GeoTrips.de

Projektdokumentation

Ein Projekt von Patrick Dinger

Betreuer: Prof. Dr. Ansgar Gerlicher, Oliver Kögler, Marc Seeger

HOCHSCHULE DER MEDIEN

1. Inhalt

2. Übersicht

GeoTrips ist eine Internetplattform deren Ziel es ist, Reisen von Benutzern aufzuzeichnen, grafisch aufzubereiten und zur Verfügung zu stellen.

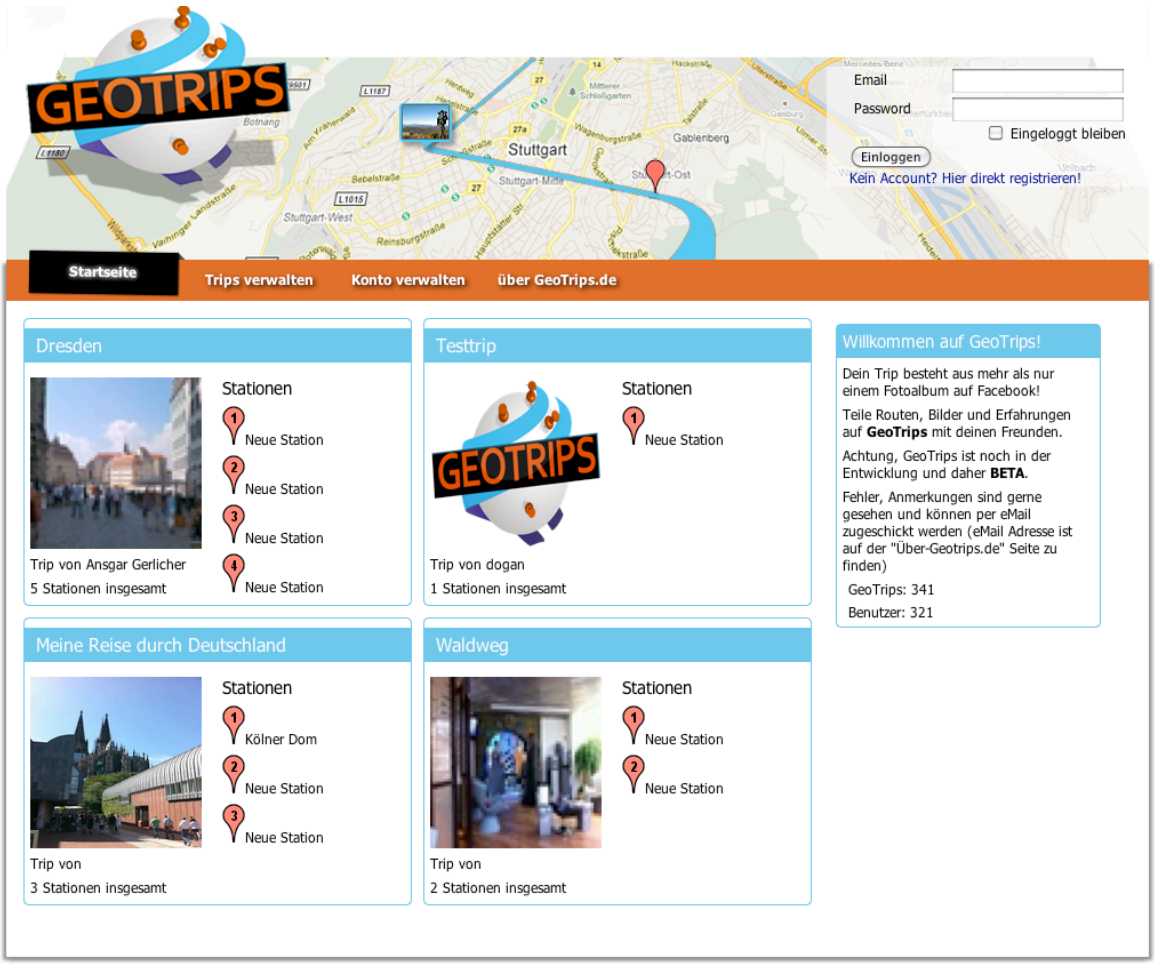

Abbildung 1: GeoTrips.de - Das Endergebnis

2.1. User-Stories

Das Ziel bei der Umsetzung des Projektes war es, folgende User-Stories zu erfüllen:

- Benutzer meldet sich an und kann Trips anlegen
- Benutzer kann Stationen seines Trips eintragen
- Benutzer kann Bilder hochladen und diese seinen Stationen zuordnen
- Benutzer kann bei Bildern mit Geotags diese Information direkt nutzen
- Benutzer kann von unterwegs über Twitter neue Stationen erstellen
- Benutzer kann seine Trips später in Form einer interaktiven Karte oder einer Slideshow betrachten
- Benutzer kann über einen öffentlichen Link seine Trips mit anderen Benutzern teilen
- Benutzer kann angelegte Trips auf andere Seiten einbetten

2.2. Zielgruppe

Als Hauptnutzer werden Web 2.0 affine Menschen zwischen 20 und 50 Jahren angesehen. Wenn das Projekt in dieser Gruppe gut ankommt wird somit auch für gute „Mundpropaganda“ gesorgt.

3. Vorgehen

3.1. Ablauf

Im ersten Schritt wurde evaluiert welche Technologien sich für dieses Projekt am besten eignen. Da es sich um ein Webprojekt handelt, wird auf Clientseite ein Browser eingesetzt. Im Vorgespräch wurde die Zielplattform genauer abgesteckt: Es wurde beschlossen, das nur aktuelle Browser verwendet werden und kein Support für alte Browser implementiert wird. Als aktuelle Browser wurden WebKit (Safari, Chrome) und Firefox eingestuft. Internet Explorer wurde außen vor gehalten.

Auf Serverseite stehen viele Technologien zur Auswahl. Hierbei fiel die Wahl relativ schnell auf RubyOnRails3. Die Veröffentlichung der Version 3 des Rails Frameworks war zu diesem Zeitpunkt erst wenige Wochen her. Da bereits gute Erfahrungen in Version 2 vorhanden waren wurde RubyOnRails 3 verwendet.

Ein weiteres wichtiges Argument für die Entscheidung für RubyOnRails3 war die immer gleiche Struktur eines Rails Projektes: Es wird immer das Model-View-Controller Pattern verwendet, ebenso ist die Ordnerstruktur standardisiert. Diese erleichtert es anderen Entwicklern Plugins für das Framework zu schreiben, die sehr einfach im eigenen Projekt wieder verwendet werden können. Plugins werden in Ruby auch als Gem bezeichnet.

Aus diesem Grund wurde im nächsten Schritt eine Recherche über bereits vorhandene Funktionalitäten in Form eines Plugins bzw. Gems gestartet. Viele wiederkehrende Aufgaben konnten so schnell und effektiv gelöst werden:

- Paperclip (Attachment Managment)
- Devise[1] (User Authorization, User Authentication)
- EXIFr[2] (EXIF Tag Reader Lib für Ruby)
- OmniAuth (OAuth Middleware für Twitter Authorization)
- Twitter (Kapselung der Twitter-API in Ruby)
- Capistrano[3] (Automatische Deployment Lösung für Rails)
- Mime-Types [4](für die Einbindung eines Flash-Uploaders in RubyOnRails3)

Wie hier zu sehen konnte beispielsweise der komplette Part der Benutzerverwaltung komplett über ein Plugin gelöst werden. Hierbei ist nicht nur die Ersparnis in Entwicklungsarbeit zu beachten, sondern auch die gesparte Zeit für das Testen. Dadurch war es möglich sich beim Testen und Entwickeln auf die Kernkomponenten von GeoTrips zu konzentrieren.

Aufgrund des Ergebnisses der Recherche wurde ein Zeitplan angelegt in Form eines Gantt Diagrammes.

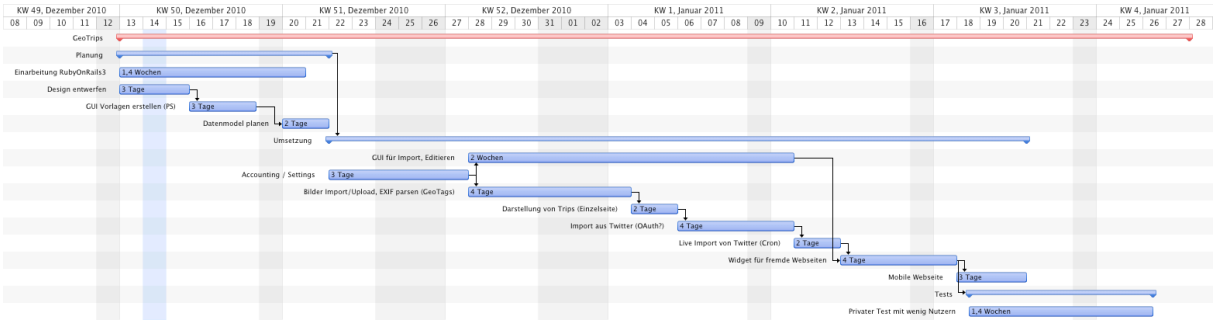

Abbildung 2: Projektplan

Die gesparte Zeit durch die Verwendung fertiger Komponenten wurde hierbei berücksichtigt, weshalb der Zeitplan mit ca. 4 Wochen relativ kurz ausfällt. Es wurde eine 6 Tage Woche zu Grunde gelegt und die Weihnachtsfeiertage als Arbeitszeit abgezogen.

3.2. Wahl der Entwicklungsumgebung

Ruby – die Programmiersprache die in der Entwicklung in RubyOnRails eingesetzt wird – ist eine dynamische Scriptsprache. Aus diesem Grund ist es für Entwicklungsumgebungen weit schwieriger den Programmierer mit Codevervollständigung zu unterstützen als dies bei stark typisierten Sprachen wie bspw. C#, Delphi oder Java der Fall ist.

Trotzdem war es wichtig eine auf RubyOnRails spezialisierte IDE (Integrated Development Environment) zu verwenden. Viele Tools die RubyOnRails mitbringt müssen umständlich über die Konsole gesteuert werden. Eine Integration in die IDE sorgt hierbei für eine enorme Steigerung der Produktivität.

Es wurden zwei IDEs evaluiert: Netbeans 6.7[5] und RubyMine[6].

Die Wahl fiel hierbei auf RubyOnMine. Zwar ist es im Gegensatz zu Netbeans eine kommerzielle Anwendungen, trumpft aber mit vielen wichtigen Funktionen auf.

Mit RubyMine lässt sich der Ruby Debugger direkt in der IDE nutzen, ebenso ist die Verwendung von GIT als Versionierungssystem von Haus aus möglich. Auch ist es möglich die Testsuite von RubyOnRails direkt über die Oberfläche zu verwenden.

3.3. Versionsverwaltung

Wie bereits erwähnt wurde für die Versionsverwaltung des Source-Codes GIT[7] verwendet. GIT ist ein dezentrales Versionierungssystem; verfügt also über kein zentrales Repository auf einem extra Server.

Zu Beginn des Projektes wurde das Repository in einem Dropbox[8]-Verzeichnis abgelegt und war somit gesichert und von mehreren Rechnern erreichbar.

In der finalen Phase wurde die Architektur erweitert.

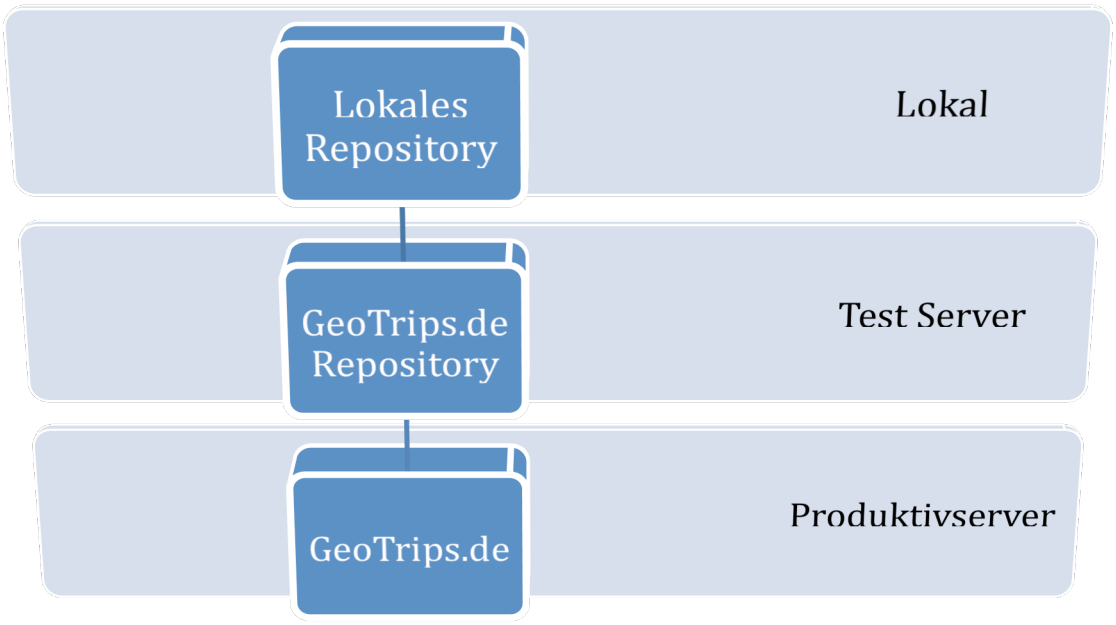

Abbildung 3: Ablauf des Deployment

Die Versionierung wurde lokal durchgeführt und vor einem Update auf das Repository des Produktivservers übertragen.

Für die Übertragung aus dem Repository auf das Produktivsystem wurde Capistrano verwendet. Es wurde hierbei ein Job erstellt, welcher es ermöglicht, die Auslieferung mit einem Kommando vom Entwicklungsrechner aus zu starten.

Hierbei werden alle anstehenden Migrationen der Datenbank durchgeführt, neue Quelldateien werden kopiert, der Cache wird gelöscht und die statischen Daten (bspw. Die hochgeladenen Bilder der Benutzer) werden in die neue Version migriert.

Diese Automatisierung ermöglicht es Änderungen sehr schnell auf den Produktivserver zu übertragen und trotzdem alle Versionen über das Versionsverwaltungssystem (Git) nachvollziehen zu können.

4. Architektur der Anwendung

4.1. Datenmodel

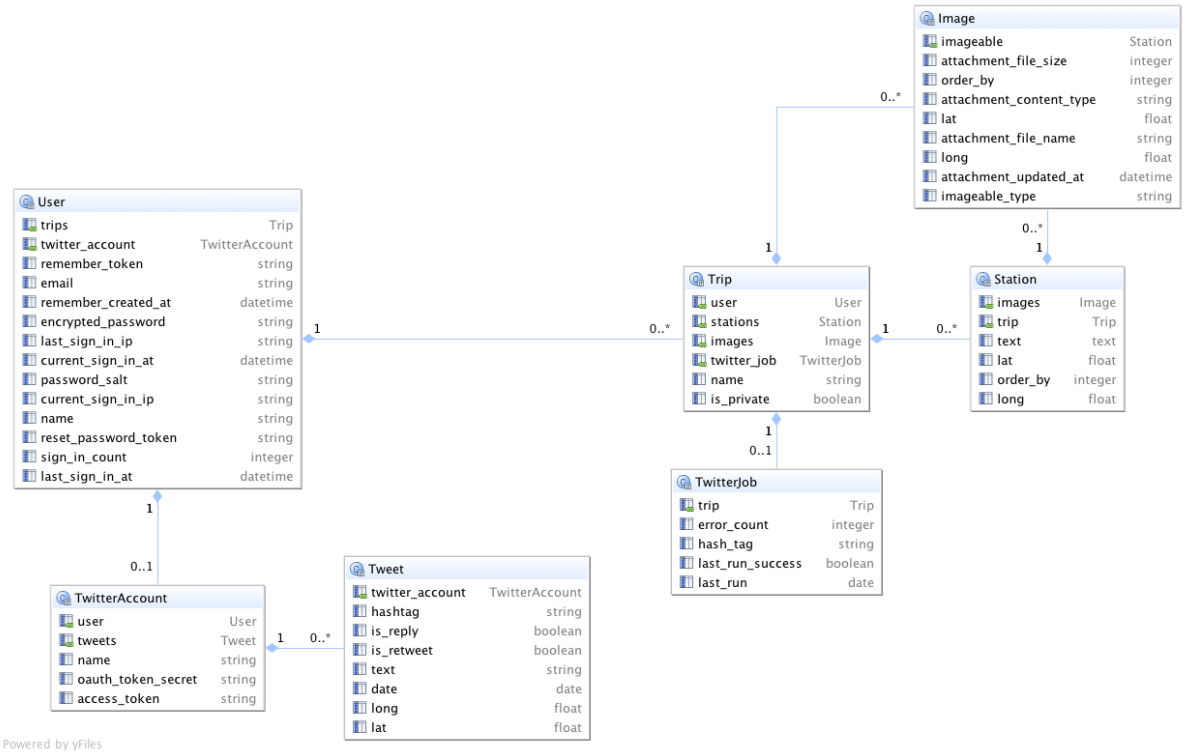

Abbildung 4: Datenmodel / ERM-Diagramm

Als erster Schritt in der Umsetzung wurde ein Datenmodel erstellt. Dieses Model diente gleichzeitig als Datenbank-Model. Da in RubyOnRails alle Datenbank-Tabellen gleichzusetzen mit Modellen im MVC-Pattern sind ist das ERM in diesem Fall ebenso eine Auflistung aller Modelle und deren Beziehungen.

Es folgt eine Erläuterung der einzelnen Modelle:

- **User**: Enthält Informationen zu den angemeldeten Benutzern, ebenso deren Anmeldedaten wie Passwort und eMail-Adresse als auch Daten die der Verwaltung oder Statistiken dienen (letztes Anmeldedatum).
- **Trip**: Ein Trip stellt eine Reise dar und besteht aus mehreren Stationen. Ein Trip gehört immer einem User.
- **Station**: Eine Station gehört immer zu einem Trip. Aus dieser Beziehung leitet sich auch der besitzende Benutzer ab. Eine Station hat einen Namen und besitzt Bilder.
- **Image**: Stellt ein hochgeladenes Bild dar. Enthält Informationen und einen Verweis auf die Datei im Dateisystem. Die Beziehung hierbei ist speziell: Sie ist polymorph. Das bedeutet: Ein Bild kann entweder einem Trip oder einer Station zugewiesen sein. Dies wird deutlich wenn man sich den Use-Case anschaut: Bilder werden zu einem Trip hochgeladen. Die Zuteilung der Bilder zu den einzelnen Stationen geschieht in der Regel später. Da ein Bild wissen muss welcher Entität es gerade zugewiesen ist, wird dieser Typ über ein Feld in der Image-Tabelle hinterlegt (imageable_type). Beim Upload werden außerdem die

Geo-Koordinaten (falls vorhanden) aus dem Bild extrahiert und in der Datenbank festgehalten.

- **TwitterAccount**: Benutzer können ihr Twitter Konto mit ihrem GeoTrips Konto verbinden. Diese Verbindung ist technisch über OAuth gelöst und benötigt verschiedene Tokens welche während des OAuth Authentifizierungsvorgangs ausgetauscht werden. Diese Tokens werden in der Datenbank festgehalten.
- **Tweet**: Diese Tabelle dient als Cache für Einträge aus Twitter. Um die Performance zu steigern werden abgerufene Tweets in dieser Tabelle zwischengespeichert und vorher bereinigt.
- **TwitterJob**: Aktiviert der Benutzer den automatischen Import von Stationen aus Twitter wird in dieser Tabelle ein neuer Eintrag erstellt. Der laufende Cronjob überprüft diese Tabelle und arbeitet die Jobs ab.

4.2. Design

Das Layout und Design von GeoTrips war ein iterierender Prozess. Jeder Entwurf wurde zuerst in Photoshop erstellt und anschließend in HTML und CSS übertragen. Nach drei Iterationen stand das finale Design fest:

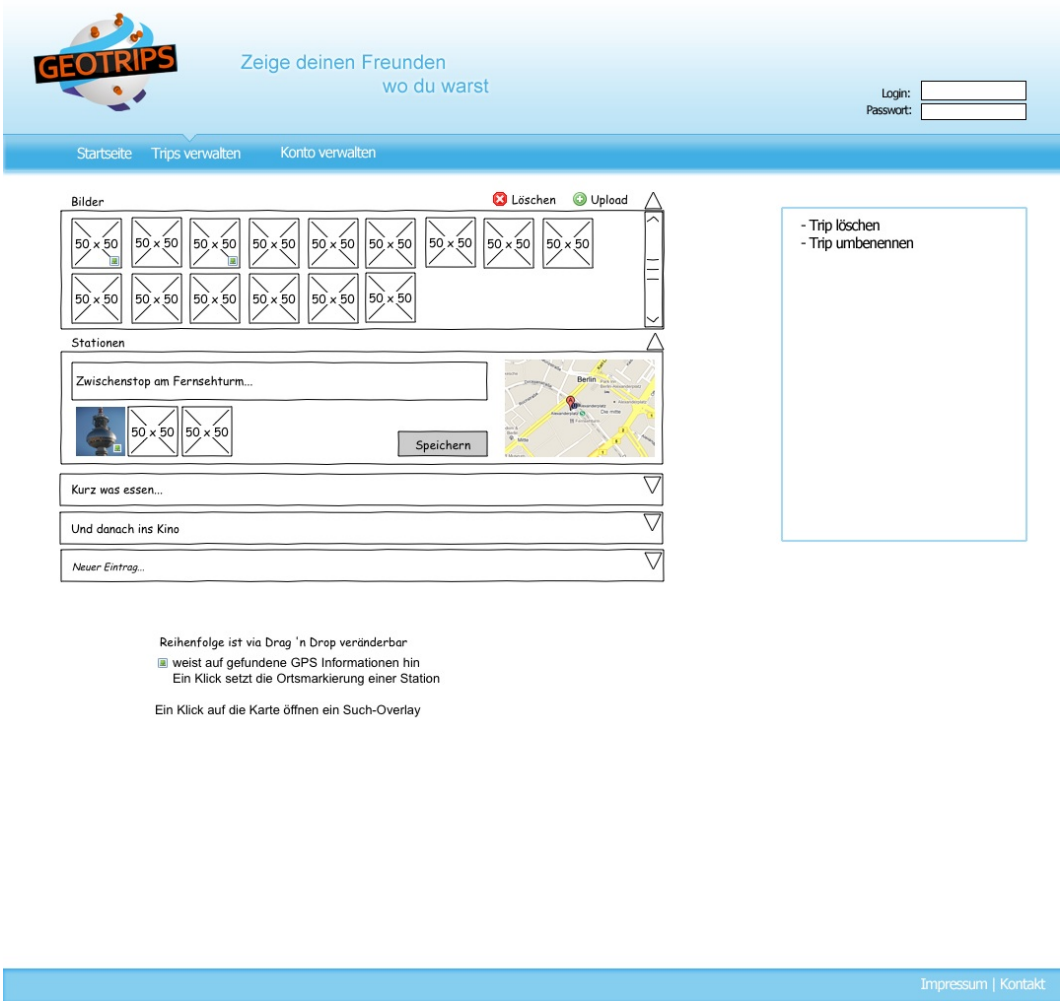

Abbildung 5: Der erste Entwurf

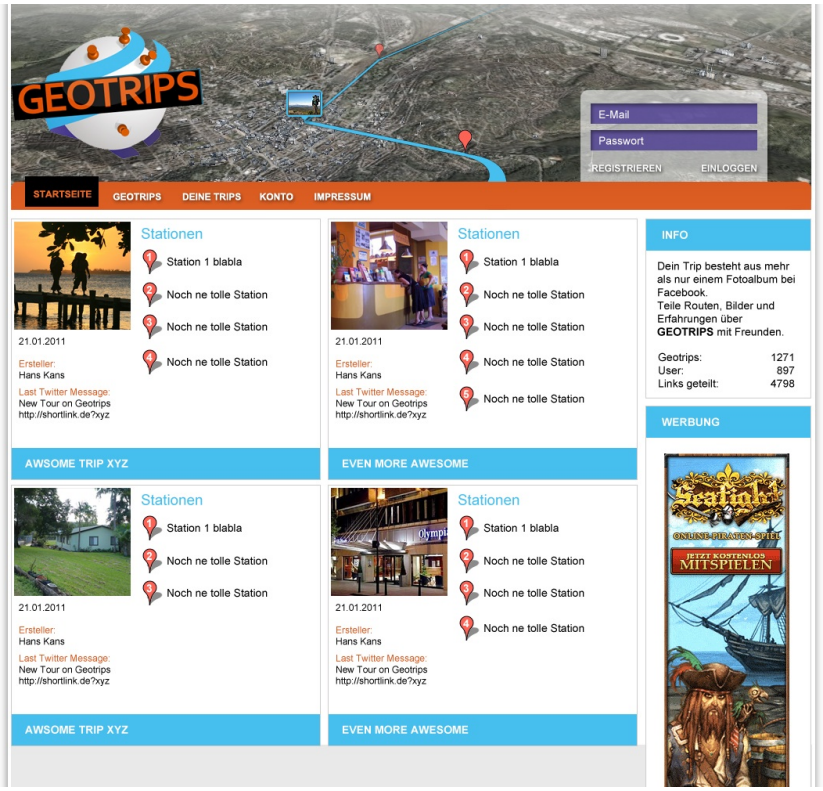

Abbildung 6: Zweite Version

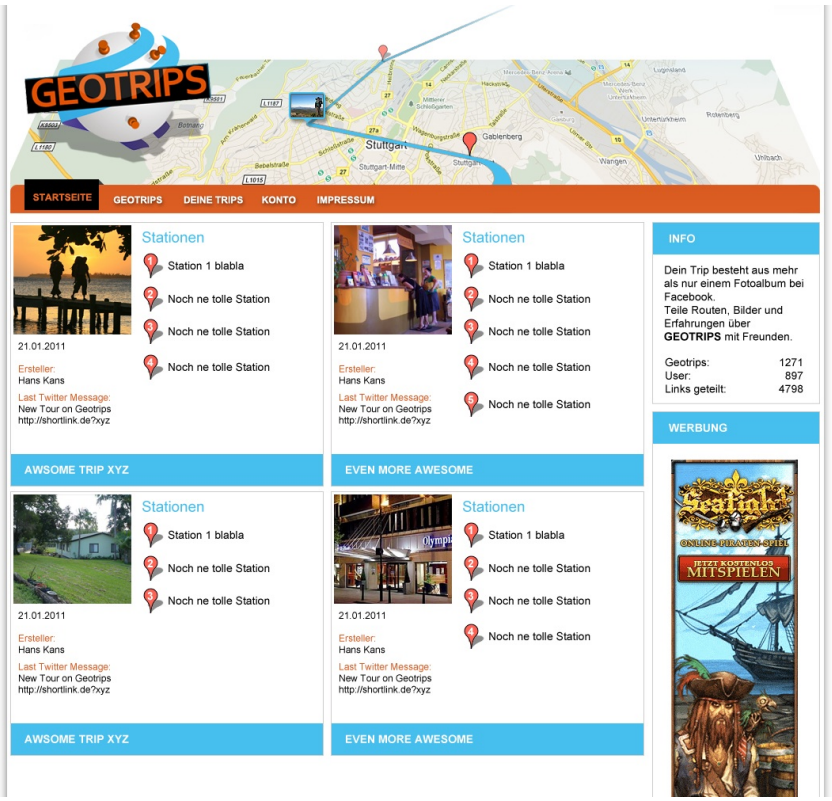

Abbildung 7: Finale Version

5. Funktionsweise der einzelnen Komponenten

5.1. Twitter und OAuth

In GeoTrips ist es möglich, Tweets direkt in GeoTrips als Stationen zu importieren. Dieser Vorgang kann auch automatisiert werden. Somit werden die eigenen Karten quasi automatisch aktualisiert wenn man von unterwegs Tweets abschickt.

Diese Funktion besteht aus drei Teilen:

5.1.1. OAuth

Der User muss seinen Account mit Twitter verbinden. Hierzu ist das OAuth[9] Verfahren bei Twitter vorgeschrieben.

Abbildung 8: Adressleiste während eines OAuth Vorganges

OAuth ist ein Verfahren welches es GeoTrips ermöglicht auf den Twitter Account des Benutzers zuzugreifen, ohne dessen Twitter Passwort zu kennen. Hierbei muss die eigene Applikation zuerst bei Twitter angemeldet werden, woraufhin man einen „consumer-key“ als auch ein „consumer-secret“ erhält. Diese zwei Tokens identifizieren die eigene Applikation.

Anschließend werden verschiedene Tokens ausgetauscht, der User wird während der Authentifizierung direkt auf die Twitter.com Seite geleitet und muss dort Bestätigen das GeoTrips.de Zugriff auf seine Daten erlangen darf.

Abbildung 9: Weiterleitung zu Twitter während des OAuth-Vorganges

Nachdem der Benutzer die Anforderung akzeptiert hat wird er zurück zu GeoTrips geleitet. Der Browser sendet hierbei zwei Tokens an den GeoTrips Server, welche gespeichert werden und für den späteren Zugriff benutzt werden können.

Technisch wurde für die Weiterleitung und die Verarbeitung der Tokens ein Plugin für Rails verwendet: Omniauth[10].

5.1.2. Zugriff auf Twitter

Für den Zugriff auf die Twitter-Daten des Nutzers, also die Tweets, wurde ebenfalls ein Gem verwendet: Twitter[11]. Dieses Plugin ist nur eine Kapselung der offiziellen Twitter-API[12].

5.1.3. Live-Tracking

Hat der Benutzer seinen Account mit GeoTrips verbunden, so kann er jeweils einen seiner angelegten Trips in den „Live-Tracking Zustand“ versetzen.

Ist dieser aktiv, werden alle vom Benutzer abgeschickten Tweets direkt aus Twitter in diesen Trip als Station importiert. Zusätzlich werden die Bilder ausgelesen, abgespeichert und die Geo-Koordinaten erfasst.

Abbildung 10: Darstellung eines Trips mit aktiviertem Live-Tracking

Zu diesem Zweck wurde ein spezieller Rake-Task genutzt. Dies ist ein Ruby-Script, welches im selben Kontext wie die GeoTrips Applikation läuft und daher auf die dort hinterlegten Daten zugreifen kann. Dieses Script wird nun alle 15 Minuten automatisch aufgerufen.

Hierbei wird folgender Ablauf ausgeführt:

1. Verbindung zur Datenbank herstellen, Tabelle „TwitterJobs“ aufrufen.
2. Alle dort hinterlegten Jobs ausführen und als „ausgeführt“ markieren.
3. Twitter kontaktieren, Überprüfen ob es neue Tweets seit der letzten Abfrage gibt.
4. Wenn ja, diese Tweet abrufen, enthaltene Bilder abspeichern und Geo-Koordinaten abrufen.
5. Erhaltene Daten als neue Station erstellen und abspeichern.

Die GPS-Koordinaten sind teil des XML Inhaltes der von Twitter zurück geliefert wird.

Die hinterlegten Bilder jedoch sind als Link im Tweet-Text hinterlegt. Um daraus das Bild zu extrahieren muss für jeden Imagehoster speziell ein Modul geschrieben werden. Zu diesem Zeitpunkt wird daher nur yfrog[13] unterstützt. Dieser Service wird bei den meisten Twitter Clients standardmäßig verwendet.

5.2. Google Maps V3, Google Maps Static API

Google bietet für seine Karten zwei APIs an. Die Javascript API erlaubt es im eigenen Projekt interaktive Karten einzubinden und diese mit Markern oder Pfaden anzupassen. Die Static API wiederum erzeugt die Karten als Grafikdateien, welche statisch eingebunden werden können.

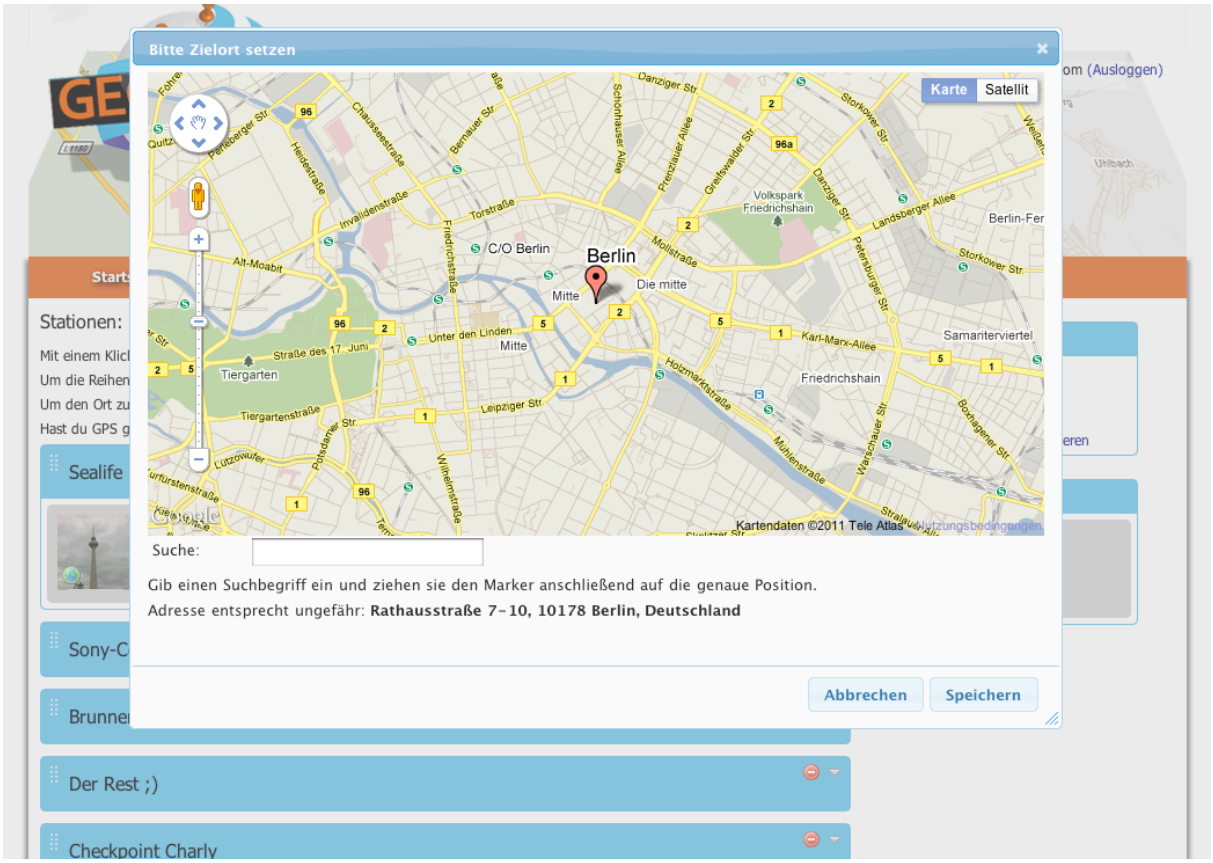

Abbildung 11: Beispiel der Nutzung der dynamischen GoogleMaps API

In GeoTrips werden beide APIs verwendet. Die statische API wird verwendet um in den einzelnen Stationen die gespeicherte Position zu zeigen. Die interaktive Karte wird verwendet wenn ein neuer Ort eingetragen werden soll. Hierbei kann der Benutzer einen Marker direkt auf die gewünschte Position ziehen. Wenn die Position gespeichert wird, werden die Daten an den Server übertragen und das Bild aus der statischen API wird aktualisiert.

5.3. Diashow

Die Architektur von GeoTrips ermöglicht mehrere verschiedene Darstellungsformen für Trips. Zum Startzeitpunkt wurden 3 verschiedene Formen eingebaut:

1. Interaktive GoogleMaps Karte mit eingezeichneter Route, klickbare Marker zeigen Bilder
2. Statische GoogleMaps Karte, gekoppelt mit einer Diashow
3. Variation der Diashow mit interaktiv wählbaren Stationen

Für die Diashows wurden viele JavaScript Frameworks gemischt und miteinander verbunden. Verwendet wurde PikaChoose[14], SlidesJs [15]und JQuery[16].

5.4. Multi-Upload

HTML unterstützt standardmäßig nur den Upload von einer Datei gleichzeitig. Um dies zu umgehen wurde eine Mischung aus Flash und Javascript verwendet um das Hochladen beliebiger Bilder zu ermöglichen.

Abbildung 12: Darstellung eines Uploads von mehreren Dateien

Verwendet wurde hierbei Uploadify[17]. Es mussten einige Änderungen an den Session-Modulen von RubyOnRails3 vorgenommen werden, da es sonst Probleme gab die Uploads auf Serverseite korrekt zu verarbeiten.

Allerdings konnte hierzu eine Beispiel-Applikation[18] im Internet gefunden werden. Dort konnten die durchzuführenden Änderungen extrahiert werden.

5.5. Bearbeiten eines Trips

Das Kernelement von GeoTrips stellt das Bearbeiten eines angelegten Trips dar. Für diese Aufgabe war es das Ziel ein komfortables, schnelles und benutzerfreundliches Interface zu entwickeln.

Im ersten Schritt wurde ein Sketch erstellt in welchem das Layout und die Funktionen als Skizze dargestellt sind.

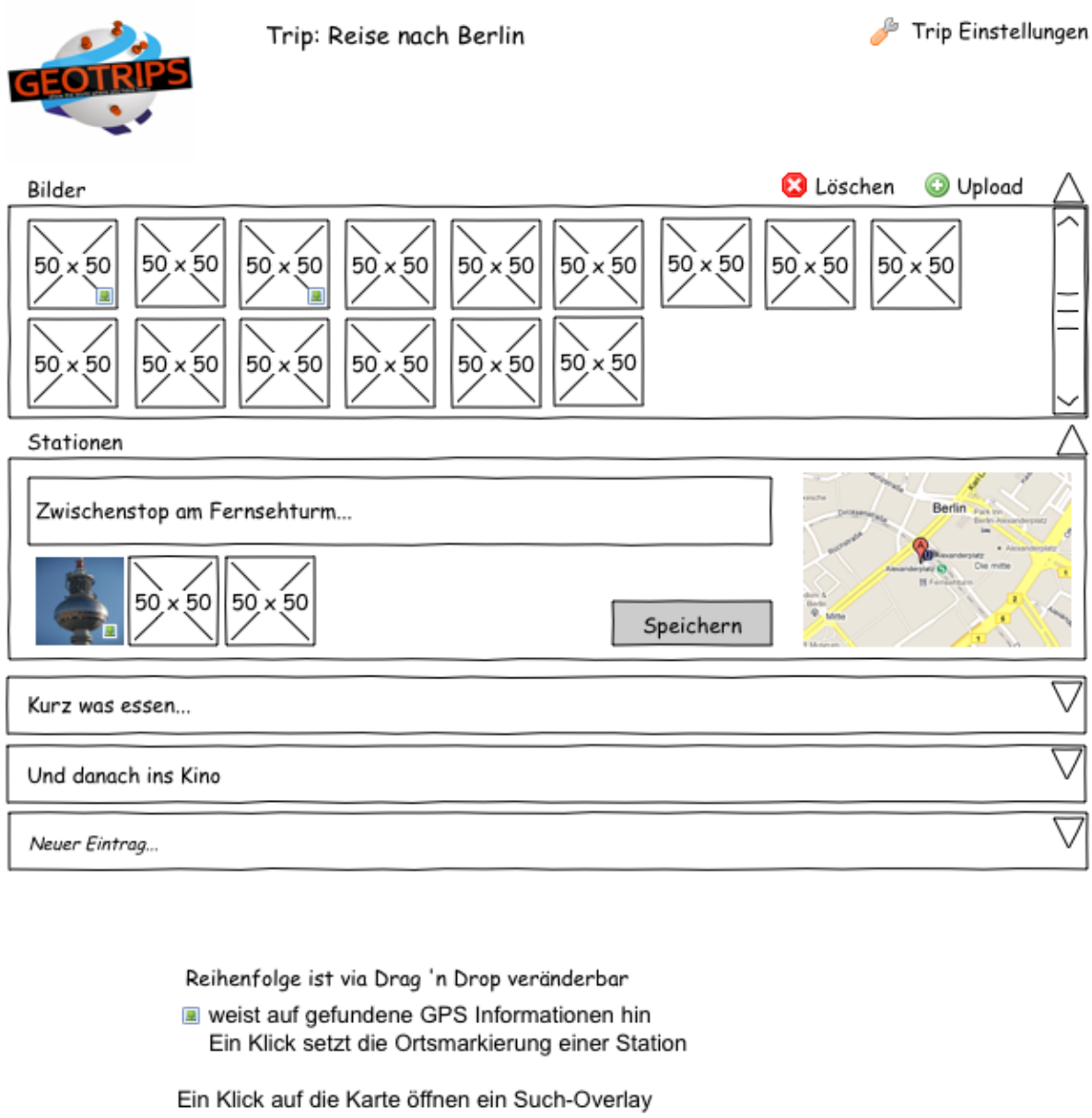

Abbildung 13: Konzeptzeichnung der Trip-Managment Seite

Bei der Realisierung wurde dieser Entwurf in einigen Punkten angepasst, damit es dem Bedienkonzept der kompletten Seite mehr entspricht. Das Endergebnis ist Folgendes:

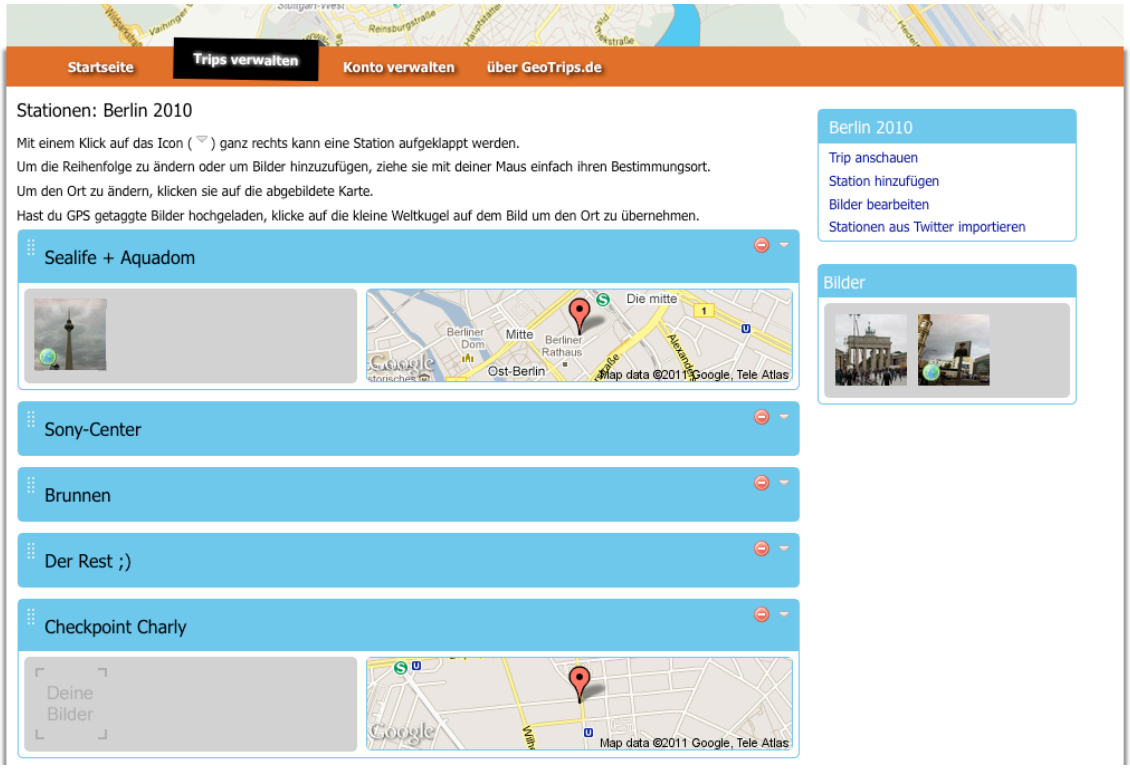

Abbildung 14: Finale Version der Trip-Managment Seite

5.5.1. Ajax

Ziel war es außerdem dem Benutzer zu ermöglichen seinen Trip möglichst schnell zu bearbeiten. Folgende Funktionalitäten sollten abgedeckt sein:

- Zuordnung von Bildern zu Stationen
- Reihenfolge der Stationen ändern
- Löschen, Umbenennen und Hinzufügen von Stationen
- Reihenfolge der Bilder in den Stationen bearbeiten

Wenn möglich sollten diese Funktionen als Drag & Drop realisiert werden. Änderungen die der Benutzer durchführt sollen direkt gespeichert werden ohne das ein Neu laden der Seite oder ein explizites speichern notwendig ist.

Für die Realisierung der Drag & Drop Funktion wurde jQuery verwendet. Die Übertragung der Daten im Hintergrund mit AJAX wurde ebenfalls mit jQuery umgesetzt.

Dies wird im Folgenden genauer erläutert:

5.5.2. Protokoll

Für die Kommunikation zwischen Browser und Client wurde ein eigenes Protokoll entwickelt. Verwendet wurde hierbei JSON. JSON ist eine Objektnotation die es erlaubt JavaScript Objekte direkt als String zu serialisieren.

Bei einem Speichervorgang werden mit Javascript im Browser die Positionen aller Elemente im DOM Baum ausgelesen. Hierbei wird die Reihenfolge der Elemente (z.B. der Stationen) berücksichtigt, sowie alle untergeordneten Elemente und deren Daten. Diese Datenstruktur sieht dann zum Beispiel folgendermaßen aus:

Abbildung 15: Beispiel eines AJAX-Datensatzes

Diese Datenstruktur wird mit Hilfe eines JSon Serializers (verwendet wurde json2.js) in einen JSon String transformiert:

```
{"stations":[{"text":"Sealife +
Aquadom","images":["61"],"id":"66","lat":"52.52","long":"13.4092"},{"text":
"Sony-
Center","images":["58","62"],"id":"67","lat":"52.5195","long":"13.4027"},{"
text":"Brunnen","images":["60"],"id":"68","lat":"52.5203","long":"13.4052"}
,{"text":"Der
Rest ;)","images":["65","66","63"],"id":"69","lat":"52.5234","long":"13.411
4"},{"text":"Checkpoint
Charly","images":[],"id":"74","lat":"52.5075","long":"13.3902"},{"text":"Br
andenburger
Tor","images":[],"id":"75","lat":"52.5163","long":"13.3782"},{"text":"Neue
Station","images":[],"id":"126","lat":"52.52","long":"13.4092"}],"trip":[{"
images":["64","59"]}]}
```

Anschließend wird dieser String an den Server geschickt. Anhand der Zielurl erkennt der Server, für welchen Trip dieses Datenpaket gedacht ist:

http://geotrips.de/trips/**16**

In diesem Beispiel ist der Trip mit der id 16 gemeint. Der Server dekodiert nun dieses Datenpaket und beginnt die Daten in der Datenbank entsprechend zu aktualisieren. Die Reihenfolge der Bilder und Stationen wird hierbei mit einem incrementellen Wert in der Datenbank festgehalten.

Zusätzlich wird überprüft ob der aktuell eingeloggte Benutzer überhaupt der Besitzer dieses Trips ist.

Wenn alles geklappt hat, wird ein positiver HTTP Code (200) an den Browser geschickt. Dieser signalisiert anschließend dem Benutzer den Erfolg der Operation.

6. Probleme und deren Lösungen

6.1. Twitter-Calpyse

In der ersten Umsetzung des Twitter-Caches kam es zu einem schwer nachvollziehbaren Fehler, welcher nur Auftrat wenn das Projekt auf dem Produktivserver ausgeführt wurde. Hierbei wurde die id von Tweets nicht richtig gespeichert.

Der signifikante Unterschied zwischen Test- und Produktivserver: Die verwendete Datenbank. Auf Testseite wurde SQLite3 verwendet, auf Server Seite mySQL.

Die Ursache war in der „Twittercalypse“ zu finden: Ab einem bestimmten Tag waren auf Twitter derart viele Tweets gespeichert, das der maximale Wert des 32bit Integer überschritten wurde. Dies war die Ursache dafür, dass in der Datenbank der Wert nicht mehr gespeichert werden konnte.

Ein Ändern des Datentypes von einem numerischen zu einem String-Datentyp in der Datenbank konnte dieses Problem lösen.

6.2. JQuery Ajax Cache Bug

Bei der Erstellung wurde die zu dem damaligen Zeitpunkt aktuelle jQuery Version 1.4.1 verwendet. Hierbei machte ein Fehler in jQuery Probleme:

Bei Absetzen einer Ajax-Anfrage vom Typ JSON muss der Cache des Browsers deaktiviert werden. Laut der Dokumentation geschieht dies automatisch wenn der Typ der Anfrage JSON ist. Das ist aber falsch: Der Cache wurde nicht deaktiviert. Daraufhin wurde folgendes seltsame Verhalten ausgelöst.

Szenario: Der Benutzer befindet sich auf der Seite für die Verwaltung der Stationen und führt dort Änderungen durch. Im Hintergrund werden nun Ajax-Anfragen durchgeführt welche diese Änderungen an den Server übermitteln. Die Zieladresse dieser Hintergrundanfragen ist in diesem Fall aber identisch zu der aktuell dargestellten Seite: Der Server entscheidet nämlich anhand des HTTP Anfragetypes was er tut (PUT oder GET), und nicht anhand der Adresse.

Surft der User nun anschließend auf eine andere Seite und klick dann den „Zurück“ Button in seinem Browser, bekommt er die Seite angezeigt welche momentan im Browser zwischengespeichert ist. Da die Ajax-Anfragen aber den Cache überschrieben haben, sieht er nun nicht mehr die Seite, sondern den Inhalt der Antwort des Servers auf die vorherigen Ajax-Anfragen!

Nach langer Recherche konnte das Problem gelöst werden, indem explizit der Cache deaktiviert wird anstatt sich darauf zu verlassen das dies automatisch geschieht wenn der Typ der Anfrage JSON ist wie es in der Dokumentation geschrieben steht.

Der Fehler wurde in den jQuey Bugtracker eingetragen, dann allerdings als Duplikat eingestuft da er schon bekannt war.

In der momentan aktuellen Version ist dieser Fehler nicht mehr enthalten.

7. Fazit

Das Projekt wurde in knapp sechs Wochen komplett durchgeführt. Meiner Meinung nach ist das enthaltene Set an Funktionen für diesen Zeitrahmen beachtlich.

Dank der Nutzung vieler fertiger Komponenten wurde viel Zeit gespart, und das ohne das man es dem Endprodukt ansieht.

Meiner Meinung nach ist das die Richtung in welche die Webentwicklung gehen muss: Es macht keinen Sinn, für jedes neue Projekt das Rad neu zu erfinden. Generische Dinge, wie beispielsweise das Management von Benutzerkonten (Register, Login, Logoff, ...) können ausgelagert und immer wieder verwendet werden.

Rails macht es durch seine einheitliche Struktur möglich, genau das zu tun.

Ich konnte in diesem Projekt alle gesteckten Ziele erreichen. Ich bin davon überzeugt, dass sich die Nutzerbasis enorm steigern wird wenn sich die Seite herum spricht. Eine Integration von Facebook ist der nächste logische Schritt. Wenn die eigenen Trips auf Facebook gepostet werden können wird dies Besucher anziehen. Der Netzwerk-Effekt tritt ein.

Ich konnte in diesem Projekt mein Wissen in Rails3 enorm steigern, was mir für zukünftige Projekte sehr hilfreich sein kann.

8. Ausblick

Hier Vorschläge für zukünftige Projekte mit dem Ziel Geotrips zu erweitern.

8.1. Mobiles Interface

Die Integration eines mobilen Interfaces wäre eine mögliche Erweiterung. Dies könnte mit Technologien wie JQuery-Touch speziell auf mobile Geräte optimiert werden und es erlauben, auch ohne die Nutzung von Twitter seine Karten von unterwegs zu manipulieren.

Dank neuer HTML5 Features ist es so auch problemlos möglich die aktuellen GPS Koordinaten auszulesen.

8.2. Support mehrerer Twitter-Image-Hoster

Momentan ist nur der Import von auf yfrog gehosteten Bildern möglich. Diese Funktion könnte modular aufgebaut werden um den Import von allen bzw. den meist genutzten Hostern zu ermöglichen.

Hierbei können Anpassungen in der Ziel-URL, der URL des exakten Bildes bis zum dynamischen Auslesen selbiges durchgeführt werden.

8.3. Einbau neuer Diashows

Es könnten neue Diashows erstellt werden. Hierbei ist auch die Nutzung proprietärer Technologien wie Flash vorzustellen.

8.4. Facebook Integration

Eigene Trips sollen auf Facebook gepostet werden können. Eventuell ist sogar eine eigene Facebook App möglich. Diese Maßnahmen sind eine gute Möglichkeit die Bekanntheit des Projektes enorm zu steigern.

9. Abbildungsverzeichnis

10. Quellen

[1] Devise Gem: https://github.com/plataformatec/devise
[2] EXIFr Gem: https://github.com/remvee/exifr
[3] Capistrano: https://github.com/capistrano/capistrano/wiki/
[4] Mime-Types Gem: https://github.com/halostatue/mime-types
[5] Netbeans: http://netbeans.org/
[6] RubyMine: http://www.jetbrains.com/ruby/
[7] Git: http://git-scm.com/
[8] Dropbox: http://dropbox.com/
[9] OAuth: http://oauth.net/
[10] OmniAuth: https://github.com/intridea/omniauth
[11] Twitter Gem: https://github.com/jnunemaker/twitter
[12] Twitter-Api: http://apiwiki.twitter.com/w/page/22554648/FrontPage
[13] yFrog: http://yfrog.com/
[14] Pikachoose: http://pikachoose.com/
[15] SlidesJs: http://slidesjs.com/
[16] jquery: http://jquery.com/
[17] http://www.uploadify.com/
[18] https://github.com/websymphony/Rails3-Paperclip-Uploadify