Tatyana Merlo – Natalie Stalder – Nadja Stadelmann

Travelling I

M133 – Webapplikation realisieren

# Einleitung

Bei einer Diskussion über unsere bisherigen Reiseziele ist uns aufgefallen, dass es schwierig sein kann, sich an alle Orte zu erinnern, welche man bisher besucht hat. Daher kam uns die Idee, dass dies in einem Webprogramm gut ersichtlich dargestellt werden kann. Ausserdem wären die Daten so von überall her abrufbar (angenommen wir stellen die Seite online).

## Systemanforderungen

Wir testen die Anwendung in Google Chrome und in Firefox. Ob die Seite in anderen Browsern ebenfalls funktioniert können wir daher nicht garantieren.

Der Webserver muss ein PHP-Interpreter installiert haben, die Browser der Clients muss JavaScript-fähig sein.

Ausserdem muss eine Verbindung vom Webserver an einen Datenbankserver vorhanden sein, damit die Daten abgespeichert und abgerufen werden können.

## Dokument

### Aufbau

* Analyse
  + Beschrieb und erste Aufsplittung in einzelne Teilbereiche
* Vorgehen
  + Abhängigkeiten, Aufgabenübersicht
* Design
  + Detaillierte Analyse DB-Design
  + Detaillierte Analyse Programmstruktur
* Implementation
* Testing
  + Testcases basieren auf UseCases

Inhalt

[Einleitung 1](#_Toc497135221)

[Systemanforderungen 1](#_Toc497135222)

[Dokument 1](#_Toc497135223)

[Aufbau 1](#_Toc497135224)

[Vision 4](#_Toc497135225)

[Erweiterungsmöglichkeiten 4](#_Toc497135226)

[Analyse 5](#_Toc497135227)

[Anforderungen 5](#_Toc497135228)

[Must have 5](#_Toc497135229)

[Nice to have 5](#_Toc497135230)

[Mockups 6](#_Toc497135231)

[Login 6](#_Toc497135232)

[Blog 6](#_Toc497135233)

Home [6](#_Toc497135234)

[Sign up 7](#_Toc497135235)

[Use Cases 8](#_Toc497135236)

[Design 11](#_Toc497135237)

[DB 11](#_Toc497135238)

[Tabellenbeschrieb 11](#_Toc497135239)

[User 11](#_Toc497135240)

[Location 11](#_Toc497135241)

[Post 11](#_Toc497135242)

[Picture 11](#_Toc497135243)

[Visited 11](#_Toc497135244)

[Designanpassungen 12](#_Toc497135245)

[Programm 12](#_Toc497135246)

[Model 12](#_Toc497135247)

[View 12](#_Toc497135248)

[Controller 12](#_Toc497135249)

[Designanpassungen 12](#_Toc497135250)

[Implementation 13](#_Toc497135251)

[Technologien 13](#_Toc497135252)

[Testing 0](#_Toc497135253)

# Vision

Man bereist die Welt und möchte möglichst viele Orte sehen. Doch sich dabei auf einen Schlag an alle Orte zu erinnern, die besten Fotos an einem zentralen Ort lagern und allenfalls Freunden eine Hotel- oder Restaurantempfehlung liefern – das fällt oftmals schwer.

Für all diese Probleme bringen wir mit dem Travelblog ‘Travelling I’ die Lösung: Auf einer Karte kann eingetragen werden, an welchem Ort man bereits gewesen ist. Das entsprechende Land wird dabei farbig eingefärbt und die Stadt mittels eines Punktes markiert. So erhält man eine Übersicht über seine vergangenen Reisen, die man überall auf der Welt über das Internet abrufen kann!

Zu jedem Ort kann auch ein Blogeintrag geschrieben werden, um tolle Hotels, Restaurants oder Ausflugsziele festzuhalten. Dazu kann man seine schönsten, lustigsten oder emotionalsten Fotos hochladen, um immer wieder in diesen Erinnerungen zu schwelgen.

Damit diese Daten nicht einfach öffentlich preisgegeben werden, muss man sich dazu zuerst registrieren und jeweils einloggen. So werden die Ferienerlebnisse vor unerwünschten Blicken geschützt.

## Erweiterungsmöglichkeiten

In weiteren Schritten stellen wir uns vor, dass sich die User Restaurants oder vorhandene Aktivitäten an den spezifischen Orten anzeigen lassen können. Da das Wetter für Ausflüge und das Gepäck ebenfalls wichtig ist, wird auch dieses eingebunden.

Um die Suche durch seine Einträge zu erleichtern, werden später Kategorien eingebunden. So kann der User zum Beispiel die Kategorie ‘Hotel’ angeben und später ganz einfach die entsprechend markierten Einträge filtern.

Doch bei den vergangenen Reisen soll es nicht bleiben! Pläne zu schmieden für neue Ferien ist ebenfalls spannend und erfreulich. Damit dies auch locker vom Hocker geht, werden den Usern entsprechende Flüge und Hotels angezeigt, die für sie geeignet sein können.

# Analyse

## Anforderungen

### Must have

1. Login / Registrierung
2. Karte – wo warst du?
   1. Nadel auf Stadt
   2. Blogeintrag mit Fotos

## Nice to have

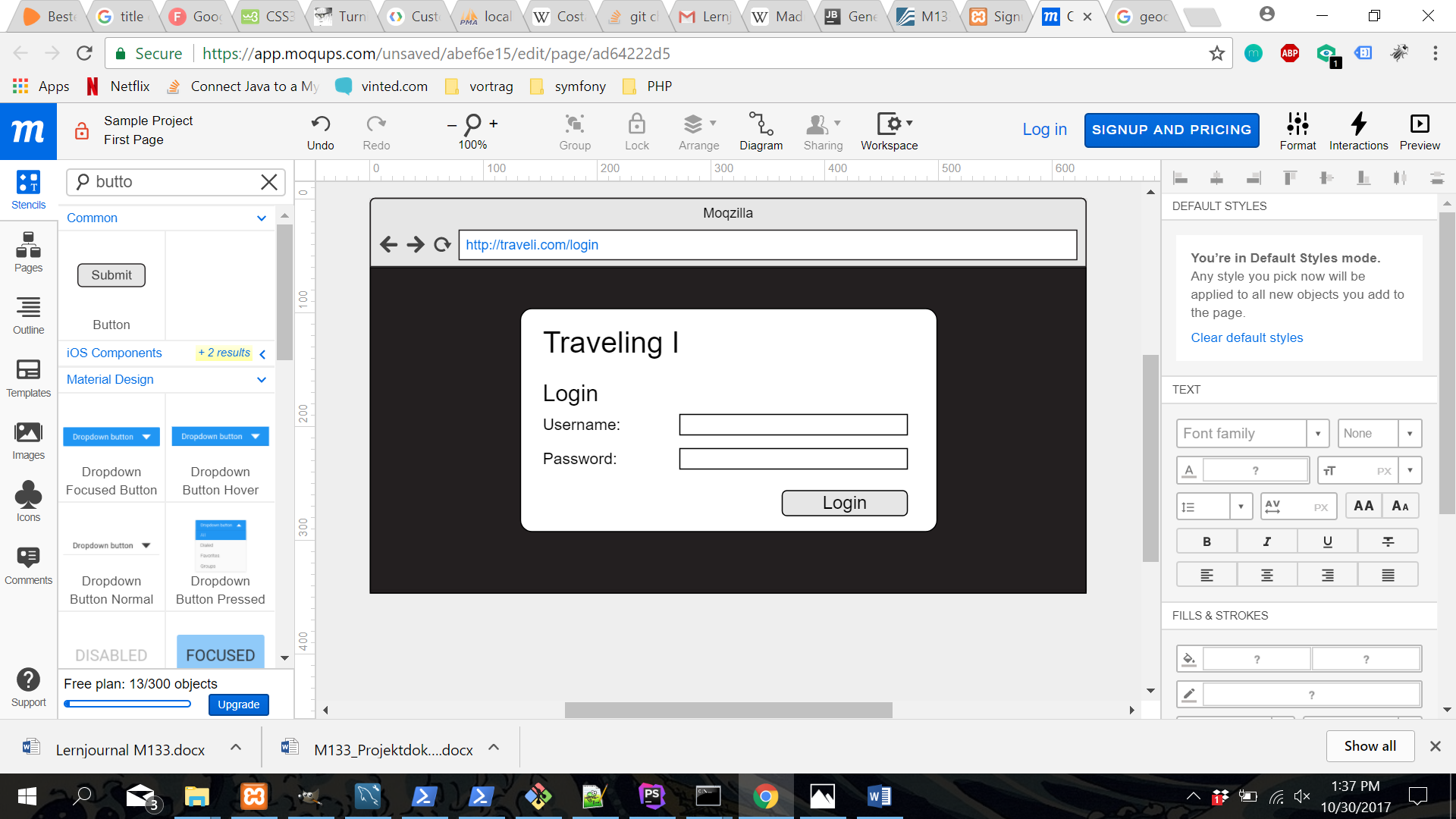
1. Google Maps einbindung
   1. Restaurants in der jeweiligen Stadt suchen
2. Wetter anzeige
   1. Name der Stadt eingeben 🡪 entsprechendes Wetter angeben
3. Vorhandene Aktivitäten anzeigen lassen
   1. Europaweit / Weltweit
   2. Tripadvisor API?
4. Blogmässige Notizen erweitern z.B. Kategorien
   1. Restaurant
   2. Hotels
   3. Festivals
   4. Einkaufscentern
   5. ….
5. Plan für nächste Reise(n)
   1. gute Flüge und Hotels anzeigen lassen

# Mockups

## Image result for geochartsLogin

Feld um Locations hinzufügen zu können.

Karte anzeigen mit Markers für die bereits hinzugefügten Locations

Home

Fehler anzeigen bei:

- Username nicht   
 vorhanden

- Falsches Passwort  
- Keine Eingaben

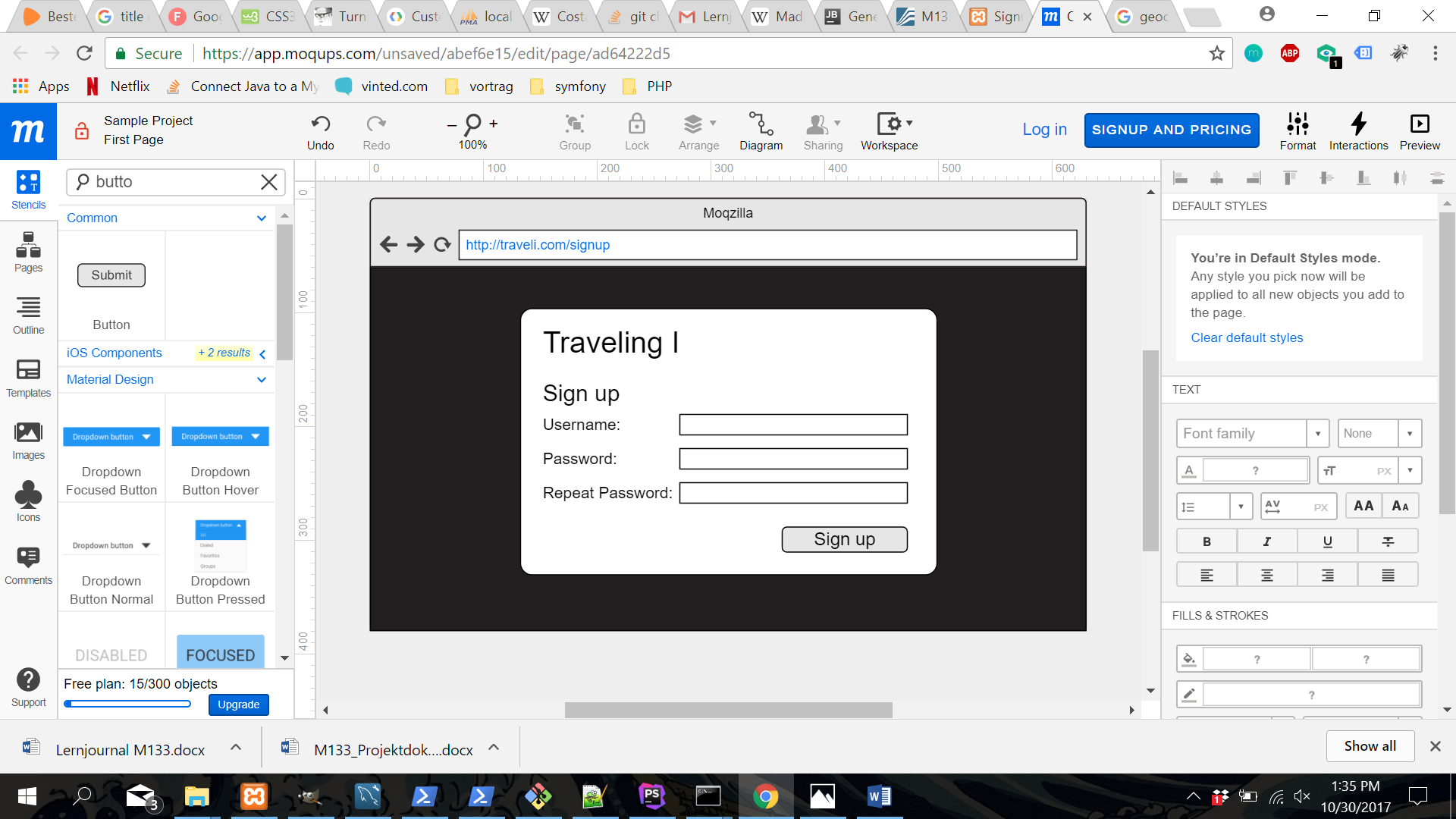
## Blog

Alle Posts anzeigen.

2 Buttons Edit und Delete für jeden Post gebraucht

## 

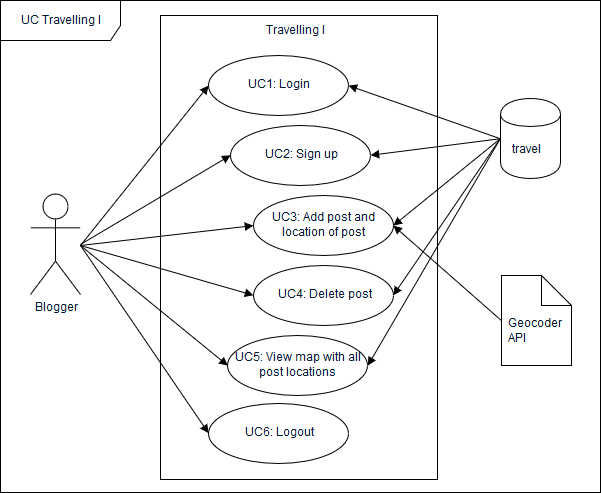
## Sign up



Überprüfung ob Passwörter gleich sind nötig, wenn nicht 🡪 Fehler anzeigen.

Username muss Unique sein 🡪 Im Backend überprüfen, wenn nicht Fehler anzeigen

## Use Cases



|  |  |
| --- | --- |
| **Identifikation** | **UC1: Login** |
| **Beschreibung** | Ein Blogger loggt sich ein. |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | Der Blogger muss bereits einen Account haben (Sign up) |
| **Trigger** | Blogger wählt "Login" |
| **Standardablauf** | 1. Link von Travelling I eingeben  2. Benutzernamen und Passwort eingeben und "Login" klicken  3. Blogger wird eingeloggt |
| **Alternative Abläufe** | 2a. Falscher (nicht in Datenbank erfasster) Benutzernamen eingegeben 🡪 Blogger kann sich nicht einloggen, Login-Seite wird erneut – mit einer Fehlermeldung - angezeigt  2b. Blogger gibt falsches Passwort an 🡪 Es wird ebenfalls eine Fehlermeldung ausgegeben und die Login-Seite neu geladen, der Benutzername bleibt.  2c. Kein Input in den Feldern 🡪 Blogger kann sich nicht einloggen, Login-Seite wird erneut – mit einer Fehlermeldung - angezeigt |

|  |  |
| --- | --- |
| **Identifikation** | **UC2: Sign up** |
| **Beschreibung** | Ein Blogger erstellt einen Account. |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | - |
| **Trigger** | Blogger wählt "Sign up" |
| **Standardablauf** | 1. Link von Travelling I eingeben  2. Auf der Login-Seite auf die Sign-Up-Seite wechseln: "Sign up here" klicken  3. Heimatsstadt suchen, Submit Button wird freigegeben  4. Benutzernamen, Passwort sowie eine Passwortbestätigung eingeben und "Sign up" wählen  5. Bloggerkonto wird erstellt und der Blogger gelangt auf die Startseite im Logged-In-Bereich |
| **Alternative Abläufe** | 4a. Bereits in Datenbank erfasster Benutzernamen eingegeben 🡪 Bloggerkonto kann nicht erstellt werden 🡪 Sign-Up-Seite wird erneut geladen mit einer Fehlermeldung  4b. Blogger gibt nicht übereinstimmende Passwortbestätigung ein 🡪 Es wird eine Fehlermeldung ausgegeben ohne die Seite neu zu laden.  4c. Blogger sendet ein leeres Formular ab 🡪 Fehlermeldung und der Blogger bleibt auf der Sign-Up-Seite  3a. Stadt nicht gefunden 🡪 Submit Button wird nicht freigegeben |

|  |  |
| --- | --- |
| **Identifikation** | **UC3: Add post and location of post** |
| **Beschreibung** | Ein Blogger fügt einen neuen Ort und den dazugehörigen Post hinzu |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | Benutzer eingeloggt |
| **Trigger** | Blogger klickt auf "Add Location to Map " |
| **Standardablauf** | 1. Auf Home Seite Ort hinzufügen, auf „Add Location to Map“ klicken  2. Posttitel, Posttext, Bilder für den Post hinzufügen  3. "Submit" wählen  4. Post wird erstellt und der Standort auf der Blogger spezifischen Weltkarte markiert |
| **Alternative Abläufe** | 1a. Location wird nicht gefunden, keine Weiterleitung auf „Edit Post“ wird vorgenommen  1b. Keine Internetverbindung vorhanden 🡪Location kann nicht gefunden werden  3a. Die ausgewählte Location hat bereits einen Posteintrag 🡪 eine Fehlermeldung wird angezeigt  3b. Die Bilder, die hochzuladen sind, sind nicht im richtigen Format/ in der richtigen Grösse etc. 🡪 Es wird ebenfalls eine Fehlermeldung ausgegeben, der Post wird ohne Bilder resp. mit den Bildern abgespeichert, welche hochgeladen werden konnten |

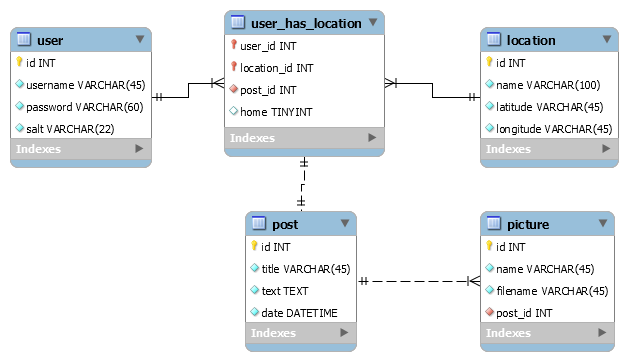
|  |  |
| --- | --- |
| **Identifikation** | **UC4: Delete Post** |
| **Beschreibung** | Ein Blogger löscht einer seiner Posts. |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | Benutzer eingeloggt |
| **Trigger** | Blogger wählt für bestimmten Post "Delete". |
| **Standardablauf** | 1. Blogger geht auf seinen Blog und wählt die "Delete"-Option, welche unter jedem Post angezeigt wird.  2. Post wurde gelöscht |
| **Alternative Abläufe** | - |

|  |  |
| --- | --- |
| **Identifikation** | **UC5: View map with all post locations** |
| **Beschreibung** | Ein Blogger sieht sich die Home Seite an |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | Benutzer eingeloggt |
| **Trigger** | Klick auf Home Menü |
| **Standardablauf** | 1. Klick auf Home Menü  2. Karte mit markierten Locations wird angezeigt |
| **Alternative Abläufe** | 2a. Internetverbindung nicht vorhanden 🡪 Karte wird nicht angezeigt |

|  |  |
| --- | --- |
| **Identifikation** | **UC6: Logout** |
| **Beschreibung** | Ein Blogger loggt sich aus. |
| **Primärer Actor** | Blogger |
| **Vorbedingungen** | Benutzer eingeloggt |
| **Trigger** | Blogger wählt die Aktion "Logout" |
| **Standardablauf** | 1. "Logout" wählen  2. Benutzer ausgeloggt |
| **Alternative Abläufe** | - |

# Design

## DB



### Tabellenbeschrieb

#### User

Der User braucht einen Usernamen sowie das Passwort. Der Username ist unique definiert. Das Passwort ist mittels BCrypt verschlüsselt – das heisst, es benötigt eine Länge von VARCHAR(60). Ausserdem wird der Salt VARCHAR(22) abgespeichert, er besteht aus einer generierten Zeichenkette.

#### Location

Hier werden die besuchten Orte abgespeichert. Wir speichern den Namen der Location, sowie die Längen- und Breitengrade VARCHAR(45) in dieser Tabelle ab, da Google GeoChart die Nadeln mittels der Koordinaten setzt.

#### Post

Im Post wird das Datum, der Titel und ein Text eingespeichert.

#### Picture

Der Bildernam, wird gespeichert. Weiter wird der Filename inklusive relativem Pfad gespeichert, damit das passende Bild geladen werden kann.

#### Visited

Zwischentabelle zwischen dem User und der Location. Mehrere User können dieselbe Stadt besuchen und ein User kann mehrere Städte besuchen. Weiter ist hier der Fremdschlüssel zum zugehörigen Post eingespeichert. Auch wird hier gespeichert, ob eine Location für den Benutzer die «Home City» ist, da diese speziell gehandhabt hat(kann nicht gelöscht werden, hat keinen dazugehörigen Post welcher angezeigt wird).

### Designanpassungen

## Programm

In unserer Filestruktur achten wir sehr darauf, dass MVC-Prinzip einzuhalten. Die verschiedenen Programmiersprachen möchten wir ebenfalls möglichst voneinander abgrenzen. So soll in unseren HTML-Files kein PHP ersichtlich sein.

### Model

Das eigentliche Datenmodell ist unsere Datenbank ‘travel’. Dennoch müssen wir natürlich auch in unserer Applikation etwas implementieren, was die Daten aus der Datenbank holt und für die Weiterverarbeitung zur Verfügung stellt. Dies machen wir via ‘Doctrine’ um eine Objektrelationale Datenbankverbindung zu verwenden. Das Konzept dahinter ist, dass eine DB-Tabelle genau eine Klasse, beziehungsweise eine Entity im PHP-Programm besitzt. Sie besteht aus Klassenvariablen mit Getter- und Setter-Methoden, wobei jede Variable für ein DB-Attribut steht. Die Verbindungen werden in den Klassen via Annotations erstellt.

### View

Unsere Views sind allesamt als HTML-Dateien abgespeichert. Zwei Templates beinhalten die Grundstruktur der Views. Eines ist für die eingeloggte Ansicht gedacht, das andere (ohne Navigation) wird für die Aktionen ausserhalb des eingeloggten Bereiches verwendet. Sprich, das Login und das SignUp erhalten ein etwas anderes Design als beispielsweise der Blog eines Users.

Allgemein ist das Layout sehr schlicht gehalten. Es soll nicht von den spannenden Reisen der User ablenken, schon gar nicht, da er seine eigenen Bilder hochladen kann.

### Controller

Der Einstiegspunkt der Applikation ist die Datei ‘index.php’, welche auf die ‘route.php’ verweist. Darin sind alle Controller gehandelt. Die abstrakte Klasse ‘Controller’ wird von allen spezifischen Controller extended. Jeder handelt einige Funktionen in einem speziellen Zuständigkeitsbereich. Der LoginController ist zum Beispiel für die Anzeige des Loginformulars zuständig sowie die Loginfunktionalität selber. Gleichzeitig checkt er die Usereingaben und logt den User bei korrekten Eingaben ein.

Die gesamte Logik ist also in den Controller eingebaut, welche die Usereingaben entgegennehmen, die entsprechenden Funktionen ausüben und danach die korrekten Views anzeigen.

In der URL sind die jeweiligen Controller und die verwendeten Methoden ersichtlich.

### Designanpassungen

//Filestruktur einfügen?

# Implementation

## Technologien

Die Sprache PHP in Kombination mit HTML wurde gemäss Vorgabe verwendet. Es wurde ausserdem eine MariaDB Datenbank erstellt.

|  |  |
| --- | --- |
| Technologie | Verwendungszweck |
| Doctrine | Objektrelationales Datenbankframework. Wurde verwendet um die Tabellen als Klassen abbilden zu können was die Verwendung in Queries erleichtert sowie auch zur Einhaltung des MVC Prinzip beiträgt da diese Klassen das Model bilden.  Ein weiterer Vorteil ist die bereits beinhaltete Sql Injection Prevention von DQL. |
| Javascript | Für AJAX Requests sowie Inputüberprüfungen verwendet. Einbindung Google GeoCoder, Google GeoCharts. |
| Google GeoCoder | Wird verwendet um die Längen- und Breitengrade von Locations Anfragen zu können. Anfrage erfolgt asynchron. |
| Google GeoCharts | Wird verwendet um die Karte anzeigen zu können. Ermöglicht Pins zu setzten und wurde gegenüber Google Maps bevorzugt da die Darstellung für unseren Zweck übersichtlicher ist. |
| AJAX | Wurde verwendet um die Abgrenzung zwischen der View und dem Controller einhalten zu können, indem die View Dateien HTML sind und nicht PHP womit Variablen nicht direkt übergeben werden können. Wird bei der Karte verwendet um die einzuzeichneten Punkte zu erhalten sowie um Locations hinzuzufügen. |

## MVC

|  |  |
| --- | --- |
| **Model:** | Entity Klassen welche die Tabellen abbilden. |
| Beispiel: (Ausschnitt von User.php) | **<?php  namespace** Travel\Entity; **use** Doctrine\ORM\Mapping **as** ORM;  */\*\*  \* User  \*  \** ***@ORM\Table****(name="user")  \** ***@ORM\Entity*** *\*/* **class** User {  */\*\*  \** ***@var*** *string  \*  \** ***@ORM\Column****(name="username", type="string", length=45, precision=0, scale=0, nullable=false,   \* unique=false)  \*/* **private $username**;  */\*\*  \* Set username  \*  \** ***@param*** *string $username  \** ***@return*** *User  \*/*  **public function** setUsername($username)  {  $this->**username** = $username;  **return** $this;  }   */\*\*  \* Get username  \*  \** ***@return*** *string   \*/*  **public function** getUsername()  {  **return** $this->**username**;  }  } |
| **View:** | HTML Klassen. Werden jeweils als Content in das Template (Entweder in template.html oder templateLogin.html eingefügt, je nachdem ob bereits ein User angemeldet ist). Felder welche mir {{ }} gekennzeichnet sind werden ersetzt. |
| Beispiel:  login.html) | <**div id="content"**>  <**h2**>Login</**h2**>  <**p id="errors" class="errors"**>{{error}}</**p**>  <**form action="?controller=Login&action=checkUserAction"   method="post"**>  <**label for="username"**>Username: </**label**>  <**input name="username" type="text" id="username"**  **value={{username}}**><**br**>  <**label for="password"**>Password: </**label**>  <**input name="password" type="password" id="password"**><**br**>  <**input class="button btn" type="submit" value="Login"**>  </**form**> </**div**>  <**p**>You don't have a Travelling I Account yet? <**a href="/travel/travel/Source%20Files/src/index.php?controller=SignUp&action=signUpAction"**>Sign up here...</**a**></**p**> |

|  |  |
| --- | --- |
| **Controller:** | Php Klassen welche Views aufrufen und Daten vom Model holen. Befinden sich im src/Controller Ordner, erben jeweils von der Oberklasse Controller, welche Grundfunktionalität wie getCurrentUser() und getEntityManager() bereitsstellt. |
| Beispiel: (LogoutController.php) | **<?php  namespace** Travel\Controller;  **class** LogoutController **extends** Controller {  **public function** logoutAction($html)  {  *session\_register\_shutdown*();  **unset**($\_SESSION[**'user'**]);  $html = *file\_get\_contents*(**"../app/resources/view/templateLogin.html"**);  $html = *str\_replace*(**"{{goToResource}}"**, **'../app/resources/'**, $html);  $login = **new** LoginController();  **echo** $login->loginAction($html);  } } |

# Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Testfall Nummer | Use Case | Testbeschrieb | Erwartetes Resultat | Tatsächliches Resultat | Tester | Testdatum | Status |
| **1** | **UC1** | **Login** |  |  |  |  |  |
| 1.1 |  | Ein Blogger loggt sich mit in der DB vorhandenem Usernamen und Passwort ein | Der Blogger wird eingeloggt, er gelangt auf seine Home-Seite | Der Blogger wird eingeloggt, er gelangt auf seine Home-Seite | Natalie | 30.10.2017 | OK |
| 1.2 |  | Ein Blogger will sich mit nicht in der DB vorhandenen Usernamen einloggen. | Der Blogger gelangt erneut auf die Login-Seite, die ihm die Fehlermeldung "Invalid Credentials." ausgibt. | Der Blogger gelangt erneut auf die Login-Seite, die ihm die Fehlermeldung "Invalid Credentials." ausgibt. | Natalie | 30.10.2017 | OK |
| 1.3 |  | Ein Blogger will sich mit in DB vorhandenem Usernamen und falschem Passwort einloggen. | Der Blogger gelangt erneut auf die Login-Seite, die ihm die Fehlermeldung "Invalid Password." ausgibt. | Der Blogger gelangt erneut auf die Login-Seite, die ihm die Fehlermeldung "Invalid Password." ausgibt. |  |  |  |
| **2** | **UC2** | **Sign Up** |  |  |  |  |  |
| 2.1 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |