

# 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

<b>1. Einleitung</b>	<b>1</b>
1.1. Projektorganisation . . . . .	1
<b>2. Architektur</b>	<b>2</b>
<b>3. Domänenmodell</b>	<b>4</b>
3.1. Architektur . . . . .	4
3.2. Benutzerrollen . . . . .	4
3.3. Begriffe . . . . .	5
<b>4. Datenbank</b>	<b>7</b>
4.1. Maps . . . . .	7
4.2. Users . . . . .	11
4.3. Views . . . . .	12
4.4. Stored Procedures . . . . .	13
<b>5. Service</b>	<b>14</b>
5.1. Allgemeine Struktur . . . . .	14
5.2. HTTP-Schnittstelle . . . . .	14
5.3. Assembly-Schnittstelle . . . . .	15
5.4. Caching . . . . .	15
5.5. Sicherheit . . . . .	15
<b>6. Collector</b>	<b>16</b>
6.1. Recherche . . . . .	16
6.2. Positionsermittlung . . . . .	21
6.3. Allgemeine Struktur . . . . .	22
6.4. Benutzeroberfläche . . . . .	22
6.5. Core Bibliothek . . . . .	23
<b>7. Admin</b>	<b>24</b>
7.1. Allgemeine Struktur . . . . .	24
7.2. Benutzeroberfläche . . . . .	26
7.3. Admin Spezifisches . . . . .	30
7.4. Maintenance Tool . . . . .	30
7.5. QR-Codes generieren . . . . .	30



<b>8. Client</b>	<b>32</b>
8.1. Allgemeine Struktur . . . . .	32
8.2. Benutzeroberfläche . . . . .	33
<b>9. Fazit</b>	<b>34</b>
<b>10. Ausblick</b>	<b>35</b>
<b>A. Benutzerverwaltung</b>	<b>36</b>
<b>B. Webservice</b>	<b>39</b>
B.1. Allgemeine Objekte . . . . .	39
B.2. Rückgabe Objekte . . . . .	45
B.3. MapsController . . . . .	48
B.4. UsersController . . . . .	53
<b>C. Verwendung des Webservices</b>	<b>55</b>
C.1. Verwendung der Benutzerschnittstelle . . . . .	55
<b>D. Admin-Bedienungsanleitung</b>	<b>56</b>
D.1. Registrieren . . . . .	56
D.2. LogIn / LogOut . . . . .	56
D.3. Passwort ändern . . . . .	57
D.4. Neue Map anlegen . . . . .	57
D.5. Neuen Floor anlegen . . . . .	57
D.6. Menühandhabung des Layers . . . . .	58

## 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

Die Architektur unseres Projekts sieht an der Benutzeroberfläche neben dem Navigations-Client einen Collector-Client und eine Admin-Oberfläche vor. Im Hintergrund werden die Informationen in einer Datenbank gespeichert und mit Hilfe eines Webservice bereitgestellt.

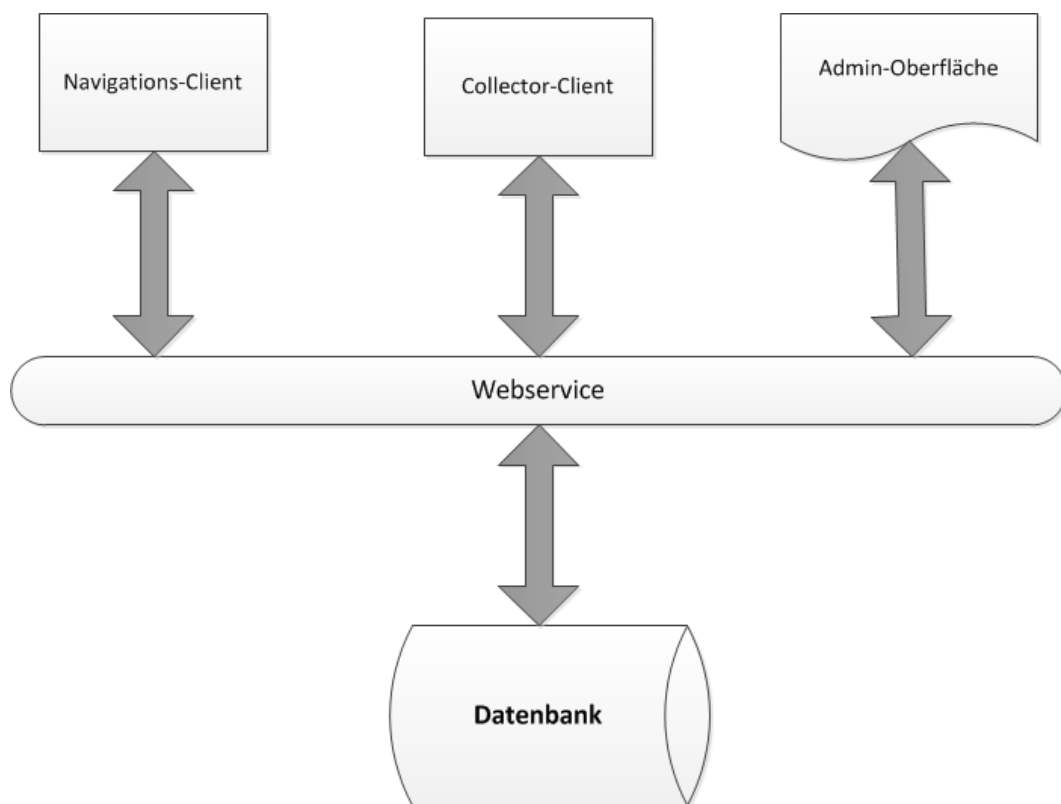


Abbildung 2.1.: Grafische Darstellung der Architektur

Der **Navigations-Client** ist die Bedienoberfläche des Benutzers. Hier wird die Karte des Gebäudes mit allen Wegpunkten und die Navigation angezeigt. Dem Benutzer werden zusätzlich zur Navigation Komfortfunktionen zur einfacheren und schnelleren Bedienung bereit gestellt.

Mit dem **Collector-Client** soll der Datenbestand des Projektes fortlaufend erweitert werden. Für die Knoten, bzw. Wegpunkte, können verschiedene, für die Navigation benötigte Informationen hinterlegt werden. Um fehlerhafte Eingaben zu vermeiden wird dieses Tool nicht vom Benutzer, sondern nur von Administratoren eingesetzt.



Die **Admin-Oberfläche** dient der Verwaltung von Karten und Benutzern. Der Administrator kann alte Karten bearbeiten oder neues Kartenmaterial einstellen. Zu diesen Karten wird hier auch der Graph mit allen Knoten und Kanten erstellt und anschließend verwaltet. Mit der Benutzerverwaltung können neue Benutzer angelegt und vorhandene bearbeitet werden oder aber alte Benutzerkonten gelöscht werden.

In der **Datenbank** werden sämtliche Informationen der Anwendung gespeichert. Sie enthält die Benutzerdaten und das gesamte Kartenmaterial. Das Kartenmaterial umfasst dabei die Kartengrafiken wie auch den zugehörigen Graphen. Über einen **Webservice** können die Daten abgerufen werden. Dieser stellt Funktionen zur Abfrage und Ablage von Navigations- und Benutzerdaten bereit.

Zu Beginn des Projekts war geplant das Projekt auf Microsoft's Cloud Platform *Windows Azure* zu hosten. Hier wäre eine gute Kompatibilität mit den von uns genutzten Technologien, sowie eine hohe Erreichbarkeit über eine öffentliche, feste IP-Adresse gegeben. Leider dauerte die Einrichtung für eine studentische Testphase seitens Microsoft zu lange. Alternativ wurde uns ein Server mit einer öffentlichen IP-Adresse an der Hochschule bereitgestellt auf welchem das System jetzt betrieben wird.



## 3. Domänenmodell

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

### 3.1. Architektur

#### 3.1.1. Webservice (StudMap.Service)

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

#### 3.1.2. Admin-Oberfläche (StudMap.Admin)

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

#### 3.1.3. Navigations-Client (StudMap.Navigator)

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

#### 3.1.4. Sammler-Client (StudMap.Collector)

App zur Eingabe von Navigationsinformationen. Wird bedient durch Administratoren.

### 3.2. Benutzerrollen

#### 3.2.1. Anwender (User)

Muss identifiziert sein und verwendet den Navigations-Client.



### 3.2.2. Administrator

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

## 3.3. Begriffe

### 3.3.1. Karte (Map)

Beschreibt das gesamte Gebäude mit allen Stockwerken.

### 3.3.2. Stockwerk (Floor)

2-dimensionale Ansicht mit allen Layern der Ebene.

### 3.3.3. 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.

### 3.3.4. Route

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

### 3.3.5. Graph

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





### 3.3.6. 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.

### 3.3.7. Kante (Edge)

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

### 3.3.8. Point of Interest (POI)

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

## 4. Datenbank

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

### 4.1. Maps

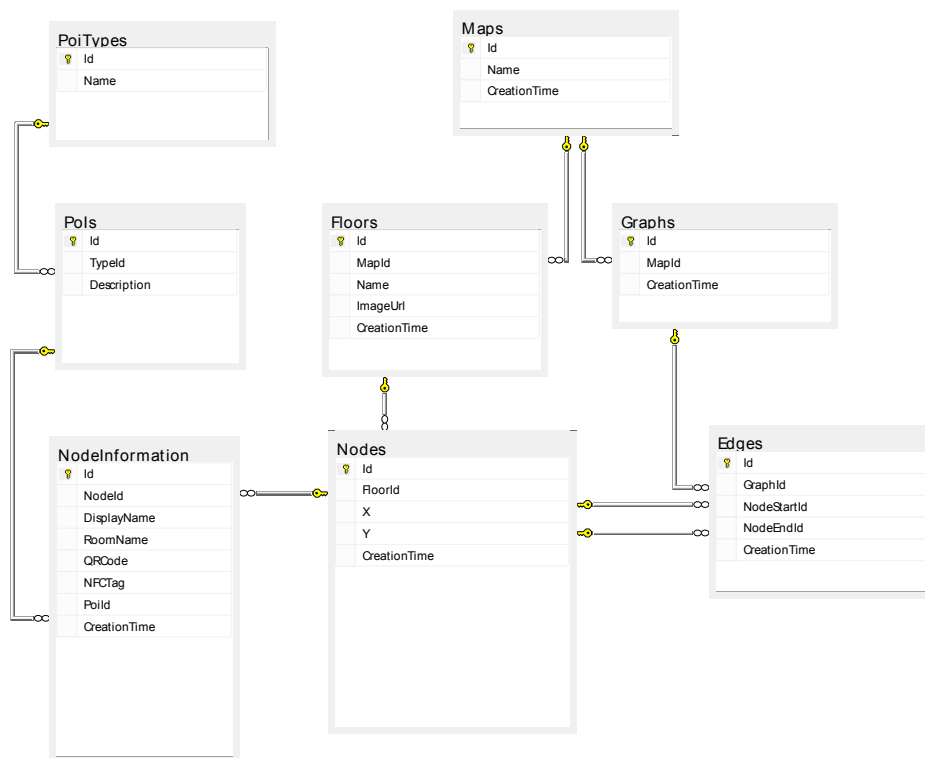


Abbildung 4.1.: Datenbankmodell für die Kartenobjekte

#### 4.1.1. Maps

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

#### 4. Datenbank

---

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

##### 4.1.2. 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

##### 4.1.3. 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

##### 4.1.4. 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



#### 4.1.5. 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

#### 4.1.6. 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)	Optional zugeordneter PoI
CreationTime	DATETIME	Erstellungszeitpunkt

#### 4.1.7. Pols

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

#### 4. Datenbank

---

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

##### 4.1.8. 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

## 4.2. Users

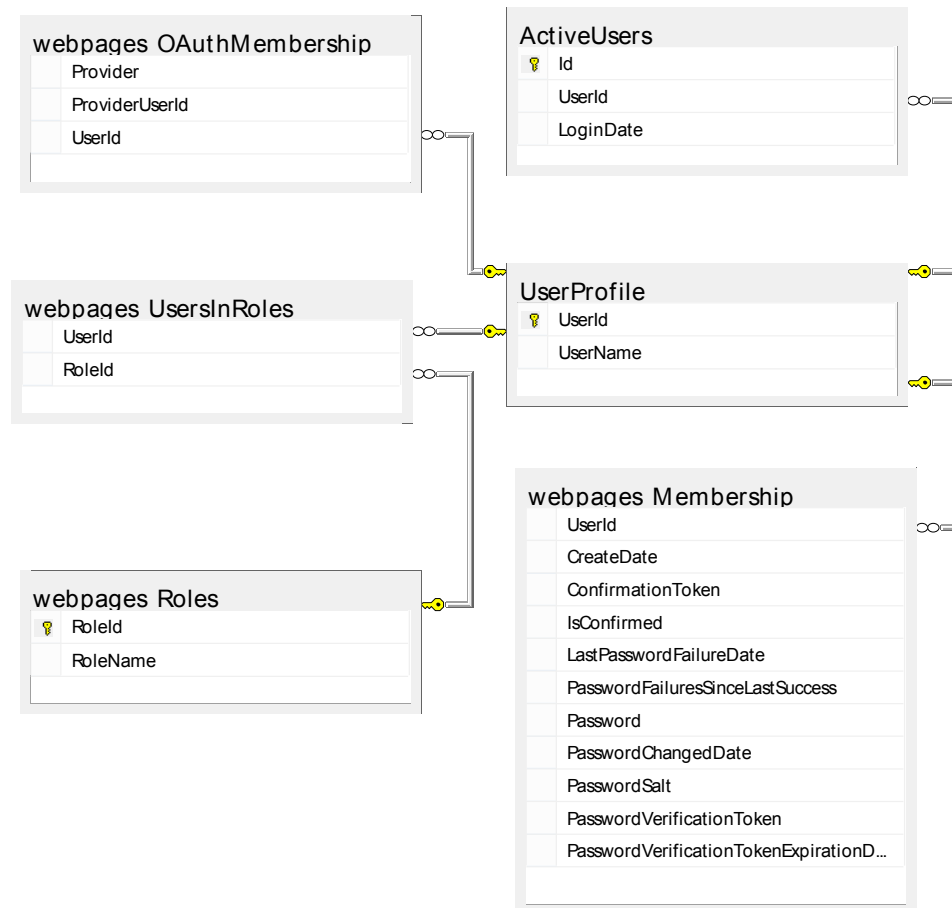


Abbildung 4.2.: Datenbankmodell für Benutzerobjekte

### 4.2.1. MVC spezifische Tabellen

Alle Tabellen aus dem Diagramm 4.2 mit Ausnahme der ActiveUsers-Tabelle werden von MVC generiert und verwaltet. Sie dienen zur Benutzerauthentifizierung und zur Autorisierung.

Relevant für StudMap sind die Tabellen “webpages\_Roles” und “webpages\_UserInRoles“. Hier werden bestehenden Nutzern Rollen zugewiesen. Um die Admin-Oberfläche nutzen zu können, muss einem Nutzer die Rolle “Admin“ zugewiesen worden sein.



#### 4.2.2. ActiveUsers

In dieser Tabelle wird festgehalten, welche Benutzer innerhalb eines festgelegten Zeitraums unsere App benutzt haben. Diese Tabelle kann z.B. um die letzte Position des Nutzers erweitert werden. Dadurch könnten andere Nutzer sehen, wer gerade in ihrer Nähe ist.

Spaltenname	Datentype	Bedeutung
Id	INTEGER (PK)	ID des Eintrags
UserId	INTEGER (FK)	ID des Nutzers
LoginDate	DATETIME	Zeitpunkt der letzten Anmeldung bzw. Nutzung der App

### 4.3. Views

Um kompliziertere Abfragen nicht im Service abhandeln zu müssen, haben wir für diese Views in der Datenbank angelegt. Die Views vereinfachen meist Joins über mehrere Tabellen.

#### 4.3.1. NodeInformationForMap

Diese View vereinigt die Informationen zu einem Knoten mit der zugehörigen Karte. Es werden nur Knoten berücksichtigt, zu denen ein Raum- oder Anzeigename hinterlegt wurde. Siehe [Maps](#) und [NodeInformation](#).

#### 4.3.2. PoisForMap

In dieser View werden die Knoteninformationen zusätzlich mit denen über mögliche PoIs angereichert. Es werden zudem alle Knoten herausgefiltert, zu denen kein PoI existiert. Siehe [PoIs](#).

#### 4.3.3. RoomsForMap

Ähnlich zu der View NodeInformationForMap liefert diese View allerdings nur den Raum- und Anzeigenamen eines Knoten.



## 4.4. Stored Procedures

Das Löschen von Tabellen kann auf Grund von Fremdschlüsselbeziehungen u.U. aufwendig und komplex sein. Daher haben wir für das Löschen des Teilgraphen auf einem Stockwerk (`sp_DeleteGraphFromFloor`), eines gesamten Stockwerks (`sp_DeleteFloor`) und einer gesamten Karte (`sp_DeleteMap`) Stored Procedures angelegt. Diese Löschen zunächst alle zugehörigen Einträge in abhängigen Tabellen und entfernen dann den Eintrag in der Haupttabelle.





## 5. Service

Der Webservice stellt die Logik für alle Clients bereit. Dies schließt den Navigations-Client, den Collector-Client und die Admin-Oberfläche ein. Er kapselt in erster Linie die Datenbankzugriffe und nutzt Caching für häufige Anfragen. Es werden Funktionen zur Benutzerverwaltung sowie zur Navigation bereitgestellt.

### 5.1. Allgemeine Struktur

Als Grundlage verwendet der Webservice ein ASP.NET Web API Projekt<sup>1</sup>. Der Datenbankzugriff erfolgt über das Entity Framework von Microsoft<sup>2</sup>.

In einem Web API Projekt können die Funktionen des Webservices auf Basis von selbst definierten Datenstrukturen erstellt werden. Damit muss sich der Entwickler nicht damit beschäftigen, wie diese Objekte vom Client geliefert werden und in welchem Format die Antwort gesendet wird. Das Serialisieren erfolgt meist in den Formaten XML oder JSON. Unser Webservice verwendet im Standard das JSON-Format, da es leichtgewichtiger ist und somit besser für mobile Clients geeignet ist.

Die Funktionalität teilt sich in drei Bereiche auf. Zum einen sind das die Karten- und Navigationsinformationen, die Benutzerverwaltung und das WLAN-Fingerprinting (nur experimentell). Diese Bereiche werden in drei Controllern abgebildet. Ein Controller ist eine Klasse, die Anfragen gegen den Webservice verarbeitet.

### 5.2. HTTP-Schnittstelle

Der Webservice bietet eine HTTP-Schnittstelle, die von den Clients genutzt wird. Diese bietet Funktionen über die HTTP-Methoden GET und POST auf bestimmte URLs an.

Der Client sendet einfache Anfrageparameter über die URL-Query-Parameter (GET). Bei komplexeren Anfragen kann er ein JSON-Dokument in den Body der Anfrage setzen (POST). Als Antwort erhält er ein JSON-Dokument<sup>3</sup>.

---

<sup>1</sup><http://www.asp.net/web-api>

<sup>2</sup>[http://msdn.microsoft.com/de-de/library/bb399567\(v=vs.110\).aspx](http://msdn.microsoft.com/de-de/library/bb399567(v=vs.110).aspx)

<sup>3</sup>Web API unterstützt auch XML, dies wird von unseren Clients allerdings nicht verwendet



Eine komplette Referenz der Webservice-Funktionen befindet sich im Anhang dieser Dokumentation (siehe [Webservice](#)).

### 5.3. Assembly-Schnittstelle

Da Anwendungen wie die Admin-Oberfläche und ein Teil der Client-Oberfläche immer auf demselben Server laufen, wie der Webservice ist ein Umweg über die HTTP-Schnittstelle aufwendiger als notwendig. Daher verwenden diese Anwendungen auch die Assembly-Schnittstelle des Webservices. Dabei werden die Anfragen direkt an die entsprechenden Controller gestellt. Das Serialisieren und Deserialisieren der Anfrage- und Antwortobjekte entfällt somit. Für dynamische Anfragen, z.B. in Javascript, nutzen diese Anwendungen allerdings weiterhin die HTTP-Schnittstelle.

Die Assembly-Schnittstelle bietet die gleiche Funktionalität wie die HTTP-Schnittstelle. Alle Funktionen des Webservice stehen als .NET Methoden der entsprechenden Controller-Klasse zur Verfügung. Deshalb verzichten wir hier auf eine eigene Dokumentation.

### 5.4. Caching

Der Webservice nutzt das Caching der HTTP-Laufzeitumgebung, um häufige Abfragen (z.B. allgemeine Karten- und Stockwerkinformationen) und komplexe Berechnungen (z.B. Routen zwischen allen Knoten) effizienter durchzuführen. Die Cache-Objekte sind nur für eine bestimmte Dauer gültig (aktuell eine Stunde) und werden invalidiert, falls sich die Stammdaten ändern.

### 5.5. Sicherheit

Aus Sicherheitsgründen macht es Sinn die HTTP-Schnittstelle vor der Veröffentlichung des Programms auf HTTPS umzustellen, da bisher jede Kommunikation unverschlüsselt erfolgt.



## 6. 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.

### 6.1. 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.

#### 6.1.1. 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.

##### 6.1.1.1. 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.

#### 6.1.1.2. 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

#### 6.1.2. 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.

##### 6.1.2.1. 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.

### 6.1.3. 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.

#### 6.1.3.1. Bibliothek tesseract

Tesseract<sup>4</sup> 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 6.1). Die Ergebnisse variieren von "Kein Text erkannt" bis hin zu "Buchstabensalat mit Raumnummer". Schneidet man den relevanten Teil per Hand aus (siehe 6.2), 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.

#### 6.1.3.2. App Google Goggles

Google Goggles<sup>5</sup> 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 API<sup>6</sup>. Diese ist zwar von Google geplant, aber nie umgesetzt worden. Auch wenn

---

<sup>4</sup><https://code.google.com/p/tesseract-ocr/>

<sup>5</sup><https://play.google.com/store/apps/details?id=com.google.android.apps.unveil>

<sup>6</sup><http://stackoverflow.com/questions/2080731/google-goggles-api>



Abbildung 6.1.: Gesamtes Bild Raumschild

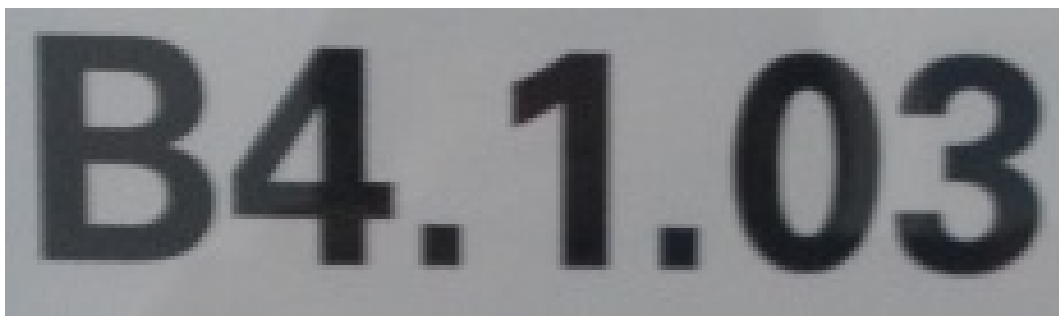


Abbildung 6.2.: Ausschnitt Raumschild

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

#### 6.1.4. 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.

##### 6.1.4.1. 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**<sup>7</sup>), 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.



---

<sup>7</sup>RSS: received signal strength wird in dBm gemessen.

#### 6.1.4.2. 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.

#### 6.1.4.3. 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.

### 6.2. 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.

#### 6.2.1. 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 }
```

### 6.2.2. 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
```

## 6.3. 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.

## 6.4. Benutzeroberfläche



## 6.5. 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



## 7. 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 Örtlichkeiten 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 beigelegt.

### 7.1. 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*.

##### 7.1.1. Model

###### AccountModels

Kapselt die benutzerspezifischen Daten in einem Model zur Authentifizierung von Benutzerprofilen. Dieses Model findet Verwendung beim Registrieren sowie beim LogIn- / LogOut-Verfahren.

###### AdminModels

Model indem der MapName gekapselt vorliegt.



### **FloorViewModel**

Dieses Model enthält Daten zu einem Floor. FloorId, MapId und FloorImageFile sind diese Daten.

### **MapViewModel**

MapId und Name werden für die MapView in diesem Model gekapselt.

## **7.1.2. View**

Die Views in unserem Projekt wurden in Views die den Account-, den Admin- und den Home-Bereich betreffen unterteilt.

### **Account**

Enthält html-Seiten zum registrieren, anmelden und verwalten den eigenen Benutzerprofils.

### **Admin**

Unter dieser Rubrik fallen die Webseiten, mit dem der Administrator Maps, Floors und Knoteninformationen zu einer Map hinzufügen kann. Sozusagen die Arbeitsfläche des Administrators.

### **Home**

Einstiegspunkt oder oberstes Element (Index) unserer Webseite.

## **7.1.3. 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

Der *AccountController* verarbeitet die Ereignisse, die vom Registrierungs-, LogIn- und LogOut-Verhalten eines Benutzers ausgelöst werden. Im Fokus stehen hierbei die **HTTP GET-** und **HTTP POST-Methoden**, die von der Klasse *AccountController* implementiert werden.

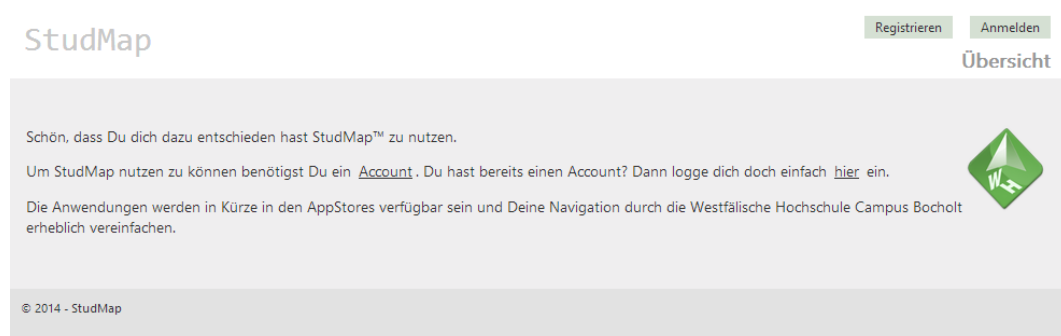
## AdminController

Der *AdminController* reagiert auf Zustandsänderungen, die beim Anlegen, Bearbeiten und Löschen von Datenmaterial in Form von Karten oder Informationen (Meta-Informationen), ausgelöst werden.

Relevante Funktionen sind die Methoden zum erstellen und löschen von Maps und Floors. Des weiteren werden über diesen Controller die Routeninformationen als Graph verarbeitet und in einer Datenbank gespeichert. Detailinformationen zu einem Knoten aus einem Graphen verarbeitet dieser Controller und speichert sie ab. Diese Detailinformationen ermöglichen besondere Orte als *Points of Interest* zu kennzeichnen.

## 7.2. Benutzeroberfläche

Die Administrator Benutzeroberfläche ist über folgenden Link <http://193.175.199.115/StudMapAdmin/> erreichbar.



Der Anwender enthält die Möglichkeit sich anzumelden oder sich zu registrieren.

## Registrieren

<http://193.175.199.115/StudMapAdmin/Account/Register>

StudMap

Registrieren Anmelden

Übersicht

Name

Passwort

Bestätige Passwort

Registrieren

© 2014 - StudMap

## Anmelden

<http://193.175.199.115/StudMapAdmin/Account/Login>

StudMap

Registrieren Anmelden

Übersicht

Name

Passwort

☐ Angemeldet bleiben

Anmelden

[Registriere](#) Dich wenn Du keinen Account hast.

© 2014 - StudMap

## Admin

Den Einstiegspunkt zum verwalten von Maps finden Sie unter folgenden Link

<http://193.175.199.115/StudMapAdmin/Admin>. Er enthält eine Auflistung aktuell

## 7. Admin

vorhandener Maps. Es können neue Maps erstellt, bzw. vorhandene entfernt werden.

StudMap

Hallo, Marcus! [Abmelden](#)

[Übersicht](#) [Admin](#)

Maps	Floors	Layer
MapId	Name	
3	Westfälische Hochschule	
<a href="#">Anlegen</a>		

© 2014 - StudMap

Die Verwaltung der einzelnen Floors zu einer Map werden in der nachfolgenden Grafik gezeigt. Durch einen Klick auf *Anlegen* kann eine neue Floor, mit der zugehörigen Kartengrundlage hinzugefügt werden.

StudMap

Hallo, Marcus! [Abmelden](#)

[Übersicht](#) [Admin](#)

Maps	Floors	Layer		
FloorId	MapId	Name	FloorImageFile	
1011	3	Ebene 0	http://193.175.199.115/StudMapAdmin/Images/Floors/RN_Ebene_0.png	
1012	3	Ebene 1	http://193.175.199.115/StudMapAdmin/Images/Floors/RN_Ebene_1.png	
1013	3	Ebene 2	http://193.175.199.115/StudMapAdmin/Images/Floors/RN_Ebene_2.png	
<a href="#">Anlegen</a>				

© 2014 - StudMap

Der Layer zeigt die Karte und einen gegebenenfalls erstellten Graphen mit Knoten und Kanten zu genau einem Floor an. Durch Verwendung spezieller Features kann der Administrator hier einen Graphen erstellen und Knoteninformationen hinterlegen. Des weiteren besteht die Möglichkeit Knoten mit Knoten anderer Floors zu verknüpfen. Basis für eine ergonomische Navigation der Karte ist die JavaScript-Bibliothek D3.


StudMap


Hallo, Marcus! Abmelden


Übersicht Admin

Maps
Floors
Layer

**Ebene 0**







**Knoteninformationen (1030)**

**Knoteninformationen**

**DisplayName:**

**RoomName:**

**QR-Code:**

**NFC Tag:**

**Pol Typ:**

**Pol Beschreibung:**

Tel.: 02871 2155-822  
 e-Mail: martin.schulten@w-hs.de

**Knoten**

Speichern
Abbrechen

Ebene 0





Über den Button *Abmelden* gelangen Sie zum Index unserer Webseite zurück.

## 7.3. Admin Spezifisches

### Neuen Benutzer mit Administrator Rechten versehen

Nachdem ein neuer Benutzer die Registratur erfolgreich abgeschlossen hat, ist dieser im Benutzerprofil noch nicht als Administrator gekennzeichnet. Derzeit ist es so, dass ein Datenbankadministrator in der Tabelle *webpages\_UsersInRoles* den Eintrag von 1 für Users auf 2 für Admins abändern muss. Nur Admins haben die Rechte Maps und Floors zu erstellen, sowie das Kartenmaterial mit Metainformationen zu bereichern.

### ELMAH

Als Logging Werkzeug haben wir das auf der .NET Plattform sehr verbreitete und OpenSource Tool *ELMAH* eingesetzt. *ELMAH*<sup>8</sup> loggt auftretende Fehler und Exceptions. Nach kurzem Installationsaufwand und Konfiguration einer Datenbank, in der die Fehler gespeichert werden, kann das Tool bereits genutzt werden. Genaue Installationsanleitungen findet Sie zu genüge im Internet<sup>9</sup>.

## 7.4. 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.

## 7.5. 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

---

<sup>8</sup>ELMAH steht für Error Logging Modules and Handlers"

<sup>9</sup><http://blog.thomasbandt.de/39/2380/de/blog/elmah-mit-aspnet-mvc-nutzen-und-fehler-loggen.html>



## *7. Admin*

---

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](https://www.qr-code.net/).

## 8. 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.

### 8.1. 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:

#### 8.1.1. 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.

#### 8.1.2. 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 eine WebView zur Anzeige gebracht wird. Dies hat für uns und den Endbenutzer den großen 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.

### 8.1.3. 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.

## 8.2. Benutzeroberfläche

Neben der bereits erwähnte WebView mit eine hübschen 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 Funktionalitäten 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 Übersichtlichkeit 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 intuitiv 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.



## 9. 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 könnten.



## 10. Ausblick

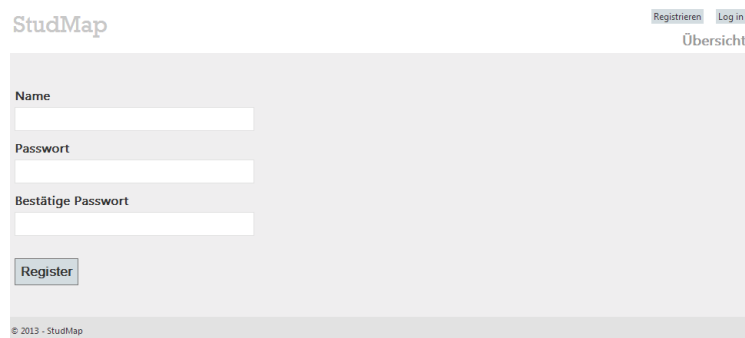
In diesem Kapitel geht es um mögliche Erweiterungen und Anpassungen, die in Zukunft sinnvoll sind. Zusätzlich soll es auch um die Möglichkeit gehen StudMap an anderen Standorten einzusetzen. Wir haben uns zwar in diesem Projekt auf das Gebäude unserer Hochschule beschränkt, jedoch kann StudMap auch in anderen Universitäten oder auch öffentlichen Gebäuden eingesetzt werden.

Ein für die Zukunft sehr interessantes Feature ist sicherlich die Positionsermittlung anderer Benutzer. Das heißt es besteht die Möglichkeit abzufragen an welcher Position sich eine Person gerade befindet. Denkbar wäre auch sich daraufhin zu dieser Person navigieren zu lassen.

Des Weiteren könnte man die Navigation noch überarbeiten und sich beispielsweise an Google Maps orientieren. Wegweisende Symbole und Angaben über die Distanz bis zum Ziel sind sicherlich weitere hilfreiche Features. Zusätzlich könnte eine Sprachausgabe implementiert werden.

## A. 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.



The screenshot shows the registration page of the StudMap application. At the top left is the 'StudMap' logo. At the top right are links for 'Registrieren', 'Log in', and 'Übersicht'. The main form area contains three input fields labeled 'Name', 'Passwort', and 'Bestätige Passwort'. Below these fields is a 'Register' button. At the bottom left of the form area, there is a copyright notice: '© 2013 - StudMap'.

Abbildung A.1.: Webseite zur Registrierung im StudMap Admin

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

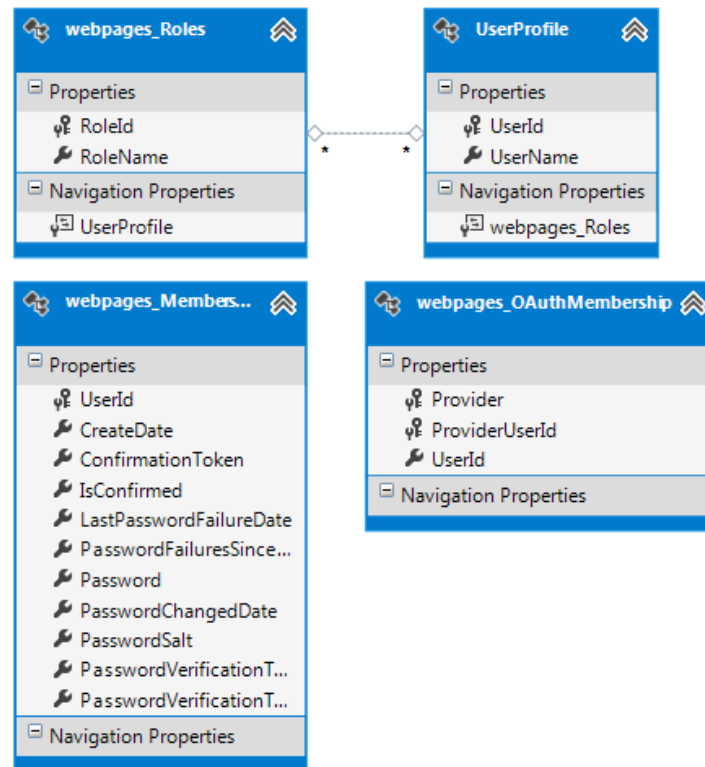


Abbildung A.2.: Datenbankstruktur der integrierten Benutzerverwaltung

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	
<input type="checkbox"/>	Properties
	Id
	UserId
	LoginDate
<input type="checkbox"/>	Navigation Properties

Abbildung A.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](#).



## B. 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`<sup>10</sup> 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.

### B.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.

#### B.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 }
```

---

<sup>10</sup> siehe: [MSDN Dokumentation](#)

### B.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 }
```

### B.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. <a href="#">Edge</a> ).
Nodes	Eine Liste von Knoten (s. <a href="#">Node</a> ).

Beispiel:

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

### B.1.4. Pathplot

Repräsentiert für die Darstellung benötigte Daten.

Bitte noch  
entspre-  
chend  
vervoll-  
ständigen!

### B. Webservice

Eigenschaften:

Id	??
Classes	??
Points	Eine Liste von <b>Node</b> Objekten.

Beispiel:

```

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

### B.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!

### B.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 }
```

### B.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 }
```

### B.1.8. Pol

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

Eigenschaften:

Type	Typ des PoIs (s. <b>PoiType</b> ).
Description	Beschreibung des PoIs.

Beispiel:

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

### B.1.9. RoomAndPol

Enthält Informationen zu einem Raum und dem zugeordneten PoI.

Eigenschaften:

Room	Rauminformationen (s. <b>Room</b> ).
PoI	PoI-Informationen (s. <b>PoI</b> ).

Beispiel:

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

### B.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 }
```

### B.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 }
```

### B.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 }
```

### B.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 }
```

### B.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 }
```

### B.1.15. User

Repräsentiert die für einen Benutzer relevanten Daten.

Eigenschaften:

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

Beispiel:

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

### B.1.16. SaveGraphRequest

Repräsentiert die Änderungen an dem Teilgraph für ein Stockwerk.

Eigenschaften:

FloorId	ID des Stockwerks.
NewGraph	Der hinzugefügte Teilgraph (s. <a href="#">Graph</a> ).
DeletedGraph	Der gelöschte Teilgraph (s. <a href="#">Graph</a> ).

Beispiel:

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

## B.2. Rückgabe Objekte

### B.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 }
```



### B.2.1.1. ResponseStatus

Wir unterscheiden zwischen:

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

### B.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.

### B.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 }
```



### B.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 }
```

## B.3. MapsController

### B.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**

### B.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**

### B.3.3. GetMaps

Liefert eine Liste aller Karten zurück.

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

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

### B.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](#)

### B.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](#)

### B.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](#)

### B.3.7. GetFloor

Liefert Informationen zu einem Stockwerk.

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

Parameter:

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

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

### B.3.8. GetFloorplanImage

Liefert die URL des Bilds zu einem Stockwerk.

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

Parameter:

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

Rückgabewert: String

### B.3.9. SaveGraphForFloor

Speichert den Graphen zu einem Stockwerk ab.

POST </api/Maps/SaveGraphForFloor>

POST Body: [SaveGraphRequest](#)

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

### B.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](#)

### B.3.11. GetGraphForFloor

Liefert den Teilgraphen für ein Stockwerk.

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

Parameter:

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

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



### B.3.12. GetFloorPlanData

Liefert die verschiedenen Schichten auf einem Stockwerk.

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

Parameter:

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

Rückgabewert: `ObjectResponse`, `FloorPlanData`

GetConnectedNodes

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

Rückgabewert: `ListResponse`, `Node`

### B.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



### B.3.15. SaveNodeInformation

Speichert zusätzliche Informationen zu einem Knoten ab.

POST </api/Maps/SaveNodeInformation>

POST Body: [NodeInformation](#)

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

### B.3.16. GetPoiTypes

Liefert eine Liste aller Typen von PoIs zurück.

GET </api/Maps/GetPoiTypes>

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

### B.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](#)

### B.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](#)

### B.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

### B.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

## B.4. UsersController

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

### B.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**





### B.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](#)

### B.4.3. Logout

Meldet einen angemeldeten Anwender vom System ab.

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

Parameter:

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

Rückgabewert: [BaseResponse](#)

### B.4.4. GetActiveUsers

Ermittelt eine Liste der aktuell am System angemeldeten Anwender.

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

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

## C. Verwendung des Webservices

### C.1. Verwendung der Benutzerschnittstelle

#### C.1.1. Registrierung

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

#### C.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.

#### C.1.3. Aktive Benutzer abfragen

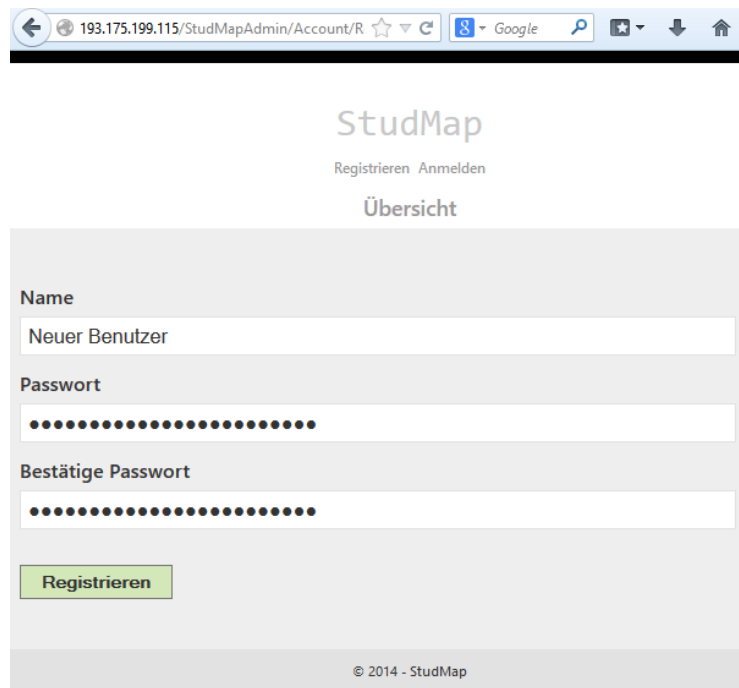
Die aktiven Benutzer können über die Funktion **GetActiveUsers** abgefragt werden. Damit die Anzeige der aktiven Benutzer im Client möglichst aktuell ist, sollte diese Abfrage ebenfalls in regelmäßigen Zeitabständen erfolgen.

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

## D. Admin-Bedienungsanleitung

### D.1. Registrieren

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Account/Register>.



The screenshot shows a web browser window with the address bar displaying [193.175.199.115/StudMapAdmin/Account/R](http://193.175.199.115/StudMapAdmin/Account/R). The page title is "StudMap" with sub-links "Registrieren" and "Anmelden". Below this is a section titled "Übersicht" containing a registration form. The form has three input fields: "Name" with the placeholder text "Neuer Benutzer", "Passwort", and "Bestätige Passwort", all filled with dots. A green "Registrieren" button is at the bottom of the form. The footer of the page reads "© 2014 - StudMap".

2. Button *Registrieren* betätigen.
3. Datenbankadministrator muss den **Neuen Benutzer mit Administrator Rechten** **versehen**.

### D.2. Login / LogOut

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Account/Login>.
2. Benutzername und Passwort eingeben.
3. Button *Anmelden* betätigen.
4. Sie können sich jederzeit über den Button *Abmelden* ausloggen.



### D.3. Passwort ändern

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Account/Manage> oder auf eigenen Benutzernamen klicken.
2. Aktuelles Kennwort und gewünschtes neues Kennwort eingeben. Dieses zusätzlich bestätigen.
3. Button *Passwort ändern* betätigen.

### D.4. Neue Map anlegen

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Admin/CreateMap> oder auf der Admin Seite auf *anlegen* klicken.
2. Anschließend Map-Namen eingeben und bestätigen.
3. Map kann über das Papierkorb-Symbol gelöscht werden.

### D.5. Neuen Floor anlegen

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Admin>.
2. Anschließend auf den Map-Namen klicken.
3. Über den Button *Anlegen* gelangen Sie nun zur Seite auf der Sie einen Floor hinzufügen können.
4. Den Eintrag MapId wird automatisch übernommen.
5. Sie geben den Floor-Namen ein und ergänzen einen Link zu Ihrer Kartengrundlage <sup>11</sup>.
6. Abschließend betätigen Sie den Button *Anlegen*.

---

<sup>11</sup>Bevorzugtes Format der Karte sind PDF und PNG Files.

## D.6. Menühandhabung des Layers

1. Aufruf der Webseite: <http://193.175.199.115/StudMapAdmin/Admin>.
2. Anschließend auf den Map-Namen klicken.
3. Dann auf den Floor-Namen klicken.
4. Sie können über das Mausrad in die Karte, bzw. aus der Karte hinaus zoomen und das Kartenwerk verschieben indem Sie die linke Maustaste gedrückt halten und die Maus hin und her bewegen.
5. Des weiteren können Sie folgende Features ausüben, um die Karte mit Meta-Informationen zu ergänzen:
  - Knoten anlegen
  - Kanten anlegen
  - Knoten löschen
  - Knoteninformationen hinterlegen
  - Knoten mit Knoten auf anderem Floor verbinden

### Knoten anlegen

1. Sie befinden sich auf der Layer Webseite und haben das Kartenwerk vor Augen.
2. Sie zoomen an die entsprechende Stelle an der Sie einen Knoten erzeugen möchten mit dem Mausrad.
3. Dann betätigen Sie in Kombination mit der *Strg-Taste* die *linke Maustaste*.
4. Abschließend müssen Sie den Button *Speichern* betätigen, den Sie sehen müssen wenn sie komplett raus gezoomt haben.

### Kanten anlegen

1. Sie befinden sich auf der Layer Webseite und haben das Kartenwerk vor Augen.
2. Zwei Knoten werden mit einander verbunden, indem Sie zuerst einen der beiden Knoten mit einem einfachen Klick mit der *linken Maustaste* markieren.
3. Anschließend navigieren Sie zum zweiten Knoten. Durch einen Klick mit der *linken Maustaste* werden beide Knoten mit einer Kante miteinander verbunden.

4. Speichern Sie Ihr Ergebnis über den entsprechenden Button ab.

## Knoten löschen

1. Sie befinden sich auf der Layer Webseite und haben das Kartenwerk vor Augen.
2. Sie zoomen zu den Knoten, den es zu entfernen gilt.
3. Markieren Sie den Knoten mit der *linken Maustaste*.
4. Betätigen Sie die Taste *Entf* auf Ihrer Tastatur. Der Knoten und anliegende Kanten sind entfernt worden.
5. Vergessen Sie nicht den aktuell vorliegenden Graphen zu speichern.

## Knoteninformationen hinterlegen

1. Sie befinden sich auf der Layer Webseite und haben das Kartenwerk vor Augen.
2. Durch einen einfachen Klick mit der *rechten Maustaste* erhalten Sie gezieltere Informationen zum Knoten.

**Knoteninformationen (1030)**

**Knoteninformationen**

DisplayName: Prof. Dr. Martin Schul

RoomName: B4.0.06

QR-Code: {"General":{"RoomNa

NFC Tag:

Pol Typ: Büro Dozent

Pol Beschreibung: Tel.: 02871 2155-822  
e-Mail: martin.schulden@w-h  
s.de

**Knoten**

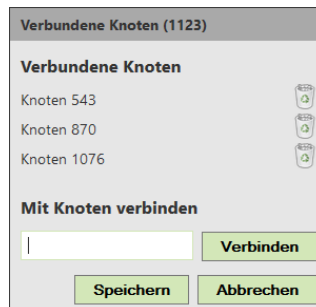
Speichern Abbrechen

3. Sie können die den Knoten um Informationen erweitern, indem Sie entsprechende Vermerke in den Textboxen vornehmen.
  - Display Name: Sprechender Name des Raumes
  - RoomName: Einmalige Raumbezeichnung
  - QR-Code: Von uns automatisierter QR-Code für diesen Raum
  - NFC-Tag: NFC-Tag ID für diesen Raum

- PoI-Type: Art des Raumes, z.B. Labor
  - PoI-Beschreibung: Informationen die diesen Raum genauer beschreiben oder die Sie diesem Knoten zusätzlich hinterlegen wollen.
4. Die Knoten Id und die genauen Koordinaten des Knotens, in Prozent, werden angezeigt nachdem Sie auf das Stichwort **Knoten** mit der Maus navigieren.
  5. Info: Ergänzen Sie einen Knoten um Informationen erst, nachdem Sie den Graphen gespeichert haben. Ansonsten gehen Ihre Eingaben verloren.


### Knoten mit Knoten auf anderem Floor verbinden


1. Sie befinden sich auf der Layer Webseite und haben das Kartenwerk vor Augen.
2. Zuerst benötigen Sie die Knoten Id's, mit den Sie Ihren Knoten verbinden möchten. Also müssen Sie sich möglicherweise die Id eines Knotens einer anderen Floor notieren.
3. Betätigen Sie die *Shift-Taste + Linke Maustaste* auf den Knoten:




Verbundene Knoten (1123)

**Verbundene Knoten**

Knoten 543 

Knoten 870 

Knoten 1076 

**Mit Knoten verbinden**

**Verbinden**

**Speichern** **Abbrechen**

4. Tragen Sie die Id ein.
5. Drücken Sie den Button *Verbinden*
6. Abschließend betätigen Sie den Button *Speichern* und zwei Knoten wurden über diesen Weg miteinander verbunden.