des Wlans seine Position bestimmen. Sind Start und Zielpunkt bekannt, berechnet der Server eine Route und teilt diese dem Client mit, welcher diese daraufhin graphisch auf der Karte zur Anzeige bringt.

Des Weiteren gibt es Suchfunktionen und Auflistungen von besonders interessanten Orten (Points of Interest). Zur Benutzung der Wlan-Positionserkennung ist eine Registrierung und Anmeldung an unserem Server von Nöten.

7.1 Allgemeine Struktur

Die Navigator-Applikation wurde wie auch der Collector in Android umgesetzt. Es wird mindestens die Version 4.1 vorausgesetzt. Es galt folgende Hauptaufgaben zu erfüllen:

7.1.1 Positionserkennung

Wie bereits im Kapitel ?? beschrieben, haben wir uns für Positionserkennung anhand von QR-Codes, NFC-Tags und Wlan-Fingerprinting entschieden. Für die Positionserkennung mittels NFC und Wlan greifen wir auf unsere Android-Bibliothek der Kernfunktionalitäten zurück. Während für die Interpretation des QR-Codes eine Fremd-Applikation zum Einsatz kommt. Diese muss bei der ersten Benutzung gegebenenfalls installiert werden.

Die letztendlich ermittelten Daten interpretieren wir mit Hilfe des Webservices, der wiederum Bestandteil unserer Kernfunktionalitäten ist.

7.1.2 Anzeige und Navigation auf einer Karte

Wir haben uns für die Benutzung einer freien Bibliothek für Karten entschieden. Nach intensiver Recherche fiel unsere Wahl dabei auf die Javascript Bibliothek d3 von [. Das hat zur Folge, dass unsere Karte lediglich in einer Website platziert ist,](#)

Infos zur
Bibliothek



die mittels einer WebView zur Anzeige gebracht wird. Dies hat für uns und den Endbenutzer den größten Vorteil, dass Änderung an der Karte nicht ein Update der Applikation nach sich zieht. Mittels einer definierten Schnittstelle lässt sich das Javascript aus unserer Android-Applikation heraus bedienen, so dass beispielsweise Positionsermittlungen automatisch auf der Karte nachgehalten werden. Dadurch ist die Navigation bei gewählten Zielpunkt vollständig.

7.1.3 Visuell ansprechende und intuitiv bedienbare Applikation

Eine Applikation sollte heute intuitiv bedienbar, übersichtlich und ansprechend sein, damit sie gerne und viel benutzt wird. Wir bauen dabei auf moderne Möglichkeiten des Android Betriebssystems. Im folgenden Kapitel werden wir diese näher erläutern.

7.2 Benutzeroberfläche

Neben der bereits erwähnten WebView mit einer hellschwarzen Karte auf Basis der d3 Bibliothek, ist unsere gesamte Applikation in dunklen Tönen gehalten und bietet klare Linien bei einer Highlight-Farbe die dem Grün der Westfälischen Hochschule sehr nahe kommt. Im Vordergrund der Applikation steht selbstverständlich die Karte, während grundlegende Funktionen wie die Suche oder der Aufruf des QR-Code-Scanners in der Actionbar geboten werden. Weitergreifende Funktionen wie das Wechseln der Ebene oder Anmelden befinden sich in einem Drawer auf der linken Seite der Applikation, der auf Wunsch in das Bild hinein gezogen werden kann.

Zusätzliche Fenster wie z.B. die Auflistung der Points of Interest werden grundsätzlich in Dialogfenstern zur Anzeige gebracht, was der übersichtlich und Navigation durch die Applikation zu Gute kommt. Die Suche ist ein besonderes Event, welches in der Actionbar ausgeführt wird und dort die Anzeige verändert, sodass grundsätzlich Ordnung in der Applikation herrscht. Neben dem intuitiven Design der Applikation ist selbstverständlich auch die Bedienung der Karte äußerst benutzerfreundlich. Neben den bekannten Möglichkeiten des Multitouch, beispielsweise zum Zoomen, ist auch die Steuerung der Navigation selbsterklärend. Einen gewünschten Punkt angeklickt, schon kann es los gehen. Optisch ansprechend wird eine Route eingezeichnet und mit der Zielflagge gekennzeichnet.



8 Fazit

Im Verlauf unseres Projektes ist deutlich geworden, dass die Navigation und die Lokalisierung innerhalb von Gebäuden ein schwieriges Thema ist. Daher wählten wir für das Problem der Lokalisierung gleich mehrere Ansätze. Jedoch mussten wir erkennen, dass sowohl die Positionierung mittels Texterkennung der Raumschilder, als auch die Positionsermittlung über WLAN Finger Prints für unsere Anforderungen nicht bzw. nicht gut genug funktioniert haben. Aus diesem Grund beschäftigten wir uns mit alternativen Möglichkeiten und entschieden uns dafür die Positionsdaten auf NFC Tags und QR Codes zu speichern und diese im Gebäude zu platzieren. So ist die zuverlässige Positionsbestimmung immer durch Scannen eines NFC Tags oder eines QR Codes möglich.

Bei der Umsetzung dieser Ideen haben wir uns für ein Backend basierend auf Microsoft Technologien entschieden. Für die Entwicklung der Datenbankstruktur verwendeten wir das Entity Framework, mit dem die Datenbank automatisch aus unserem Datenmodell generiert werden konnte. Zusätzlich konnten komplexe Datenbankabfragen einfach umgesetzt und ausgewertet werden. Dadurch haben wir gerade zu Beginn des Projektes eine Menge Arbeit und Zeit gespart. Des Weiteren haben wir uns bei der administrativen Oberfläche für eine ASP.NET Webapplikation entschieden, in der wesentliche Bestandteile der Benutzeroberfläche und eine Benutzerverwaltung bereits integriert waren. Auch dadurch haben wir uns viel Arbeit erspart und konnten uns auf die Wesentlichen Probleme konzentrieren.

Begeistert von diesen vielen Möglichkeiten entschieden wir uns unsere Anwendung in der Windows Azure Cloud zu hosten und meldeten daher einen entsprechenden Studenten Account bei Microsoft an. Leider erhielten wir über Wochen kein eindeutiges Feedback von Microsoft weshalb wir uns letztendlich für einen eigenen Windows Server innerhalb der Hochschule entschieden haben.

Abschließend blicken wir auf ein komplexes Projekt mit vielen Herausforderungen zurück. Durch die unterschiedlichen Technologien haben wir alle etwas Neues kennengelernt und weitere wichtige Erfahrung sammeln können. Innerhalb des Projekts gab es allerdings nicht nur technische Herausforderungen, auch die Projektorganisation selbst, sowie die Zusammenarbeit im Team ist in jedem Projekt eine Herausforderung. Gemeinsam haben wir es, trotz der vielen Schwierigkeiten, geschafft unser Projektziel zu erreichen. So sind insgesamt drei Anwendungen zur Navigation innerhalb unserer Hochschule entstanden, die mit entsprechenden administrativen Aufwand auch wirklich eingesetzt werden konnten.



9 Ausblick



10 Domänenmodell

Durch das Domänenmodell legen wir Begriffe fest, mit denen die Kommunikation im Projektteam einfacher wird.

10.1 Architektur

10.1.1 Webservice (StudMap.Service)

Stellt Funktionen zur Ablage und Abfrage von Navigationsinformationen und Benutzerdaten öffentlich bereit.

10.1.2 Admin-Oberfläche (StudMap.Admin)

Weboberfläche zum Anlegen, Bearbeiten von Navigationsinformationen und Benutzerdaten. Die Weboberfläche kann nur von der Benutzerrolle Administrator bedient werden.

10.1.3 Navigations-Client (StudMap.Navigator)

App zur Anzeige von Karten und Navigation zwischen Wegpunkten. Wird bedient durch den Anwender.

10.1.4 Sammler-Client (StudMap.Collector)

App zur Eingabe von Navigationsinformationen. Wird bedient durch Administratoren.

10.2 Benutzerrollen

10.2.1 Anwender (User)

Muss identifiziert sein und verwendet den Navigations-Client.



10.2.2 Administrator

Verwendet Admin-Oberfläche und den Sammler-Client. Muss registriert sein.

10.3 Begriffe

10.3.1 Karte (Map)

Beschreibt das gesamte Gebäude mit allen Stockwerken.

10.3.2 Stockwerk (Floor)

2-dimensionale Ansicht mit allen Layern der Ebene.

10.3.3 Schicht (Layer)

Es gibt mehrere Schichten, die jeweils Detailinformationen zu einem Stockwerk enthalten.

- Bild-Layer: Enthält grafische Darstellung des Stockwerks.
- Graph-Layer: Enthält Kanten und Knoten für Routen.
- POI-Layer: Zusatzinformationen zu speziellen Orten.
- Routen-Layer: Darstellung grafischer Elemente zur Navigation.
- Personen-Layer: Darstellung anderer Anwender.

10.3.4 Route

Hat einen Start- und einen Endknoten. Verbindet diese beiden Knoten über Zwischenknoten und Kanten.

10.3.5 Graph

Gesamtheit aller Knoten und Kanten der Karte (Stockwerk-übergreifend).



10.3.6 Knoten (Node)

Besteht aus eindeutigem Identifier, X- und Y-Koordinate und Stockwerk. Zu dem Knoten können zusätzliche Informationen hinterlegt werden: Name, Raumnummer, NFC-Tag, QR-Tag, ..., Verweis auf POI.

10.3.7 Kante (Edge)

Verbindung zweier Knoten. Bedeutet, dass man von einem Punkt zum anderen laufen kann.

10.3.8 Point of Interest (POI)

Ort besonderen Interesses (z.B. Bibliothek, Mensa, ...)

10.4 Recherche

Für die Navigation innerhalb von Gebäuden konnten wir nicht auf die gängigen Standards zurückgreifen, sondern mussten uns andere Wege überlegen, wie wir die Nutzer innerhalb der Gebäude lokalisieren.

Außerhalb von Gebäuden ist die Lokalisierung mittels GPS sehr verbreitet und auch einfach und akkurat. Um eine ähnliche Lokalisierung unserer Nutzer zu ermöglichen haben wir uns die folgenden Möglichkeiten überlegt.

10.4.1 QR-Codes

QR-Codes sind weit verbreitet, einfach zu erstellen und mit vielen Geräten einzulesen. Dadurch bieten QR-Codes die Möglichkeit Informationen einfach an Orten anzubringen und von Maschinen einzulesen.

Wir haben uns zwei Konzepte überlegt, QR-Codes in unserem Projekt einzubauen. Dazu haben wir einmal die Möglichkeit betrachtet die Auswertung des QR-Codes am Server zu realisieren und mit der Möglichkeit verglichen die Auswertung direkt am Client des Nutzers zu implementieren.

10.4.1.1 Serverseitig

Für die serverseitige Umsetzung hat gesprochen, dass das Smartphone keine Rechenleistung benötigt um die QR-Codes zu dekodieren. Des Weiteren wird durch eine serverseitige Implementierung vermieden, dass der Nutzer weitere Apps auf seinem Smartphone installieren muss.

Für die Implementierung in den Webservice haben wir uns für die offene Bibliothek [ZXing.NET](#) benutzt. Allerdings ist uns bei der Implementierung und Testen der Bibliothek direkt aufgefallen, dass die Bilder zuvor am Smartphone verkleinert werden müssen um Bandbreite zu sparen und die Laufzeit der Bibliothek zu verringern. Dadurch wird, obwohl es ein Ziel der serverseitigen Umsetzung war, Rechenleistung benötigt. Darüber hinaus mussten wir feststellen, dass die Bibliothek keine zuverlässige Dekodierung der QR-Codes bietet.

10.4.1.2 Clientseitig

Im Gegensatz zur serverseitigen Umsetzung wird bei dieser Implementierung die gesamte Dekodierung am Client vorgenommen. Dadurch wird die Netzlast verringert, der Aufwand am Client aber erhöht.

Hier bot sich zum einen an eine Bibliothek in den Client aufzunehmen, oder eine

externe Anwendung zum Dekodieren der QR-Codes zu benutzen. Wir haben uns schlussendlich dazu entschieden

Entscheidung
beim QR-
Code
Reader
aufschrei-
ben und
warum

10.4.2 NFC-Tags

NFC ist eine Technologie, auf die wir im Alltag immer häufiger stoßen, sei es bei Werbung, Bezahl- oder Ticketsystemen. NFC steht für Near Field Communication und besteht aus zwei Komponenten, der passiven, dem NFC-Tag, und der aktiven Komponente, beispielsweise dem Smartphone.

Die als Datenspeicher fungierenden NFC-Tags werden immer preiswerter und sind zudem relativ klein, was eine Anbringung an den gewünschten Orten problemlos ermöglicht. Auf der anderen, der aktiven Seite stehen immer mehr Smartphones bereit, die diese Technologie unterstützen.

Die steigende Beliebtheit der NFC-Technologie verdankt diese dem Komfort. Wie der Name bereits aussagt, reicht schon die Nähe der aktiven Komponente zur passiven um Daten zu kommunizieren. Diesen Komfort bieten wir dem Benutzer, um seine Position dem Navigator mitzuteilen.

Des Weiteren lassen sich auf dem NFC-Tag zusätzliche Informationen hinterlegen, die Unwissende auf die Navigationsmöglichkeit aufmerksam machen.

10.4.2.1 Umsetzung

Der gesamte Prozess der Positionsermittlung findet clientseitig statt. Zum einen ist das Client-Gerät unumgänglich für die Kommunikation mit dem NFC-Tag und zum anderen sind die Datenmengen und der Aufwand der Interpretation sehr gering.

Die Android NFC API ermöglicht uns die native Umsetzung der Positionsermittlung für Android-Geräte.

10.4.3 OCR der Raumschilder

Anstatt QR- oder NFC-Tags an den Räumen der FH anzubringen, besteht die Möglichkeit die bereits angebrachten Türschilder zu verwenden. Hierzu ist eine Erkennung der Raumnummer auf dem Türschild notwendig. Die OCR (Optical Character Recognition) kann über eine Bibliothek sowohl auf Client-, als auch auf Serverseite erfolgen. Erfolgt die Erkennung auf dem Client, dann könnten ebenfalls bestehende Apps zur Texterkennung verwendet werden.



Abbildung 10.1: Gesamtes Bild Raumschild

10.4.3.1 Bibliothek tesseract

Tesseract⁵ ist eine native Bibliothek für die Erkennung von Text in Bilddateien. Es gibt sowohl für .NET als auch für Java (Android) entsprechende Wrapper, die von uns verwendet werden können. Bei den Tests zu der OCR-Bibliothek haben sich allerdings einige Schwächen gezeigt. Tesseract ist für die Texterkennung von gescannten Dokumenten gedacht und arbeitet deshalb nur stabil, wenn sich auf dem Bild ausschließlich Text befindet. Dies wird an folgenden Beispielbildern deutlich.

Auf dem Bild mit Raumschild und Wand wird nur unzuverlässig Text erkannt (siehe 10.1). Die Ergebnisse variieren von "Kein Text erkannt" bis hin zu "Buchstabensalat mit Raumnummer". Schneidet man den relevanten Teil per Hand aus (siehe 10.2), wird der Text einwandfrei erkannt.

Damit die Raumschilder zuverlässig erkannt werden, ist es notwendig, dass aufgenommene Bilder zunächst auf den Bereich mit der Raumnummer zugeschnitten werden. Dies erfordert einen hohen Entwicklungsaufwand.

⁵<https://code.google.com/p/tesseract-ocr/>

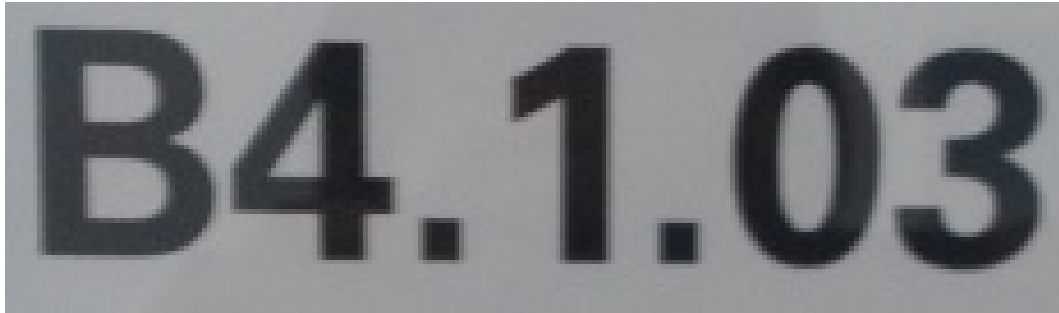


Abbildung 10.2: Ausschnitt Raumschild

10.4.3.2 App Google Goggles

Google Goggles⁶ ist eine Android-App, die zur Erkennung von Text, Symbolen und QR-Tags verwendet werden kann. Diese App lieferte in Tests auch bei suboptimalen Bildern gute Ergebnisse. Außerdem ist die App gut in das Android-Umfeld eingebettet und lässt sich leicht bedienen.

Allerdings gibt es für die Verwendung von Google Goggles noch keine öffentliche API⁷. Diese ist zwar von Google geplant, aber nie umgesetzt worden. Auch wenn die App eine komfortable Möglichkeit zur OCR-Erkennung bietet, kann diese ohne API nicht in unserem Projekt verwendet werden.

10.4.4 WLAN Fingerprinting

Bei der Positionierung des Nutzers mittels WLAN haben wir über eine für den Nutzer passive Positionierung recherchiert. Alle anderen Positionierungsmethoden benötigten eine Eingabe des Nutzers. Wir haben hier die Eingabe der Position auf einer Karte und das Einlesen von QR-Codes oder NFC-Tags behandelt. Die Positionierung ermöglicht es allerdings im Hintergrund zu laufen und ohne Eingabe des Nutzers die Position zu bestimmen.

WLAN ist zur Positionierung innerhalb von Gebäuden geeignet, da es zum einen eine weit verbreitete Infrastruktur ist, auf vielen mobilen Plattformen verfügbar ist, Wände durchdringt und Standard WLAN Access Points bereits eine Lokalisierung auf Raum-Genauigkeit ermöglicht.

Aus all diesen Gründen haben wir uns mit der Positionierung mittels WLAN beschäftigt.

⁶<https://play.google.com/store/apps/details?id=com.google.android.apps.unveil>

⁷<http://stackoverflow.com/questions/2080731/google-goggles-api>

10.4.4.1 Sammeln von WLAN Fingerprints

Um später Vergleiche im Client anstellen zu können mussten wir zuerst Daten des Netzwerkes sammeln. Ein Access Point wird dabei eindeutig durch eine **BSSID** gekennzeichnet und der Client gibt Auskunft über die empfangene Signalstärke (**RSS**⁸), welche beobachtet und aufgezeichnet werden kann.

Ziel dieser Phase war es an möglichst vielen Punkten in der Hochschule die *RSS* zu messen und diese zu einem Punkt auf der Karte der Hochschule zu speichern.



10.4.4.2 Kalibrierung

Da das Sammeln der WLAN Fingerprints mit einem Smartphone realisiert wird und wir davon ausgehen mussten, dass nicht jeder Nutzer das gleiche Smartphone besitzt, mussten wir uns eine Möglichkeit der Kalibrierung überlegen. Dazu haben wir überlegt, dass der Nutzer zuerst in einer Kalibrierungsphase selbst einen Fingerprint erstellt von einem von uns festgelegten Ort und diesen mit dem von uns gemessenen Fingerprint vergleicht. Dadurch bekommen wir einen Faktor um den das Smartphone des Nutzer von unserem Gerät abweicht. Da wir vermuten, dass die WLAN Antennen der Smartphones auch in verschiedenen Bereichen, hohe, mittlere und niedrige Signalstärke, sich stark unterscheiden berechnen wir diesen Faktor für die gerade genannten Bereiche.

Dieser Teilbereich ist in der *StudMap-App* umgesetzt.

10.4.4.3 Positionierung mittels WLAN Fingerprints

Um die Position eines Nutzers ermitteln zu können muss dieser, wie der *Collector* einen Fingerprint des WLANs an seiner aktuellen Position erstellen. Diesen Fingerprint und seine Faktoren, welche während der Kalibrierung ermittelt wurden, schickt

⁸RSS: received signal strength wird in dBm gemessen.



10 Domänenmodell

der Client zum Server, welcher durch Vergleiche den Standpunkt ermittelt und zurückgibt.

Dieses Feature ist auch in der *StudMap-App* umgesetzt.

Wie werden die FP verglichen.

11 Positionsermittlung

In diesem Kapitel wird beschrieben, wie die Position eines Anwenders auf der Karte bestimmt wird. Dazu werden die im Kapitel [Recherche](#) beschriebenen Verfahren verwendet.

11.1 QR-Tags

An den Räumen werden QR-Tags angebracht, die eine Zuordnung zu einem Knoten auf der ermöglicht. Hier ist zu beachten, dass die Informationen auf dem QR-Tag auch für andere Anwendungen nützlich sein sollen. Deshalb kann hier nicht nur eine Knoten-ID hinterlegt werden.

Folgendes JSON-Format ist ein möglicher Kandidat:

```
1 {  
2   "General": {  
3     "Label": "A2.1.10",  
4     "Name": "Aquarium"  
5   },  
6   "StudMap": {  
7     "NodeId": "12",  
8     "Url": "https://code.google.com/p/studmap/"  
9   }  
10 }
```

11.2 NFC-Tags

Neben den QR-Tags werden auch NFC-Tags an den Räumen angebracht. Auf diese werden aktuell keine Informationen geschrieben. Die Knoten werden über die NFC-ID des Chips zugeordnet.

Um Benutzer, die die StudMap-App nicht installiert haben zu informieren wird eine URL auf den NFC-Tags gespeichert. Diese leitet auf eine Seite mit Informationen über den gescannten Raum und einen Link auf unsere Projektseite. In Zukunft kann hier auch ein Link auf den Google Play Store hinterlegt werden, um eine einfache Installation zu ermöglichen.

Die URL sieht z.B. so aus (für das Aquarium):

```
1 http://193.175.199.115/StudMapAdmin/Admin/NodeInfo?nodeId=847
```

12 Datenbank

Die Daten für StudMap werden in einer zentralen Datenbank gespeichert. In diesem Kapitel werden die einzelnen Tabellen thematisch gruppiert und Besonderheiten erläutert.

12.1 Maps

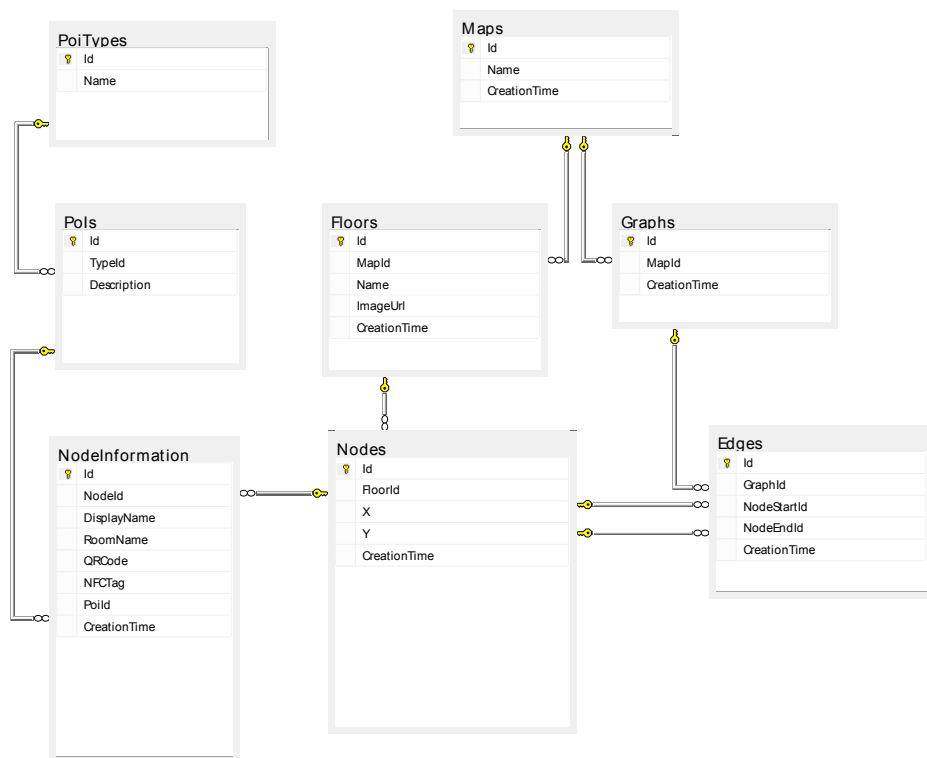


Abbildung 12.1: Datenbankmodell für die Kartenobjekte

12.1.1 Maps

Für jede Karte wird ein Eintrag in dieser Tabelle erzeugt. Zu jeder Karte wird ein frei vergebener Name und der Erstellungszeitpunkt gespeichert.



Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Karte
Name	NVARCHAR(255)	Name der Karte
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.2 Floors

Für jedes Stockwerk wird ein Eintrag in dieser Variable angelegt. Zu jedem Stockwerk wird ein frei vergebenen Name, eine URL auf ein Bild des Stockwerks und ein Erstellungszeitpunkt gespeichert. Ein Stockwerk ist genau einer Karte zugeordnet

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des Stockwerks
MapId	INTEGER (FK)	ID der zugeordneten Karte
Name	NVARCHAR(255)	Name des Stockwerks
ImageUrl	NVARCHAR(MAX)	URL des Bilds
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.3 Graphs

Ein Graph beschreibt die Knoten- und Kantenstruktur auf einer Karte. Dazu werden alle Kanten mit dem Graphen verknüpft. Über die Kanten sind auch die Knoten mit dem indirekt verknüpft.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des Graphen
MapId	INTEGER (FK)	ID der zugeordneten Karte
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.4 Edges

Eine Kante verknüpft zwei Knoten in einem Graphen.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Kante
GraphId	INTEGER (FK)	ID der zugeordneten Graphen
NodeStartId	INTEGER (FK)	ID des Startknotens
NodeEndId	INTEGER (FK)	ID des Endknotens
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.5 Nodes

Die Position eines Knoten wird durch die Zuordnung zu einem Stockwerk und seine X/Y-Koordinaten auf diesem Stockwerk bestimmt. Außerdem wird der Erstellungszeitpunkt eines Knotens gespeichert.

Die X- und Y-Koordinaten werden im Bereich 0.0 bis 1.0 gespeichert. Dabei bedeutet 0.0 ganz links (X) oder ganz oben (Y) und 1.0 ganz rechts (X) bzw. ganz unten (Y) auf dem Bild des Stockwerks.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knotens
FloorId	INTEGER (FK)	ID der zugeordneten Stockwerks
X	DECIMAL(18,17)	X-Koordinate auf dem Stockwerk
Y	DECIMAL(18,17)	Y-Koordinate auf dem Stockwerk
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.6 NodeInformation

Zu einem Knoten können noch weitere Informationen hinterlegt werden. Diese sind optional und werden nur zu wichtigen Knoten wie Seminarräumen, Büros und Toiletten. Über die Knoteninformationen kann auch ein PoI mit dem Knoten verknüpft werden.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knoteninformation
NodeId	INTEGER (FK)	ID der zugeordneten Knotens
DisplayName	NVARCHAR(50)	Name der angezeigt werden soll
RoomName	NVARCHAR(255)	Offizieller Raumname (z.B. B4.0.1.11)
QRCode	NVARCHAR(255)	Hinterlegter QR-Code
NFCTAG	NVARCHAR(50)	Hinterlegtes NFC-Tag
PoiId	INTEGER (FK)	Optionalen zugeordneter PoI
CreationTime	DATETIME	Erstellungszeitpunkt

12.1.7 Pols

Ein PoI (Point of Interest) kategorisiert für den Benutzer relevante Knoten. Hier kann z.B. nach Dozentenbüros, Mensa, Bibliothek und Toiletten gefiltert werden.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID der Knotens
TypeId	INTEGER (FK)	ID des PoI-Typs
Description	NVARCHAR(MAX)	Zusätzliche Beschreibung des PoIs

12.1.8 PoiTypes

Diese Tabelle enthält die möglichen Typen von PoIs.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des PoI-Typs
Name	NVARCHAR(255)	Name des PoI-Typs

12.2 Users

ä

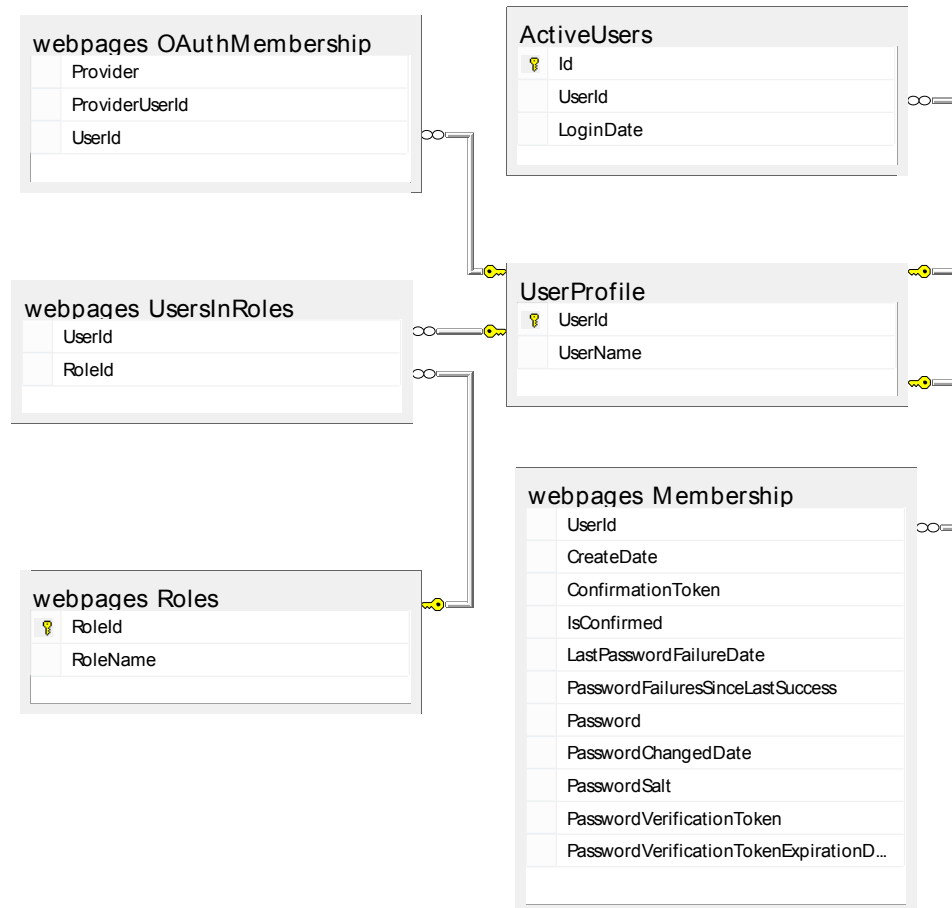


Abbildung 12.2: Datenbankmodell für Benutzerobjekte

13 Benutzerverwaltung

In einem ASP.NET MVC 4 Projekt ist bereits eine vollständige Benutzerverwaltung integriert, die wir auch in unserem Projekt benutzen wollen. Durch die integrierte Benutzerverwaltung sind Webseiten zur Registrierung und für den Login / Logout bereits fertig.

Abbildung 13.1: Webseite zur Registrierung im StudMap Admin

Für die Benutzerverwaltung verwendet das ASP.NET MVC 4 Projekt folgende Datenbankstruktur:

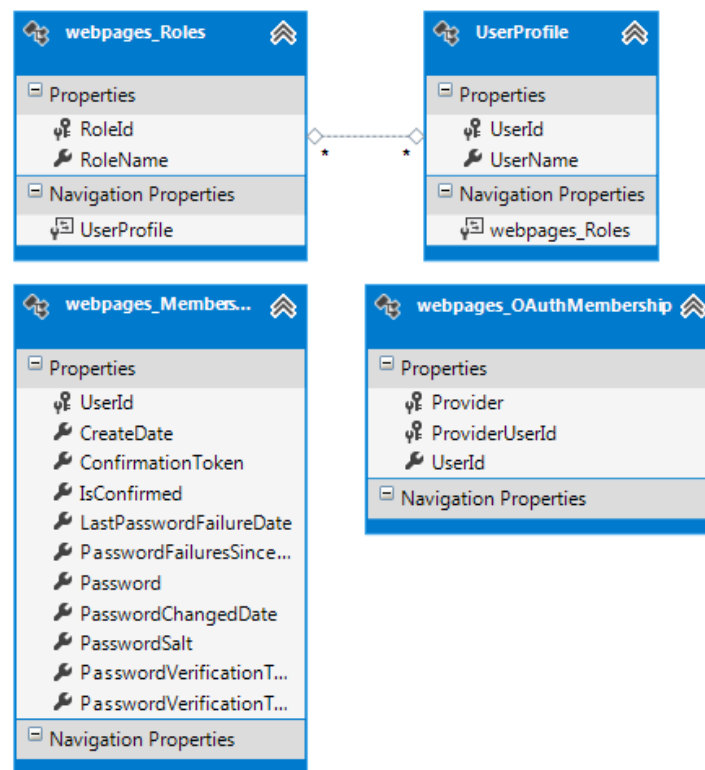
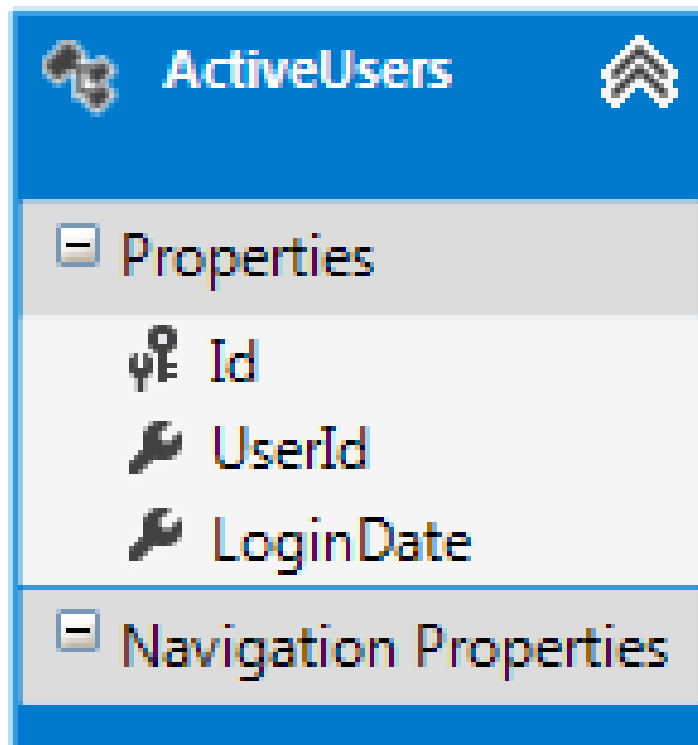


Abbildung 13.2: Datenbankstruktur der integrierten Benutzerverwaltung

Für unser Projekt sind nur die drei Tabellen `UserProfile`, `webpages_Roles` und `webpages_Membership` relevant. Wie im Domain Model bereits beschrieben unterscheiden wir zwischen den Benutzerrollen Benutzer und Administrator. Jeder Anwender kann sich in mehreren Benutzerrollen befinden. Zusätzlich sind in der Tabelle `webpages_membership` weitere Anwenderdaten wie beispielsweise das Datum der Registrierung das Passwort hinterlegt. Damit ist die Benutzerverwaltung für den Administrationsbereich vollständig.

Um die (beispielsweise über das Smartphone) am System angemeldeten Clients zu verwalten haben wir eine weitere Tabelle `ActiveUsers` hinzugefügt:



ActiveUsers	
Properties	
Id	
UserId	
LoginDate	
Navigation Properties	

Abbildung 13.3: Tabelle ActiveUsers

Für den StudMap Admin genügt die bereits integrierte Benutzerverwaltung. Allerdings benötigen wir noch eine Schnittstelle, damit auch die Web bzw. Smartphone Clients auf die Benutzerverwaltung zugreifen können. Siehe dazu Kapitel [Verwendung der Benutzerschnittstelle](#).



14 Webservice

Für den Zugriff auf die Daten stellen wir den Webservice `StudMap.Service` zur Verfügung. Der Webservice besteht dabei aus zwei Schnittstellen in Form von sogenannten Controller Klassen, die jeweils von der Klasse `ApiController`⁹ abgeleitet sind:

- `MapsController`: Verwaltung von Karten- und Routeninformationen
- `UsersController`: Verwaltung von Benutzerinformationen

Bevor nun die Funktionen der jeweiligen Controller Klasse erläutert werden, folgt eine Übersicht, über die verschiedenen Rückgabe Werte und ihre Bedeutung.

14.1 Allgemeine Objekte

Im folgenden werden die im Webservice verwendeten Objekte aufgeführt. Für jedes Objekt werden Eigenschaften und die Repräsentation der Daten im JSON-Format aufgelistet.

14.1.1 Edge

Repräsentiert eine Kante in einem Graphen.

Eigenschaften:

StartNodeId	ID des Start-Knotens der Kante.
EndNodeId	ID des End-Knotens der Kante.

Beispiel:

```
1 {  
2   "StartNodeId": 12,  
3   "EndNodeId": 6  
4 }
```

⁹siehe: [MSDN Dokumentation](#)

14.1.2 Node

Repräsentiert einen Knoten in einem Graphen.

Eigenschaften:

Id	ID des Knotens.
X	X-Koordinate auf dem Bild des Stockwerks. Wertebereich: 0.0 - 1.0. 0.0 bedeutet linker Bildrand. 1.0 bedeutet rechter Bildrand.
Y	Y-Koordinate auf dem Bild des Stockwerks. Wertebereich: 0.0 - 1.0. 0.0 bedeutet oberer Bildrand. 1.0 bedeutet unterer Bildrand.
FloorId	ID des Stockwerks auf dem sich der Knoten befindet.
HasInformation	true, wenn es Raum- oder PoI-Daten zu dem Knoten gibt. Ansonsten false.

Beispiel:

```
1 {  
2   "Id": 12,  
3   "X": 0.45,  
4   "Y": 0.76,  
5   "FloorId": 2,  
6   "HasInformation": true  
7 }
```

14.1.3 Graph

Repräsentiert für ein Stockwerk den entsprechenden Teilgraphen. Eigenschaften:

FloorId	ID des Stockwerks, das der entsprechende Graph repräsentiert.
Edges	Eine Liste von Kanten (s. Edge).
Nodes	Eine Liste von Knoten (s. Node).

Beispiel:

```
1 {  
2   "FloorId": 12,  
3   "Edges": { {...}, {...}, {...} },  
4   "Nodes": { {...}, {...}, {...} }  
5 }
```

14.1.4 Pathplot

Repräsentiert für die Darstellung benötigte Daten.

Bitte noch
entspre-
chend
vervoll-
ständigen!

Eigenschaften:

Id	??
Classes	??
Points	Eine Liste von Node Objekten.

Beispiel:

```

1 {
2   "Id": "flt-1",
3   "Classes": "planned",
4   "Points": { {...}, {...}, {...} }
5 }
```

14.1.5 FloorPlanData

Repräsentiert Daten die auf einem Bild dargestellt werden können.

Eigenschaften:

Pathplot	
Graph	

Beispiel:

```

1 {
2   "Pathplot": {...},
3   "Graph": {...}
4 }
```

Bitte noch
entspre-
chend
vervoll-
ständigen!

14.1.6 Room

Repräsentiert Daten die für einen Raum relevant sind.

Eigenschaften:

NodeId	ID des Knotens an dem sich der Raum befindet.
DisplayName	Der Anzeigename für den Raum (z.B. Aquarium)
RoomName	Der eigentliche Raumname (z.B. A 2.0.11).
FloorId	Id des Floors, auf welchen sich der Room befindet.

Beispiel:

```

1 {
2   "NodeId": 12,
3   "DisplayName": "Aquarium",
4   "RoomName": "A2.0.11",
5   "FloorId": 1
6 }
```


14.1.7 PoiType

Repräsentiert einen **Poi** Typen, wie beispielsweise Mensa oder Bibliothek.

Eigenschaften:

Id	ID des Poi Typs.
Name	Der Name des Poi Tpes (z.B. Mensa).

Beispiel:

```
1 {  
2   "Id": 12,  
3   "Name": "Mensa"  
4 }
```

14.1.8 Poi

Repräsentiert einen Point Of Interest, wie beispielsweise eine Mensa oder eine Bibliothek.

Eigenschaften:

Type	Typ des PoIs (s. PoiType).
Description	Beschreibung des PoIs.

Beispiel:

```
1 {  
2   "Type": 1,  
3   "Description": "In der Mensa kann man essen."  
4 }
```

14.1.9 RoomAndPoi

Enthält Informationen zu einem Raum und dem zugeordneten PoI.

Eigenschaften:

Room	Rauminformationen (s. Room).
Poi	PoI-Informationen (s. Poi).

Beispiel:

```
1 {  
2   "Room": {...},  
3   "Poi": {...}  
4 }
```

14.1.10 NodeInformation

Repräsentiert die für einen Knoten relevanten Daten.

Eigenschaften:

DisplayName	Anzeigename für den Knoten (z.B. Dr. Schulten, Martin).
RoomName	Raumname für den Knoten (z.B. B2.0.03).
Node	NodeInformation repräsentiert diesen Knoten.
PoI	PoI Informationen zu diesem Knoten.
QRCode	Dem Knoten zugeordnetem QR Code.
NFCTag	Dem Knoten zugeordnetem NFC Tag.

Beispiel:

```
1 {  
2   "DisplayName": "Dr. Schulten, Martin",  
3   "RoomName": "B2.0.03",  
4   "Node": {...},  
5   "PoI": {...},  
6   "QRCode": "...",  
7   "NFCTag": "..."  
8 }
```

14.1.11 QRCode

Repräsentiert die QRCode-Daten für den Knoten.

Eigenschaften:

General	Grundsätzliche Informationen zu einem Room.
StudMap	Allgemeine Informationen zum Projekt sind in StudMap hinterlegt.

Beispiel:

```
1 {  
2   "General": {...},  
3   "StudMap": {...}  
4 }
```

14.1.12 FullNodeInformation

Repräsentiert die für einen Knoten relevanten Daten.

Eigenschaften:

Map	Informationen zur gesamten Karte.
Floor	Repräsentiert die für dieses Stockwerk relevanten Daten.
Info	NodeInformation zum aktuellen Knoten.

Beispiel:

```
1 {  
2   "Map": {...},  
3   "Floor": {...},  
4   "Info": {...}  
5 }
```

14.1.13 Floor

Repräsentiert die für ein Stockwerk relevanten Daten.

Eigenschaften:

Id	ID des Stockwerks.
MapId	ID der Karte zu dem das Stockwerk gehört.
Name	Name des Stockwerks.
ImageUrl	Der Dateipfad auf dem Server zum Bild des Stockwerks.
CreationTime	Zeitstempel, an dem das Stockwerk erstellt wurde.

Beispiel:

```
1 {  
2   "Id": 1011,  
3   "MapId": 3,  
4   "Name": "Ebene 0",  
5   "ImageUrl": "Images/Floors/RN_Ebene_0.png",  
6   "CreationTime": "2013-11-18 14:36:24.607"  
7 }
```

14.1.14 Map

Repräsentiert die für eine Karte relevanten Daten.

Eigenschaften:

Id	ID der Karte.
Name	Name der Karte.

Beispiel:

```
1 {  
2   "Id": 3,  
3   "Name": "Westfälische Hochschule",  
4 }
```

14.1.15 User

Repräsentiert die für einen Benutzer relevanten Daten.

Eigenschaften:

Name	Name des Benutzers.
------	---------------------

Beispiel:

```
1 {  
2   "Name": "Daniel",  
3 }
```

14.1.16 SaveGraphRequest

Repräsentiert die Änderungen an dem Teilgraph für ein Stockwerk.

Eigenschaften:

FloorId	ID des Stockwerks.
NewGraph	Der hinzugefügte Teilgraph (s. Graph).
DeletedGraph	Der gelöschte Teilgraph (s. Graph).

Beispiel:

```
1 {  
2   "FloorId": 2,  
3   "NewGraph": {...},  
4   "DeletedGraph": {...}  
5 }
```

14.2 Rückgabe Objekte

14.2.1 BaseResponse

Allgemeine Rückgabe vom Service, die einen Status und ggf. einen Fehler enthält.
Die Daten werden im JSON Format zurück gegeben.

Beispiel:

```
1 {  
2   "Status":1,  
3   "ErrorCode":0  
4 }
```

14.2.1.1 ResponseStatus

Wir unterscheiden zwischen:

- `None` = 0: Defaultwert
- `Ok` = 1: Funktion erfolgreich ausgeführt
- `Error` = 2 Fehler bei Funktionsausführung

14.2.1.2 ResponseError

Ist beim `ResponseStatus` `Error` gesetzt. Es werden folgende Fehlerszenarien unterschieden:

Allgemein:

- 001 - `DatabaseError`:
Fehler bei der Ausführung einer Datenbankabfrage.
- 002 - `Unknown`:
Unbekannter Fehler.

Alle genauen Fehler-typen beschreiben.

Registrierung:

- 101 - `UserNameDuplicate`:
Der Benutzername ist bereits vergeben.
- 102 - `UserNameInvalid`:
Der Benutzername ist ungültig.
- 103 - `PasswordInvalid`:
Das Passwort ist ungültig.

Anmeldung:

- 110 - `LoginInvalid`:
Die Logindaten (Name oder Passwort) sind ungültig.

Maps:

- 201 - `MapIdDoesNotExist`:
Zur angeforderten `MapId` existiert keine Map.
- 202 - `FloorIdDoesNotExist`:
Zur angeforderten `FloorId` existiert kein Floor.
- 203 - `NodeIdDoesNotExist`:
Zur angeforderten `NodeId` existiert kein Node.

Navigation:

- 301 - NoRouteFound:
Es konnte keine Route gefunden werden.
- 302 - StartNodeNotFound:
Der angegebene Startknoten existiert nicht.
- 303 - EndNodeNotFound:
Der angegebene Endknoten existiert nicht.

Information:

- 401 - PoiTypeIdDoesNotExist:
Zur angegebenen PoiTypeId existiert kein PoiType.
- 402 - NFCTagDoesNotExist:
Das angegebene NFC-Tag wurde nicht gefunden.
- 403 - QRCodeDoesNotExist:
Der angegebene QR-Code wurde nicht gefunden.
- 404 - PoiDoesNotExist:
Zur angegebenen PoiId existiert kein Poi.
- 405 - QRCodeIsNullOrEmpty:
Es wurde kein QR-Code angegeben.
- 406 - NFCTagIsNullOrEmpty:
Es wurde kein NFC-Tag angegeben.
- 407 - NFCTagAlreadyAssigned:
Das NFC-Tag ist bereits einem anderen Knoten zugeordnet.

14.2.2 ObjectResponse

Eine generische Klasse die `BaseResponse` um ein Feld `Object` erweitert, indem die Nutzdaten gespeichert werden.

Beispiel:

```
1 {  
2   "Status": 1,  
3   "ErrorCode": 0,  
4   "Object": {...}  
5 }
```



14.2.3 ListResponse

Eine generische Klasse die **BaseResponse** um eine Liste **List** erweitert, indem Nutzdaten in Form einer Collection gespeichert werden.

Beispiel:

```
1 {  
2   "Status": 1,  
3   "ErrorCode": 0,  
4   "List": [...]  
5 }
```

14.3 MapsController

14.3.1 CreateMap

Erstellt eine neue Karte mit dem vorgegebenen Namen.

POST [/api/Maps/CreateMap?mapName=WHS](#)

Parameter:

mapName	Sprechender Name der Karte.
---------	-----------------------------

Rückgabewert: **ObjectResponse**, **Map**

14.3.2 DeleteMap

Löscht eine Karte mit der angegebenen ID.

POST [/api/Maps/DeleteMap?mapId=2](#)

Parameter:

mapId	ID der Map, die gelöscht werden soll.
-------	---------------------------------------

Rückgabewert: **BaseResponse**

14.3.3 GetMaps

Liefert eine Liste aller Karten zurück.

GET [/api/Maps/GetMaps](#)

Rückgabewert: **ListResponse**, **Map**



14.3.4 CreateFloor

Erstellt einen Floor zu einer Map mit einem sprechenden Namen.

POST [/api/Maps/CreateFloor?mapId=2&name=Erdgeschoss](#)

Parameter:

mapId	ID der Map, zu der ein Floor angelegt wird.
name	Sprechender Name des Floors.

Rückgabewert: [ObjectResponse](#), [Floor](#)

14.3.5 DeleteFloor

Löscht einen Floor mit der angegebenen ID.

POST [/api/Maps/DeleteFloor?floorId=3](#)

Parameter:

floorId	ID des Floors, der gelöscht werden soll.
---------	--

Rückgabewert: [BaseResponse](#)

14.3.6 GetFloorsForMap

Liefert alle Stockwerke einer Karte.

GET [/api/Maps/GetFloorsForMap?mapId=2](#)

Parameter:

mapId	ID der Karte, von der die Stockwerke abgefragt werden sollen.
-------	---

Rückgabewert: [ListResponse](#), [Floor](#)

14.3.7 GetFloor

Liefert Informationen zu einem Stockwerk.

GET [/api/Maps/GetFloor?floorId=3](#)

Parameter:

floorId	ID des Stockwerks.
---------	--------------------

Rückgabewert: [ObjectResponse](#), [Floor](#)

14.3.8 GetFloorplanImage

Liefert die URL des Bilds zu einem Stockwerk.

GET </api/Maps/GetFloorplanImage?floorId=3>

Parameter:

floorId	ID des Stockwerks.
---------	--------------------

Rückgabewert: String

14.3.9 SaveGraphForFloor

Speichert den Graphen zu einem Stockwerk ab.

POST </api/Maps/SaveGraphForFloor>

POST Body: [SaveGraphRequest](#)

Rückgabewert: [ObjectResponse](#), [Graph](#)

14.3.10 DeleteGraphForFloor

Löscht den Teilgraphen auf einem Stockwerk. Die Teilgraphen auf anderen Stockwerken werden nicht verändert. Kanten die Stockwerke mit diesem Stockwerk verbinden, werden ebenfalls entfernt.

POST </api/Maps/DeleteGraphForFloor?floorId=2>

Parameter:

floorId	ID des Stockwerks, dessen Teilgraph gelöscht werden soll.
---------	---

Rückgabewert: [BaseResponse](#)

14.3.11 GetGraphForFloor

Liefert den Teilgraphen für ein Stockwerk.

GET </api/Maps/GetGraphForFloor?floorId=2>

Parameter:

floorId	ID des Stockwerks.
---------	--------------------

Rückgabewert: [ObjectResponse](#), [Graph](#)



14.3.12 GetFloorPlanData

Liefert die verschiedenen Schichten auf einem Stockwerk.

GET </api/Maps/GetFloorPlanData?floorId=2>

Parameter:

floorId	ID des Stockwerks.
---------	--------------------

Rückgabewert: `ObjectResponse`, `FloorPlanData`

GetConnectedNo

14.3.13 GetRouteBetween

Liefert die Route zwischen zwei Knoten, wenn diese existiert.

GET </api/Maps/GetRouteBetween?mapId=2&startNodeId=12&endNodeId=46>

Parameter:

mapId	ID des Karte, auf der die Route bestimmt werden soll.
startNodeId	ID des Startknotens.
endNodeId	ID des Zielknotens.

Rückgabewert: `ListResponse`, `Node`

14.3.14 GetNodeInformationForNode

Liefert weitere Informationen zu einem Knoten. Diese umfassen Raumnummern, zugeordnete NFC- und QR-Tags, usw.

GET </api/Maps/GetNodeInformationForNode?nodeId=12>

Parameter:

nodeId	ID des Knotens.
--------	-----------------

Rückgabewert: `ObjectResponse`, `Graph`

GetNodeInformationForNode
GetFull-NodeInformationForNode



14.3.15 SaveNodeInformation

Speichert zusätzliche Informationen zu einem Knoten ab.

POST </api/Maps/SaveNodeInformation>

POST Body: [NodeInformation](#)

Rückgabewert: [ObjectResponse](#), [NodeInformation](#)

14.3.16 GetPoiTypes

Liefert eine Liste aller Typen von PoIs zurück.

GET </api/Maps/GetPoiTypes>

Rückgabewert: [ListResponse](#), [PoiType](#)

14.3.17 GetPolsForMap

Liefert eine Liste aller PoIs auf einer Karte zurück.

GET </api/Maps/GetPoIsForMap?mapId=2>

Parameter:

mapId	ID der Karte.
-------	---------------

Rückgabewert: [ListResponse](#), [RoomAndPoi](#)

14.3.18 GetRoomsForMap

Liefert eine Liste aller Räume auf einer Karte zurück.

GET </api/Maps/GetRoomsForMap?mapId=2>

Parameter:

mapId	ID der Karte.
-------	---------------

Rückgabewert: [ListResponse](#), [Room](#)

14.3.19 GetNodeForNFC

Sucht auf einer Karte nach einem Knoten mit einem bestimmten NFC-Tag.

GET </api/Maps/GetNodeForNFC?mapId=2&nfcTag=46A6CG739ED9>

Parameter:

mapId	ID der Karte.
nfcTag	NFC-Tag, nach dem gesucht werden soll.

Rückgabewert: **ObjectResponse**, **Node**

SaveNFCForNode

14.3.20 GetNodeForQRCode

Sucht auf einer Karte nach einem Knoten mit einem bestimmten QR-Code.

GET </api/Maps/GetNodeForQRCode?mapId=2&qrcode=46A6CG739ED9>

Parameter:

mapId	ID der Karte.
qrCode	QR-Code, nach dem gesucht werden soll.

Rückgabewert: **ObjectResponse**, **Node**

SaveQRCodeForNode

14.4 UsersController

Der **UserController** stellt klassische Funktionen zur Benutzerverwaltung zur Verfügung.

14.4.1 Register

Registriert einen neuen Anwender in der Benutzerrolle Benutzer.

POST </api/Users/Register?userName=test&password=geheim>

Parameter:

userName	Der Benutzername.
password	Passwort im Klartext.

Rückgabewert: **BaseResponse**



14.4.2 Login

Meldet einen bereits registrierten Anwender am System an.

POST [/api/Users/Login?userName=test&password=geheim](#)

Parameter:

userName	Der Benutzername.
password	Passwort im Klartext.

Rückgabewert: [BaseResponse](#)

14.4.3 Logout

Meldet einen angemeldeten Anwender vom System ab.

GET [/api/Users/Logout?userName=test](#)

Parameter:

userName	Der Benutzername.
----------	-------------------

Rückgabewert: [BaseResponse](#)

14.4.4 GetActiveUsers

Ermittelt eine Liste der aktuell am System angemeldeten Anwender.

GET [/api/Users/GetActiveUsers](#)

Rückgabewert: [ListResponse](#), [User](#)



15 Verwendung des Webservices

15.1 Verwendung der Benutzerschnittstelle

15.1.1 Registrierung

Bevor sich ein Benutzer am StudMap System anmelden kann, muss er sich zunächst über die Funktion **Register** registrieren.

15.1.2 Aktive und inaktive Benutzer

Im StudMap System wird zwischen aktiven und inaktiven Benutzern unterschieden. Nachdem sich ein Benutzer am System registriert hat gilt dieser als inaktiv. Über die Funktion **Login** kann er sich am System anmelden und gilt somit als aktiv.

Damit der angemeldete Benutzer auch aktiv bleibt, sollte sich dieser in einem Zeitintervall von fünf Minuten über die Methode **Login** am System aktiv melden. Nach einer Inaktivität von 15 Minuten wird der Benutzer automatisch inaktiv.

Über die Funktion **Logout** kann sich ein Benutzer wieder vom System abmelden und wird somit inaktiv.

15.1.3 Aktive Benutzer abfragen

Die aktiven Benutzer können über die Funktion **GetActiveUsers** abgefragt werden. Damit die Anzeige der aktiven Benutzer im Client möglichst aktuell ist, sollte diese Abfrage ebenfalls in regelmäßigen Zeitabständen erfolgen.

Best
Practices
"Programmier-
Handbuch" für
MapsCon-
troller
schreiben.



16 Maintenance Tool

Während der Entwicklung des StudMap-Projektes ist es uns aufgefallen, dass wir zur Wartung und Ausführung bestimmter Aufgaben ein einfaches Tool hilfreich wäre. Daher haben wir eine WPF-Anwendung entwickelt, welche die folgenden Aufgaben erledigt.

16.1 QR-Codes generieren

Eine Möglichkeit der Positionierung innerhalb des StudMap-Projektes ist das Einlesen von QR-Codes. In diesen QR-Codes stehen die im Kapitel [QR-Tags](#) beschriebenen Daten.

In der Anwendung werden alle [NodeInformation](#) ausgelesen und in einer Tabelle angezeigt. Anschließend kann in dieser Tabelle noch nach einem [Floor](#), oder dem Namen des Raums gefiltert werden.

Abschließend kann für alle ausgewählten Knoten ein QR-Code als PNG generiert werden. Zur Generierung der QR-Codes nutzen wir die Bibliothek [QrCode.Net](#).