Masterprojekt Dokumentation

“Where2Fly”

Phil Taubert

Daniel Mertens

16 MIM - HTWK Leipzig

Betreuer: Herr Prof. Dr. Michael Frank

Inhaltsverzeichnis

1. Projektziel 2
1a. Allgemeines Ziel 2
1b. Funktionen 2
1c. Zielgruppe 3
2. Verwendete Technologien 4
2a. Meteor 4
2b. MongoDB 5
2c. Google Maps Javascript API 5
3. Umsetzung 6
3a. Projektstruktur 6
3b. Packages 7
3c. Codebeispiele 7
4. Design 9
4a. Bootstrap mit SCSS und Meteor 9
4b. Struktur des Templates 10
5. Anmerkungen 11
5a. Probleme bei der Umsetzung 11
5b. Sicherheit 12
5c. Bildrechte 12
6. Zusammenfassung/Ausblick 13
7. Arbeitsnachweis 14

1. Projektziel

1a. Allgemeines Ziel

Ziel des Projektes ist eine Website zu entwickeln, welche sich als (Austausch-) Plattform an Drohnenbesitzer richtet. Es sollen Erfahrungen und Tipps ausgetauscht werden können zu den Punkten:

- Sicherer Luftraum für mehr Klarheit und über Rechtslage an bestimmten Orten
- Schnelles finden von neuen geeigneten Flugplätzen und Motiven
- genereller Austausch mit anderen Piloten zu Technik, Bedienung, etc.

Dafür werden die Informationen von Nutzern zusammengetragen, überprüft und auf einer Karte dargestellt. Dies könnten Plätze (Bsp.: Sehenswürdigkeiten) oder Bereiche (Bsp.: Seen) sein. Als technische Grundlage dienen hierfür die Koordinaten. Informationen zu den Plätzen sind:

- Objektname
- öffentlich / Privatgelände
- Flug (nur mit vorheriger Absprache) erlaubt?
- Einschränkungen
- Bilder
    - Möglichkeit eigens gehostete Bilder über URL anzeigen
    - API eines externen Imagehosters (Imgur) verwenden
- Bemerkungen

1b. Funktionen

- Startseite
    - Suche direkt nach Objekten
        - Objekt bereits eingetragen → Kurzinfo und Link zu Detailseite
        - Objekt nicht eingetragen → Möglichkeit neuen Datensatz anzulegen
    - in der Nähe eines bestimmten Ortes
    - durch html5-Lokalisierung beim aktuellen Standort des Nutzers

- “Feed”, Sammlung letzter Beiträge
    - letzte angelegte Objekte
    - letzte Bemerkungen
    - letzte Bilder
- Profilseite
    - angelegte Plätze
    - Favorisierte Plätze
    - verfasste Bemerkungen
    - hochgeladene Fotos
    - Nutzerdaten einsehen
        - Name, Kontaktdaten, Link zu Socialmedia-Profilen, gewerbliche Nutzung, UAV-Informationen
    - auf eigenem Profil besteht die Möglichkeit alle Daten zu editieren oder zu löschen
- Detailseite eines Standortes
    - Informationen zum Standort
        - Profil des Nutzers, Koordinaten, FlightLight, öffentliches/privates Gelände
    - Interaktionsmöglichkeiten
        - Ort favorisieren, Bemerkung schreiben, Foto hochladen, Informationen zum Ort bearbeiten
- Seitenübergreifende Funktionen
    - Sprache ändern (Deutsch/Englisch)
    - Melden bei Verstoß gegen Rechte (Bsp.: Persönlichkeitsrecht/Flugrecht)

1c. Zielgruppe

Die Zielgruppe beschränkt sich nicht nur auf private und gewerbliche Piloten von Multicoptern mit Kameras, sondern schließt auch Personen die Luftaufnahmen von bestimmten Orten suchen mit ein. Durch das Profil von jedem Nutzer besteht die Möglichkeit der direkten Kontaktaufnahme. So entsteht zusätzlich eine Plattform über die Piloten für Luftaufnahmen gefunden werden können.

2. Verwendete Technologien

2a. Meteor

In der Planungsphase des Projekts wurde die Umsetzung mit Drupal als Grundlage beschlossen. Drupal ist ein flexibles und professionelles Content Management System, geschrieben in PHP.
Im Laufe des Semesters haben wir im Modul “Webtechnologien” jedoch das Meteor Framework kennengelernt, mit welchem die Programmierung von interaktiven Webapplikationen stark vereinfacht wird. Meteor ist ein Full-Stack Framework, damit werden Backend sowie Frontend entwickelt. Dabei wird ausschließlich Javascript als server- und clientseitige Programmiersprache verwendet, was die Integration der Google Maps Javascript API erleichtert. Da bei Where2Fly Interaktionen mit einer Karte im Vordergrund stehen, wurde Meteor anstatt Drupal als Basis verwendet.
Ein weiterer Vorteil des Meteor Frameworks ist die große Anzahl einfach zu integrierender, von Drittentwicklern geschriebener Pakete (“Meteor Packages”), welche eine einfache Implementierung grundlegender Funktionen wie eine Benutzeranmeldung ermöglichen.
Außerdem legt Meteor viel Wert auf die so genannte Reaktivität, durch welche Änderungen an persistenten Daten sofort auf sämtlichen verbundenen Clients sichtbar werden. Dies wird durch das eigens dafür entwickelte *Distributed Data Protocol* (DDP) realisiert, welches auf einer Publish-Subscribe Architektur basiert. Dabei wird mittels Javascript eine Verbindung über Websockets zum Server aufgebaut, über welche die Datenbank abgefragt wird sowie Ergebnisse und Veränderungen der Daten sofort gesendet werden.[1]

[1] https://blog.meteor.com/introducing-ddp-6b40c6aff27d

2b. MongoDB

Meteor bietet eine tiefgehende Integration mit dem dokumentenbasierten Datenbanksystem MongoDB. Dieser wird als persistente Datenspeicherung, aber auch als sich ständig veränderte Datenschicht angesehen. So werden alle Daten, die bei der Interaktion mit der Applikation anfallen, in MongoDB verwaltet.
Bei diesem Projekt betrifft dies unter anderem die Nutzer mit gespeicherten Orten und sämtlichen Informationen. Diese werden als *Collections gespeichert*, deren einzelne Einträge Javascript-Objekten ähneln. Diese können ohne vordefiniertes Schema gespeichert und abgerufen werden.[2]

2c. Google Maps Javascript API

Für die Darstellung der Karten sowie Markierungen der gespeicherten Orte wird Googles Kartendienst mit der Google Maps Javascript API integriert. Diese bettet eine Landkarte in verschiedenen Konfigurationen auf der Website ein und nimmt Parameter für Markierungen entgegen, die dann auf der Karte an den richtigen Stellen angezeigt werden.
Die API ist bis zu 25.000 geladenen Karten am Tag kostenlos, was für die Entwicklung sowie eine lange Zeit nach der Veröffentlichung ausreichend sein wird.[3]

[2] https://guide.meteor.com/collections.html#mongo-collections
[3] https://developers.google.com/maps/pricing-and-plans/#details

3. Umsetzung

3a. Projektstruktur

Zur gemeinsamen Entwicklung und Versionskontrolle wurde GitHub verwendet, wo der gesamte Code verfügbar ist.[4]
Die Ordnerstruktur eines Meteor Projekts besteht im wesentlichen aus zwei Teilen: dem Server- und dem Client-Code. In den Ordnern *client* und *server* liegt jeweils eine Javascript-Datei mit dem Namen *main.js* als Einstiegspunkt. Hier werden nun alle benötigten Dateien importiert. Diese anwendungsspezifischen Dateien befinden sich im Ordner *imports*. Dies wird in der Dokumentation von Meteor empfohlen, da somit nur die Dateien, welche wirklich benötigt werden, vom Meteor build System gebündelt und ausgeliefert werden.
In diesem Ordner befinden sich 3 Unterordner. In *api* befindet sich die Konfiguration der MongoDB-Collections. Der Ordner *startup* wird zur Konfiguration von Funktionen wie der Nutzerverwaltung und URL-Pfade verwendet. Im dritten Ordner *ui* wird das Frontend implementiert.
Um die Struktur übersichtlich zu halten, wurden in Letzterem weitere Unterordner für einzelne Bereiche der Website angelegt. So befindet sich beispielsweise im Ordner *detail* eine Template-Datei *detail.html* sowie eine Javascript-Datei *detail.js*. In der Template-Datei wird die Struktur der Komponente mit HTML festgelegt. Darin kann mit einer Template-Engine namens *Blaze* auf diverse Daten und Funktionen der zugehörigen Javascript-Datei über doppelt geschweifte Klammern zugegriffen werden. Hierbei spielt die oben angesprochene Reaktivität eine Rolle. Man benötigt keine Methoden zur Modifizierung des HTML-DOMs bei einer Veränderung von Daten, da diese vom Framework automatisch generiert werden.
Im Ordner *.meteor* befinden sich Konfigurationsdateien für das Framework an sich, beispielsweise eine Datei *packages* welche alle installierten Meteor-Bibliotheken listet.
Der Ordner *public* enthält öffentliche Dateien wie Logos und Bilder, welche von allen Clients abrufbar sein sollen.

[4] https://github.com/Tapematch/Where2Fly

3b. Packages

Vorgefertigte Programmpakete, die Meteor Packages, erleichtern die Integration häufig genutzter Funktionen. Sie werden über die Kommandozeile mit dem Befehl *meteor add <packagename>* installiert.
So sind beispielsweise verschiedene Pakete für die Nutzeranmeldung verfügbar. Das Paket *accounts-password* implementiert eine Registrierung von Nutzern mit E-Mail-Adresse und Passwort. Ergänzend wurde das Paket *useraccounts:bootstrap* installiert, welche die Einbindung einer Login- und Registrier-Oberfläche mit Bootstrap über den Blaze-Aufruf *{{> atForm}}* ermöglicht.
Ein weiteres verwendetes Paket ist *tap:i18n*. Dies ermöglicht eine einfache Internationalisierung der Webanwendung. Hierfür werden Sprachdateien im Ordner *i18n* im Projektverzeichnis abgelegt, welche automatisch vom Paket geladen werden. Mit dem Paket *tap:i18n-ui* und dem Blaze-Aufruf *{{> i18n_dropdown}}* wird eine Sprachauswahl geöffnet. Je nachdem welche Sprache ausgewählt wird, wird mit *{{_ "stringName"}}* der entsprechende Text aus der aktuellen Sprachdatei geladen.
Neben standardmäßig bei jedem neuen Projekt integrierten Paketen wie Blaze und die Anbindung an MongoDB wurden weitere zusätzliche Pakete installiert. Diese steuern beispielsweise das Rendern der richtigen Komponenten nach URL-Pfaden oder integrieren die Anbindung der Imgur-API, welche für den Bilderupload verwendet wird.

3c. Codebeispiele

Alle Orte werden mit ihren Koordinaten, dem Besitzer und sämtlichen weiteren Eigenschaften in einer Collection namens *Places* gespeichert, welche in *imports/api/places.js* deklariert wird:

```
export const Places = new Mongo.Collection('places');
```

Im Template, welches die Karte auf der Startseite generiert, wird nun auf diese Collection zugegriffen. Folgender Befehl sucht alle Orte mit vorher festgelegten Filtereinschränkungen heraus und liefert einen Cursor, über welchen sämtliche Daten abgerufen werden können:

```
places = Places.find({$and: [privatePropertyFilter, filter]});
```

Jeder Ort dieses Cursors wird nun mit den Koordinaten an die Google Maps API übergeben, welche über ein Paket beim Rendern der Seite geladen wird und die Orte als Markierungen auf der Karte darstellt. Dabei werden reaktive Funktionen genutzt, um neue Marker ohne ein Neuladen der Seite anzuzeigen oder herausgefilterte Marker verschwinden zu lassen.

Besucht man die Detailseite eines Ortes, indem man einen Marker anklickt, werden verschiedene Daten zum jeweiligen Ort angezeigt, wie die gespeicherten Bilder. Diese werden in einer eigenen Collection *Photos* gespeichert. Im Javascript der Detailseite unter *imports/ui/detail/detail.js* werden in einem Helper alle Fotos zur entsprechenden Orts-ID abgerufen:

```
photos() {
    var pid = FlowRouter.getParam("pid");
    return Photos.find({"place": pid}, {sort: {createdAt: -1}});
}
```

Auf diesen Helper greift das Template *detail.html* im gleichen Ordner über einen Blaze-Aufruf zu. Dabei wird für jedes gefundene Photo das Template *photo* aufgerufen, welches unter *imports/ui/photo* definiert wird.

```
<h3>{{_ "photos"}}</h3> <!-- Internationalisierte Überschrift -->
<div class="row">
    <div class="gallery">
        {{#each photos}}
            {{> photo}}
        {{/each}}
    </div>
</div>
```

In diesem Template wird dann ein Vorschaubild, Verweise zum Ort und dem Profil des Besitzers, sowie verschiedene Aktionen zum Löschen und Melden des Fotos angezeigt. Aufgrund der Reaktivität von Meteor wird ein neues Foto bei allen Clients ohne Seitenreload sofort angezeigt, indem das DOM modifiziert und das zusätzliche Template in die angezeigte Seite geladen wird.
Mit diesem Prinzip werden auch alle weiteren Daten wie Bemerkungen und Profile geladen und angezeigt.

4. Design

Das Design für die Website wurde aus dem Logoentwurf abgeleitet. Die Idee war einen typischen Marker von Karten wie Google Maps so zu modifizieren, dass die Bedeutung der Drohne verständlich wird. Erreicht wurde dieses Vorhaben durch die Abbildung eines Propellers im Kopf des Standortmarkers. Anhand des Logos wurden anschließend die Formen und Farben auf das Template abgeleitet. Das Template wurde mit Hilfe des Bootstrap-Frameworks realisiert und ist somit responsive und nach aktuellen technischen Standards entwickelt, um auf verschiedenen Endgeräten ein optimales Nutzererlebnis zu gewährleisten.

4a. Bootstrap mit SCSS und Meteor

Um Bootstrap mit Meteor zu verbinden, bietet die Meteor-Community ein Bootstrap-Package an, welches mit *meteor add twbs:bootstrap* installiert wird. Um SCSS in den Stylesheets zu verwenden müssen noch zwei weitere Pakete installiert werden:
meteor add fourseven:scss
meteor add reywood:bootstrap3-sass
Nachdem die Bibliotheken installiert sind, werden sie in der main.scss über *@import '{reywood:bootstrap3-sass}/bootstrap';* in das Projekt geladen und stehen dem Entwickler zur Verfügung.
In unserer Projektstruktur befinden sich die Stylesheets unter */imports/ui/scss/**. Für eine bessere Übersicht wurde für jedes Template eine eigene Datei angelegt, die in der main.scss importiert wird.

4b. Struktur des Templates

Das Hauptaugenmerk der Website liegt auf den Geoinformationen und gekennzeichneten Plätzen. Deshalb wird auf jeder Seite ganz oben die Karte mit den jeweiligen relevanten Informationen geladen, wie z.B. die neuesten Plätze oder eigene Favoriten auf der Profilseite. Darunter befinden sich weitere relevante Informationen auf die jeweilige Absicht des Nutzers abgestimmt. So entsteht ein einheitliches Bild und der Benutzer muss sich auf den Unterseiten nicht neu orientieren.
Alle Formularbearbeitungen finden in sogenannten Modals statt. Wenn beispielsweise ein Ort bearbeitet werden soll, wird die Website ausgegraut und das Formular sichtbar über die Website gelegt. So bleibt der Nutzer in der Kenntnis, auf welcher Seite er ist und verliert die Orientierung nicht. Nach dem Absenden des Formulars wird das Modal geschlossen und der Datensatz ist ohne Browser-Aktualisierung auf dem neusten Stand.
Die Farben für die Buttons sind wie folgt festgelegt:

- grün (primarycolor): Weiter/nächster Schritt
- gelb (warning): Abbruch einer Aktion
- rot (danger): Löschen (unwiderruflich)
- grau (report): Melden von Fehlern/Rechtsverstößen usw.

Des Weiteren sind die Buttons mit zusätzlichen Icons versehen um die Funktion des jeweiligen Buttons zu verdeutlichen.

5. Anmerkungen

5a. Probleme bei der Umsetzung

1. Einarbeitung in Meteor
   Die größte Herausforderung war die Einarbeitung in Meteor. Es waren nur wenig praktische Erfahrungen vorhanden, auf welche wir zurückgreifen konnten. Daher war eine tiefergehende Einarbeitung in die Struktur und in die technischen Abläufe des Frameworks notwendig.
2. Umstellung auf MongoDB
   Meteor arbeitet ausschließlich mit MongoDB und somit war auch eine Einarbeitung in diese neue Technologie notwendig. Die Beispiele und Dokumentation von Meteor hat uns grundlegend unterstützt. Dennoch stellte uns der Umgang mit der Datenbank und der GoogleMaps-Api vor eine Hürde, welche dementsprechend auch viel Zeit gekostet hat.
3. Meteors Template-Engine
   Zu jedem modularen Template gibt es eine .html und eine .js Datei. Die .js Datei enthält die Intelligenz, die .html strukturiert die Ausgaben. Es war schwierig einen Überblick zu bekommen, wie diese Dateien miteinander kommunizieren, z.B. beim Auswerten von Formulareingaben oder dem Umgang bei bestimmten Events.
4. Einbindung externer Libraries und Dienste
   Bei der Verwendung externer Programmpakete und APIs ist man immer von anderen Entwicklern abhängig. Es muss sichergestellt werden, dass die Bibliotheken auch bei Veränderungen der Voraussetzungen, beispielsweise eine Versionsänderung von Meteor, weiterhin wie erwartet funktionieren. Zudem ist zu beachten, dass auch eingebundene Schnittstellen wie die Imgur API oder die Google Maps Javascript API in Zukunft Veränderungen unterlegen sein können. Eine Gewährleistung, dass sämtliche externen Dienste jederzeit funktionieren ist nicht gegeben. Sämtliche Funktionen müssen also regelmäßig kontrolliert und gegebenenfalls angepasst werden.

5b. Sicherheit

Bei der Entwicklung einer interaktiven Webapplikation mit Meteor und dessen Benutzersystem müssen zwei Sicherheitsaspekte beachtet werden.
Zur Erstellung eines ersten Prototyps wird bei jedem neuen Meteor Projekt das Paket *insecure* installiert. Dies ermöglicht eine Modifikation der gesamten angebundenen Datenbank von allen Clients aus. Bei einer Veröffentlichung stellt dies ein Sicherheitsrisiko dar, da serverseitig nicht geprüft werden kann, ob ein Nutzer zu einer bestimmten Aktion berechtigt ist.
Aus diesem Grund muss dieses Paket vor der Veröffentlichung auf einem Produktionssystem entfernt werden und alle Transaktionen in Methoden auf dem Server ausgelagert werden. Diese Methoden werden dann serverseitig ausgeführt, sodass darin eine Überprüfung des angemeldeten Benutzers stattfinden kann. In diesem Projekt wurden die Methoden in den jeweiligen Collections unter */imports/api* definiert und können an beliebiger Stelle über *Meteor.call()* aufgerufen werden.
Ein weiteres Paket, welches die Erstellung des Prototypen vereinfacht, ist *autopublish*. Dies bewirkt, dass alle Collections der Datenbank (bis auf sensible Daten der Benutzer) für jeden Client sichtbar sind, da sie automatisch vom Server veröffentlicht (“publish”) werden. Gibt es private Daten, welche nur für bestimmte Nutzer sichtbar sein sollten, müsste man auch dieses Paket entfernen und die Veröffentlichung der Daten serverseitig manuell implementieren. Da es in diesem Projekt bisher keine privaten Daten gibt und sämtliche Orte, Bilder, Kommentare und Profile von allen Nutzern eingesehen werden dürfen, konnte dieser Aspekt vorerst vernachlässigt werden. Bei jeder Erweiterung des Funktionsumfangs sollte jedoch auf diese Funktion geachtet werden.

5c. Bildrechte

Damit die Plattform keine Bildrechte verletzt wurde Wert darauf gelegt, dass die Bilder nicht direkt auf dem Server gehostet werden. Dieses Problem wurde umgangen, indem nur eine Verlinkung zu einem Bild oder der Upload zum Image-Hoster imgur.com erlaubt wird. Imgur stellt eine Programmierschnittstelle bereit, in der ein Bild entgegengenommen, gespeichert und die Bild-URL zurückgegeben wird. In der Datenbank des Projekts wird somit nur die URL gespeichert. Weitere rechtliche Aspekte werden in der Nutzungsvereinbarung beim Erstellen eines Accounts und im Datenschutz geregelt.

6. Zusammenfassung/Ausblick

Abschließend können wir auf ein sehr lehrreiches und zukunftsorientiertes Projekt zurückblicken. Die Einarbeitung in das Meteor-Framework war schwierig, dennoch denken wir, dass diese Art von Frameworks in Zukunft in der Webentwicklung eine große Rolle spielt. Gerade durch das eigene Protokoll, welches Daten lesen und schreiben kann ohne das Neuladen der Seite, ist besonderer Vorteil. Des Weiteren bietet Meteor auch eine Funktion an, welche eine entwickelte Anwendung direkt in eine native iOS- oder Android-App exportieren kann, was eine sehr komfortable Lösung ist um noch mehr Menschen mit seiner Anwendung zu erreichen.
Wir wollen die Anwendung veröffentlichen, sobald alle Sicherheitsaspekte und rechtlichen Fragen geklärt sind. Sie wird unter http://where2fly.net/ erreichbar sein.
Ein Problem in Zukunft könnte die Abhängigkeit von Drittentwicklern der Packages sein. Sie müssen sicherstellen, dass ihre Pakete auch in Zukunft mit neueren Meteorversionen funktionieren. Genauso verhält es sich mit der API von GoogleMaps und IMGUR.

7. Arbeitsnachweis

ungefährer Zeitaufwand	Projektplanung
10 h	Ausarbeitung Projektidee / Projektplanung
20 h	Einarbeitung in das Thema Drupal/Meteor, Entscheidung über verwendete Technologien
16 h	Einarbeitung in Meteor
4 h	Projektplanung in Meteor (Datenhaltung, Struktur)

ungefährer Zeitaufwand	GoogleMaps API
3 h	Einarbeitung GoogleMaps API
5 h	GoogleMaps API einbinden
15 h	Standorte auf Karte anzeigen
15 h	neue Standorte hinzufügen, Marker auf Karte verschieben
6 h	Filterfunktion für die Karte

ungefährer Zeitaufwand	Funktionen
10 h	Nutzer-/Rechtesystem
20 h	Bilderupload
3 h	Bemerkungen zu jedem Standort
5 h	Internationalisierung
2 h	Meldefunktion
10 h	Favorisierte Orte
20 h	Neueste Aktivitäten (Feed)
20 h	Profilseiten der Nutzer
10 h	URL-Routing

ungefährer Zeitaufwand	Design/Frontendentwicklung
15 h	Design und Template Erstellung für Start und Unterseiten
13 h	verschiedene Template Sections eingerichtet (Maps, Profile, Feed, Detailseiten)
2 h	Integration von Bootstrap und Bootstrap-SCSS
3 h	Implementation von Modals für Formulare
3 h	FlightLight und PrivateProperty realisiert

ungefährer Zeitaufwand	Webserver
4 h	Einarbeitung und Analyse der Anforderungen an den Webserver
4 h	Installation und Einrichtung des Webservers und der Datenbank
2 h	Bündeln und Deployment der Anwendung

ungefährer Zeitaufwand	Gesamt
50 h	Projektplanung
100 h	Funktionen
44 h	GoogleMaps API
36 h	Design/Frontendentwicklung
10 h	Webserver
240 h	**Insgesamt**