



Fachhochschule Köln  
Cologne University of Applied Sciences

FACHHOCHSCHULE KÖLN FAKULTÄT FÜR INFORMATIK UND  
INGENIEURWISSENSCHAFTEN

---

ENTWICKLUNGSPROJEKT INTERAKTIVE SYSTEME

## Meilenstein 4

Campus Gummersbach  
im Studiengang  
Medieninformatik

Betreut von:

Prof. Dr. Kristian Fischer  
Prof. Dr. Gerhard Hartmann  
Robert Gabriel, B. Sc.

ausgearbeitet von:

DERYA ERGUEL  
SINEM KAYA

26. Mai 2015

## Inhaltsverzeichnis

<b>1 Datenstrukturen</b>	<b>2</b>
1.1 XML-Schemata . . . . .	2
1.2 Entity-Relationship-Modellierung . . . . .	7
<b>2 WBA-Modellierung</b>	<b>9</b>
2.1 Ressourcen . . . . .	9
2.1.1 Profil . . . . .	9
2.1.2 Karte . . . . .	9
2.1.3 Nachricht . . . . .	10
2.1.4 Wetter . . . . .	10
2.1.5 Ressourcentabelle . . . . .	11
<b>3 Prototyp UI</b>	<b>12</b>
3.1 Gestaltungslösungen . . . . .	12
3.1.1 Paperbased-Prototyping . . . . .	12
<b>4 Literaturverzeichnis</b>	<b>22</b>
<b>5 Projektplan</b>	<b>23</b>

# 1 Datenstrukturen

In diesem Kapitel wird die Strukturierung der Daten ausführlich besprochen, die notwendig sind, um die technische Umsetzung zu realisieren und das Repräsentieren der Daten gerecht realisieren zu können.

## 1.1 XML-Schemata

Da eine Android-Applikation entwickelt werden soll, ist die Repräsentation der Daten im Zusammenhang mit dem Repräsentationsformat XML die beste Möglichkeit für dieses Projekt, da das Format in der Entwicklung bereits involviert ist und in dem Modul 'Web-basierte Anwendungen 2' praktisch umgesetzt wurde. Aus diesem Grund soll die Strukturierung der Daten anhand von XML-Schemata dargestellt und erläutert werden.

Als Beispiel wird der Abruf der Wetterdaten als XML-Schema präsentiert, da dies am ausgeprägtesten ist. Die restlichen XML-Schemata haben einen sehr ähnlichen Aufbau, weshalb auf diese nicht weiter eingegangen wird.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="15dp"
        android:text="@string/location" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_alignParentTop="true"
        android:layout_alignLeft="@+id/editText2"
        android:layout_alignStart="@+id/editText2" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/country"
        android:layout_marginTop="36dp"
        android:layout_below="@+id/button1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

```

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_alignLeft="@+id/editText3"
    android:layout_alignStart="@+id/editText3">
    <requestFocus />
</EditText>

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="34dp"
    android:text="Temperatur:"
    android:layout_below="@+id/textView2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignLeft="@+id/editText4"
    android:layout_alignStart="@+id/editText4" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Feuchtigkeit:"
    android:layout_below="@+id/editText3"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="38dp" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Feuchtigkeit:"
    android:layout_below="@+id/editText3"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="38dp" />

<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:layout_marginLeft="26dp"
    android:layout_marginStart="26dp"
    android:layout_alignBaseline="@+id/textView4"
    android:layout_alignBottom="@+id/textView4"
    android:layout_toRightOf="@+id/textView4"
    android:layout_toEndOf="@+id/textView4" />
```

```

    <TextView
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Luftdruck:"
        android:layout_alignBaseline="@+id/editText5"
        android:layout_alignBottom="@+id/editText5"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <EditText
        android:id="@+id/editText5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_below="@+id/editText4"
        android:layout_alignLeft="@+id/editText4"
        android:layout_alignStart="@+id/editText4"
        android:layout_marginTop="25dp" />

    <EditText
        android:id="@+id/editText6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_below="@+id/editText5"
        android:layout_alignLeft="@+id/editText5"
        android:layout_alignStart="@+id/editText5" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="open"
        android:text="Wetter"
        android:layout_below="@+id/editText1"
        android:layout_alignLeft="@+id/editText1"
        android:layout_alignStart="@+id/editText1" />

</RelativeLayout>

```

Abbildung 1: XML-Schema 'activity\_wetter.xml'

Um nicht weiter aus zu schweifen, werden nur die relevanten Zeilen erläutert. Die Activity beinhaltet 6 verschiedene Eingabefelder (EditText), 6 TextViews für die Texte, die angezeigt werden müssen und einen Button. Jedes Element erhält eine ID wie z.B. 'android:id="@+id/textView1"' zur Identifikation. Dadurch kann jedes Element eindeutig angesprochen werden. Durch 'android:text="@string/location"' wird der Variable ein Datentyp zugeordnet. In diesem Beispiel wäre es ein String. Dadurch kann der Text, den das TextView beinhalten soll, in der 'string.xml' separat erfasst werden. Durch das Betätigen des Buttons werden die Informationen zum Wetter abgerufen und in den jeweiligen EditText gespeichert und angezeigt. Durch die Zeile 'android:onClick="open"' beim Button wird der OnClickListener aufgerufen, der den Button klickbar macht und durch 'open' wird die Methode 'open' in der Wetter.java aufgerufen, der sich um das Parsen der Daten kümmert.

```

* Created by sinemkaya on 06.05.15.
*/
public class HandleXML {

    private String country = "country";
    private String temperature = "temperature";
    private String humidity = "humidity";
    private String pressure = "pressure";
    private String clouds = "clouds";
    private String urlString = null;
    private XmlPullParserFactory xmlFactoryObject;
    public volatile boolean parsingComplete = true;

    public HandleXML(String url) { this.urlString = url; }
    public String getCountry() { return country; }
    public String getTemperature() { return temperature; }
    public String getHumidity() { return humidity; }
    public String getPressure() { return pressure; }
    public String getClouds() { return clouds; }

    public void parseXMLAndStoreIt(XmlPullParser myParser) {
        int event;
        String text=null;
        try {
            event = myParser.getEventType();
            while (event != XmlPullParser.END_DOCUMENT) {
                String name=myParser.getName();
                switch (event){
                    case XmlPullParser.START_TAG:
                        break;
                    case XmlPullParser.TEXT:
                        text = myParser.getText();
                        break;

                    case XmlPullParser.END_TAG:
                        if(name.equals("country")){
                            country = text;
                        }
                        else if(name.equals("humidity")){
                            humidity = myParser.getAttributeValue(null,"value");
                        }
                        else if(name.equals("pressure")){
                            pressure = myParser.getAttributeValue(null,"value");
                        }
                        else if(name.equals("clouds")){
                            clouds = myParser.getAttributeValue(null, "value");
                        }
                        else if(name.equals("temperature")){
                            temperature = myParser.getAttributeValue(null,"value");
                        }
                        else{
                            }
                        break;
                }
                event = myParser.next();
            }
            parsingComplete = false;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
    public void fetchXML(){
        Thread thread = new Thread(new Runnable(){
            @Override
            public void run() {
                try {
                    URL url = new URL(urlString);
                    HttpURLConnection conn = (HttpURLConnection)
                        url.openConnection();
                    conn.setReadTimeout(10000 /* milliseconds */);
                    conn.setConnectTimeout(15000 /* milliseconds */);
                    conn.setRequestMethod("GET");
                    conn.setDoInput(true);
                    conn.connect();
                    InputStream stream = conn.getInputStream();

                    xmlFactoryObject = XmlPullParserFactory.newInstance();
                    XmlPullParser myparser = xmlFactoryObject.newPullParser();

                    myparser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES
                        , false);
                    myparser.setInput(stream, null);
                    parseXMLAndStoreIt(myparser);
                    stream.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();
    }
}

```

Abbildung .2: Klasse 'HandleXML' für das Parsen der Wetter-Daten

Die XML-Dokumente müssen geparsst werden, um die Daten nutzen zu können. Diese Aufgaben übernimmt die Klasse 'HandleXML'. Die Klasse parst das Dokument mit Hilfe der Schnittstelle 'XMLPullParser'(1), die in die Klasse importiert wird. 'XMLPullParser' stellt notwendige Funktionen zum Parsen zur Verfügung.

Die Methode 'parseXMLAndStoreIt' ermöglicht das Parsen der XML-Daten. Die Wetterdaten, die in der Anwendung angezeigt werden sollen, werden in der Methode verglichen und die Daten werden mittels der GET-Methode in der Methode 'fetch-XML' abgerufen.

XML-Schema	Klasse	ID's	Bezeichnung
activity wetter	TextView	textview1 textview2 textview3 textview4 textview5 textview6	Standort Land Temperatur Feuchtigkeit Luftdruck Wolken
activity wetter	EditText	edittext1 edittext2 edittext3 edittext4 edittext5 edittext6	Standort Land Temperatur Feuchtigkeit Luftdruck Wolken
activity wetter	Button	button1	Wetter

Tabelle .1: XML-Schema 'activity wetter.xml' - einfache Darstellung der Datenstruktur

Die Tabelle 1 soll eine einfache Strukturierung des XML-Schemas darstellen. Es soll aufweisen, welche Klassen innerhalb des XML-Schemas benutzt werden, welche ID's vorhanden sind. Durch die zum ID zugehörige Bezeichnung sollte klar werden, um welche Daten es sich bei der Repräsentation handelt.

## 1.2 Entity-Relationship-Modellierung

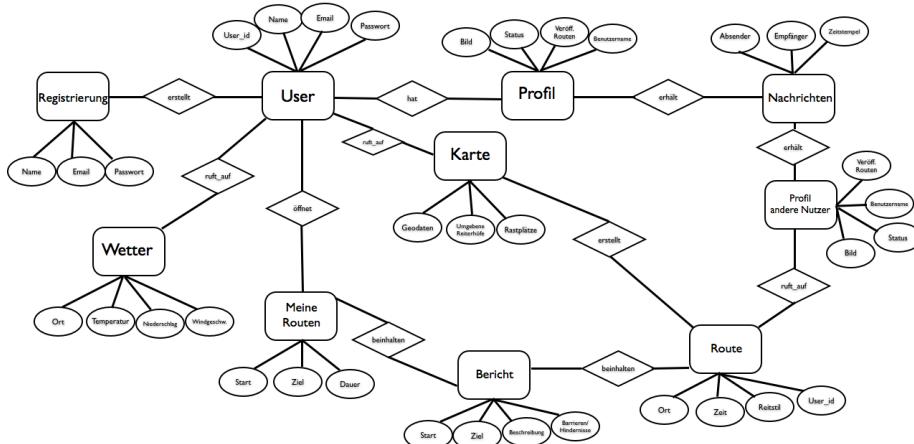


Abbildung .3: ER-Modellierung zur Datenstruktur

Das-Entity-Relationship-Modell dient zur Modellierung von Daten. Dabei werden technische Aspekte nicht berücksichtigt. Das ER-Diagramm besteht hauptsächlich aus den Entitäten, den Attributen und den Beziehungstypen. Bei Attributen handelt es sich um die Eigenschaften von Entitäten. Die Entitäten sind durch Beziehungstypen miteinander verbunden. (2) Die gesamte Datenmodellierung der Applikation soll durch ein Entity-Relationship-Diagramm verdeutlicht werden. Die Entität 'User' hat beispielsweise die Attribute 'User id', 'Name', 'E-Mail', 'Passwort' und kann die Entität 'Karte' aufrufen, die die Attribute 'Geodaten', 'Umgebene Reiterhöfe' und 'Rastplätze' enthält. Durch die Modellierung soll dargestellt werden, welche Entitäten vorhanden sind und wie diese in Zusammenhang stehen.

## 2 WBA-Modellierung

### 2.1 Ressourcen

#### 2.1.1 Profil

Die Profil ressource enthält alle Profil angaben eines Users. Dabei kann jeder Nutzer das eigene Profil erstellen, ändern, erweitern und löschen. Das Profil beinhaltet Texte, ein Profilbild und veröffentlichte Routen. Über das Profil kann jeder User an andere User Nachrichten schreiben und erhalten. Jedes mal wenn der Nutzer eine Route erstellt erhält dieser anliegende Routen von anderen Nutzern. Somit kann er über die Verlinkung auf der Karte die Profile aufrufen und in Kontakt treten um weiter Informationen zu erhalten oder Erfahrungen auszutauschen.

**Methoden:** GET,PUT,POST,DELETE

**Attribute:**

- Name
- Status
- Profilbild
- Veröffentlichte Routen

**Einsatz:** Erstellung eines Profil,ändern von Angaben, löschen des Accounts

**URI:** UserID/ProfilID

#### 2.1.2 Karte

Die Ressource Karte erlaubt das anzeigen und aufrufen von Standorten auf einer Karte. Der Nutzer hat dabei die Möglichkeit über diese Anzeige Routen zu erstellen. Auf der Karte erkennt der Nutzer die Umgebung und kann sich anhand dieser orientieren. die auf der Karte angezeigten Icons sind beinhalten Informationen und können übers Display an getippten werden. Sie erhalten damit Texte oder die Nutzer können sich an den Punkt navigier lassen.

**Methoden:** GET,POST

**Attribute:**

- Geodaten
- Umgebungsinformationen
- Straßennamen

- Wege
- ?????

**Einsatz:** Anzeige der aktuellen Position auf einer Karte,navigieren zu einem bestimmten Punkt, abrufen von Informationen zur Umgebung, Umgebung auf einer Karte anzeigen lassen

**URI:**UserID/KarteId

#### 2.1.3 Nachricht

Die Nachricht Ressource repräsentiert die einzelne Mitteilung die ein Nutzer von einem Nutzer als Text bekommt. Möchte eine Nutzer einem anderen Nutzer etwas mitteilen kann er die übers Profil des jeweiligen. Eine Nachricht kann abgerufen, erstellt und gelöscht werden.

**Methoden:** GET,POST,DELETE

**Attribute:**

- Sender
- Absender
- Inhalt
- Uhrzeit

**Einsatz:** Nachricht lesen,schreiben,löschen

**URI:**nachrichtenlisteID/nachrichtID

#### 2.1.4 Wetter

Die Ressource Wetter beinhaltet den dienst der aktuelle Wetterlage angibt. Dabei werden diese über Query-Parameter gefiltert.Der User hat die Möglichkeit zu jedem Ort sich die Wetterdaten anzeigen zulassen. Weiterhin kann der Nutzer Vorhersage in Stunden anzeigen lassen.

**Methoden:** GET

**Einsatz:** Abrufen von Wetterdaten, Filterung nach Temperatur,Niederschlag und Windgeschwindigkeit,Land,Bewölkung

**Query-Parameter:** Temperatur=[Celsius], Niederschlag=[prozent],Windgeschwindigkeit=[km/h], Land=[Land],Bewölkung=[xy]

**URI:**UserID/WetterID

Ressource	Methode	Semantik
User/ID	Get	"Karte anzeigen"
User/ID	Post	"Route Erstellen"
User/ID	Put	"Route ändern"
User/ID	Delete	"Route löschen"
Karte/ID	Put	"Karte erzeugen"
Profil/ID	Get	"Profil anzeigen"
Profil/ID	Post	"Profil erstellen"
Profil/ID	Put	"Profil ändern"
Profil/ID	Delete	"Profil löschen"
Wetter/ID	Get	"Wetter abrufen"
Nachrichten/ID	Get	"Nachricht anzeigen"
Nachricht/ID	Post	"Nachricht Schreiben"
Nachrichtenliste/ID	Get	"Nachrichtenliste anzeigen"
Nachrichten/ID	Delete	"Nachrichten löschen"
Route/ID	Get	"Route anzeigen"
Route/ID	Put	"Route ändern"
Route/ID	Delete	"Route löschen"
Routenliste/ID	Get	"Routenliste anzeigen"

Tabelle .2: Ressourcen Übersicht

### 2.1.5 Ressourcentabelle

In der folgenden Tabelle werden die ermittelten Ressourcen mit der angewendeten Methode und festgelegten URI dargestellt.

## 3 Prototyp UI

### 3.1 Gestaltungslösungen

In Meilenstein 2 wurde festgelegt, dass das Paperbased-Prototyping-Verfahren zum Ermitteln der Gestaltungslösung verwendet werden soll. Die Gründe beruhen auf der Vielfältigkeit des Verfahrens. Es erfordert keinerlei Kosten und ist für allgemeine Änderungen und für Änderungen des Interfaces sehr geeignet, da es sich nur um Zeichnungen auf Papier handelt. Dadurch sind mögliche Iterationen schneller und einfacher umsetzbar.

#### 3.1.1 Paperbased-Prototyping

Im Folgenden werden die erstellten papierbasierten Prototypen in Reihenfolge dargestellt und erläutert:



Abbildung .4: Paperbased Prototyp - Login

Ist der Benutzer bereits registriert, kann er durch die Eingabe von Benutzernamen und Passwort in das System gelangen und sich somit einloggen. Ist der Benutzer nicht registriert, so muss er zunächst eine Registrierung vollziehen.

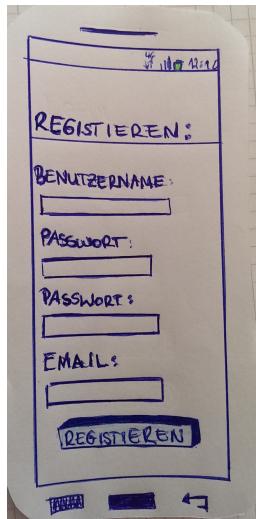


Abbildung .5: Paperbased Prototyp - Registrierung

Die Registrierung erfolgt durch die Eingabe von Benutzername, Passwort und E-Mail.

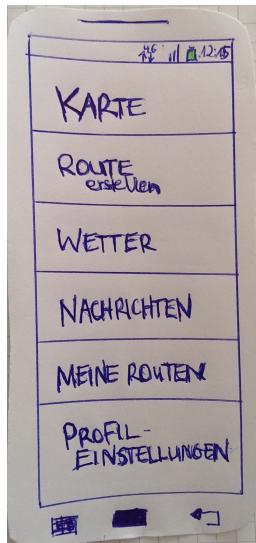


Abbildung .6: Paperbased Prototyp - Hauptmenü

Das Hauptmenü besteht aus 6 Buttons: Karte, Route erstellen, Wetter, Nachrichten, Meine Routen, Profil-Einstellungen

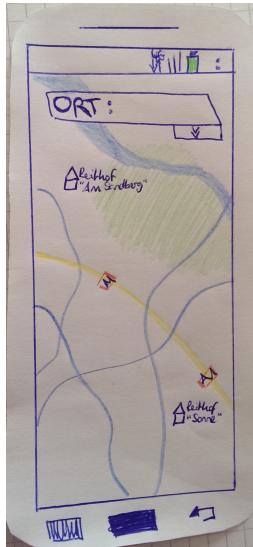


Abbildung .7: Paperbased Prototyp - Karte

Sobald der Benutzer den Button 'Karte' tätigt, öffnet sich die Karte mit den Umgebungsinformationen und dem aktuellen Standort. Zusätzlich kann er andere Orte erforschen, indem er den erwünschten Ort in das Eingabefeld eingibt.



Abbildung .8: Paperbased Prototyp - Route erstellen 1

Tätigst der Benutzer den Button 'Route erstellen', gelangt er ebenfalls auf die Karte. Dort kann er die erwünschte Reitzeit angeben und den Reitstil.

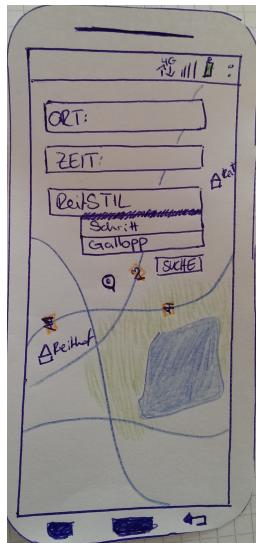


Abbildung .9: Paperbased Prototyp - Route erstellen 2

Durch Tätigkeiten des Reitstils öffnen sich Optionen, die sich in 'Sprint' und 'Gallopp' unterscheiden. Jeder Reitstil enthält konstante Mittel-Geschwindigkeiten (3), die als Kriterien zur Suche der Routen genutzt werden. Durch den Button 'Suche' wird nach der Besten Alternativ-Route gesucht.

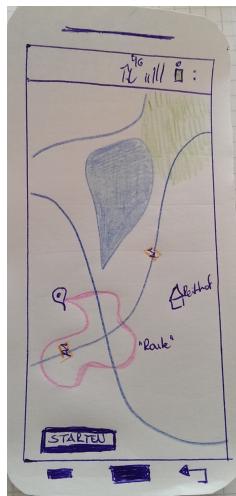


Abbildung .10: Paperbased Prototyp - Route erstellen 3

Die Route wird gefunden und angezeigt. Durch den Button 'Starten' kann die Route gestartet werden.

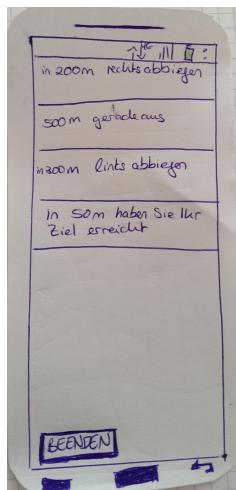


Abbildung .11: Paperbased Prototyp - Route erstellen 4

Die Route wird schriftlich navigiert und gleichzeitig auditiv wiedergegeben. Durch den Button 'Beenden' wird die Route beendet.

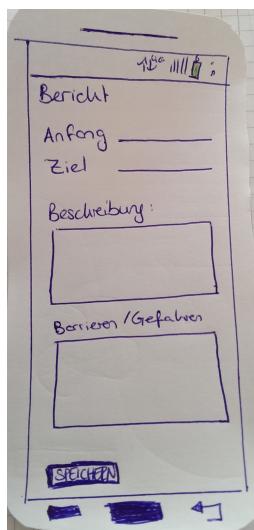


Abbildung .12: Paperbased Prototyp - Route erstellen 5

Nach Beenden der Route erscheint ein Fenster zum Erfassen eines Berichts. Dies ist für die Benutzer optional.



Abbildung .13: Paperbased Prototyp - Wetter 1

Tätigt der Benutzer im Hauptmenü den Button 'Wetter', so erscheint dieses Fenster. Durch Eingabe des Ortes und anschließend durch das Tätigen des 'Wetter'-Buttons werden die Werte für den erwünschten Ort angezeigt.

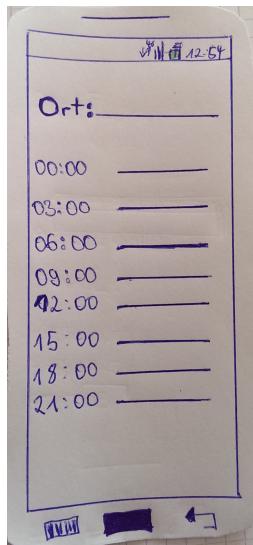


Abbildung .14: Paperbased Prototyp - Wetter 2

Hier wird die Temperatur und der Niederschlag im 3-Stunden Takt angezeigt.

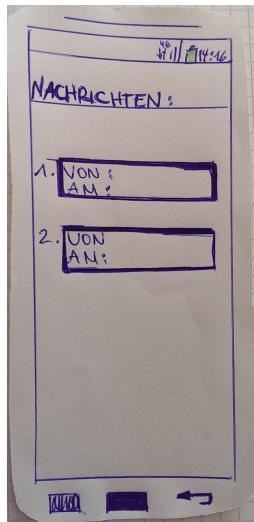


Abbildung .15: Paperbased Prototyp - Nachrichten 1

Tätigt der Benutzer den 'Nachrichten'-Button, so erscheint das Nachrichten-Fenster. Hier kann der Benutzer auf seine Nachrichten zugreifen und die neuesten Nachrichten lesen.

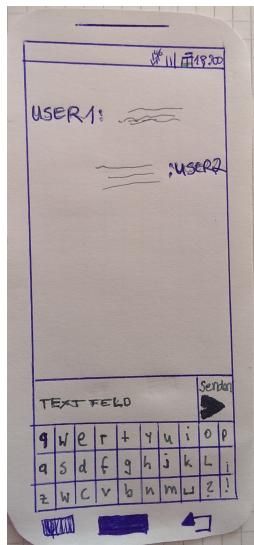


Abbildung .16: Paperbased Prototyp - Nachrichten 2

Abbildung 16 stellt die Kommunikation durch Nachrichtenaustausch zwischen zwei unterschiedlichen Benutzern dar.



Abbildung .17: Paperbased Prototyp - Meine Routen

Durch den 'Meine Routen'-Button im Hauptmenü, werden dem Benutzer die bereits erstellten Routen angezeigt.

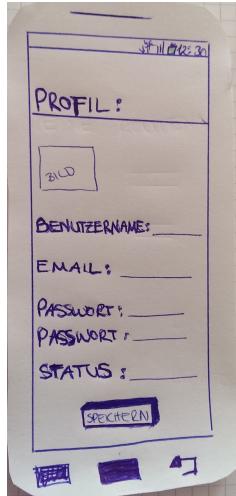


Abbildung .18: Paperbased Prototyp - Profileinstellungen Benutzer Ansicht

Durch das Tätigen des 'Profil-Einstellungen'- Buttons gelangt der Benutzer in die Einstellungen und kann dort sein Profil-Foto, Benutzername, Passwort und sein Status ändern und speichern.

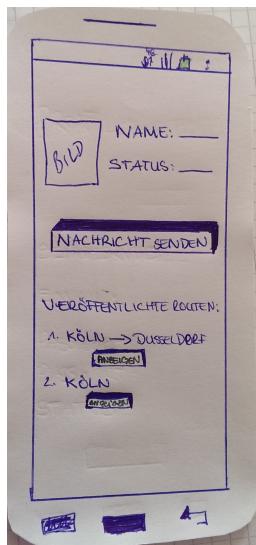


Abbildung .19: Paperbased Prototyp - Profilansicht von anderen Benutzern

Dies ist die Sicht eines Benutzers, der das Profil eines anderen Benutzers aufruft. Durch den Button 'Nachricht senden' kann der Benutzer eine Nachricht senden (siehe Abbildung 16). Zusätzlich sieht er erstellte Routen des Benutzers, die er auf der Karte abrufen kann.

## Abbildungsverzeichnis

.1	XML-Schema 'activity wetter.xml' . . . . .	4
.2	Klasse 'HandleXML' für das Parsen der Wetter-Daten . . . . .	6
.3	ER-Modellierung zur Datenstruktur . . . . .	7
.4	Paperbased Prototyp - Login . . . . .	12
.5	Paperbased Prototyp - Registrierung . . . . .	13
.6	Paperbased Prototyp - Hauptmenü . . . . .	13
.7	Paperbased Prototyp - Karte . . . . .	14
.8	Paperbased Prototyp - Route erstellen 1 . . . . .	14
.9	Paperbased Prototyp - Route erstellen 2 . . . . .	15
.10	Paperbased Prototyp - Route erstellen 3 . . . . .	15
.11	Paperbased Prototyp - Route erstellen 4 . . . . .	16
.12	Paperbased Prototyp - Route erstellen 5 . . . . .	16
.13	Paperbased Prototyp - Wetter 1 . . . . .	17
.14	Paperbased Prototyp - Wetter 2 . . . . .	17
.15	Paperbased Prototyp - Nachrichten 1 . . . . .	18
.16	Paperbased Prototyp - Nachrichten 2 . . . . .	18
.17	Paperbased Prototyp - Meine Routen . . . . .	19
.18	Paperbased Prototyp - Profileinstellungen Benutzer Ansicht . . .	19
.19	Paperbased Prototyp - Profilansicht von anderen Benutzern . . .	20
.20	Projektplan . . . . .	23

## Tabellenverzeichnis

.1	XML-Schema 'activity wetter.xml' - einfache Darstellung der Da- tenstruktur . . . . .	7
.2	Ressourcen Übersicht . . . . .	11

## 4 Literaturverzeichnis

- [1] <http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html> - Sichtungsdatum: 21.05.2015
- [2] <http://de.wikipedia.org/wiki/Entity-Relationship-Modell> - Sichtungsdatum: 25.05.2015
- [3] [http://equivetinfo.de/html/eckdaten\\_pferd.html](http://equivetinfo.de/html/eckdaten_pferd.html) - Sichtungsdatum: 25.05.2015

## 5 Projektplan



Abbildung .20: Projektplan