

Fakultät Geoinformationswesen
Studiengang Kartographie und Geomatik

Praktikum Datenbanken
Prof. Dr. Hans Kern
SS 2008

Studienarbeit

Verkehrsauskunftssystem für Singapur

„SINIS – Singapore Network Information System“

Bearbeitet von
Arne Johannessen
Volker von Nathusius

1	Einführende Informationen	4
2	Projektanalyse	5
2.1	Sichtung, Korrektur und Ergänzung der Datenbank	5
2.2	Überprüfung der Abfragen	7
2.3	Bewertung der vorhandenen Daten und Abfragen	8
3	Konzeption	9
3.1	Systementwurf	9
3.2	Datenbankschicht	10
3.2.1	Datenmodell	10
3.2.2	Schnittstelle zur Datenbank	11
3.3	Anwendungsschicht	12
3.3.1	Übersicht zu den Modulen	12
3.3.2	dbis.include – die Fassade der Anwendungsschicht	12
3.4	Dialogschicht	13
3.5	Klassifizierung der verschiedenen Abfragen	13
3.6	Semantische Auszeichnung	14
3.7	Graphische Gestaltung der Benutzeroberfläche	15
3.8	Konventionen	16
4	Klassifizierung der Abfragen der Dialogschicht	18
4.1	Grundstruktur aller Abfragen	18
4.2	Abfrage mit interaktiven Dialogfeldern	18
4.3	Statische Abfrage	22
4.4	Abfragen mit interaktiver Karte	22
4.5	Ergebnisausgabe als Karte	24
4.6	Klassenzugehörigkeit der einzelnen Abfragen	25
5	Abfragen der Dialogschicht	26
5.1	singapore.php: statische Abfrage	26
5.2	map.php: statische Abfrage	27
5.3	network_totalRange.php: statische Abfrage	27
5.4	network_lineRange.php: statische Abfrage	28
5.5	line_howManyStops.php: statische Abfrage	28
5.6	routing.php: Abfrage mit interaktiven Dialogfeldern	28
5.7	line_atStop.php: Abfrage mit interaktiver Karte	30
5.8	line_map.php: Ergebnisausgabe als Karte	30

5.9	stop_nameQuery.php: Abfrage mit interaktiven Dialogfeldern.....	32
5.10	stop_howManyLines.php: Abfrage mit interaktiven Dialogfeldern	33
5.11	stop_nearPosition.php: Abfrage mit interaktiver Karte	34
5.12	stop_reachable.php: Abfrage mit interaktiver Karte	35
5.13	stop_atLine.php: Abfrage mit interaktiven Dialogfeldern	36
5.14	stop_nearPoi.php: Abfrage mit interaktiven Dialogfeldern	36
5.15	poi_info.php: Abfrage mit interaktiven Dialogfeldern	37
5.16	poi_nearStop.php: Abfrage mit interaktiven Karten	37
5.17	poi_map.php: Ergebnisausgabe als Karte	37
5.18	dataset_info.php: Abfrage mit interaktiven Dialogfeldern	38
6	Komplexe Abfragen der Dialogschicht	40
6.1	Google Maps–Einbindung	40
6.1.1	Schnittstelle zwischen Google Maps und der Datenbank	40
6.1.2	Zusammenspiel der Schichten	41
6.2	Routing-Algorithmus	43
6.2.1	Umsetzung in PHP und SQL.....	43
6.2.2	Ergebniskritik.....	46
7	Installation der Web-Anwendung	47
7.1	Mindest-Systemanforderungen	47
7.2	Empfohlene Umgebung und Kompatibilität	47
7.3	Datenbank vorbereiten	48
7.4	Web-Anwendung installieren.....	48
8	Schlussbemerkungen	50
8.1	Zählung der Abfragen	50
8.2	Verwendete Software	50
8.3	Ausblick	50
9	Literatur.....	51
10	Lizenz	52
11	Lizenz (deutsche Übersetzung)	53

1 Einführende Informationen

Die Studienarbeit baut auf auf der Studienarbeit von Daniel Pfeifer aus dem Wintersemester 2007/08.

Es wurden uns übergeben:

- die Dokumentation von Daniel Pfeifer
- fünf csv-Tabellen namens karte, haltest, linie, poi und haltest_linie
- Screenshots von Abfrageergebnissen
- Fotografien der MRT-Züge in Singapur
- ein veralteter Liniennetzplan von Singapur
- das Entity-Relationship-Modell von Daniel Pfeifers Datenbank

Die Aufgabe ist wie folgt:

- Sichtung und Bewertung der vorhandenen Daten.
- Ergänzung/Vervollständigung der vorhandenen Daten.
- Erstellen einer SQL-Ladefdatei für MySQL
- Konzeption eines Layouts für eine nutzerfreundliche Webseite
- Realisierung von 20 Abfragen an die MySQL-Datenbank auf der Webseite, wobei die Formularelemente Checkbox, Radiobutton, Auswahlliste und Textfeld verwendet werden sollen.
- Interaktive Darstellung von Punktinformationen mit Google Maps.

2 Projektanalyse

2.1 Sichtung, Korrektur und Ergänzung der Datenbank

Anhand der Dokumentation konnte einigermaßen nachvollzogen werden, auf welche Weise die uns übergebenen Tabellen und Abfragen angelegt worden waren. Die Projektanalyse ergab folgende wichtige Hinweise, die die Richtung der Studienarbeit maßgeblich beeinflussten:

- Die Tabelle „karten“, die fünf nicht näher beschriebene Karten mit Blattmaß, Kantenecken und Maßstab aufführte, stellte sich als komplett überflüssig heraus: Gemäß der ursprünglichen Aufgabenstellung wurde bereits in der frühen Planungsphase ein Proof-of-Concept der Einbettung von Google Maps über die von Google bereitgestellte API in das Projekt entwickelt. Wie sich herausstellte, ist damit eine Umsetzung von Karten, die durch ihre Eckpositionen definiert sind, nicht ohne Weiteres möglich; vielmehr bezieht sich Google Maps auf eine Zentrumsposition und eine Skalierungsklasse, so dass die in der Tabelle „karten“ vorhandenen Daten nicht wiederverwendbar waren. Infolgedessen fiel schon früh der Entschluss, zugunsten einer guten Google Maps–Einbindung auf die Tabelle „karten“ vollends zu verzichten.
- Sämtliche Koordinaten in der Datenbank liegen im UTM-System vor (Zone 48N). Für Google Maps werden aber geographische Koordinaten benötigt. Ein Professional Paper des amerikanischen USGS (Snyder 1987) enthält zur Umrechnung geeignete Formeln. Die Implementierung der Umrechnung erfolgte für beide Rechenrichtungen gemeinsam in einer eigenen PHP-Klasse.
- Weil Englisch neben Chinesisch, Malaiisch und Tamilisch auch die Verkehrssprache in Singapur ist, und eine gewisse Nutzerorientierung im Projekt hergestellt werden sollte, haben wir uns dafür entschieden, komplett auf Englisch vorzugehen. Da das gesamte Namensgut in den Tabellen auf Englisch verfasst worden war, fiel die Entscheidung leicht, datenbankintern sowie auf der noch zu erstellenden Benutzeroberfläche komplett auf Englisch umzusteigen.
- Aufgrund dieser Entscheidung wurde die Tabelle „linie“ in „line“ umbenannt, „haltest“ in „stops“ und „haltest_linie“ in „stop_has_line“. „poi“ konnte mit dem bestehenden Namen bestehenbleiben. Auch die Spaltennamen wurden entsprechend geändert (vergl. zur folgenden Attributtabelle die Ausarbeitung von Daniel Pfeifer):

Tabellenname	Attribut	Schlüssel	Type	Erläuterung
stop	id	Primär	int (8)	ID der Haltestelle
	name		varchar (50)	Name der Haltestelle
	utm_e		int (6)	Abszisse in UTM
	utm_n		int (7)	Ordinate in UTM
line	id	Primär	int (8)	ID der Linie
	color		varchar (20)	Farbe der Linie (CSS-Syntax)
	name		varchar (50)	Name der Linie
stop_has_line	id_stop	Verbund	int (8)	ID der Haltestelle

	id_line	Verbund	int (8)	ID der Linie
	order		int (8)	Reihenfolge der Haltestellen
	distance		int (8)	Distanz ab der Anfangshaltestelle
	duration		int (8)	Fahrtdauer ab Anfangshaltestelle
poi	id	Primär	int (8)	ID des POI
	name		varchar (50)	Name des POI
	type		text (20)	Art des POI
	utm_e		int (6)	Abszisse der POI-Koordinate in UTM
	utm_n		int (7)	Ordinate der POI-Koordinate in UTM

- Im Rahmen dieser Umarbeitung der Tabellen wurden auch schwerwiegende Fehler korrigiert: So verfügte die Tabelle „haltest_linie“ über den Primärschlüssel „id“: Jeder Beziehung zwischen Haltestellen und Linien wurde eine eigene ID zugewiesen, neben den Verknüpfungs-IDs „h_id“ und „l_id“. Die neue Tabelle „stop_has_line“ hingegen verwendet als Primärschlüssel die Kombination der beiden Fremdschlüssel „id_line“ und „id_stop“. Ferner waren in der ursprünglichen Tabelle „haltest_linie“ zwei Umsteigehaltestellen nicht erfasst (aus Sicht der Datenbank hielt dort somit nur je eine Linie anstatt zweien), außerdem existierten in der Tabelle „haltest“ mehrere Duplikate einzelner Haltestellen unter unterschiedlichen IDs. Dadurch hatte Daniel Pfeifer erreicht, dass es genausoviele Haltestellen wie Haltestellen-Linien-Beziehungen in seinen Tabellen gab, obwohl es in den drei von ihm erfassten Linien sechs Umsteigepunkte gab!
- Ein besonders grober Fehler war die bisherige Handhabung der grünen Linie (East-West-Line): Wie aus dem Liniennetzplan ersichtlich, zweigt an der Haltestelle „Tanah Merah“ eine Nebenlinie ab, das so genannte „Changi Airport Shuttle“. Diese Linie war in die East-West-Line integriert worden, und zwar ausgerechnet mit „Expo“ als unmittelbarer Nachfolger der Haltestelle „Boon Lay“, ganz am anderen Ende der Linie. Anstelle dieser fehlerhaften Modellierung wurde eine neue Linie angelegt, die die drei Haltestellen „Tanah Merah“, „Expo“ und „Changi Airport“ enthält.
- Nach dieser Korrektur wurden die vier bis dato in Betrieb genommenen LRT-Nebenlinien mit weiteren 40 Haltestellen neu in die Datenbank aufgenommen. Durch die Änderung der SQL-Abfrage des oben erwähnten Google Maps–Proof of Concepts von SELECT auf INSERT konnte dies effizient bewerkstelligt werden. Insgesamt sind damit nunmehr knapp 100 Haltestellen in acht Linien erfasst.
- Aufgrund ihrer Schleifenstruktur erwiesen sich die LRTs als nicht besonders einfach zu modellieren, wie am Beispiel „Sengkang“ gezeigt werden kann: Diese Haltestelle hat nunmehr fünf Einträge in der „stop_has_line“-Korrelation: als Durchgangsstation der North East Line, sowie jeweils als Anfangs- und Endstation der Sengkang LRT West Loop und der Sengkang LRT East Loop.
- Auffallend war in diesem Zusammenhang, dass die IDs nicht nur je Tabelle einmalig, sondern in der gesamten Datenbank einmalig waren (so enthielt die Tabelle „haltest“ die Haltestellen 1–68, die Tabelle „linie“ die Linien 69–71, die Tabelle „poi“ die POIs 79–94...). Diesem Umstand wurde nicht weiter Beachtung geschenkt: Neu erfasste Haltestellen erhielt die IDs 69 aufwärts, neu erfasste Linien die IDs 72 aufwärts. Die Lücken, die durch Löschung von Haltestellen entstanden, wurden nicht neu aufgefüllt.

- Sehr gut für unsere Zwecke erwies sich die Tatsache, dass Daniel Pfeifer die Koordinaten der Points of Interest sowie der Haltestellen bereits in metrischen UTM-Koordinaten angegeben hatte, sodass sich Luftlinien-Entfernungsberechnungen sehr einfach realisieren ließen.
- Schließlich wurde die SQL-Startdatei aus den nunmehr vier Excel-Tabellen erzeugt. Sie ist unter dem Namen singapore.sql auf der CD zu finden.

2.2 Überprüfung der Abfragen

Schließlich wurden auch die bereits vorhandenen Abfragen durchgeschaut, wobei natürlich alle Bezeichner auf ihren neuen Namen hin umgeschrieben werden mussten. Diese Umarbeitung war zudem gut für die Wiedereinarbeitung in die SQL-Syntax. Folgende Abfragen waren uns übergeben worden:

1. Ausgabe der Gesamtstrecke für alle Linien
2. Welche herausragenden Museen gibt es in Singapur?
3. Welche Haltestellen beginnen mit dem Buchstaben C?
4. Durch welche Haltestellen fährt Linie 69, in welcher Reihenfolge, wie weit ist deren Entfernung und wie lange ist die Fahrtzeit?
5. Wie viele Haltestellen haben die einzelnen Linien?
6. Welche Linien halten an der Haltestelle Jurong East?
7. Welches sind die 5 Haltestellen an denen die meisten Linien halten und welche Linien sind das?
8. Welches sind die fünf nächstgelegenen Haltestellen zum Parliament House (gemessen an der Luftlinie), und wie weit sind diese entfernt?
9. Wie lange dauert die Fahrt von der Haltestelle Yew Tee zur Haltestelle Newton mit der Nord-Süd-Linie und wie lang ist die zurückgelegte Strecke?
10. Auf welchen Karten liegt der POI Little India?
11. Auf welcher Karte liegen wieviele POIs?
12. Auf welcher Karte liegen wieviele Haltestellen?
13. Welche POIs gibt es im Umkreis von 500m um die Haltestelle Little India?
14. Wie groß ist die Entfernung (Luftlinie in m) von der Raffles Statue zu allen Haltestellen in Singapur?
15. Wie kommt man von der Haltestelle Novena zur Haltestelle Bugis?
16. Welche Haltestellen werden ohne umzusteigen von der Haltestelle Redhill erreicht?
17. Welche Haltestellen erreicht man von der Haltestelle Pasir Ris mit 1x umsteigen?

Von diesen 17 Abfragen fielen gleich zu Beginn drei Abfragen weg, die sich auf die Karten bezogen, die uns nicht zur Verfügung standen. Ferner ist auffallend, dass die Abfrage 8, 13 und 14 auf der gleichen Fragestellung beruhen, und nur eine unterschiedliche Anzahl von Ergebnissen ausgeben oder statt der Entfernung von POI zu Haltestelle die Entfernung Haltestelle zu POI liefern.

Außerdem gab es Abfragen, die nur eine sehr kurze und nicht besonders weiterführende Ergebnisliste ausgaben, wie etwa die Fragen 3 und 6, die sich sehr gut kombinieren ließen: Welche Haltestellen beginnen mit <Buchstabenkombination> und an welcher Linie liegen sie?

Als weniger sinnvoll erwiesen sich schließlich die Abfragen 9, 15, 16 und 17:

Sie alle sind Teilschritte eines Routing-Algorithmus, wobei Abfrage 9 und 16 den Fall „nicht umsteigen“ und Abfrage 15 und 17 den Fall „genau einmal umsteigen“ abdecken.

Während in Abfrage 15 das Beispiel von Daniel Pfeifer zwar korrekt „von Novena nach Raffles Place“ exakt eine Umsteigehaltestelle ausgibt, führt bereits eine Abfrage „von Novena

nach Newton“ (zwei direkt nebeneinanderliegende Haltestellen auf der gleichen Linie, keine davon eine Umsteigestelle!) zu 25 Umsteigehaltstellen, an denen aber meist nur die North-South-Line hält. Abfragen zu Wegen, bei denen mehr als einmal umgestiegen werden muss, führen dagegen zu null Treffern. Es kann damit konstatiert werden, dass die Abfragen speziell auf die Situation eines Drei-Linien-Netzes zugeschnitten, in dem jede Linie jede andere mindestens einmal kreuzt. Die erweiterte Datenbank kann es aber möglich machen, dass man bis zu sechsmal umsteigen muss, um von einem LRT-Subnetz in ein anderes zu gelangen.

Da sich die vier oben genannten Abfragen nicht für eine Nutzeroberfläche verallgemeinern lassen, muss eine komplett andere Herangehensweise an diese Problemstellung gefunden werden: Es wäre weitaus nutzerfreundlicher, falls es gelingen sollte, sie alle in einem einzigen PHP-Dokument zu realisieren.

2.3 Bewertung der vorhandenen Daten und Abfragen

Die übergebenen Daten waren eher mager, vor allem im Vergleich zu den Datenbeständen anderer Projekte in diesem Semester. Trotz der diversen Fehler waren sie eine gute Grundlage, auf der unsere Studienarbeit aufbauen konnte. Die Abfragen erwiesen sich ebenfalls als weitgehend brauchbar, auch wenn einige Abfragen wie z. B. Nr. 15 nicht verallgemeinerbar waren.

3 Konzeption

3.1 Systementwurf

In Anbetracht der eine Vielzahl von eigenständigen PHP-Dokumenten mit jeweils unterschiedlichen Datenbankabfragen fordernden Aufgabenstellung wurde zunächst überlegt, wo welche Redundanzen zu erwarten sind und wie man sie vermeiden kann. Offensichtlich haben alle Dokumente und Abfragen eine Reihe von Gemeinsamkeiten. Insbesondere sind dies:

- Zugriff auf dieselbe Datenbank
- viele Abfragen ähneln sich in Teilen
- Fehler (etwa im Falle von Problemen beim Aufbau der Verbindung zur Datenbank) sind gleich zu behandeln
- gleichartige Möglichkeit zur Navigation zwischen Dokumenten
- ähnlicher semantischer Aufbau und identischer semantischer Rahmen
- gleiche Optik der Benutzerschnittstelle (Corporate Design)
- ähnliche (möglichst gleiche) Funktion der Bedienung

Es wurden große Mühen investiert, die Entstehung solcher Redundanzen von vornherein zu vermeiden. Um diesem Ziel nahe zu kommen, wurden während des gesamten Projektzeitraumes so viele im Entstehen begriffene Redundanzen wie möglich in einem andauernden Vorgang aggressiven Refactorings umgangen. Nur in wenigen Einzelfällen wurde bewusst eine Redundanz zugelassen, um eine bessere Lesbarkeit und Wartbarkeit des Codes zu erreichen.

Es wurde für den Entwurf des Systems ein Schichtenmodell aufgestellt.

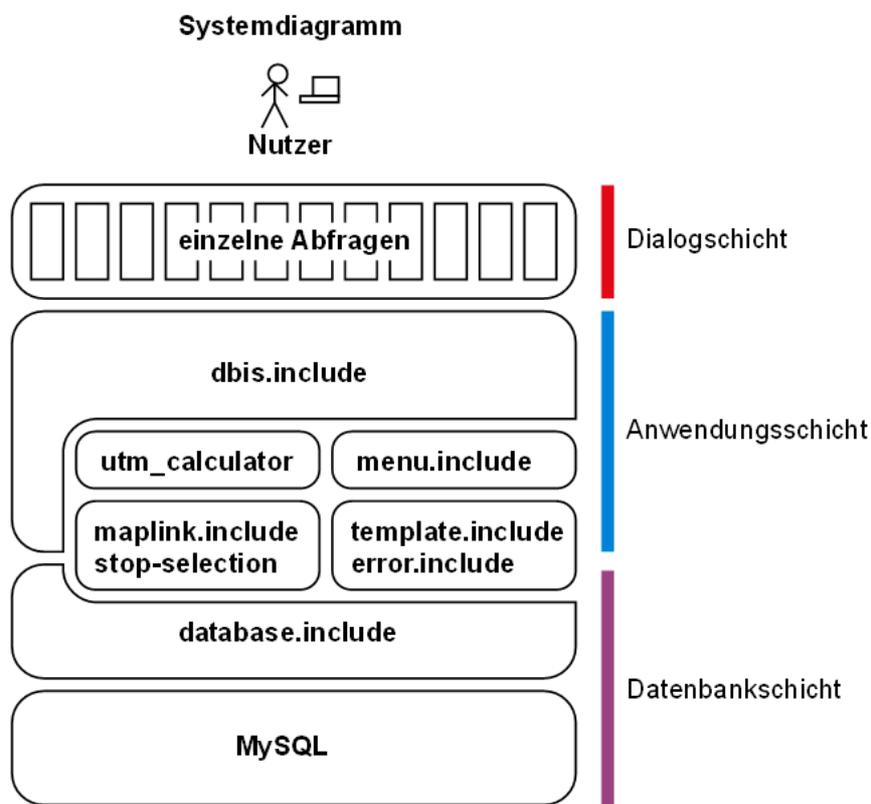


Abb. 1: Systemdiagramm

Wie aus dem Systemdiagramm hervorgeht, ergeben sich aus den genannten Anforderungen drei Schichten, in denen jeweils unterschiedliche Web-Technologien zum Einsatz kommen:

1. Datenbankschicht (SQL, PHP)
2. Anwendungsschicht (SQL, PHP, HTTP, JavaScript, XML)
3. Dialogschicht (SQL, PHP, HTTP, JavaScript, HTML, CSS)

Im Folgenden werden diese Schichten und deren Aufbau in dieser Reihenfolge einzeln im Detail besprochen.

3.2 Datenbankschicht

Der wichtigste Bestandteil der Datenbankschicht ist – natürlich – die Datenbank selbst. Zusätzlich ist der Schicht eine Sammlung von PHP-Funktionen zugeordnet, welche die Skripte der höheren Schichten vor der Notwendigkeit beschützen, unmittelbar auf die Datenbank zuzugreifen. Diese PHP-Funktionen sind in einer einzigen Datei zusammengefasst; es ist dies die `database.include.php`.

Konsequenzen:

- Code in den höheren Schichten wird vereinfacht und dadurch leichter zu warten.
- Änderungen der Datenbank-API (z. B. durch Wechsel des Backends von MySQL auf Oracle) ziehen nur wenige, im Idealfall keine, Änderungen an höheren Schichten nach sich.
- Zusätzliche Funktionalität ist auch in fortgeschrittenem Stadium des Projekts mit vertretbarem Aufwand zu realisieren.

3.2.1 Datenmodell

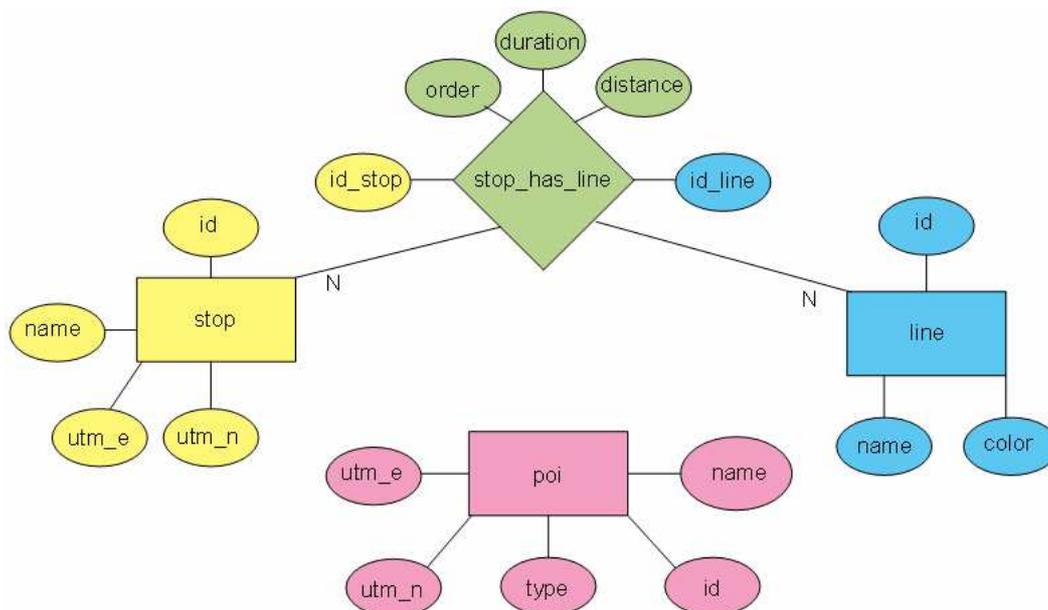


Abb. 2: Entity-Relationship-Diagramm

Wie aus dem ER-Diagramm ersichtlich ist, ist das zugrundeliegende Datenmodell ein recht einfaches: Zwei Tabellen `stop` und `line` mit Haltestellen und Linien sind über eine n:m-Relation miteinander verknüpft. Die diese Relation abbildende Assoziativtabelle `stop_has_line` enthält drei zusätzliche Attribute, welche die Relation genauer beschreiben; es wird damit der Verlauf der Linien dargestellt, die bestimmte Haltestellen in einer bestimmten Reihenfolge bedienen.

Die Tabelle `poi` steht im vorliegenden Datenmodell nicht in einer direkten Relation mit den anderen drei Tabellen. Sie enthält Sehenswürdigkeiten und andere besondere Einrichtungen mit hohem Publikumsverkehr („Points of Interest“).

Die Attributpaare `utm_e` und `utm_n` in den Tabellen `poi` und `stop` beziehen sich zwar auf das gleiche Koordinatensystem, was die algorithmische Herstellung eines Zusammenhangs erlaubt, jedoch besteht keine direkte Verknüpfung Datenbankebene. Um eine solche Relation zu schaffen, wäre die Zusammenfassung aller Koordinatenpaare in einer eigenen Tabelle erforderlich; die Tabellen `poi` und `stop` könnten dann auf ein bestimmtes Koordinatenpaar in jener Tabelle verweisen und würden die eigenen Attribute für Koordinaten nicht mehr benötigen. Weil dies für die vorgesehenen Abfragen jedoch keinen Vorteil brächte (es existieren keine Redundanzen in den Koordinatenpaaren), wurde eine solche Tabelle jedoch nicht eingerichtet.

3.2.2 Schnittstelle zur Datenbank

Oberstes Ziel bei der Entwicklung war es, die Datenbankschnittstelle so einfach wie möglich zu halten. Dies konnte erreicht werden, indem nur eine sehr begrenzte Anzahl Funktionen überhaupt bereitgestellt wurde:

- `databaseQuery(<string>)` sendet eine SQL-Abfrage an die Datenbank und liefert das Ergebnis
- `databaseNumRows()` liefert die Anzahl der Zeilen im Ergebnis der letzten Abfrage
- `databaseNextRow()` liefert die nächste Ergebniszeile der letzten Abfrage, oder einen Fehlerwert nach der letzten Zeile

Eigene Funktionen zum Auf- oder Abbau der Datenbankverbindung existieren nicht, da die Schnittstelle dies automatisch durchführt. Zur Initialisierung genügt es, das Include-Modul mit `require_once('database.include.php')` einzubinden.

Lediglich zwischen mehreren Anfragen mit ungewöhnlich großen Ergebnistabellen kann es bei MySQL sinnvoll sein, die alten Ergebnisse freizugeben; hierzu wurde zusätzlich die Funktion `databaseFreeResult()` bereitgestellt.

Außer den genannten ist zuletzt noch die Funktion `databaseEscape(<string>)` Betrachtung wert: Zur Vermeidung der Gefahr von SQL-Injections bedürfen alle vom Nutzer kommenden Daten besonderer Behandlung, bevor sie in SQL-Abfragen eingebaut werden. Diese Funktion übernimmt dies für Zeichenketten.

Abschließend sei betont, dass der Klient dieser Schnittstelle ausschließlich die Anwendungsschicht ist, während die Dialogschicht mit ihren zahlreichen PHP/HTML-Dateien niemals direkt auf sie zugreifen soll (Kapselung).

3.3 Anwendungsschicht

Die Anwendungsschicht besteht aus zahlreichen Include-Modulen, die verschiedene Verantwortungsbereiche haben. Nicht jede Situation erfordert die Benutzung aller Include-Module.

3.3.1 Übersicht zu den Modulen

- `constants.include.php` enthält „Site-Konstanten“, deren Wert von der Umgebung abhängt, in der das System eingerichtet ist. Dazu gehören beispielsweise der Hostname des Datenbankservers und dessen Zugangsdaten, der sog. „Google Maps API Key“ und lokale Pfadangaben.
- `error.include.php` enthält Routinen zur Fehlerbehandlung. Einmal mit `require_once('error.include.php')` eingebunden, werden Fehler abgefangen, die durch PHP oder durch andere Module gemeldet werden. Je nach Einstellung in den „Site-Konstanten“ werden verschiedene Fehlertypen ignoriert, protokolliert oder angezeigt.
- `menu.include.php` gibt das Hauptnavigationenmenü aus. Struktur und Inhalt dieses Menüs werden in einem assoziativen hierarchischen Array in `menu.include.php` festgelegt. Beim Aufruf der Funktion `printMenu()` wird ein Menü in HTML erzeugt, das ausgehend vom aktuellen Pfad automatisch den gerade ausgewählten Menüpunkt sowie ferner alle aufgrund von fehlenden Dateien unerreichbaren Menüpunkte entsprechend ausgezeichnet hat.
- `template.include.php` bietet Funktionen an, welche die immer gleichen Bestandteile der HTML-Seiten ausgeben – DOCTYPE-Deklaration, HTML-Starttag und STYLE-Element, weiterhin Masthead mit Logo, das Menü und einige zusätzliche Verweise.
- `maplink.include.php` erlaubt es, mit einfachen Mitteln einen Hyperlink zu Google Maps einer bestehenden Ergebnistabelle hinzuzufügen. (Siehe Kapitel 4.2)
- `utm_calculator.class.php` ist eine PHP-Klasse, die Koordinaten im UTM-Netz in geographische Koordinaten nach Länge und Breite umrechnen kann und auch die umgekehrte Richtung beherrscht. Integriert ist eine vollständige Implementierung der Breiten-Zonen im UTM-System, einschließlich der norwegischen Ausnahmen.
- `stop-selection.js` setzt die Auswahl einer Haltestelle per interaktiver Google Maps-Karte um. Dem Nutzer wird eine Karte mit allen Haltestellen präsentiert und die Möglichkeit gegeben, durch einfachen Klick eine davon auszuwählen.
- `stop-selection.php` ist ein PHP-Modul, das `stop-selection.js` unterstützt, indem es verschiedene Datenbankauszüge im XML-Format liefert.

Die Dokumentation der Programmierschnittstelle (API) der Anwendungsschicht erfolgt überwiegend durch phpdoc-Kommentare an den einzelnen Methoden in den Dateien. Mittels des Werkzeugs phpdoc kann diese Dokumentation in HTML umgewandelt werden.

Soweit die Komplexität einen konzeptuellen Überblick nötig macht, findet sich die Beschreibung in diesem Kapitel. Ausnahme ist das Modul `stop-selection`, das im Kapitel 6.1 ausführlich dokumentiert ist.

3.3.2 dbis.include – die Fassade der Anwendungsschicht

Nachdem die diversen oben genannten Module über mehr als ein halbes Dutzend Dateien verstreut sind und damit eher unübersichtlich sind, bietet es sich an, durch Annäherung an das Façade-Pattern eine einfach zu benutzende Schnittstelle zur Dialogschicht zu schaffen. Dies gelang in `dbis.include.php`.

Für die Dialogschicht reicht es regelmäßig aus, lediglich `dbis.include.php` zu importieren, um die volle Funktionalität der Anwendungs- und Datenbankschicht zur Verfügung stehen zu haben.

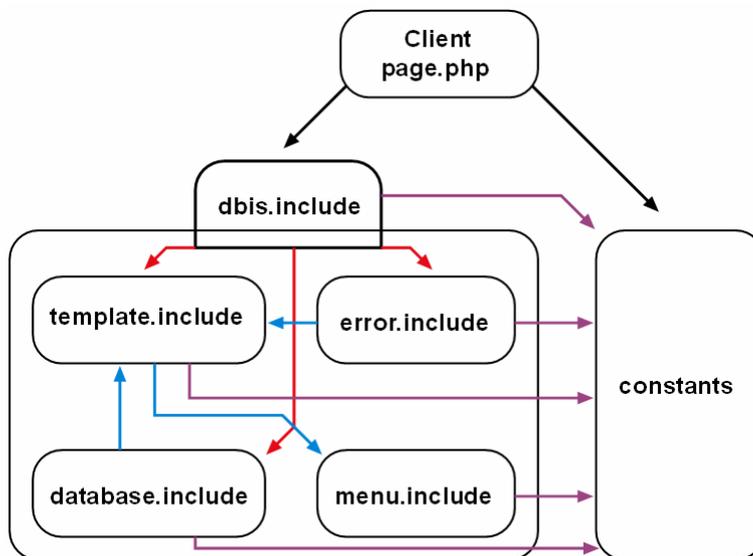


Abb. 3: `dbis.include` bildet als Fassade die Schnittstelle der Anwendungsschicht. Pfeile kennzeichnen „benutzt“-Relationen (z. B. „der Klient benutzt `dbis.include`“).

Die Funktionsaufrufe von `dbis.include.php` werden zum überwiegenden Teil einfach an die zuständigen anderen Module der Anwendungsschicht weitergeleitet.

Auch `dbis.include.php` bietet jedoch zumindest eine interessante Routine: `dbisPrintTableRow(<array>)` gibt einen Array als HTML-Tabellenzeile aus. Spätestens daran ist offensichtlich, dass es sich nicht um eine „echte“ Implementierung des Façade-Patterns nach Gamma et al. handelt, sondern lediglich die Idee aufgegriffen und in angepasster Form umgesetzt wird.

3.4 Dialogschicht

Im Gegensatz zu klassischen Desktop-Anwendungen ist im Kontext der Aufgabenstellung eine Trennung von Programmlogik und graphischer Benutzeroberfläche nicht ohne Weiteres möglich. Die Vorgabe, je Abfrage nur eine PHP-Datei zu verwenden, führt fast zwangsweise zu einer engen Kopplung der Logik mit der Oberfläche.

Dies ist aber bei Web-Skriptsprachen im allgemeinen ohnehin beinahe ein notwendiges Übel, das in vielen Fällen nur unter großen Umständen beseitigt werden kann und daher häufig schlichtweg von Entwicklern ignoriert wird.

In unserem Fall ergibt sich aus dieser Vorgabe der Vorteil einer übersichtlicheren Anzahl von Dateien und einem insgesamt einfacheren Systementwurf, andererseits aber auch der Nachteil einer höheren Komplexität innerhalb der Dateien selbst.

Um diesen Nachteil kontrollieren zu können, bietet sich eine Klasseneinteilung und Schematisierung der Abfragen an.

3.5 Klassifizierung der verschiedenen Abfragen

Die Einteilung der einzelnen Abfragen in Klassen fällt nicht schwer, da die Benutzeroberfläche und Funktionalität direkt auch Auswirkungen auf die Struktur des Quellcodes haben. So kann beispielsweise eine Abfrage, die keine Eingabewerte vom Nutzer bekommt, auf den Code zur Überprüfung solcher Eingabewerte verzichten.

Konkret sind anhand ihrer Ein- und Ausgabewerte und des Grads der Interaktion die folgenden Klassen von Abfragen zu erkennen:

- keine Nutzereingabe
- Nutzereingabe mit SELECT-Element etc. / freie Nutzereingabe
- Eingabe durch interaktive Karte
- Ergebnis ist Ausgabe als Karte

Zusätzlich existiert der Typ der komplexen Abfrage, bei der mehrere SQL-Abfragen durch einen in PHP implementierten Algorithmus aufgerufen werden – zum Teil mehrmals – und zu einer gemeinsamen Ausgabe synthetisiert werden.

Diese Abfrageklassen werden in Kapitel 3 erklärt.

3.6 Semantische Auszeichnung

Die Auszeichnung (engl. „markup“) erfolgt durch die Hypertext Markup Language (HTML) in der Version 4.01 Strict. Diese Entscheidung beruht auf der Feststellung, dass XHTML seinen einzigen Vorteil – nämlich die Möglichkeit, verschiedene Syntaxen wie etwa MathML und SVG im selben Dokument miteinander zu kombinieren – beim vorliegenden Projekt nicht ausspielen kann.

Zu den Nachteilen von XHTML gehören unter anderem:

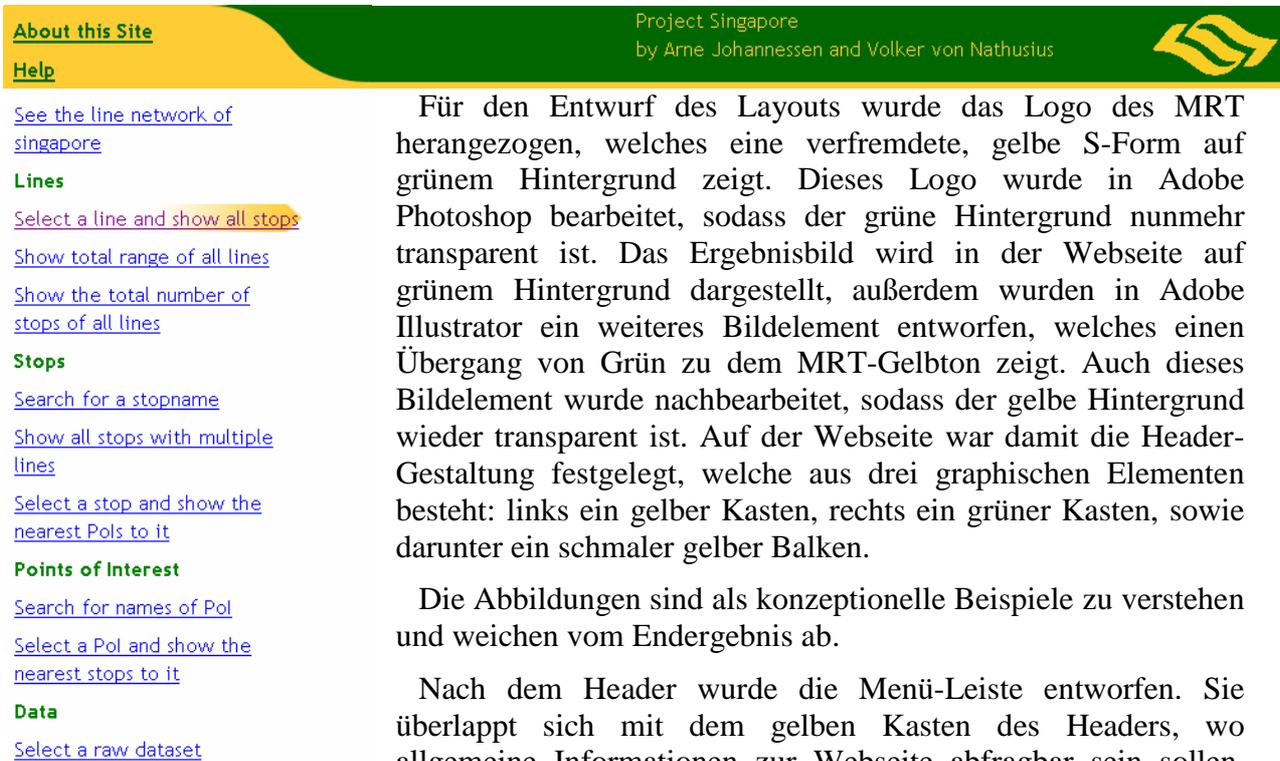
- XHTML ist inkompatibel zum Microsoft Internet Explorer in allen Versionen.
- XHTML benötigt mehr Speicherplatz, weil es im Gegensatz zu HTML keine Elemente mit optionalen Tags gibt (`HTML`, `HEAD`, `P`, `TBODY` etc.) und die Nutzung der SGML-Minimierungsfunktion `ATTRIB OMITNAME` nicht erlaubt ist (`checked=„checked“`).
- XHTML ist nicht abwärtskompatibel (als Vorgänger wird HTML 4 angesehen).
- XHTML ist nicht aufwärtskompatibel (der Nachfolger von XHTML 1 wird XHTML 2 sein).
- XHTML ist weniger robust als HTML (das Einbinden externer Ressourcen, etwa beim „Mashup“ mit Google Maps, bringt nicht kontrollierbare Quellen für well-formedness errors in das eigene XHTML-Dokument).

Unter einem „XHTML-Dokument“ versteht der Fachmann eine Resource, die mit dem Content-Type `application/xhtml+xml` an den User-Agent ausgeliefert wird. Es besteht zwar auch die technische Möglichkeit, ein dem XHTML-Syntax entsprechendes HTML-Dokument auszuliefern, falls es Appendix C–Konformität besitzt; das wird jedoch von den betreffenden W3C-Arbeitsgruppen nicht empfohlen.

Bei der grundsätzlichen Frage, ob Frames in der Webseite auftauchen sollen, entschieden wir uns dagegen. Die Aufteilung der Seite in Header, Menü und Inhalt kann alleine durch CSS bewerkstelligt werden, welches ohnehin für eine einheitliche Darstellung von Stilen benötigt wird. Ferner haben Frames die allgemein bekannten Mängel hinsichtlich Barrierefreiheit und Referenzierbarkeit (z. B. beim Ablegen einer Adresse als Bookmark).

3.7 Graphische Gestaltung der Benutzeroberfläche

Ausgehend von der in HTML erfolgten Auszeichnung des Inhalts konnte selbiger leicht mittels CSS-Formatvorlagen graphisch frei gestaltet werden.



About this Site Project Singapore
by Arne Johannessen and Volker von Nathusius

Help

[See the line network of singapore](#)

Lines

[Select a line and show all stops](#)

[Show total range of all lines](#)

[Show the total number of stops of all lines](#)

Stops

[Search for a stopname](#)

[Show all stops with multiple lines](#)

[Select a stop and show the nearest Pols to it](#)

Points of Interest

[Search for names of Pol](#)

[Select a Pol and show the nearest stops to it](#)

Data

[Select a raw dataset](#)

Für den Entwurf des Layouts wurde das Logo des MRT herangezogen, welches eine verfremdete, gelbe S-Form auf grünem Hintergrund zeigt. Dieses Logo wurde in Adobe Photoshop bearbeitet, sodass der grüne Hintergrund nunmehr transparent ist. Das Ergebnisbild wird in der Webseite auf grünem Hintergrund dargestellt, außerdem wurden in Adobe Illustrator ein weiteres Bildelement entworfen, welches einen Übergang von Grün zu dem MRT-Gelbton zeigt. Auch dieses Bildelement wurde nachbearbeitet, sodass der gelbe Hintergrund wieder transparent ist. Auf der Webseite war damit die Header-Gestaltung festgelegt, welche aus drei graphischen Elementen besteht: links ein gelber Kasten, rechts ein grüner Kasten, sowie darunter ein schmaler gelber Balken.

Die Abbildungen sind als konzeptionelle Beispiele zu verstehen und weichen vom Endergebnis ab.

Nach dem Header wurde die Menü-Leiste entworfen. Sie überlappt sich mit dem gelben Kasten des Headers, wo allgemeine Informationen zur Webseite abfragbar sein sollen.

Darunter folgt eine Liste mit Ergebnissen, die dem Nutzer der Webseite zur Verfügung gestellt werden sollen. Die aktive Seite wird jeweils mit einem gelben Pfeil visuell hervorgehoben, wie in der Abbildung erkennbar.

Der Content der jeweiligen Seite wird schließlich rechts neben dem Menü angezeigt.

Das Layout-Konzept besteht darin, dass der gesamte Seiteninhalt (BODY-Element) einen linken Rand von 19 Geviert hat. In diesen Rand werden dann die Menüs positioniert. Aufgrund der Länge (im Sinne von Höhe) der Menüs kann dies nicht fest (`position:fixed`), sondern muss absolut (`position:absolute`) geschehen (relative oder statische Positionierung wären nicht sinnvoll, weil dabei die Menüs weiterhin Raum im „Flow“ beanspruchen würden, so dass unten im Layout ein großer freier Raum entstünde).

Hinsichtlich Funktionsweise der absoluten Positionierung sei auf Abschnitte 10.3.7 und 10.6.4 („Visual Formatting Model Details“) der CSS-Spezifikation verwiesen; die Barrierefreiheit dieser Positionierungsmethode ergibt sich aus der Abfolge der HTML-Elemente.

Der „Masthead“ mit dem „SINIS“-Schriftzug sowie das „sitemenu“ haben beide eine Höhe von 59 Pixel. Obwohl diese relative Länge nicht von der Schriftgröße, sondern vom Anzeigegerät abhängt (vgl. 4.3.2:6 CSS), stellt dies in der Praxis kein Problem dar, weil bei üblichen Schriftgrößen der vertikale Platz mehr als ausreicht.

Das Layoutkonzept wird umseitig auf einem Bildschirmfoto graphisch angedeutet.

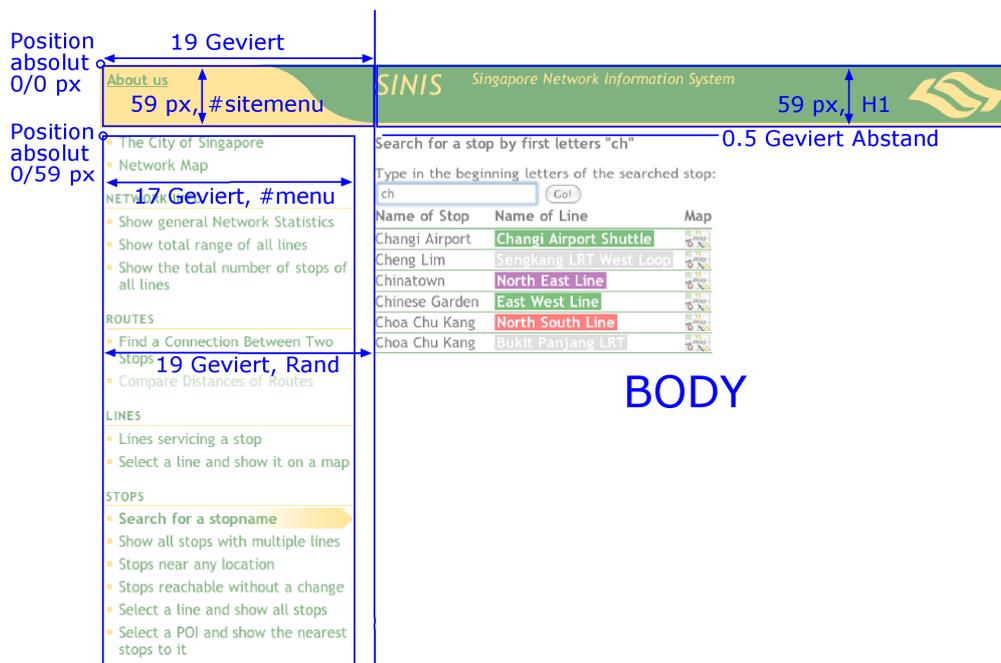


Abb. 5: CSS-Layoutkonzept

3.8 Konventionen

Prinzipiell wurde in PHP folgender Programmier- und Formatierungsstil angewendet:

```
<?php
// PHP-Code
?>
```

PHP-Code ist als solcher deutlich gekennzeichnet durch php-Processing Instructions, die niemals eingerückt stehen.

In PHP gilt grundsätzlich „eine Anweisung pro Zeile“; es kann ein Kommentar folgen. Die Ausnahme hiervon bilden PHP-Teile, die nur eine Zeile umfassen, wie etwa im folgenden Beispiel, wo innerhalb eines HTML-Attributs PHP verwendet wird:

```
<INPUT TYPE="text" NAME="poi" VALUE="<?php echo @$_GET['poi']; ?>">
```

Bedingte Anweisungen und Schleifen werden nach dem Java-Stil geklammert und eingerückt (man beachte die doppelte Einrückung im Falle eines Umbruchs):

```
if (<Bedingung>) {
    <Anweisung>;
    foreach (<array> as <arrayelement>) {
        <Anweisung
            mit
            Zeilenumbrüchen>;
    }
}
```

Optionale geschweifte Klammern werden immer gesetzt:

```
if (<Bedingung>) {
    <Anweisung>;
}
```

SQL-Befehle werden nach den Schlüsselwörtern SELECT, FROM, WHERE, GROUP BY, ORDER BY und LIMIT ausgetrennt. Subselects werden eingerückt:

```
SELECT shl.id_stop AS stop_id, COUNT(*) AS line_count
FROM line, stop, (
    SELECT DISTINCT id_stop, id_line
    FROM stop_has_line
```

```
) AS shl  
WHERE line.id = shl.id_line  
GROUP BY shl.id_stop  
ORDER BY COUNT(*) DESC;
```

Bei Schachtelungen von PHP-Code im Rahmen von HTML wurde im Projekt darauf geachtet, dass der spätere Quellcode der HTML-Seite mit möglichst wenig Einrückungen versehen ist (zwecks besserer Lesbarkeit und kleinerer Dateigröße), aber die vorhandenen Einrückungen pragmatisch korrekt geschachtelt sind. Dass PHP-Code nicht wie HTML-Code eingerückt steht, sondern über eine eigene Einrückungszählung verfügt, ist eine notwendige direkte Folge der korrekten HTML-Einrückung, in Kombination mit den beschriebenen Konventionen für PHP:

```
<UL>  
  <LI><STRONG>[Text ]</STRONG></LI>  
<?php  
while (<Bedingung>) {  
  <Anweisung>;  
}  
?>  
</UL>
```

Variablen, die aus der Datenbank oder vom Nutzer kommende Werte enthalten (insbesondere HTTP-GET-Parameter), müssen vor der Ausgabe mit der Standard-PHP-Funktion `htmlspecialchars` von gefährlichen Zeichen wie `<`, `>`, `&` befreit werden, um das Risiko eines XSS-Angriffs zu vermeiden (Cross-Site-Scripting).

Per Konvention beginnen in diesem Projekt Variablennamen, deren enthaltene Werte sicher nach HTML ausgegeben werden dürfen, in der Regel mit „safe“, während Variablen, deren Werte unbehandelt aus der Datenbank oder aus Nutzereingaben kommen, in der Regel Namen haben, die mit „unsafe“ beginnen (ausführliche „ungarische Notation“).

4 Klassifizierung der Abfragen der Dialogschicht

4.1 Grundstruktur aller Abfragen

Die PHP-Dokumente sind prinzipiell in drei große Blöcke unterteilt. Diese wären:

1. Aufbau der Seite und Abfragen des Formulars (falls vorhanden)
2. Durchführung der SQL-Abfrage
3. Anzeige der Ergebnisse

4.2 Abfrage mit interaktiven Dialogfeldern



Abb. 6: Beispiel für interaktive Dialogfelder

Nachfolgend wird der Aufbau einer interaktiven PHP-Abfrage erläutert. Dabei werden zuallererst `dbis.include.php` und `maplink.include.php` definiert, da von dort einige Ausgabebefehle übernommen werden. Danach wird überprüft, ob möglicherweise gefährliche Eingaben durch den Benutzer getätigt wurden. Schließlich wird der Titel des Browserfensters benannt. Hier handelt es sich um die Datei „`stop_atLine`“, welche von einer ausgewählten Linie alle Haltestellen anzeigen soll. Entsprechend steht nach diesen Zeilen im Browserfenster entweder, dass es sich um ein Auswahlformular (line selection form) handelt, oder welche Linie man ausgewählt hat (Your chosen line: North-South-Line). Schließlich wird mit einer `dbis.include`-Funktion die grüne Titelleiste ausgegeben.

```
<?php
require_once('dbis.include.php');
require_once('maplink.include.php');

/***** data entry and prep *****/

$isAllValuesPresent = array_key_exists('line_selection', $_GET);
$unsafeLine = @dbisHttpGet('line_selection');
$safeLine = htmlspecialchars($unsafeLine);

// print page title and header
if ($isAllValuesPresent) {
    dbisPrintHead('Your chosen line: '.$_GET["line_selection"]);
}
else {
    dbisPrintHead('line selection form');
```

```
}  
?>
```

In diesem Beispiel folgt nun das Eingabeformular für den Benutzer, in mehreren Dateien, die keine Interaktivität erfordern, geht es direkt mit der SQL-Abfrage weiter. Der Nutzer verwendet dieses Formular für die Eingabe. Je nach Abfrage kann dieser Teil aus Checkboxen, Radiobuttons, Eingabefeldern oder Auswahllisten bestehen. Beim Beispiel „stop_atLine“ handelt es sich um ein <SELECT>-Element, welches in einer MySQL-Abfrage die Füllwerte, nämlich die Namen der Linienelemente, selektiert. Anschließend wird die Selektion mit einer While-Schleife durchlaufen, und jeder Zeile der Ausgabetable als Auswahloption in das Selektionsmenü eingefügt.

Bemerkenswert ist das leere ACTION-Attribut des FORM-Elementes: Die URI-Spezifikation sieht in Abschnitt 4.2 „Same-document References“ ausdrücklich vor, dass auch im Falle des FORM-Elementes in HTML ein leerer Wert für dasselbe Dokument steht. Wörtlich: „[I]f the URI reference occurs in [...] HTML's FORM element, then an empty URI reference represents the base URI of the current document and should be replaced by that URI when transformed into a request.“ Dieses Verhalten wird auch so von den Browsern implementiert.

```
<FORM METHOD="GET" ACTION="">  
  <LABEL>Select the line you want to show<BR>  
  <SELECT NAME="line_selection" SIZE="1" onChange="submit()">  
<?php  
// following query fills the selection menu with the necessary data for the later query  
dbisQuery("SELECT name FROM line ORDER BY name;");  
while ($row = dbisNextRow()) {  
  $name = htmlspecialchars($row['name']);  
  echo DBIS_HTML_INDENT, DBIS_HTML_INDENT, '<OPTION VALUE="', $name;  
  if ($safeLine == $name) {  
    echo '" SELECTED>';  
  }  
  else {  
    echo '">';  
  }  
  echo $row['name'];  
  echo "</OPTION>\n";  
}  
?>  
  </SELECT></LABEL>  
  <INPUT TYPE='submit' VALUE='Go! '>  
</FORM>
```

Falls noch kein Element ausgewählt wurde, soll die HTML-Ausgabe der Seite hier beendet werden, wofür die Funktion dbisPrintMenu von menu.include.php via der internen Verlinkung von dbis.include.php aufgerufen wird.

```
// only proceed with the actual DB query if we have all necessary info  
if (! $isAllValuesPresent) {  
  dbisPrintMenu();  
  exit();  
}
```

Andernfalls soll mit der SQL-Abfrage fortgefahren werden, die den zweiten Teil des Dokuments darstellt. Dabei werden die gewählten Parameter von PHP ausgelesen und als SQL-Variablen an die Datenbank übergeben. Nun wird der Datenbank die Abfrage nach genau diesen Variablen befohlen, wozu die normale SQL-Syntax verwendet wird.

```
// input processing
$line = dbisEscape($unsafeLine);
dbisQuery("SET @input = '$line'");

// run the query
dbisQuery("
SELECT stop_has_line.`order` + 1 as `order`, stop.name AS stopname,
line.name AS linename, line.color as linecolor, stop_has_line.distance as distance,
stop_has_line.duration as duration, stop.utm_e AS utm_e, stop.utm_n AS utm_n
FROM stop_has_line, stop, line
WHERE line.id=stop_has_line.id_line AND stop.id=stop_has_line.id_stop AND line.id = (
    SELECT id
    FROM line
    WHERE name = @input OR id = @input
)
ORDER BY stop_has_line.`order`;
");
```

Mit einer kurzen Prüfung wird vor der Ausgabe noch abgefragt, ob ein leerer Datensatz als Ergebnis der SQL-Abfrage herausgekommen ist. In diesem Fall soll die HTML-Ausgabe der Seite hier mit einer knappen Fehlermeldung beendet werden, und nicht etwa ein leerer Tabellenkopf ohne Tabellenkörper ausgegeben werden. Zur Fehlermeldung wird die Standard-PHP-Funktion `trigger_error()` <nachricht>, <typ> verwendet, der Fehler wird daraufhin von `error.include` abgefangen und – abhängig von den Einstellungen in `constants.include` – dort behandelt.

```
// defend against empty results
if (dbisNumRows() <= 0) {
    trigger_error('Line "'.$safeLine.'" does not exist', E_USER_ERROR);
}
```

Falls die Abfrage hingegen ein Ergebnis gebracht hat, wird, im dritten Teil der PHP-Seite, die Ausgabe in Form einer Tabelle gestartet. Die Tabellenköpfe wurden manuell erstellt. Dies ist zwar weniger elegant als die Definition der Tabellenköpfe bereits in SQL und ein anschließendes Auslesen via PHP, doch es reduziert das Fehlerpotential während der SQL-Abfrage (etwa aufgrund von Leer- und Sonderzeichen im Variablennamen etc.) enorm. Wie aus der obigen SQL-Abfrage ersichtlich, wurden die Tabellenköpfe trotzdem aussagekräftig benannt, und mit ihnen soll auch später in der Ausgabe gearbeitet werden.

Es werden Vorkehrungen für Barrierefreiheit (SCOPE- und ABBR-Attribute), schnelles Laden und Rendern (COL-Elemente) sowie vollständige Semantik (THEAD-, TH-, TD-Elemente) und scrollbare Tabellen (THEAD- und TBODY-Elemente) getroffen. (Letzteres wird allerdings von gängigen graphischen User-Agents nicht implementiert und kann auch nicht durch CSS emuliert werden.)

```
// make query results into a table
?>
<TABLE>
  <COL><COL><COL><COL><COL><COL>
<THEAD>
  <TR>
    <TH></TH>
    <TH SCOPE="col" ABBR="Stop">Name of Stop</TH>
    <TH SCOPE="col" ABBR="Line">Name of Line</TH>
    <TH SCOPE="col">Distance</TH>
    <TH SCOPE="col">Duration</TH>
    <TH SCOPE="col">Map</TH>
  </TR>
</TBODY>
```

Nach dieser Definition der Tabellenköpfe wird über PHP die Tabelle zeilenweise ausgegeben. Allerdings kann es vorkommen, dass Zahlen im Ergebnis auftauchen. Die Zahlen werden standardmäßig linksbündig ausgegeben, was jedoch gerade bei sehr großen Zahlenwerten äußerst unschön ist und sofort auffällt.

Für den Fall, dass Zahlen im Ergebnis auftauchen, soll also dafür gesorgt werden, dass sie rechtsbündig ausgegeben werden. Dafür wird über `dbis.include.php` eine zuvor definierte Funktion von `menu.include.php` aufgerufen, welche eine Klasse `numericvalue` vergibt, welche in CSS als rechtsbündig definiert wurde. Diese gibt für jede Reihe zunächst ein `TableRow`-Tag aus. In jedem `TableRow`-Tag werden dann Zellen definiert (`<TD>`), denen jeweils eine Klasse zugewiesen wird. Falls an dieser Stelle im PHP-Dokument ein numerischer Wert definiert worden ist, steht hier später im Quelltext `<TD CLASS = "numericvalue">`, was von der CSS-Datei erkannt wird und darum rechtsbündig angeordnet wird.

Die Klasse „noentities“ wird hingegen im späteren Verlauf des Seitenaufbaus gelöscht, weil sie nicht in HTML oder CSS benötigt wird: Da die Spalten `maplink` (mit Links nach Google Maps), `distance`, `duration` und `linename` HTML-Tags enthalten, darf bei ihnen keine automatische Umwandlung von Sonderzeichen wie `<` und `>` in Entities erfolgen; die Klasse „noentities“ legt dieses Verhalten fest.

```
$columnClass = array(
    'order' => 'numericvalue',
    'distance' => 'numericvalue noentities',
    'duration' => 'numericvalue noentities',
    'linename' => 'linename noentities',
    'maplink' => 'noentities');
```

Bei der zeilenweisen Ausgabe, die in einer `while`-Schleife erfolgt, wird vor der eigentlichen Zeile noch der Tabelleninhalt formatiert: die Spalte, in der der Liniename (`linename`) gespeichert wurde, soll die Farbe erhalten, die für die entsprechende Linie in der Datenbank gespeichert wurde. Da dies ansonsten keinerlei Semantik beinhaltet, wird dies mit einem semantisch neutralen `SPAN`-Element durchgeführt. Anschließend wird die Farbspalte aus der Ausgabe gelöscht (`unset`), und die Ganzzahlwerte von Dauer und Distanz bekommen Einheiten angehängt.

Mit der Funktion `convertCoordinatesToMapLink` wird anschließend eine Funktion von `maplink.include.php` aufgerufen, welche am Anfang des Dokuments eingebunden wurde. Diese Funktion konvertiert die in den Spalten `utm_e` und `utm_n` ausgegebenen UTM- in geographische Koordinaten und erstellt einen Link zu den Seiten von GoogleMaps, welcher die Form eines kleinen Kartenbildchens hat. Die Spalten `utm_e` und `utm_n` werden direkt in dieser Funktion mit `unset` gelöscht.

```
while ($row = dbisNextRow()) {
    $row['linename'] = '<SPAN STYLE="background-color:
'.htmlspecialchars($row['linecolor']).' ">'.htmlspecialchars($row['linename']).'</SPAN>';
    unset($row['linecolor']);
    $row['distance'] .= ' <ABBR TITLE="meter">m</ABBR>';
    $row['duration'] .= ' <ABBR TITLE="minutes">min</ABBR>';
    convertCoordinatesToMapLink($row, 'utm_e', 'utm_n');
    dbisPrintTableRow($row, $columnClass); // output one table row
}
?>
```

Am Ende der Seite wird schließlich das Menü eingebunden, wobei es sich abermals um die `menu.include`-Funktion handelt, die zuvor nur beim Abbruch aufgrund von fehlender Nutzereingabe oder Fehlermeldungen aufgerufen werden sollte.

```
<?php
dbisPrintMenu();
?>
```

Nach diesem Muster sind die meisten PHP-Seiten unseres Projekts aufgebaut, weswegen in den einzelnen Seitendokumentationen i.d.R. nur noch folgende zwei Blöcke dokumentiert werden: Das interaktive Formular und die SQL-Abfrage.

4.3 Statische Abfrage

The screenshot shows the SINIS (Singapore Network Information System) interface. On the left, there are navigation menus for 'About us', 'NETWORK INFO', and 'ROUTES'. The 'Show total range of all lines' option is highlighted. The main content area displays a table titled 'Total length of each line' with the following data:

Name of Line	Total Length
North South Line	43176 m
East West Line	38687 m
North East Line	19221 m
Bukit Panjang LRT	10469 m
Sengkang LRT West Loop	6331 m
Changi Airport Shuttle	5972 m
Punggol LRT East Loop	5275 m
Sengkang LRT East Loop	4418 m

Abb. 7: Beispiel für eine statische Abfrage

Die statische Abfrage ist, wie bereits oben angeklungen ist, ein Sonderfall unserer interaktiven Abfrage, da hier kein Formular verwendet wird, und darum auch keine SQL-Variablen verwendet werden können. Die oben gegebene Beschreibung einer Abfrage mit interaktiven Dialogfeldern soll darum genügen. Die zugrundeliegenden SQL-Befehle können in den einzelnen Seitendokumentationen gefunden werden.

4.4 Abfragen mit interaktiver Karte

The screenshot shows the SINIS interface with the 'Lines servicing a stop' query selected. A map of the Sengkang area is displayed with a red dot indicating the selected stop. Below the map, the following lines are listed as servicing the stop:

- North East Line
- Sengkang LRT West Loop
- Sengkang LRT East Loop

Abb. 8: Beispiel für eine Abfrage mit interaktiver Karte

Abfragen mit interaktiver Karte bieten sich vor allem zur Eingabe einer Position an. Zwar ist es auch grundsätzlich möglich, linienhafte und flächenhafte Objekte mit fast beliebiger Komplexität in Google Maps anzuzeigen, auswählen zu lassen und sogar von Nutzer selbst zeichnen zu lassen, allerdings werden dabei leider die dazu nötigen JavaScript-Skripte leicht ebenfalls fast beliebig komplex.

Überdies ist die Zahl der Linien wesentlich kleiner als die Zahl der Haltestellen, während gleichzeitig sich häufig sowohl Linien als auch Haltestellen räumlich jeweils untereinander eng beieinander befinden. Infolgedessen ist zur Auswahl von Linien eine Auswahlliste oder ein Pop-Up-Menü deutlich besser geeignet als allein eine Karte.

Auch bieten sich Karten selbstverständlich zur Darstellung beliebiger Ergebnisse an. Leider ist aufgrund der Datenlage eine lagetreue Visualisierung des Linienverlaufs nicht möglich. Eine ansprechende und nicht verwirrende Darstellung wird nur bei kleinen Maßstäben erreicht.

In der Konsequenz ist eine Karte im Rahmen dieses Projekts vor allem geeignet

- zur Eingabe einer Position,
- zur Darstellung einer Linie (vgl. Kapitel 4.5),
- zur Auswahl eines Punktes und
- zur Darstellung eines Punktes (vgl. Kapitel 4.2).

Letzteres wird umgesetzt, wo immer es möglich ist und ist fester Bestandteil aller Abfrageklassen (siehe Klasse „Abfrage mit interaktiven Dialogfeldern“).

Eingabe einer Position und Auswahl eines Punktes ist konzeptionell im Wesentlichen identisch; der einzige Unterschied ist, dass auszuwählende Punkte zwangsläufig zuerst einmal dargestellt werden müssen, bevor der Nutzer einen davon auswählen kann. An dieser Stelle wird als repräsentatives Beispiel für diese Klasse die Abfrage `line_atStop.php` erläutert.

Anfang und Ende dieser Abfrage sind bis auf unwesentliche Unterschiede identisch zur Klasse der Abfragen mit interaktiven Dialogfeldern. Anstelle eines HTML-Formulars mit Dialogfeldern wird jedoch das DOM für die Karte vorbereitet.

Im ersten Schritt wird dabei das Laden des Moduls „stop-selection“ seitens des Browsers angestoßen. In HTML sind SCRIPT-Elemente auch als Kinder des BODY-Elements erlaubt. Eine detaillierte Beschreibung der Interaktionen mit dem Skript findet sich im Kapitel „Google Maps–Einbindung“.

```
<SCRIPT SRC="stop-selection.js" TYPE="text/javascript"></SCRIPT>
```

Durch die Vorgabe per HTML werden ins DOM per default drei benachbarte Elemente mit den IDs `legendabove`, `selectionmap` und `out` eingefügt.

Ersteres ist ein leerer Absatz, der lediglich eine minimalistische Legende zusammen mit einer Handlungsaufforderung enthalten soll: „klicken Sie auf eine Haltestelle!“ Weil aber das Klicken in die Karte bei abgeschaltetem JavaScript nicht möglich ist, wird dieses Element erst durch JavaScript mit Inhalt gefüllt.

```
<P ID="legendabove"></P>
```

Das zweite Element („`selectionmap`“) soll die Karte beinhalten. Google Maps ersetzt den Inhalt dieses Elements per DOM-Manipulation mit allem, was für die Karte benötigt wird. Daher ist es möglich, einen Ersatzinhalt für den Fall, dass JavaScript oder Google Maps nicht zur Verfügung stehen, innerhalb dieses Elements zu platzieren („`graceful degradation`“). In unserem Fall handelt es sich dabei um ein einfaches HTML-Formular, welches das Problem kurz erklärt und die exakte Eingabe des Namens der Haltestelle erwartet.

Besser wäre es, hier ein `stop_nameQuery.php` (siehe Kapitel 5.8) ähnelndes Verhalten anzubieten, so dass die Eingabe des Namens nicht exakt zu erfolgen hat. Dies wurde jedoch aus Zeitgründen nicht implementiert.

```
<DIV ID="selectionmap">
  <FORM ACTION="" METHOD="GET">
    <P><LABEL>Name of stop: <INPUT TYPE="text" NAME="stopname" VALUE="<?php echo
$safeStopName; ?>"></LABEL></P>
    <P><INPUT TYPE="submit" VALUE="Go!"><INPUT TYPE="hidden" NAME="noscript" VALUE="1">
  </FORM>
  <P>Normally, an interactive map would appear in this space. However, your browser or
JavaScript engine appears to be <STRONG>incompatible with Google Maps,</STRONG> or there is no
connection to Google's server. Therefore we offer the simple Web form above as alternative
means to run this page's query.</P>
```

```
<P>Note that you have to enter the stop's name using the correct spelling here. You may  
mix case as you like though.</P>  
</DIV>
```

Jegliche Resultate, die nach erfolgtem SQL-Query an den Nutzer ausgegeben werden sollen, werden per „Ajax“ (Asynchronous JavaScript and XMLHttpRequest) nachgeladen und dann im DOM verankert. Am einfachsten geschieht letzteres wie oben durch den Ersatz des Inhalts eines bestehenden DOM-Nodes. Hierfür wird ein leerer, semantisch neutraler Container (DIV) bereitgestellt („out“).

```
<DV ID="out"></DIV>
```

Weil sich laut Entwicklungsvorgabe der gesamte Code zur selben Abfrage in der Dialogschicht in derselben Datei befinden soll, erfolgt die Bereitstellung der Daten für den Container „out“ dadurch, dass das JavaScript-Modul `stop-selection` zum Nachladen der Resultate per Ajax genau diese Abfragendatei `line_atStop.php` wieder aufruft, jedoch im Gegensatz zum Aufruf über das Menü durch den Nutzer nun mit einem HTTP-Query-String, der den Namen der vom Nutzer gewählten Haltestelle enthält. Anhand dieses Namens und der PHP-Variable `$isAllValuesPresent` können beide Zustände leicht unterschieden werden.

Eine Sonderbehandlung erfolgt allerdings für den Fall, dass JavaScript nicht zur Verfügung steht: Zur Erzeugung der korrekten Ausgabe nach HTML wird im Falle des Absendens des Ersatz-Formulars ein zusätzlicher Query-Parameter namens `noscript` eingeführt. Ist sowohl `noscript` als auch ein Haltestellenname im HTTP-Query enthalten, wird ein Menü zusammen mit dem Ergebnis ausgegeben, während mit JavaScript aufgrund der Ajax-Einbettung des Resultats in die schon existierende Webseite kein zusätzliches Menü ausgegeben werden darf.

```
$isAllValuesPresent = array_key_exists('stopname', $_GET);  
$noScript = array_key_exists('noscript', $_GET) && dbisHttpGet('noscript') == 1;  
  
if (! $isAllValuesPresent || $noScript) {  
    // we're not called as part of an Ajax request  
    // no particular stop has been requested yet  
  
    // [prepare default Google Maps screen and replacement HTML form ...]  
}  
  
if ($isAllValuesPresent) {  
    // the client requested a specific stop  
  
    // [run the SQL query ...]  
    // [put query results into a table ...]  
}  
  
if (! $isAllValuesPresent || $noScript) {  
    // we're not called as part of an Ajax request  
  
    dbisPrintMenu();  
}
```

4.5 Ergebnisausgabe als Karte

Da mit dem Modul `stop-selection` bereits eine leistungsfähige Möglichkeit zur Darstellung einer Google Maps-Karte mit Inhalt zur Verfügung steht, bietet es sich an, dieses Modul auch für andere Karten zu verwenden. Dazu muss allerdings der `onLoad`-Handler des Dokuments überschrieben werden, weil `stop-selection` dort standardmäßig das Nachladen und Darstellen aller Haltestellen in die Wege leitet. Gleichzeitig kann im überschreibenden `onLoad`-Handler für andere Anfangsdarstellungen gesorgt werden.

```
<SCRIPT TYPE="text/javascript" SRC="stop-selection.js"></SCRIPT>  
<SCRIPT TYPE="text/javascript"><!--  
window.onload = function () {  
    init("", <?php echo dbisGoogleMapsDefaultView(); ?>);  
    // [additional changes to the default presentation]
```

```
};  
// on click, do nothing  
mapClickListener = function () {  
};  
/--></SCRIPT>
```

Ist ein Abfrage für die Ausgabe ihres Ergebnisses in Form einer Karte mit Google Maps vorgesehen, so ist die Bereithaltung einer barrierefreien Alternative nicht ohne Weiteres möglich. Es wäre eine völlig neue Konzeption der Abfrage notwendig; zum Teil ist eine nicht-visuelle Darstellung der Geodaten sogar überhaupt nicht sinnvoll automatisiert möglich (etwa im Falle eines dargestellten Linienverlaufs). Steht Google Maps nicht zur Verfügung, wird daher nur eine kurze Erklärung zusammen mit einer Entschuldigung ausgegeben.

```
<DIV ID="selectionmap" CLASS="maponly">  
  <P>Normally, an interactive map would appear in this space. However, your browser or  
  JavaScript engine appears to be <STRONG>incompatible with Google Maps,</STRONG> or  
  there is no connection to Google's server. Because the point of this page's query  
  is to display a location map, there is no alternative to be using Google Maps.</P>  
  <P>We're sorry for the inconvenience.</P>  
  <P>The latest versions of Firefox, Opera, Safari and Internet Explorer have all been  
  tested and found to support the Web standards used by this page's query.</P>  
</DIV>  
<DIV ID="out"></DIV>  
<P ID="legendabove"></P>
```

Die eigentliche Ausgabe des Ergebnisses der SQL-Abfrage auf der Karte erfolgt wiederum par JavaScript. Je nach Art der Karte (statisch oder dynamisch) wird das Skript unmittelbar an das HTML-Dokument angehängt und damit sofort beim Laden ausgeführt, oder aber später nach einer Aktion des Nutzers (z. B. Klick auf Knopf in Formular) per Ajax nachgeladen.

Dieses Skript enthält typischerweise eine mit PHP erzeugte Datenliste in Form eines JavaScript-Arrays sowie eine Schleife oder Funktion, mit der diese Datenliste zur Darstellung in der Karte verwendet wird.

Näheres findet sich in den die beiden zu dieser Klasse gehörenden Abfragen beschreibenden Kapiteln 5.8 und 5.17.

4.6 Klassenzugehörigkeit der einzelnen Abfragen

Die Zuordnung der einzelnen Abfragen aus der Dialogschicht zu den hier vorgestellten Klassen wird im nächsten Kapitel vorgenommen. Dort sind auch eventuelle Besonderheiten und Ausnahmen erläutert, so weit sie für den Gesamtprozess oder das Verständnis von Bedeutung sind.

5 Abfragen der Dialogschicht

5.1 singapore.php: statische Abfrage

Diese Seite ist als Start- und Begrüßungsseite vorgesehen. Gezeigt wird ein Bild der Skyline sowie ein einführender Text in englischer Sprache. Die meisten Textpassagen stammen aus der englischen Wikipedia, wurden aber z.T. umformuliert, um auf das Projekt zu passen.

Es wurden hier auch zwei Abfragen implementiert: Einerseits wird die Länge des Liniennetzes aus der Datenbank ausgelesen, andererseits die Entfernung Singapurs vom Äquator. Diese Abfragen wurden in den Informationstext eingebaut.

```
$northingResult = dbisQuery("
    SELECT AVG(utm_e) AS average_e, AVG(utm_n) AS average_n
    FROM (
        SELECT utm_e, utm_n
        FROM stop
        UNION
        SELECT utm_e, utm_n
        FROM poi
    ) AS all_positions;
");

$lengthResult = dbisQuery("
    SELECT SUM(distance) AS total_range
    FROM (
        SELECT MAX(stop_has_line.distance) AS distance
        FROM stop_has_line
        INNER JOIN line ON line.id = stop_has_line.id_line
        GROUP BY stop_has_line.id_line
    ) AS line_range;
");
```

```
/****** print result *****/

$northingRow = dbisNextRow($northingResult);
$geographic = $calculator->toGeographic($northingRow['average_e'],
$northingRow['average_n'], DBIS_UTM_ZONE);
$kmPerDegreeOfLatitude = $calculator->ellipsoidSemiMajorAxis / 1000 * (M_PI / 2 / 90);
$kmNorthOfEquator = $geographic['latitude'] * $kmPerDegreeOfLatitude;

$lengthRow = dbisNextRow($lengthResult);
$kmCurrentNetwork = $lengthRow['total_range'] / 1000;

// make query results into a text
?>

<P><IMG SRC="images/CityView.jpg" CLASS="fullwidth" ALT=""></P>
<P>The <STRONG>Republic of Singapore (Mandarin: Xīnjiāpō)</STRONG> is an island nation
located at the southern tip of the Malay Peninsula. It lies <EM><?php echo
round($kmNorthOfEquator); ?> kilometres north of the equator,</EM> south of the Malaysian
state of
<!--following text not included in the documentation-->
```

Um lizenzrechtliche Probleme zu vermeiden, wurde noch ein Hinweis auf die Lizenzen angefügt, unter denen die Wikipedia das Bild und die Texte veröffentlicht hat. Nötig ist dies, weil die Nutzungsbedingungen der Wikipedia vorschreiben, dass für eine Veröffentlichung von Kopien dieselbe Lizenz genutzt werden soll.

```
<P CLASS="license"><IMG SRC="http://creativecommons.org/images/public/somerights20.png"
ALT=""> The <A HREF="images/CityView.jpg" TYPE="image/png">photograph above</A> was retrieved
on 3 July 2008 <A REL="copyright"
HREF="http://commons.wikimedia.org/wiki/Image:Marina_bay_new_IR.jpg">from Wikimedia
Commons</A>; copyright © 2006 Calvin Teo. It is licensed under the <A REL="license"
HREF="http://creativecommons.org/licenses/by-sa/2.5/">Creative Commons Attribution-Share Alike
2.5 License</A>.</P>
```

```
<P CLASS="license">The text on this page uses material from the Wikipedia articles <A  
REL="copyright"  
HREF="http://en.wikipedia.org/w/index.php?title=Singapore&oldid=222714482">Singapore</A>  
and <A REL="copyright"  
HREF="http://en.wikipedia.org/w/index.php?title=Mass_Rapid_Transit_%28Singapore%29&oldid=2  
2225269">Mass Rapid Transit (Singapore)</A>; copyright © 2001-2008 authors of <A  
HREF="http://en.wikipedia.org/w/index.php?title=Singapore&action=history">Singapore</A>  
and <A REL="copyright"  
HREF="http://en.wikipedia.org/w/index.php?title=Mass_Rapid_Transit_%28Singapore%29&action=  
history">Mass Rapid Transit (Singapore)</A> in Wikipedia, © 2008 Volker von Nathusius, Arne  
Johannessen. Permission is granted to copy, distribute and/or modify this text under the terms  
of the <A REL="license" HREF="http://www.gnu.org/copyleft/fdl.html">GNU Free Documentation  
License</A>, version 1.2 published by the <A HREF="http://www.fsf.org/">Free Software  
Foundation</A>; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A  
<A HREF="singapore-text.LICENSE.txt" TYPE="text/plain">copy of the license</A> is  
available.</P>
```

5.2 map.php: statische Abfrage

Bei dieser Seite wird zwar über die dbis.include-Elemente eine Verbindung zur Datenbank aufgebaut, es werden jedoch keine weiteren Abfragen vorgenommen, sondern lediglich ein derzeit aktueller Liniennetzplan von Singapur präsentiert, der wie das oben gezeigte Foto auf den Seiten von Wikipedia:Commons zur freien Verfügung verlinkt wurde. Auch hier wurde ein lizenzrechtlicher Verweis eingebaut.

```
<P><A HREF="images/NetworkMap.png" TYPE="image/png" TITLE="Show larger version">  
<IMG SRC="images/NetworkMap.png" CLASS="fullwidth" ALT="[Line Network Map of Singapore]">  
</A></P>  
<P><A HREF="images/NetworkMap.png" TYPE="image/png">Show larger version</A></P>  
  
<HR>  
<P CLASS="license"><IMG SRC="http://creativecommons.org/images/public/somerights20.png"  
ALT=""> <A HREF="images/NetworkMap.png" TYPE="image/png">This image</A> was retrieved on 21  
June 2008 <A REL="copyright"  
HREF="http://en.wikipedia.org/wiki/Image:Mrt_lrt_system_map_%28current%29.png">from  
Wikipedia</A>. It is licensed under the <A REL="license"  
HREF="http://creativecommons.org/licenses/by-sa/2.5/">Creative Commons Attribution-Share Alike  
2.5 License</A>.</P>
```

5.3 network_totalRange.php: statische Abfrage

Diese Abfrage liefert die Gesamtlänge des Liniennetzes zurück. Eine Besonderheit an dieser Seite ist es, dass keine Tabelle ausgegeben wird; außerdem werden alle Angaben der Seite in nur einer einzigen Abfrage realisiert, die folglich mehrere Subabfragen beinhaltet. Alle Subqueries liefern einen einzigen Wert zurück, sodass die an die PHP-Seite zurückgelieferte Rohtabelle folgende Spaltennamen aufweist: total_range, line_count, stop_count, interchanges_per_line, stops_per_line und interchanges_percent (die letzteren beiden werden in der Hauptabfrage durch Kombination der vier vorhergehenden Subqueries erstellt). Die Tabelle hat nur eine einzige Spalte.

```
SELECT *, (stop_count / line_count) AS stops_per_line,  
(interchanges_per_line * line_count / stop_count) AS interchanges_percent  
FROM (  
  SELECT (  
    -- total track length  
    SELECT SUM(distance)  
    FROM (  
      SELECT MAX(stop_has_line.distance) AS distance  
      FROM stop_has_line  
      INNER JOIN line ON line.id = stop_has_line.id_line  
      GROUP BY stop_has_line.id_line  
    ) AS line_range  
  ) AS total_range, (  
    -- number of lines  
    SELECT COUNT(*)  
    FROM line  
  ) AS line_count, (  
    -- stop count  
    SELECT COUNT(*)  
    FROM stop_has_line  
  ) AS stop_count,  
  -- interchanges per line  
  SELECT COUNT(*)  
  FROM line  
  WHERE line.interchanges_per_line > 0  
  ) AS interchanges_per_line,  
  -- stops per line  
  SELECT COUNT(*)  
  FROM stop_has_line  
  WHERE stop_has_line.line_id > 0  
  ) AS stops_per_line  
  ) AS network_data
```

```
-- number of stops
SELECT COUNT(*)
FROM stop
) AS stop_count, (
-- average number of interchanges per line
SELECT AVG(line_count)
FROM (
  SELECT COUNT(line_id) AS line_count
  FROM (
    -- full table of interchange stops
    SELECT DISTINCT sh11.id_line AS line_id, sh11.id_stop
    FROM stop_has_line sh11
    INNER JOIN stop_has_line sh12 ON sh11.id_stop = sh12.id_stop
    WHERE sh11.id_line <> sh12.id_line
  ) AS interchange_stops
  GROUP BY line_id
) AS line_interchange_count
) AS interchanges_per_line
) AS results;
");
```

Es folgt ein Fließtext, bei dem jeweils die einzige Zeile aus den Spalten der SQL-Rückgabetable in den Fließtext eingebaut wird.

```
<P>The total length of the whole network of Singapore is <?php echo $results['total_range'];
?> <ABBR TITLE="meter">m</ABBR>.</P>
<P>The total count of stops in the entire network is <?php echo $results['stop_count']; ?>
and the total number of lines is <?php echo $results['line_count']; ?>. That makes for a
computed average of <?php echo round($results['stops_per_line'], 1); ?> stops per line.</P>
<P>An average of <?php echo round($results['interchanges_per_line'], 2); ?> stops per line
are interchange stops; over the entire network, <?php echo
round($results['interchanges_percent'] * 100, 1); ?> % of stops are interchange stops.</P>
```

5.4 network_lineRange.php: statische Abfrage

Auch auf dieser PHP-Seite wird keine dynamische Abfrage geregelt, sondern lediglich die vorher definierte SQL-Abfrage durchgeführt, die die Gesamtlänge aller Linien formatiert ausgibt.

```
SELECT line.name AS linename, MAX(stop_has_line.distance) AS distance,
line.color AS linecolor
FROM stop_has_line, line
WHERE line.id = stop_has_line.id_line
GROUP BY line.name
ORDER BY distance DESC;
```

5.5 line_howManyStops.php: statische Abfrage

Auch diese Seite ist nicht dynamisch. Hier wird angezeigt, welche Linien wie viele Haltestellen haben, was formatiert ausgegeben wird.

```
SELECT line.name AS linename, COUNT(*) AS number, line.color AS linecolor
FROM stop_has_line, line
WHERE line.id = stop_has_line.id_line
GROUP BY id_line
ORDER BY count(*) DESC;
```

5.6 routing.php: Abfrage mit interaktiven Dialogfeldern

Hierbei handelt es sich um die umfangreichste Abfrage im Projekt, welche die in der Projektanalyse erwähnten vier SQL-Abfragen zur Erreichbarkeit von Haltestellen im Liniennetz ersetzen sollte. Der zur Routensuche verwendete Algorithmus wird im Kapitel „Routing-Algorithmus“ erklärt, mitsamt den dazu nötigen Datenbank-Abfragen.

Von der eigentlichen Routensuche selbst abgesehen ist diese Abfrage allerdings ein recht typischer Vertreter ihrer Klasse, weswegen der Aufbau des PHP-Skripts hier nicht viel detaillierter als für die anderen Abfragen in diesem Kapitel beschrieben wird.

Der Dateneingabe- und Vorbereitungsteil ist nicht nur aufgrund des mehrelementigen Formulars etwas umfangreicher, sondern auch, weil sofort die IDs der eingegebenen Haltestellen-Namen für die spätere Verwendung im Routing-Algorithmus ermittelt und in PHP-Variablen gespeichert werden. Grundsätzlich Neues enthält dieser Teil aber nicht.

```
$isAllValuesPresent = array_key_exists('from', $_GET) && array_key_exists('to', $_GET);
if ($isAllValuesPresent) {

    // input processing
    $unsafeFromStopName = dbisHttpGet('from');
    $safeFromStopName = htmlspecialchars($unsafeFromStopName, ENT_NOQUOTES);
    $unsafeToStopName = dbisHttpGet('to');
    $safeToStopName = htmlspecialchars($unsafeToStopName, ENT_NOQUOTES);

    dbisQuery("SELECT id FROM stop WHERE name = '$unsafeFromStopName'");
    if ($row = dbisNextRow()) {
        $fromStopId = $row['id'];
    }
    dbisQuery("SELECT id FROM stop WHERE name = '$unsafeToStopName'");
    if ($row = dbisNextRow()) {
        $toStopId = $row['id'];
    }
}
```

Eine Ausnahme sind die im Formular enthaltenen Schaltflächen zur Anzeige einer Haltestellen-Auswahlkarte, die sehr einfach über das Modul stop-selection realisiert werden konnte. Diese Auswahlkarte wird im DOM im DIV mit der ID mapdialog verankert. Nach Auswahl einer Haltestelle wird der Name der ausgewählten Haltestelle in das Formular-Element, das zuvor durch die BUTTON-Schaltflächen vorgegeben wurde, zurückgeschrieben.

```
<BUTTON TYPE="button" onClick="showModalMap(&quot;from&quot;)">Show map</BUTTON>
<BUTTON TYPE="button" onClick="showModalMap(&quot;to&quot;)">Show map</BUTTON>
```

Die Stelle der Datenbankabfrage nimmt nun der Aufruf des Routing-Algorithmus ein:

```
// begin route search
$routes = findRoutes($fromStopId, $toStopId);
```

Die Ergebnisausgabe ist wieder etwas umfangreicher. Zunächst wird, falls vom Nutzer gewünscht, das Ergebnis anhand von vordefinierten und über die URL änderbaren (allerdings nicht auf die Benutzeroberfläche hinaus geführten) Filterparameter auf die besten Ergebnisse beschränkt.

```
$safeFilter = (array_key_exists('filter', $_GET) || !
    (array_key_exists('change-minutes', $_GET) && array_key_exists('from', $_GET)
    && array_key_exists('to', $_GET))) && @dbisHttpGet('filter') != '0';

if ($safeFilter) {
    $routes = filterRoutes($routes);
}
```

Anschließend werden bei den übriggebliebenen Verbindungen IDs in Namen umgewandelt (aus Effizienzgründen findet diese Umwandlung nicht wie hier gezeigt in SQL, sondern in PHP statt):

```
SELECT name FROM stop WHERE id = '{ $leg['from'] }'
```

Die eigentliche Ausgabe der Routen unterscheidet sich von den anderen Abfragen lediglich darin, dass statt einer Tabelle hier eine Liste von Tabellen ausgegeben wird, auf die jeweils noch eine einzeilige Zusammenfassung folgt. In der Zusammenfassung wird darauf geachtet, dass der Numerus in der englischen Sprache korrekt wiedergegeben wird (nicht „1 changes“).

Ferner wird dbisPrintTableRow hier zusätzlich eine Einrückungsinformation in \$indent mitgegeben, so dass das HTML trotz der zusätzlichen Schachtelung der Tabellen in eine Liste korrekt eingerückt ist.

```
<UL>
<?php

$columnClass = array(
    'distance' => 'numericvalue noentities',
    'duration' => 'numericvalue noentities',
    'line' => 'linename noentities');
$indent = DBIS_HTML_INDENT.DBIS_HTML_INDENT.DBIS_HTML_INDENT;

foreach ($routes as &$route) {
?>
    <LI>
        <TABLE>
            <COL><COL><COL><COL><COL>
            <THEAD>
                <TR>
                    <TH SCOPE="col">From</TH>
                    <TH SCOPE="col">To</TH>
                    <TH SCOPE="col" ABBR="on">On Line</TH>
                    <TH SCOPE="col" ABBR="for">Travel Time</TH>
                    <TH SCOPE="col">Distance</TH>
                </TR>
            <TBODY>
        <?php

            foreach ($route['legs'] as &$leg) {
                dbisPrintTableRow($leg, $columnClass, $indent);
            }

        ?>
        </TABLE>
        <P><?php echo 'Total journey time ', $route['duration'], ' minute',
(count($route['duration']) == 2 ) ? ', ' : 's, ', count($route['legs']) - 1, ' change',
(count($route['legs']) == 2 ) ? '.' : 's.'; ?></P>
        </LI>
    <?php
    }
?>
</UL>
```

Näheres zu den hier aufgerufenen, im selben Dokument enthaltenen PHP-Funktionen ist im Kapitel „Routing-Algorithmus“ dokumentiert.

5.7 line_atStop.php: Abfrage mit interaktiver Karte

Mittels Google Maps bietet diese Abfrage dem Nutzer die Möglichkeit zur Auswahl einer Haltestelle. Es werden daraufhin alle Linien ermittelt, die an dieser Haltestelle halten.

```
SET @input = '$stopname';

SELECT DISTINCT line.name AS linename, line.color AS color
FROM line, stop, stop_has_line
WHERE stop.name = @input
      AND stop.id = stop_has_line.id_stop AND line.id = stop_has_line.id_line;
```

5.8 line_map.php: Ergebnisausgabe als Karte

Zur Darstellung als Karte wird auf das stop-selection-Modul zurückgegriffen und das Verhalten bei Start so angepasst, dass die Karte weder im Maßstab noch im Ausschnitt verändert werden kann. Diese Einschränkung ist sinnvoll, da die Linien nicht über ausreichend viele Stützpunkte für eine lagetreue Darstellung verfügen.

```
// display no stops, disallow view change
window.onload = function () {
    init("", <?php echo dbisGoogleMapsDefaultView(); ?>);
    map.removeControl(zoomControl);
    map.disableDoubleClickZoom();
    map.disableDragging();
};
```

Um die Darstellung anstoßen zu können, wird dem Nutzer folgendes Formular per HTML bereitgestellt:

```
<SCRIPT TYPE="text/javascript"><!--  
// handle the form  
function lineSelected (button) {  
  if (googleMapsEnabled) {  
    var cuddleIe7 = "&nbsp;";  
    retrieveAndPrintUri("?id="+button.value, cuddleIe7);  
  }  
}  
//-->  
</SCRIPT>  
  
<FORM ACTION="http://none.invalid/">  
  <P>Select the line you want to see:</P>  
  <P>  
    <INPUT TYPE="radio" NAME="line_selection" ID="line-75" VALUE="75"  
      onClick="lineSelected(this)"> <LABEL FOR="line-75" CLASS="linename"  
        STYLE="background-color: silver">Bukit Panjang LRT</LABEL><BR>  
    <INPUT TYPE="radio" NAME="line_selection" ID="line-76" VALUE="76"  
      onClick="lineSelected(this)"> <LABEL FOR="line-76" CLASS="linename"  
        STYLE="background-color: green">Changi Airport Shuttle</LABEL><BR>  
    <!-- [...] -->  
  </P>  
</FORM>
```

Die Ergebnisdarstellung funktioniert ohnehin nur mit JavaScript, weswegen Skriptfähigkeit hier vorausgesetzt werden und auf einen herkömmlichen Submit-Button verzichtet werden darf. Wird auf einen Radio-Button geklickt, so wird mit dessen ID das Nachladen und Ausführen des zur Darstellung der Linie nötigen JavaScripts angestoßen. Dies geschieht durch die bereits in `stop-selection` vorhandene Funktion `retrieveAndPrintUri(<uri>, <textoutput>)`.

Aufgrund eines Bugs im Internet Explorer 7 muss zusätzlicher Inhalt ausgegeben werden, damit das Skript ins DOM eingefügt und ausgeführt werden kann. Hierzu wird ein nicht umbrechendes Leerzeichen verwendet.

Das oben gezeigte HTML-Formular ist natürlich nicht statisch, sondern wird zur Laufzeit aus der Datenbank mittels PHP erzeugt. Die Verwendung von `DBIS_HTML_INDENT` dient einer konsistenten Einrückung (mit gleich vielen Tabs oder Leerzeichen) im HTML-Code; in der Auslieferungsversion ist diese Konstante mit drei Leerzeichen vorbelegt.

```
// following query fills the selection list with the necessary data for the later query  
dbisQuery("SELECT id, name, color FROM line ORDER BY name;");  
while ($row = dbisNextRow()) {  
  $id = (int)$row['id'];  
  echo DBIS_HTML_INDENT, DBIS_HTML_INDENT;  
  echo '<INPUT TYPE="radio" NAME="line_selection" ID="line-', $id, ' VALUE=', $id;  
  echo ' onClick="lineSelected(this)"> <LABEL FOR="line-', $id;  
  echo ' CLASS="linename" STYLE="background-color: ', $row['color'], '>',  
    htmlspecialchars($row['name']), "</LABEL><BR>\n";  
}
```

Das nachgeladene Skript, welches die Kartendarstellung der Linie erzeugt, füllt zunächst einen JavaScript-Array mit den Positionen der Linienstützpunkte (also in unserem Fall mit denen der Haltestellen). Dieser Array wird dann verwendet, um zwei Polylinien so darzustellen, dass sich eine Linearsignatur mit weißem, leicht transparentem Rand und einer der in der Datenbank gespeicherten Linienfarbe entsprechenden Füllfarbe ergibt.

Die Darstellung der tatsächlichen Linienfarbe in Form eines CSS-Farbnamens (wie im Datenmodell vorgesehen) wird leider nicht von Google Maps „native“ unterstützt; es ist eine Rendering-Engine erforderlich, deren SVG- oder VML-Fähigkeiten von Google Maps unterstützt werden. Zu diesem Zeitpunkt sind dies lediglich Gecko (z. B. Firefox) und Trident (z. B. Internet Explorer für Windows). In allen anderen Browsern kommt die Google Maps–

Default-Darstellung in Form einer hellblauen Linie zur Anwendung, was als sehr guter Ersatz angesehen wird.

```
<SCRIPT TYPE="text/javascript">
var lineData = [];
lineData.push(new GLatLng(1.334125, 103.741798));
lineData.push(new GLatLng(1.349316, 103.749915));
lineData.push(new GLatLng(1.358968, 103.751923));
// [...]

map.clearOverlays();
map.addOverlay(new GPolyline(lineData, "#ffffff", 10, .75)); // outline
map.addOverlay(new GPolyline(lineData, "red", 6, 1)); // line
// :BUG: named colors degrade gracefully to blue if neither SVG nor VML is supported
</SCRIPT>
```

Zur Erzeugung dieses Skripts dient wiederum jeweils eine SQL-Datenbank-Abfrage mit anschließender Ausgabe in PHP für die Linienfarbe und die Linien-Koordinaten:

```
<?php
/***** run database query *****/
$safeId = (int)@dbisHttpGet('id');
dbisQuery("SET @input = '$safeId'");
$colorResult = dbisNextRow(dbisQuery("
    SELECT color
    FROM line
    WHERE id = @input;
"));
dbisQuery("
    SELECT stop.utm_e AS utm_e, stop.utm_n AS utm_n
    FROM stop
    INNER JOIN stop_has_line ON stop.id = stop_has_line.id_stop
    WHERE stop_has_line.id_line = @input
    ORDER BY stop_has_line.`order`;
");

/***** show result on map *****/
?>
<SCRIPT TYPE="text/javascript">
var lineData = [];
<?php
$calculator = new UtmCalculator;
while ($row = dbisNextRow()) {
    $geographic = $calculator->toGeographic($row['utm_e'], $row['utm_n'], DBIS_UTM_ZONE);
    printf("lineData.push(new GLatLng(%08.6f, %08.6f));\n",
        $geographic['latitude'], $geographic['longitude']);
}
?>

map.clearOverlays();
map.addOverlay(new GPolyline(lineData, "#ffffff", 10, .75));
map.addOverlay(new GPolyline(lineData, "<?php echo $colorResult['color']; ?>", 6, 1));
</SCRIPT>
```

5.9 stop_nameQuery.php: Abfrage mit interaktiven Dialogfeldern

Diese einfache Abfrage erlaubt dem Nutzer die Suche nach dem Namen einer Haltestelle, anhand der Anfangsbuchstaben, die in ein Eingabefeld eingegeben werden können.

```
<FORM METHOD="GET" ACTION="">
  <LABEL>Type in the beginning letters of the searched stop:</LABEL><BR>
  <INPUT TYPE="text" NAME="stop" VALUE=""<?php echo @$_GET['stop']; ?>>
  <INPUT TYPE="submit" VALUE="Go!">
</FORM>
```

Mit einem LIKE-Operator werden sie mit den Haltestellennamen in der Datenbank verglichen, und dann gemeinsam mit der dazugehörigen Linie ausgegeben. Es sollen keine doppelten Einträge ausgegeben werden, falls ein Punkt (aufgrund von Schleifen) zweimal auf der gleichen Linie liegt.

```
SELECT DISTINCT s.name AS name, l.name as linename, l.color as linecolor,
    s.utm_e AS utm_e, s.utm_n AS utm_n
FROM stop s, line l, stop_has_line shl
WHERE s.name LIKE @input AND l.id = shl.id_line AND s.id = shl.id_stop
ORDER BY s.name;
```

5.10 stop_howManyLines.php: Abfrage mit interaktiven Dialogfeldern

Auf dieser Seite wird ermittelt, welche Haltestellen von den meisten Linien angefahren werden. Da es im gesamten Netz von Singapur nur 9 Umsteigehaltestellen unter 100 Haltestellen gibt, erschien es nicht sinnvoll, dem Nutzer hier eine Auswahlmöglichkeit zu geben, sondern stattdessen einfach alle Haltestellen mit mehreren Linien auszuwählen. Es sollte ursprünglich eine statische Abfrage realisiert werden, doch nach längerer Überlegung wurde die Abfrage um eine Checkbox erweitert, die dem Nutzer die Möglichkeit gibt, sich auch die Haltestellen mit nur 1 Linie anzeigen zu lassen. Standardmäßig ist die Checkbox deaktiviert, behält ihren Zustand aber, falls sie aktiviert wird.

Mit JavaScript wird das Formular automatisch abgeschickt, wenn sich der Zustand der Checkbox ändert.

```
<FORM METHOD="GET" ACTION="">
  <P>The list below only lists stops serviced by more than one line per default.
  <BR><LABEL>
  <?php
  echo DBIS_HTML_INDENT, DBIS_HTML_INDENT, DBIS_HTML_INDENT;
  echo 'If you want to show all, check this field:&nbsp;<INPUT TYPE="checkbox" NAME="showAll"
onChange="submit()" ' ;
  if(array_key_exists('showAll', $_GET)) {
    echo "CHECKED>";
  }
  else {
    echo ">";
  }
  ?>
  </LABEL>
  <INPUT TYPE="submit" VALUE="Go!">
</FORM>
```

Die folgende Abfrage unterscheidet nun anhand der Checkbox zwischen den zwei Möglichkeiten. Es werden nicht nur die Haltestellen ausgewählt, sondern auch die dazugehörigen Linien. Doppelte Stopps einer Linie an einer Haltestelle werden nicht ausgegeben (dieser Fehler tritt ansonsten bei den Linien-Schleifen auf).

```
if(array_key_exists('showAll', $_GET)) {
  dbisQuery("
    SELECT DISTINCT stop.name AS name, line_count, line.name AS line_name,
      line.color AS line_color, stop.utm_e AS utm_e, stop.utm_n AS utm_n
    FROM stop_has_line, line, stop, (
      SELECT shl.id_stop AS stop_id, COUNT(*) AS line_count
      FROM line, (
        SELECT DISTINCT id_stop, id_line
        FROM stop_has_line
      ) AS shl
      WHERE line.id = shl.id_line
      GROUP BY shl.id_stop
      ORDER BY COUNT(*) DESC
    ) AS lines_per_station
    WHERE stop.id = stop_id AND id_stop = stop_id AND id_line = line.id
    ORDER BY line_count DESC, name ASC;
  ");
}
else {
  dbisQuery("
    SELECT DISTINCT stop.name AS name, line_count, line.name AS line_name,
      line.color AS line_color, stop.utm_e AS utm_e, stop.utm_n AS utm_n
    FROM stop_has_line, line, stop, (
      SELECT shl.id_stop AS stop_id, COUNT(*) AS line_count
      FROM line, (
```

```
SELECT DISTINCT id_stop, id_line
FROM stop_has_line
) AS shl
WHERE line.id = shl.id_line
GROUP BY shl.id_stop
ORDER BY COUNT(*) DESC
) AS lines_per_station
WHERE stop.id = stop_id AND id_stop = stop_id AND id_line = line.id
AND line_count > 1
ORDER BY line_count DESC, name ASC;
");
}
```

Der einzige Unterschied zwischen den beiden Teilen ist die zusätzliche Zeile im 2. Abschnitt, wo der line-count größer als 1 sein muss. Hier hätte zwar leicht eine weitere Variable eingeführt werden können, welche die Zahl der anzeigbaren Haltestellen regelt, und mit der verglichen wird. Dass die Abfrage trotzdem “doppelt” im Code steht wird, hat mehrere Hintergründe: Sie erlaubt eine bessere Wartbarkeit, ist zudem sicherer als das Hantieren mit einer weiteren Variable, und es ist besser lesbar.

5.11 stop_nearPosition.php: Abfrage mit interaktiver Karte

Diese Abfrage nutzt das stop-selection-Framework zur Darstellung und Handhabung der Karte. Allerdings geht es nicht um die Auswahl einer Haltestelle, sondern vielmehr um die Eingabe einer beliebigen Position; aus diesem Grund werden anfangs keine Haltestellen dargestellt und bei einem Klick wird statt der entsprechenden Auswahl-Routine in stop-selection lediglich unsere eigene SQL-Abfrage durchgeführt (mittels Ajax). Das Verfahren ähnelt dem in der Klasse der Abfragen mit Ergebnisausgabe als Karte (siehe Kapitel 4.5).

```
<SCRIPT TYPE="text/javascript"><!--
var clickOverlay;

// display no stops to begin with
window.onload = function () {
    init("Click on the map to choose a location!",
        <?php echo dbisGoogleMapsDefaultView(); ?>);
};

// on click, just run the SQL query instead of the stop-selection routine
mapClickListener = function (overlay, point) {
    if (point) {
        var resultCaption = "<P>Stops near your location:\n
        <IMG SRC=\"images/pin.png\" WIDTH=\"22\" HEIGHT=\"20\" ALT=\"\"></P>";
        retrieveAndPrintUri("?latitude="+point.lat()+"&longitude="+point.lng(),
            resultCaption);

        // show a marker to indicate click position
        if (clickOverlay) {
            map.removeOverlay(clickOverlay);
        }
        clickOverlay = new GMarker(point,
            new MarkerOptions(undefined, new MarkerIcon("pin"), 5));
        map.addOverlay(clickOverlay);
    }
};

//-->
</SCRIPT>
```

Nach einem Klick auf die Karte wird die Klickposition von der retrieveAndPrintUri-Funktion mittels XMLHttpRequest an dieses PHP-Skript geschickt und dort mit Hilfe des UtmCalculators von geographischen in Datenbank-(UTM)-Koordinaten konvertiert. Der Reply enthält daraufhin eine Tabelle mit den Namen, Abständen und Koordinaten der jeweils nächsten Haltestellen (wie bei interaktiven Karten üblich). Zugrundegelegt wird folgende SQL-Abfrage:

```
SET @easting = '$easting';
SET @northing = '$northing';
SET @range = 2500;

SELECT *
FROM (
  SELECT stop.utm_e AS easting, stop.utm_n AS northing, stop.name AS name,
    round(sqrt(POW((stop.utm_n - @northing), 2) + POW(stop.utm_e - @easting, 2)))
    AS beeline
  FROM stop
  ORDER BY beeline
) AS beeline_all_stops
WHERE beeline <= @range;
```

Außerdem enthält der Reply ein JavaScript, das diese Haltestellen in der Karte darstellt:

```
<SCRIPT TYPE="text/javascript">
if (googleMapsEnabled) {

  var markerData = [];
  <?php
  $calculator = new UtmCalculator;
  foreach ($resultStops as $stop) {
    $geographic = $calculator->toGeographic($stop['easting'], $stop['northing'],
      DBIS_UTM_ZONE);
    echo DBIS_HTML_INDENT,
      'markerData.push({name: "', htmlspecialchars($stop['name']), '", ' ;
    printf("latitude: %08.6f, longitude: %08.6f});\n",
      $geographic['latitude'], $geographic['longitude']);
  }
  ?>

  displayResultData(markerData);
}
</SCRIPT>
```

Die Funktion `displayResultData` gehört dabei zum Modul `stop-selection` und stellt je Haltestelle einen auffälligen Google Map-Marker in der Karte dar.

5.12 stop_reachable.php: Abfrage mit interaktiver Karte

Mittels Google Maps bietet diese Abfrage dem Nutzer die Möglichkeit zur Auswahl einer Haltestelle. Es werden daraufhin alle anderen Haltestellen ermittelt, die von der gewählten Haltestelle aus ohne Umsteigen zu erreichen sind.

```
SET @input = '$stopname';

SET @sid = (
  SELECT id
  FROM stop
  WHERE name = @input
);

SELECT DISTINCT stop.name AS name, line.name AS linename, line.color AS color,
  stop.utm_e AS utm_e, stop.utm_n AS utm_n
FROM stop_has_line, stop, line
WHERE stop_has_line.id_stop = stop.id AND stop_has_line.id_line = line.id
  AND stop_has_line.id_line IN (
  SELECT l.id
  FROM line l, stop_has_line shl
  WHERE l.id = shl.id_line AND shl.id_stop = @sid
)
ORDER BY line.name ASC;
```

Außerdem enthält der Reply ein JavaScript, das diese Haltestellen in der Karte darstellt:

```
<SCRIPT TYPE="text/javascript">
if (googleMapsEnabled) {

  var markerData = [];
  <?php
  $calculator = new UtmCalculator;
  foreach ($resultStops as $stop) {
```

```
$geographic = $calculator->toGeographic($stop['utm_e'], $stop['utm_n'], DBIS_UTM_ZONE);
echo DBIS_HTML_INDENT,
    'markerData.push({name: ', htmlspecialchars($stop['name']), ', ', ' ';
printf("latitude: %08.6f, longitude: %08.6f});\n",
    $geographic['latitude'], $geographic['longitude']);
}
?>

displayResultData(markerData);
}
</SCRIPT>
```

Die Funktion `displayResultData` gehört dabei zum Modul `stop-selection` und stellt je Haltestelle einen auffälligen Google Map-Marker in der Karte dar.

5.13 `stop_atLine.php`: Abfrage mit interaktiven Dialogfeldern

Diese Datei diente bereits als Beispiel für eine Abfrage mit interaktiven Dialogfeldern in Kapitel 3. Eine nochmalige Aufführung erübrigt sich darum.

5.14 `stop_nearPoi.php`: Abfrage mit interaktiven Dialogfeldern

Auf dieser Seite wird ein Formular aufgebaut, welches wie das `stop_atLine.php` eine Auswahlliste zur Verfügung stellt. Ziel ist es, die fünf nächstgelegenen Haltestellen zu einem vom Nutzer ausgewählten Point of Interest herauszusuchen.

Mit JavaScript wird das Formular automatisch abgeschickt, wenn sich der Zustand der Auswahlliste ändert.

```
$isAllValuesPresent = array_key_exists('poi_selection', $_GET);
$unsafePoi = @dbisHttpGet('poi_selection');
$safePoi = htmlspecialchars($unsafePoi);
[...] gekürzt
<FORM METHOD="GET" ACTION="">
    <LABEL>Select a Point of Interest to show the nearest stops<BR>
    <SELECT NAME="poi_selection" SIZE="1" onChange="submit()">
<?php

// following query fills the selection menu with the necessary data for the later query
dbisQuery("SELECT name FROM poi ORDER BY name;");
while ($row = dbisNextRow()) {
    $name = htmlspecialchars($row['name']);
    echo DBIS_HTML_INDENT, DBIS_HTML_INDENT, '<OPTION VALUE="' . $name;
    if ($$safePoi == $name) {
        echo '" SELECTED>';
    }
    else {
        echo '">';
    }
    echo $row['name'];
    echo "</OPTION>\n";
}
?>
</SELECT></LABEL>
<INPUT TYPE='submit' VALUE='Go! '>
</FORM>
```

Über die UTM-Koordinaten werden hier nach dem Pythagorassatz die nächstgelegenen Haltestellen herausgesucht.

```
SELECT @input AS poiname, stop.name AS stopname,
    round(sqrt(POW((stop.utm_n - (SELECT utm_n FROM poi WHERE name = @input)), 2) +
    POW(stop.utm_e - (SELECT utm_e FROM poi WHERE name = @input), 2))) AS beeline,
    utm_e, utm_n
FROM stop
ORDER BY beeline
LIMIT 5;
```

5.15 poi_info.php: Abfrage mit interaktiven Dialogfeldern

Diese Abfrage hat die gleiche Intention wie die zuvorige Abfrage stop_nameQuery.php, allerdings wird hier nicht nur der Name, sondern auch die Art des Point of Interest verglichen.

```
<FORM METHOD="GET" ACTION="">
  <LABEL>Type in the Point of Interest name or type you want to show</LABEL><BR>
  <INPUT TYPE="text" NAME="poi" VALUE="<?php echo $safePoi; ?>">
  <INPUT TYPE="submit" VALUE="Go!">
</FORM>
```

```
SET @input = '%$poi%';
SELECT name, type, utm_e, utm_n
FROM poi
WHERE type LIKE @input OR name LIKE @input
ORDER BY name ASC;
```

5.16 poi_nearStop.php: Abfrage mit interaktiven Karten

Mittels Google Maps bietet diese Abfrage dem Nutzer die Möglichkeit zur Auswahl einer Haltestelle. Es werden daraufhin alle anderen Haltestellen ermittelt, die von der gewählten Haltestelle aus ohne Umsteigen zu erreichen sind.

```
SET @input = '$stopname';

SET @stop_utm_n = (
  SELECT utm_n
  FROM stop
  WHERE name = @input
);
SET @stop_utm_e = (
  SELECT utm_e
  FROM stop
  WHERE name = @input
);

SET @distance_limit = 1000;
SELECT poi.name AS name, poi.type AS type,
  round(sqrt(POW((poi.utm_n - @stop_utm_n), 2) + POW(poi.utm_e - @stop_utm_e, 2)))
  AS distance, poi.utm_e AS utm_e, poi.utm_n AS utm_n
FROM poi
WHERE round(sqrt(POW((poi.utm_n - @stop_utm_n), 2) + POW(poi.utm_e - @stop_utm_e, 2)))
  < @distance_limit
ORDER BY distance ASC;
```

5.17 poi_map.php: Ergebnisausgabe als Karte

Zur Darstellung als Karte wird auf das stop-selection-Modul zurückgegriffen und das Verhalten bei Start so erweitert, dass die Karte über volle Kontrollelemente zur Änderung von Maßstab, Ausschnitt und Art der Hintergrunddarstellung verfügt. Dies ist gerade bei Points of Interest sinnvoll, um dem Nutzer die besten Informationsmöglichkeiten über die einzelnen Punkte unmittelbar zur Verfügung zu stellen.

Die Points of Interest werden sofort beim Öffnen des Dokuments mit Ajax (in Form eines JavaScripts) geladen und dargestellt.

```
// display no stops or anything, but load POIs
window.onload = function () {
  init("", <?php echo dbisGoogleMapsDefaultView(); ?>);
  retrieveAndPrintUri("?data=1", "Click on a marker to see more information!");
  map.addMapType(G_PHYSICAL_MAP);
  map.removeControl(zoomControl);
  map.addControl(new GLargeMapControl());
  map.addControl(new GMapTypeControl());
};
```

Das nachgeladene Skript, welches die Kartendarstellung der Linie erzeugt, füllt zunächst einen JavaScript-Array mit den Positionen und Metadaten der POIs. Dieser Array wird dann verwendet, um für jeden POI je einen Marker in Google Maps zu erzeugen und bei allen jeweils als Behandlung eines Klicks das Öffnen eines „Info-Fensters“ in Google Maps vorzusehen.

```
<SCRIPT TYPE="text/javascript">
function createPoiMarker (marker) {
    var options = {title: marker.name, icon: marker.icon, clickable: true};
    var position = new GLatLng(marker.latitude, marker.longitude);
    var overlay = new GMarker(position, options);
    map.addOverlay(overlay);

    GEvent.addListener(overlay, "click", function() {
        var info = "<P><STRONG>"+marker.name+"</STRONG></P><P>"+marker.type+"</P>";
        map.openInfoWindowHtml(position, info);
    });
}

if (googleMapsEnabled) {
    var markerIcon = new MarkerIcon("pin+shadow");

    createPoiMarker({name: "Victoria Concert Hall and Theatre", icon: markerIcon,
        type: "Theatre", latitude: 1.288464, longitude: 103.851301});
    createPoiMarker({name: "Singapore City Hall", icon: markerIcon,
        type: "Sightseeing", latitude: 1.290770, longitude: 103.851299});
    createPoiMarker({name: "Singapore Art Museum", icon: markerIcon,
        type: "Museum", latitude: 1.297409, longitude: 103.850685});
}

</SCRIPT>
```

Zur Erzeugung der einzelnen createPoiMarker-Aufrufe kommt wie üblich eine PHP-Schleife zum Einsatz:

```
<?php
$calculator = new UtmCalculator;
while ($row = dbisNextRow()) {
    $geographic = $calculator->toGeographic($row['utm_e'], $row['utm_n'], DBIS_UTM_ZONE);
    echo DBIS_HTML_INDENT, 'createPoiMarker({name: "', addslashes($row['name']), "'});
    echo ', icon: markerIcon, type: "', addslashes($row['type']), "'}, '";
    printf("latitude: %08.6f, longitude: %08.6f");\n",
        $geographic['latitude'], $geographic['longitude']);
}
?>
```

5.18 dataset_info.php: Abfrage mit interaktiven Dialogfeldern

Da man hier in der URL-Zeile auch Tabellennamen angeben kann, die nicht in der Datenbank enthalten sind, werden im data-entry-Bereich der Seite erlaubte Werte für die Abfrage vordefiniert. Hält sich der Nutzer an die ihm zur Verfügung gestellten Schaltflächen auf der Seite selbst, kann es jedoch zu keinen Fehlermeldungen kommen.

```
$unsafeDataset = @dbisHttpGet('dataset');
$safeDataset = htmlspecialchars($unsafeDataset);

$legalDatasetValues = array('line', 'stop', 'poi', 'stop_has_line');
if (! in_array($unsafeDataset, $legalDatasetValues)) {
    trigger_error('"' . $safeDataset . '" is not a legal dataset name', E_USER_ERROR);
}
```

Mit dem folgenden Formular soll erreicht werden, dass vier anklickbare Radiobuttons auf der Seite erscheinen. Bei der Seitenübergabe soll der jeweils ausgewählte Button weiterhin aktiviert bleiben. Es soll keinen Defaultwert geben, sodass das Formular beim erstmaligen Aufruf noch keine Tabelle anzeigt.

```
<FORM METHOD="GET" ACTION="">
  <P>Select the raw dataset you want</P>
  <P>
    <LABEL><INPUT type="radio"
      <?php if ('line' == $unsafeDataset) { echo "CHECKED "; } ?>
      NAME="dataset" VALUE="line"> Lines</LABEL><BR>
    <LABEL><INPUT type="radio"
      <?php if ('stop' == $unsafeDataset) { echo "CHECKED "; } ?>
      NAME="dataset" VALUE="stop"> Stops</LABEL><BR>
    <LABEL><INPUT type="radio"
      <?php if ('poi' == $unsafeDataset) { echo "CHECKED "; } ?>
      NAME="dataset" VALUE="poi"> Points of Interest</LABEL><BR>
    <LABEL><INPUT type="radio"
      <?php if ('stop_has_line' == $unsafeDataset) { echo "CHECKED "; } ?>
      NAME="dataset" VALUE="stop_has_line"> Stop-Line-Korrelation</LABEL><BR>
    <INPUT TYPE="submit" VALUE="Show raw dataset">
  </P>
</FORM>
```

Die zugehörige SQL-Abfrage ist besonders einfach gehalten, zumal die Überprüfung des Werts von \$dataset hier bereits abgeschlossen ist („unsafe“ bezieht sich überdies auf XSS-Sicherheitslücken, die aber nur bei der Ausgabe nach HTML zum Tragen kommen, in SQL hingegen keine Rolle spielen).

```
SELECT * FROM ` $unsafeDataset `;
```

6 Komplexe Abfragen der Dialogschicht

6.1 Google Maps–Einbindung

Das Prinzip der Google Maps–Einbindung ist recht einfach. Kurz zusammengefasst: beim Laden der HTML-Seite wird Google Maps initialisiert und in einem dafür bereitgestellten Container die Karte angezeigt. Gleichzeitig wird veranlasst, dass Google Maps die Positionen sämtlicher Haltestellen aus der Datenbank nachlädt und dem Kartenbild hinzufügt.

Klickt der Nutzer die Karte an (Einfach-Klick), wird durch Google Maps die geographische Position des Klicks ermittelt und veranlasst, dass diejenige Haltestelle, die dieser Position am nächsten ist, mitsamt aller Attribute ermittelt wird.

Der Rest der Abfrage in der Dialogschicht ist konzeptionell identisch mit den Abfragen in der Klasse „Abfrage mit interaktivem Dialogfeld“; siehe Ausführungen zu „Abfragen mit interaktiver Karte“ in Kapitel 3.

6.1.1 Schnittstelle zwischen Google Maps und der Datenbank

Wie zuvor beschrieben, muss Google Maps in zwei Situationen auf die Datenbank zugreifen. Dies ließ sich einfach realisieren, indem per asynchronem XMLHttpRequest die Ausgabe eines entsprechenden PHP-Skripts nachgeladen wurde, welche das Ergebnis der gewünschten Datenbankabfrage im XML-Format enthält:

```
GET /stop-selection.php?latitude=1.334&longitude=103.740 HTTP/1.1

[...]
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<resultset>
  <stop latitude="1.334125" longitude="103.741798" beeline="201">Jurong East</stop>
</resultset>
```

Dieses PHP-Skript soll hier kurz beschrieben werden. Grundsätzlich ähnelt es der Klasse der statischen Abfragen, jedoch mit dem Unterschied, dass von dem URI ein Query-Parameter verwendet wird, um zwischen mehreren Abfragen auszuwählen:

```
$isAllValuesPresent = array_key_exists('latitude', $_GET)
    && array_key_exists('longitude', $_GET);
$range = (int)@$_GET['range'];

if ($isAllValuesPresent) {
    if ($range > 0) {
        // stops within range only
        dbisQuery("
            SELECT *
            FROM (
                SELECT stop.utm_e AS easting, stop.utm_n AS northing, stop.name AS stopname,
                    round(sqrt(POW((stop.utm_n - @northing), 2)
                        + POW(stop.utm_e - @easting, 2))) AS beeline
                FROM stop
                ORDER BY beeline
            ) AS beeline_all_stops
            WHERE beeline <= '$range';
        ");
    }
    else {
        // the one nearest stop only
        dbisQuery("
            SELECT stop.utm_e AS easting, stop.utm_n AS northing, stop.name AS stopname,
                round(sqrt(POW((stop.utm_n - @northing), 2)
                    + POW(stop.utm_e - @easting, 2))) AS beeline
            FROM stop
            ORDER BY beeline
            LIMIT 1;
        ");
    }
}
```

```
}  
}  
else {  
    // all stops  
    dbisQuery("  
        SELECT stop.utm_e AS easting, stop.utm_n AS northing, stop.name AS stopname  
        FROM stop;  
        ");  
}
```

Zuvor müssen allerdings noch die von Google Maps kommenden geographischen Koordinaten in UTM-Koordinaten für die Datenbank umgerechnet werden. Bei der anschließenden Ausgabe nach XML muss dementsprechend wieder von UTM-Koordinaten in geographische Koordinaten umgewandelt werden. Dies stellt aber unter Zuhilfenahme der Klasse `UtmCalculator` keine Hürde dar:

```
require_once('utm_calculator.class.php');  
$calculator = new UtmCalculator;  
  
$utm = $calculator->toUtm((double)$_GET['latitude'], (double)$_GET['longitude']);  
dbisQuery("SET @easting = '". $utm['easting'] ."'");  
dbisQuery("SET @northing = '". $utm['northing'] ."'");  
  
// [Datenbank-Query ...]  
  
while ($row = dbisNextRow()) {  
    $geographic = $calculator->toGeographic($row['easting'], $row['northing'],  
        DBIS_UTM_ZONE);  
  
    // output stop as XML ...  
}
```

Die Ausgabe selbst erfolgt nicht unter Verwendung von `dbisPrintTableRow`, da ja keine HTML-Tabelle ausgegeben werden soll, sondern XML-Elemente:

```
<resultset>  
<?php  
while ($row = dbisNextRow()) {  
    $geographic = $calculator->toGeographic($row['easting'], $row['northing'],  
        DBIS_UTM_ZONE);  
  
    // output stop as XML  
    printf(' <stop latitude="%08.6f" longitude="%08.6f",  
        $geographic['latitude'], $geographic['longitude']);  
    if (array_key_exists('beeline', $row)) {  
        echo ' beeline="' . $row['beeline'] . '";  
    }  
    echo '>', htmlspecialchars($row['stopname'], ENT_NOQUOTES), "</stop>\n";  
}  
?>  
</resultset>
```

6.1.2 Zusammenspiel der Schichten

An die Stelle einer umfangreichen textlichen Erklärung soll hier ein Interaktionsdiagramm treten, welches das Zusammenarbeiten der verschiedenen beteiligten Agenten verständlich macht.

Das Diagramm (UML-Sequenzdiagramm) befindet sich auf der nächsten Seite.

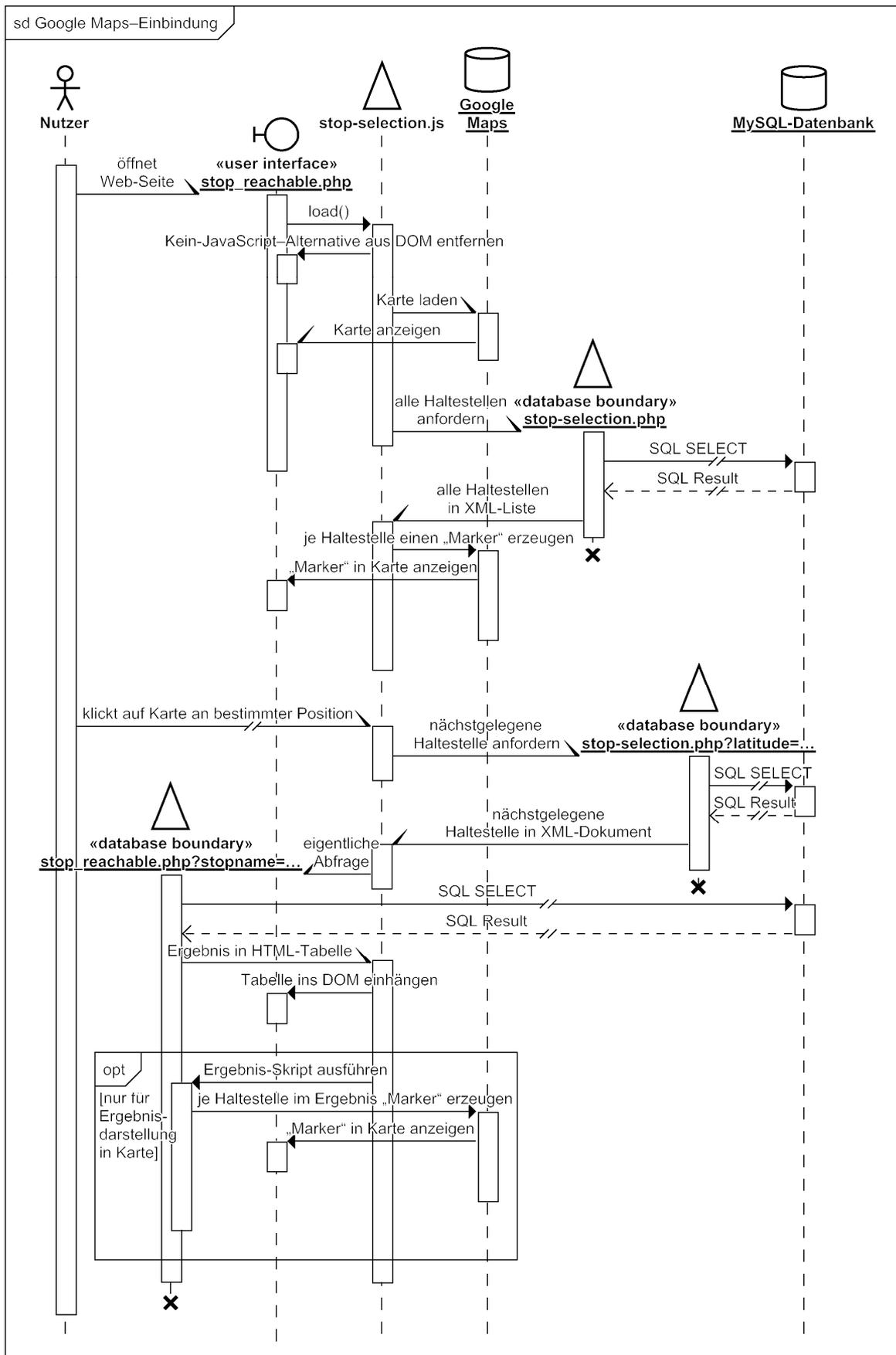


Abb. 9: UML-Sequenzdiagramm des typischen Falls der Google Maps-Einbindung

6.2 Routing-Algorithmus

Der Algorithmus ist speziell auf die Bedürfnisse von Liniennetzen zugeschnitten, bei denen es bekanntlich üblicherweise sehr viele Haltestellen, aber nur wenige Umsteigepunkte gibt. Beispielsweise gibt es im Liniennetz von Singapur 96 Haltestellen; nur neun davon sind Umsteigepunkte. Daher wurde auch der Dijkstra-Algorithmus früh verworfen: Er verlangt die Modellierung des kompletten Netzes mit allen Haltestellen. Das ist weniger für die Entwicklung problematisch (die entsprechenden Schleifen sind recht trivial) als vielmehr für die Skalierbarkeit der Web-Anwendung auf große Liniennetze sowie auf großen Nutzerzahlen und die daraus resultierende nennenswerte Zahl gleichzeitiger Zugriffe. Wir befürchten, dass insbesondere in Anbetracht der in PHP zur Verfügung stehenden Datenstrukturen unter solchen Bedingungen der Dijkstra-Algorithmus nicht performant genug sein könnte.

Das Ermitteln der Routen erfolgt in dieser Web-Anwendung mit folgendem Algorithmus:

1. Identifiziere Anfangs- und Zielhaltestelle anhand der Nutzerangaben.
2. Definiere aktuelle Linie als diejenige Linie, auf welcher der Anfangspunkt liegt.
3. Lege aktuelle Linie auf den Umsteige-Stapel.
4. Falls die aktuelle Linie die Zielhaltestelle bedient, speichere den Umsteige-Stapel als Ergebnis.
5. Für jede Umsteigemöglichkeit auf der aktuellen Linie: Setze aktuelle Linie auf die Linie, in die umgestiegen werden kann. Gehe zu Schritt 3.
6. Entferne den letzten Umstieg vom Umsteige-Stapel.
7. Gehe zu Schritt 5 und fahre mit der nächsten Umsteigemöglichkeit fort, sofern noch nicht alle durchlaufen wurden.

Dabei sind Umsteigemöglichkeiten alle Haltestellen, die noch nicht auf dem Umsteigestapel liegen (Schleifenabbruchbedingung), mit Ausnahme des letzten Umsteigepunkts und der Zielhaltestelle.

6.2.1 Umsetzung in PHP und SQL

Die PHP-Implementierung der Schritte 3 bis 7 erfolgte in einer rekursiven Funktion:

```
function findRouteLeg ($currentStopId, $destinationStopId, $currentLineId) {
    global $changeStack, $routeList, $interchangeStops;
    $changeStack[] = array('from' => $currentStopId, 'line' => $currentLineId); // push

    // is our destination on this line?
    if (lineServicesStop($currentLineId, $destinationStopId)) {
        // yes: we've found a route, so let's log it
        $routeList[] = array('legs' => $changeStack);
    }

    // try to change lines:
    // iterate through all interchange stations on the current line
    foreach ($interchangeStops[$currentLineId] as $interchange) {

        // did we already change here for this connecting line?
        if (isChangeInStack($interchange['stop'], $interchange['line'], $changeStack)) {
            // yes: no recursion for that line now, avoiding infinite loop
            continue;
        }

        // no changing at current or destination stop
        if ($interchange['stop'] == $currentStopId
            || $interchange['stop'] == $destinationStopId) {
            continue;
        }

        // recurse, searching route from interchange station
        findRouteLeg($interchange['stop'], $destinationStopId, $interchange['line']);
    }
}
```

```
    array_pop($changeStack);  
}
```

Aus Gründen der Performance werden die Datenbankabfragen nicht synchron in Echtzeit von `findRouteLeg` aus angestoßen. Dies wurde zwar zunächst versucht, jedoch stellte sich beim Geschwindigkeits-Profilung heraus, dass dann etwa 99,8 % der benötigten Rechenzeit mit MySQL-Abfragen verbracht wird. Anstattdessen werden daher nun alle Abfragen nur ein je einziges Mal durchgeführt und die Ergebnistabellen in PHP in assoziativen Arrays als Cache abgelegt.

Zum Anlegen der Caches werden folgende SQL-Abfragen bei Bedarf ausgeführt:

```
-- $linesServiceStops  
SELECT id_stop, id_line, distance, duration  
FROM stop_has_line  
ORDER BY id_line ASC, `order` ASC;  
  
-- $interchangeStops  
SELECT DISTINCT shl2.id_line AS line_id, shl2.id_stop AS stop_id, shl1.id_line AS line_id1  
FROM stop_has_line shl1  
INNER JOIN stop_has_line shl2 ON shl1.id_stop = shl2.id_stop  
WHERE shl1.id_line <> shl2.id_line;  
  
-- $stops  
SELECT id, name  
FROM stop;  
  
-- $lines  
SELECT id, name, color  
FROM line;
```

Die Funktion `findRouteLeg` verwendet eine Funktion `lineServicesStop(<line>, <stop>)`, welche den Cache `$linesServiceStops` interpretiert. Diese Funktion wird für die Bedingung in Schritt 4 benötigt und liefert `TRUE` genau dann, wenn die angegebene Linie die angegebene Haltestelle bedient:

```
function lineServicesStop ($lineId, $stopId) {  
    global $linesServiceStops;  
    foreach ($linesServiceStops[$lineId] as $item) {  
        if ($item['stop'] == $stopId) {  
            return TRUE;  
        }  
    }  
    return FALSE;  
}
```

Schritte 1 und 2 erfolgen in einer gesonderten Funktion namens `findRoutes`, die ihrerseits `findRouteLeg` aufruft, um die Routenfindung einzuleiten. `findRoutes` ist diejenige Funktion, die initial zum Anstoß des Routing aufgerufen wird; siehe Beschreibung der Dialogschicht.

Die Funktion `findRoutes` bereitet zunächst mit Schritt 1 und 2 die Rekursion vor. Allerdings muss bei dieser Implementierung der Algorithmus komplett ab Schritt 2 für jede der Linien, welche die Anfangshaltestelle bedienen, wiederholt werden. Die Liste dieser Linien wird daher zunächst in einer eigenen SQL-Abfrage ermittelt, bevor in die Rekursion `findRouteLeg` eingestiegen wird.

```
SELECT DISTINCT id_line FROM stop_has_line WHERE id_stop = '$beginStopId';
```

Nach Abschluss der Rekursion werden alle ermittelten Routen für die Ausgabe vorbereitet, denn zu diesem Zeitpunkt bestehen sie lediglich aus Kopien des Umsteigestapels (einer internen Datenstruktur, die zur Ausgabe nicht gut geeignet ist). Dabei wird auch die Liste der Routen sortiert und bei jedem Teilstück Strecke und Dauer hinzugefügt. Letzteres geschieht mittels der Funktion `quickestConnection` (so benannt, weil bei Schleifenlinien immer die kürzere der beiden Richtungen größer null gewählt wird, um unnötiges Spazierenfahren zu vermeiden).

```
function quickestConnection ($fromStopId, $toStopId, $lineId) {
    global $linesServiceStops;

    // get all instances of the from and to stop IDs (normally just one each)
    $frops = array();
    $stos = array();
    foreach ($linesServiceStops[$lineId] as $key => $item) {
        if ($item['stop'] == $fromStopId) {
            $frops[] = $key;
        }
        if ($item['stop'] == $toStopId) {
            $stos[] = $key;
        }
    }

    if (count($frops) == 0 || count($stos) == 0) {
        // stops 1 and 2 are not both on the given line
        return NULL;
    }

    // calculate duration and distance for each pair, searching for quickest
    // note that normally there is exactly one pair,
    // thus  $O(n^2)$  :  $n=1 \Rightarrow O(1)$  in the average case
    $quickest = array('duration' => NULL, 'distance' => NULL);
    foreach ($frops as $from) {
        foreach ($stos as $to) {
            $duration = abs($linesServiceStops[$lineId][$to]['duration']
                - $linesServiceStops[$lineId][$from]['duration']);
            if ($quickest['duration'] === NULL || $duration < $quickest['duration']) {
                $quickest['duration'] = $duration;
                $quickest['distance'] = abs($linesServiceStops[$lineId][$to]['distance']
                    - $linesServiceStops[$lineId][$from]['distance']);
            }
        }
    }
    return $quickest;
}
```

Soll nach Abschluss der Routenfindung gefiltert werden, so geschieht dies über die Funktion `filterRoutes`. Diese besteht aus einer einfachen Schleife mit Bedingungsprüfung; ist die Filterbedingung nicht erfüllt, wird die jeweilige Route als Ergebnis übernommen.

Gefiltert wird so, dass neben der schnellsten Verbindung all diejenigen Verbindungen übernommen werden, die weniger Umstiege haben und unter allen Verbindungen mit der gleichen Zahl Umstiegen die schnellsten sind.

```
function quickestRoute ($routes, $maxChanges) {
    $quickestRoute = NULL;
    foreach ($routes as $routeKey => $route) {
        if (($quickestRoute === NULL
            || $route['duration'] < $routes[$quickestRoute]['duration']
            && ($maxChanges === NULL || count($route['legs']) <= $maxChanges)) {
            $quickestRoute = $routeKey;
        }
    }
    return $quickestRoute;
}

function filterRoutes ($routes) {
    // find quickest routes with as few changes as possible
    $recommendedRoutes = array();
    $changes = NULL;
    while (($quickestRoute = quickestRoute($routes, $changes)) !== NULL) {
        $recommendedRoutes[] = $routes[$quickestRoute];
        $changes = count($routes[$quickestRoute]['legs']) - 1;
    }
    return $recommendedRoutes;
}
```

6.2.2 Ergebniskritik

Bei der Unteruschung der Gesamtimplementierung in `routing.php` scheint es, als werde relativ viel Zeit in unnötigen Schleifen verbracht. Dieses Verhalten ist sicherlich verbesserungswürdig.

Designziel war es, den Routing-Algorithmus selbst (also alle PHP-Funktionen in der Datei) ausschließlich auf IDs arbeiten zu lassen und demnach auch als Eingabe- und Ausgabewerte nur IDs zu verwenden. Die vielen Schleifen sind eine direkte Folge dieses Designs: in `findRouteLeg` müssen erst umständlichst alle Routen in ein Format umgewandelt werden, das ausgabetauglich ist, und mit fehlenden Werten versehen werden. Das Ergebnis davon ist dann aber doch noch nicht ausgabetauglich, weil überall nur IDs stehen. Die müssen dann wieder mit Zusatzschleifen in Namen umgewandelt werden, und obendrein müssen plötzlich die Spalten umsortiert werden, weil die vorherige Formatumwandlung der Routen doch nicht sehr toll war.

Es ist aus diesem Grund in Frage zu stellen, ob diese konkrete Implementierung des beschriebenen Algorithmus gut auf große Umgebungen skaliert.

Geschwindigkeits-Profiling des jetzt vorliegenden Algorithmus im vorliegenden Netz ergab auf einer Referenz-Maschine (Intel Core 2 Duo 2,4 GHz, Mac OS X 10.5) eine durchschnittliche Ausführzeit in der Größenordnung von 10 ms. Bei angenommenen 6 Abfragen pro Nutzer pro Minute erlaubt diese Anwendung damit rein rechnerisch bis zu 1000 gleichzeitige Nutzer (in der Praxis wird es jedoch schon etwas früher zu einer Auslastung des Systems kommen).

Diese Zahlen deuten drauf hin, dass der Algorithmus für den Produktiveinsatz in sehr großen Umgebungen in jedem Falle überdacht oder wenigstens neu gemessen werden sollte. Für die Zwecke des Datenbanken-Praktikums scheint die Leistungsfähigkeit jedoch mehr als ausreichend zu sein. Aus pragmatischen Gesichtspunkten wäre daher eine weitergehende Optimierung erst beim Auftreten von tatsächlichen Geschwindigkeitsproblemen erforderlich.

Abschließend sei angemerkt, dass der Dijkstra-Algorithmus, der als Alternative diskutiert wurde, nur dann besonders effizient arbeitet, wenn die verwendeten Datenstrukturen dies unterstützen. Benötigt wird insbesondere eine performante Priority Queue. Weil eine Priority Queue aber offensichtlich nicht performant in Form eines Arrays umgesetzt werden kann und weil die objektorientierten Fähigkeiten von PHP 5 bekanntlich stark beschränkt sind, sollte eine – selbst theoretische – Geschwindigkeitssteigerung durch Dijkstra nicht a priori angenommen werden, sondern bedarf durchaus einer empirischen Untersuchung.

7 Installation der Web-Anwendung

7.1 Mindest-Systemanforderungen

Für diese Web-Anwendung wird – laut Dokumentation der eingesetzten Technologien – mindestens benötigt:

- MySQL 5
- Apache 1.3
- PHP 4.3 (mod_php oder CGI)
- ein Web-Browser

Es wurde großer Wert darauf gelegt, so wenig Annahmen wie möglich zu machen, die die Portabilität negativ beeinflussen, und gezielte Schichten zur Abtrennung eventuell auszutauschender Komponenten eingeführt.

Ein gutes Beispiel ist `database.include.php`, über das die gesamte Kommunikation mit MySQL läuft: Es wurde grundsätzlich ein SQL-Syntax verwendet, der recht nah an ANSI-SQL liegt, um die Möglichkeit der Portierung auf ein anderes Datenbanksystem zu erhalten. Sollte zu einem späteren Zeitpunkt z. B. Oracle statt MySQL verwendet werden, so kann – falls trotzdem nötig – `database.include.php` auch leicht durch einen Adapter ersetzt werden, der die über ihn laufenden SQL-Queries auf den Oracle-Syntax umbaut.

Infolgedessen besteht die Möglichkeit, dass unsere Anwendung auch in niedrigeren oder völlig andersartigen Umgebungen mit minimalen Änderungen funktioniert. Dieser Fall wurde aber nicht weiter untersucht.

7.2 Empfohlene Umgebung und Kompatibilität

Um zu garantieren, dass diese Web-Anwendung in vollem Funktionsumfang und in der beabsichtigten Art und Weise verwendet werden kann, wird empfohlen:

- MySQL 5.0 oder höher
- Apache 2.2.8 oder höher
- mod_php 5.2.5 (Apache-Plugin für PHP) oder höher
- Browser mit Unterstützung von HTTP 1.1
- Browser mit Unterstützung von HTML 4 oder höher
- Browser mit Unterstützung von CSS Level 2 Revision 1 (CSS 2.1) oder höher
- Browser mit Unterstützung von JavaScript 1.7 oder höher (oder einem äquivalenten ECMAScript-Dialekt)
- Browser mit Unterstützung von DOM Level 2

Steht keine oder keine ausreichende JavaScript- oder DOM-Unterstützung zur Verfügung, kann die Einbindung von Google Maps nicht funktionieren. In diesem Fall wird, wo immer möglich, stattdessen eine alternative Eingabemöglichkeit angeboten, die auch ohne JavaScript und DOM-Unterstützung einen vollwertigen Ersatz bietet.

Steht keine oder keine ausreichende CSS-Unterstützung zur Verfügung, leidet die optische Darstellung. Die meisten graphischen Browser fallen auf ihre Default-Darstellung zurück. Weil alle Inhalte über eine sinnvolle semantische Auszeichnung verfügen, bleibt die Anwendung trotzdem uneingeschränkt benutzbar – von einer möglichen Einschränkung des Bedienkomforts einmal abgesehen.

Steht keine oder keine ausreichende HTML-Unterstützung zur Verfügung, könnte die Funktionalität leiden. Bei Browsern, die zumindest HTML 2.0 voll unterstützen, ist die

Kernfunktionalität jedoch nicht eingeschränkt. In der Praxis trifft dieser Punkt auf alle Browser seit Mitte der 90er Jahre zu.

Steht keine oder keine ausreichende HTTP-Unterstützung zur Verfügung, könnte die Kernfunktionalität leiden. Da HTTP/1.1 aber die Kerntechnologie des Webs ist und seit mehr als einem Jahrzehnt nicht mehr verändert wurde, ist davon auszugehen, dass in den für Endnutzer gedachten Browsern eine weitgehend fehlerfreie HTTP-Implementierung existiert. Im Test konnte diese Vermutung in Bezug auf diese Web-Anwendung bestätigt werden.

Für die Installation selbst sind weiterhin erforderlich:

- Schreibzugriff auf den Document Root des Web-Servers
- ein Texteditor
- Zugriff auf das MySQL-Prompt

Im Folgenden geht diese Installationsanleitung davon aus, dass alle Serverdienste bereits gestartet sind und ordnungsgemäß laufen.

7.3 Datenbank vorbereiten

Nachdem eine den obigen Anforderungen entsprechende Systemumgebung vorbereitet wurde, kann mit der eigentlichen Installation begonnen werden. Zunächst muss dafür die Datenbank im MySQL-Server erstellt und mit Daten gefüllt werden.

Dies geschieht durch Abarbeitung der SQL-Ladefdatei `sql/singapore.sql` an der Unix-Shell wie folgt:

```
# in sql-Verzeichnis der Projektdistribution wechseln
$ cd sql
# den MySQL-Client starten mit den richtigen Zugangsdaten
$ mysql -u root
# SQL-Ladefdatei ausführen
> source singapore.sql;
```

Sofern keine Fehler aufgetreten sind und angezeigt werden, ist an diesem Punkt die Datenbank samt allen Tabellen erzeugt und mit den korrekten Daten gefüllt.

In vielen Fällen erlauben kommerzielle Webhoster es nicht, per SQL-Anweisung Datenbanken zu löschen oder neu anzulegen. Gegebenenfalls muss daher die Datei `sql/singapore.sql` vor dem Laden entsprechend angepasst werden.

7.4 Web-Anwendung installieren

Als erstes sollte sich der Installierende einen Google Maps-API-Schlüssel für den Hostnamen besorgen, über den er letztendlich die Web-Anwendung betreiben möchte. Google erlaubt dies – nach Registrierung – über ein einfaches Web-Formular. Soll auf die Registrierung verzichtet werden, so ist auch ein eingeschränkter Betrieb ohne Google Maps möglich.

Nun gilt es, das Verzeichnis `www` in den Document Root des Web-Servers zu legen. Dies kann wahlweise durch Kopieren, Verschieben oder Änderung der Apache-Konfiguration geschehen.

Jetzt ist es bereits möglich, die Dokumente per Browser aufzurufen. Dies sollte zunächst durch Laden des Dokuments `index.php` überprüft werden. In einem die obigen Anforderungen erfüllenden Browser erscheint links ein (wahrscheinlich noch deaktiviertes) Menü und oben ein grün/gelbes Menü mit dem Schriftzug „SINIS – Singapore Network Information System“.

Nachdem die grundsätzliche Erreichbarkeit per Browser bestätigt wurde, kann nun die Konfiguration der Web-Anwendung für die Systemumgebung erfolgen. Hierzu ist die Datei

`constants.setup.php` im Browser zu laden und die Datei `constants.include.php` im Texteditor zu öffnen.

`constants.include.php` muss nun mit den folgenden Werten versehen werden:

- `DBIS_HOST` – MySQL-Server (IP-Adresse oder DNS-Hostname, z. B. „127.0.0.1“)
- `DBIS_USER` – Nutzernamen für den MySQL-Server (z. B. „root“)
- `DBIS_PASSWORD` – Kennwort für den MySQL-Server
- `DBIS_DATABASE` – Name der MySQL-Datenbank (normalerweise „singapore“)
- `DBIS_GOOGLE_KEY` – Google Maps-API-Schlüssel (voreingestellt für localhost)
- `DBIS_BASE_URI` – Pfadanteil der URI von `index.php`, endend in einem Pfadtrenner (/)
- `DBIS_BASE_PATH` – POSIX-Pfad zum `www`-Verzeichnis, endend in einem Pfadtrenner (/)

Die korrekten Werte der letzten beiden Einstellungen können aus `constants.setup.php` im Browser abgelesen werden.

Die Datei `constants.include.php` enthält einige weitere Einstellungs-Konstanten, die aber im Rahmen einer gewöhnlichen Installation in der Regel nicht geändert zu werden brauchen.

8 Schlussbemerkungen

8.1 Zählung der Abfragen

Laut Aufgabenstellung sollten grundsätzlich 20 Abfragen in der Dialogschicht („Menüeinträge“) vorhanden sein. Aufgrund der besonderen Komplexität der Implementierung eines Routing-Algorithmus gelte dieser jedoch in Folge mündlicher Absprache mit dem Dozenten als drei Abfragen in der Dialogschicht. Aus dem gleichen Grund gelte die Implementierung der interaktiven Karten in stop-selection.php ebenfalls als zusätzliche Abfrage in der Dialogschicht.

Menüeinträge sind 18 vorhanden, so dass wir im Kontext der Aufgabenstellung über 21 Abfragen in der Dialogschicht verfügen.

Im Projekt sind insgesamt 32 volle SQL-SELECT-Abfragen enthalten.

8.2 Verwendete Software

Das Projekt wurde in Windows XP mit dem Texteditor Notepad++ 4.9 und in Mac OS X 10.5.3 mit der Entwicklungsumgebung Coda 1.1 sowie dem Unix-Stream-Editor sed(1) entwickelt. Außerdem kam bei der Layoutgestaltung CSSEdit 2.6 zum Einsatz.

Zur Bearbeitung der Bilder und Grafiken wurden Freehand MX, Illustrator CS3 und Photoshop CS3 eingesetzt.

Das Testen erfolgte unter Verwendung des Datenbankservers MySQL mehreren Versionen, darunter 5.0.5, und des Web-Servers Apache in mehreren Versionen, darunter 2.0 und 2.2.8, mit mod_php in mehreren Versionen, darunter 5.2.5. Client-seitig wurden die drei Rendering Engines WebKit 5525.18, Gecko/20080201 und Trident (IE7) während der Entwicklung zum laufenden Testen eingesetzt. Zusätzliche Tests wurden mit den Browsern Firefox 2.0, Firefox 3.0, iCab 3.0.5, iCab 4.1.1, Internet Explorer 7, Lynx 2.8.5, Opera 9.51, Opera Mini 9.51, Safari 3.1.1 und SeaMonkey 1.1.8 durchgeführt.

8.3 Ausblick

In den kommenden 10 bis 15 Jahren soll der Schienennetzverkehr von Singapur massiv ausgebaut werden, wofür z. T. schon konkret geplant oder sogar gebaut wird. So wurde kürzlich die Punggol LRT West Loop fertiggestellt, dies wurde aber erst eine Woche vor Ende des Projekts festgestellt. Befahren wird die Strecke ohnehin noch nicht, da sie ein noch weitgehend unbesiedeltes Neubaugebiet erschließt. Im kommenden Jahr werden einige Erweiterungsbahnhöfe an der East West Line hinter Boon Lay angeschlossen. Insbesondere in der Innenstadt sind weitere MRT-Linien mit zahlreichen Umsteigehaltestellen geplant. Ein dichteres Liniennetz erfordert die Integration der neuen Haltestellen in dieses Modell.

9 Literatur

(Die Sortierung dieser Liste ist thematisch orientiert.)

- MySQL-Dokumentation
<http://dev.mysql.com/doc/refman/5.0/en/>
- HTTP-Spezifikation (IETF RFC 2616), 1999
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- Uniform Resource Identifiers (URI): Generic Syntax (IETF RFC 2396), 1998
<http://www.ietf.org/rfc/rfc2396.txt>
- Reserved Top Level DNS Names (IETF RFC 2606), 1999
<http://www.ietf.org/rfc/rfc2606.txt>
- Apache-Dokumentation
<http://httpd.apache.org/docs/2.2/>
- SGML: Standard Generalized Markup Language (ISO 8879), 1986
- HTML 2–Spezifikation (MIT/W3C), 1995
http://www.w3.org/MarkUp/html-spec/html-spec_toc.html
- HTML 4–Spezifikation (W3C Recommendation), 1999
<http://www.w3.org/TR/html4/>
- XML: Extensible Markup Language 1.0, 4th Edition (W3C Recommendation), 2006
<http://www.w3.org/TR/REC-xml/>
- XHTML™ 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0 (W3C Recommendation), 2000
<http://www.w3.org/TR/xhtml1/>
- XHTML Media Types (W3C Note), 2002
<http://www.w3.org/TR/xhtml-media-types/>
- Ian Hickson: Sending XHTML as text/html Considered Harmful, 2002
<http://hixie.ch/advocacy/xhtml>
- W3C HTML Working Group discussion 2007-2008
<http://lists.w3.org/Archives/Public/public-html/>
- Joe Clark: Building Accessible Websites. New Riders, 2002
<http://joelclark.org/book/>
- Web Content Accessibility Guidelines 1.0, W3C Web Accessibility Initiative, 1999
<http://www.w3.org/TR/WAI-WEBCONTENT/>
- CSS 2.1–Spezifikation (W3C Candidate Recommendation), 2007
<http://www.w3.org/TR/CSS21/>
- PHP-Dokumentation
<http://www.php.net/docs.php>
- Erich Gamma, Richard Helm, Ralph E. Johnson: Design patterns: Elements of reusable object-oriented software. Addison-Wesley, Reading, MA, 1995
- Donald Knuth: Structured Programming with Go To Statements
http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf
- ECMAScript Language Specification (ECMA-262), 1999
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- Google Maps API Reference
<http://code.google.com/apis/maps/documentation/reference.html>

10 Lizenz

License applicable to the Singapore project implemented by Arne Johannessen and Volker von Nathusius (henceforth "the authors") in the database lab at the University of Applied Sciences, Karlsruhe.

Scope of this License:

This license is applicable to all code (PHP, JavaScript and SQL queries), visual style (CSS) and marked-up content (HTML) in the project, except where noted otherwise. It is not applicable to the information stored in the database (SQL load file), as that was largely supplied by the lecturer. Any changes to the the database itself are considered to be in the Public Domain; the authors disclaim any copyright that might apply to the database.

Copyright © 2008 Arne Johannessen, Volker von Nathusius
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation or other materials provided with the distribution.
- The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11 Lizenz (deutsche Übersetzung)

Folgende Lizenz gilt für das SINIS-Projekt, implementiert durch Arne Johannessen und Volker von Nathusius (fortan als „Autoren“) während des Praktikums Datenbanken an der Hochschule Karlsruhe. Rechtlich bindend ist die englische Form dieser Lizenz.

Lizenzumfang:

Diese Lizenz gilt für den gesamten Code (PHP, JavaScript und SQL-Abfragen), für die graphische Gestaltung (CSS) und den ausgezeichneten Inhalt (HTML) des Projekts, ausgenommen die als solche ausgewiesenen Stellen. Insbesondere ist die Lizenz nicht anwendbar für den Inhalt der Dateien NS_logo.png, singapore.php, CityView.jpg und NetworkMap.png. Des Weiteren ist die Lizenz nicht anzuwenden auf die in der Datenbank gespeicherten Daten (SQL-Ladefdatei), die großteils vom Projektvorgänger übernommen wurde. Gesetzliches Datenbankrecht mag gelten, doch die Autoren verzichten auf das Copyright der Datenbank-Zusammenstellung, eingeschlossen, aber nicht beschränkt auf, alle Datenänderungen selbst.

Copyright (c) 2008 Arne Johannessen, Volker von Nathusius. Alle Rechte vorbehalten.

Weitergabe und Benutzung in Quellcode und binärer Form, mit oder ohne Veränderung, sind unter folgenden Bedingungen zugelassen:

- Weitergabe von Quellcode muss den oben angegebenen Copyright-Hinweis enthalten, diese Liste von Bedingungen und den folgenden Haftungsausschluss.
- Weitergabe in binärer Form muss den oben angegebenen Copyright-Hinweis wiedergeben, diese Liste von Bedingungen und den folgenden Haftungsausschluss in der Dokumentation oder in anderen dargebotenen Unterlagen.
- Die Namen der Autoren dürfen nicht ohne vorhergehende, eindeutige schriftliche Zustimmung verwendet werden, um aus dieser Software abgeleitete Produkte zu unterstützen oder zu bewerben.

DIESE SOFTWARE WIRD DURCH DIE AUTOREN OHNE MÄNGELGEWÄHR ANGEBO-
TEN UND JEDLICHE AUSDRÜCKLICHE ODER IMPLIZITE GEWÄHRLEISTUNG,
EINGESCHLOSSEN, ABER NICHT BESCHRÄNKT AUF, STILLSCHWEIGEND ANGE-
NOMMENE GEWÄHRLEISTUNG AUF GEBRAUCHSTAUGLICHKEIT UND TAUG-
LICHKEIT FÜR EINEN BESTIMMTEN ZWECK WERDEN VON HAFTUNG AUSGE-
SCHLOSSEN. IN KEINEM FALL SIND DIE AUTOREN VERANTWORTLICH FÜR
JEDLICHE DIREKTE, INDIREKTE, BEILÄUFIG ENTSTANDENE, KONKRETE ODER
STRAFGEBUNDENE SCHÄDEN (EINGESCHLOSSEN, ABER NICHT BESCHRÄNKT
AUF, DIE BEREITSTELLUNG VON ERSATZGÜTERN ODER -DIENSTEN; ENT-
GANGENEN NUTZEN; DATENVERLUST; VERDIENSTAUSFALL; ODER BETRIEBS-
STÖRUNG), DIE WIE AUCH IMMER VERURSACHT UND IN JEDER THEORETISCH
DENKBAREN HAFTUNGSFORM, SEI ES DURCH AUFTRAG, KAUSALHAFTUNG
ODER VERSCHULDUNGSHAFTUNG (EINGESCHLOSSEN FAHRLÄSSIGKEIT ODER
SONSTIGES VERSCHULDEN), DURCH JEDLICHE BENUTZUNG DIESER SOFT-
WARE ENTSTEHEN KÖNNEN, SOGAR FALLS DIE MÖGLICHKEIT SOLCHER
SCHÄDEN BEKANNT GEWESEN IST.