

Webbasierte Anwendungen 2

Netzbasierte Anwendungen

Dokumentation für Phase 2

Verfasst von Lutz Schönfelder und Pascal Büchle

Inhaltsverzeichnis

1 Vorbereitung und Ideenfindung.....	S.4
2 Die Idee von RentMyStuff.....	S.4
2.1 Kommunikationsabläufe und Interaktion	
3 Szenarien.....	S.5
3.1 Anmelden	
3.2 Freunde	
3.3 Objekte	
3.4 Verleih-System	
4 Projektbezogenes XML Schema/Schemata.....	S.6
4.1 XML Schema Benutzer	
4.2 XML Schema Objekte	
5 Ressourcen und die Semantik der HTTP-Operationen.....	S.9
5.1 HTTP Operationen	
6 RESTful Webservice	S.9
6.1 Generieren der Klassen	
6.2 Marshalling/Unmarshalling	
6.3 Implementierung von GET, POST, DELETE	
6.3.1 Get	
6.3.2 POST	
6.3.3 DELETE	
6.4 Query und PathParams	
6.4.1 QueryParams	
6.4.2 PathParams	
7 XMPP Server und Client	S.11
7.1 Topics und Publisher Subscriber	
7.2 XMPP Server	

7.3 XMPP Client

8 Client EntwicklungS.12

8.1 Interface

1 Vorbereitung und Ideenfindung

Wir haben uns für diese App entschieden, da unsere vorigen Ideen leider nicht genügend Potenzial für einen synchronen und asynchronen Ablauf hatten.

Als erstes entstand die Idee einer Gaming-App. Diese sollte dem User bei Kaufentscheidungen helfen, indem er in dieser App Bewertungen und Beschreibungen der Games lesen kann. Es wäre auch möglich mit anderen Usern privat zu Kommunizieren mit Hilfe eines Chats oder ähnlichem. Dies wurde aber aufgrund von mangelnder Begeisterung der Betreuer und aufgrund der Tatsache dass diese App wenig Sinn mache, da es im Internet genügend Gaming-Seiten gibt wo man sich eben solche Informationen holen kann. Somit wurde diese Idee verworfen.

Unsere zweite Idee war die Entwicklung einer Blitzer App, welche einem Blitzer in der Nähe meldet die vorher von anderen Usern in der App eingetragen wurden. Das besondere an dieser App sollte sein, dass die Blitzer nicht nur mit Standorten gespeichert werden sondern optional auch mit Bildern und Beschreibungen. Diese Idee wurde jedoch aufgrund der mangelnden Kommunikation und der Tatsache das es schon viele solcher Apps gibt nicht weiter verfolgt.

Als keine andere und noch nicht bereits vorhandene Idee einer App gefunden wurde, bekamen wir von unseren Betreuern den Vorschlag eine App zu entwickeln mit der es möglich ist Gegenstände zu verleihen. Diesen Vorschlag wollten wir nutzen und entwickelten dazu Ideen und Erweiterungen, welche das Nutzen dieser App sinnvoll macht.

So kamen wir auf eine App, welche es ermöglicht den Freunden in seinem Freundeskreis eigene Objekte zum Verleih anzubieten aber auch selber welche zu leihen.

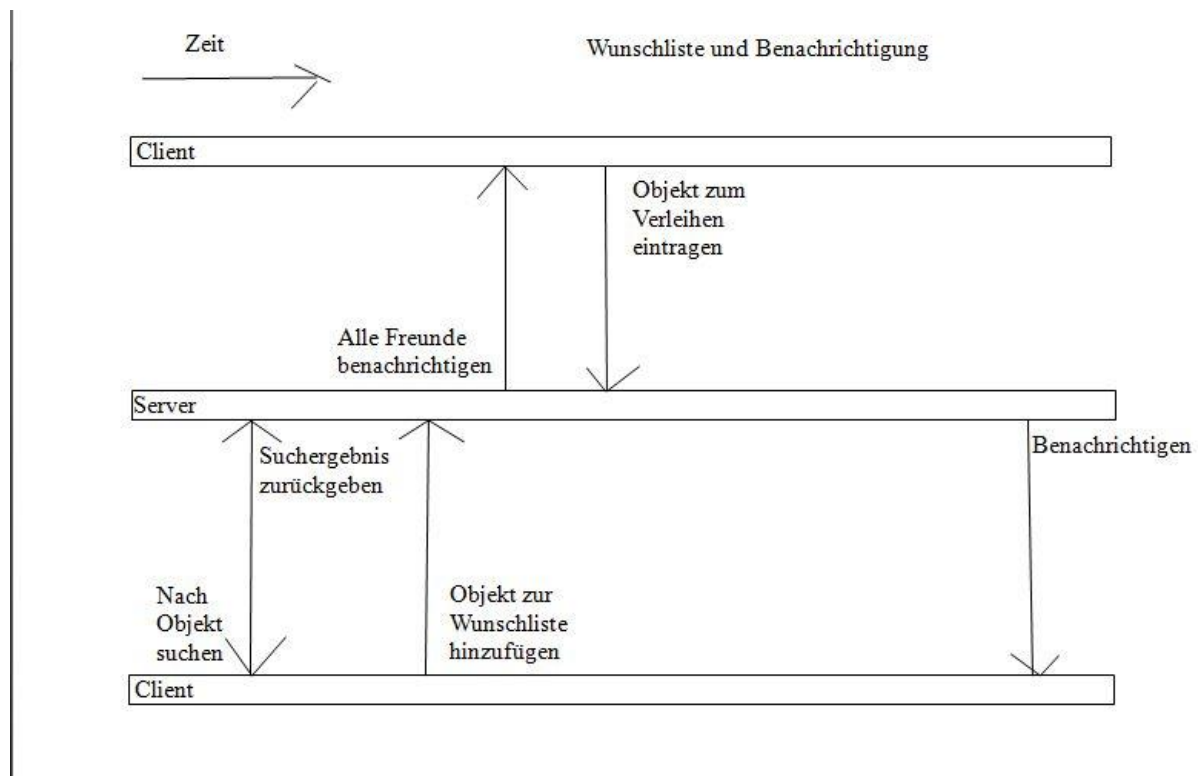
2 Die Idee von RentMyStuff

Die Idee von RentMyStuff ist es Gegenstände zu leihen und verleihen. Dies ist nur innerhalb der eigenen Freundesliste möglich, um zu gewährleisten, dass man diese Personen auch persönlich kennt und somit ein gewisses Vertrauen vorhanden ist. Ist man erst einmal angemeldet ist es einfach, schnell und effektiv eigene Gegenstände zu verleihen oder Objekte von Anderen auszuleihen.

2.1 Kommunikationsabläufe und Interaktion

Bevor richtig mit dem Projekt begonnen werden kann müssen erst einmal die groben Kommunikationsabläufe und Interaktionen Skizziert und festgehalten werden. Hier sehen sie den Kommunikationsablauf anhand des Beispiels von Wunschliste und Benachrichtigung.

Die einfachen Pfeile stellen einen asynchronen Ablauf da und die beidseitigen Pfeile den synchronen.



3 Szenarien

3.1 Anmelden

Um die App nutzen zu können ist es nötig, dass der User sich bei dieser anmeldet.

Bei der Anmeldung muss der Nutzer seinen Namen und Vornamen angeben, um es anderen zu ermöglichen ihn zu finden und als Freund zu adden. Zu diesem Zweck wird auch ein Profilbild verlangt. Desweiteren wird eine E-Mail-Adresse benötigt um den User kontaktieren zu können. Natürlich muss auch ein Passwort erstellt werden, um den Zugang von Dritten zu verhindern. Außerdem wird noch der Wohnort benötigt, welcher bei Interaktionen als Verleih-Ort (falls nicht anders angegeben) dient. Ist dies alles abgeschlossen kann der User direkt eine Wunschliste an Gegenständen erstellen oder zu einem beliebigen Zeitpunkt seiner Wahl.

3.2 Freunde

Freunde spielen in dieser App eine wichtige Rolle. Es ist Usern nicht möglich ihre Gegenstände an "fremde" Leute zu verleihen, was bedeutet, dass die Person an die man etwas leiht oder von der man etwas geliehen bekommt, in der Freundesliste vorhanden sein muss. Es wurde sich dafür entschieden, da somit ein gewisses Maß an Vertrauen gegeben ist und die User wissen worauf sie sich einlassen. Würde man die Gegenstände an jemanden verleihen den man nicht kennt oder nicht in der Freundesliste hat besteht die Gefahr dass dieser seine "Anonymität" ausnutzt und der besagten

Person schadet. Freunde kann man auf verschiedenen Weisen zu seiner Liste hinzufügen. Entweder durch die direkte Suche oder über das Profil von jemandem.

3.3 Objekte

Die Objekte die zum Verleih freigegeben werden sollen müssen mit folgenden Informationen angegeben werden: Das zu verleihende Objekt benötigt einen Namen. Auch ein Bild kann optional angegeben werden, damit die Interessenten besser beurteilen können, um was es sich handelt und in welchem Zustand es sich befindet. Desweiteren ist eine Beschreibung gefordert, welche kurz und knapp Informationen zu dem Gegenstand wiedergibt. Um das Finden des Objektes zu erleichtern wird es in eine Kategorie eingeteilt und mit Tags versehen. Optional kann auch ein gewisser Geldbetrag als Pfand gefordert werden. Zuletzt muss der Zeitraum angegeben werden, innerhalb welchen man das Objekt leihen kann.

3.4 Verleih-System

Das Verleih-System dieser App soll es den Usern ermöglichen gesuchte Objekte schnell und persönlich zu leihen. Es gibt mehrere Möglichkeiten ein Objekt zu finden. Am einfachsten ist es eine Wunschliste an Objekten zu erstellen, welche aus all den Gegenständen besteht an denen man Interesse hat. Ist die Liste einmal erstellt ist es weiterhin möglich sie um Gegenstände zu erweitern oder zu verkürzen. Diese Liste wird von der App immer mit den zum Verleih stehenden Gegenständen von Freunden verglichen. Wird ein Objekt gefunden das gesucht wird, wird man per Push-Nachricht darüber informiert. Nun kann der Nutzer den Verleiher kontaktieren und weitere Details über die Interaktion abklären.

Dem Verleiher steht es frei auf eine Anfrage ein zu gehen oder nicht. Dies teilt er dem Interessenten mit einer Nachricht mit.

4 Projektbezogenes XML Schema/Schemata

Ein Grundlegender Schritt zur Entwicklung eines Clients war es die XML Schemata zu entwickeln, auf dessen Basis alles andere aufbaut. Zu diesem Zweck wurden zwei Schemata entwickelt: Eins für die Benutzer und ein Zweites für die Objekte.

4.1 XML Schema Benutzer

Im folgenden Codeauszug ist das Schema der Benutzer zu sehen. Es können zwischen 0 und unendlich vielen Benutzern angelegt werden.

Der Wohnort enthält Postleitzahl, Stadt, Straße und Hausnummer.

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="benutzerliste">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="benutzer" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="Name" type="xs:string" />
              <xs:element name="Passwort" type="xs:string" />
              <xs:element name="Email" type="xs:string" />

              <xs:element name="Wohnort">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Stadt" type="xs:string" />
                    <xs:element name="PLZ" type="xs:string" />
                    <xs:element name="Strasse" type="xs:string" />
                    <xs:element name="HausNr" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>

              <xs:element name="Bild" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Als nächstes wird die Wunschliste dargestellt. Hier wurde minOccurs="0" gesetzt da diese Wunschliste nicht zwingend vorhanden sein muss. Das maxOccurs wird ="unbound" gesetzt, da es eine unbegrenzte anzahl an Wunschobjekten gibt.

```

    <xs:element name="Wunschliste">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Wunsch" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

4.2 XML Schema Objekte

In diesem Code Ausschnitt werden die Objekte samt ihrer Informationen definiert. Das minOccurs der Objekte wurde ="0" gesetzt, da es möglich ist kein Objekt zu haben. In folge dessen ist maxOccurs="unbounded" gesetzt denn es ist ja möglich eine unbegrenzte Anzahl an Objekten zu erstellen.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="objekte">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="objekt" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="Name" type="xs:string" />
              <xs:element name="Verleiher" type="xs:string" />

              <xs:element name="Tags">
                <xs:complexType>
                  <xs:sequence>
```

Im folgenden Ausschnitt sollte minOccurs eigentlich 0 sein, da es Sinn macht, mindestens einen Tag zu haben.

```
        <xs:element name="Tag" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Kategorie" type="xs:string" />
  <xs:element name="Beschreibung" type="xs:string" />
  <xs:element name="ZeitraumStart" type="xs:string" />
  <xs:element name="ZeitraumEnde" type="xs:string" />
  <xs:element name="Pfand" type="xs:decimal" />

  <xs:element name="Bilder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Bild" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```


5 Ressourcen und die Semantik der HTTP-Operationen

Dieser Schritt dient der Auseinandersetzung im Kontext mit RESTful Webservices und bildet die Grundlage für den nächsten Schritt in dem ein RESTful Webservice in Java erstellt wird.

HTTP Operationen

	GET	POST	PUT	DELETE
/Verleihobjekte	Gibt eine Liste der Verleihobjekte zurück	Legt ein neues Verleihobjekt an	Aktualisiert die Liste	unused
/Verleihobjekte/{id}	Gibt die Eigenschaften eines bestimmten Objektes zurück	unused	Aktualisiert die Eigenschaften	löscht das Objekt aus der Liste
/Benutzer	Gibt eine Liste der Benutzer zurück	Legt einen neuen Benutzer an	Aktualisiert die Liste	unused
/Benutzer/{id}	Gibt die Eigenschaften eines bestimmten Benutzers zurück	nused	Aktualisiert die Eigenschaften	löscht den Benutzer aus der Liste

6 Ressourcen und die Semantik der HTTP Operationen

Dieser Schritt dient der Auseinandersetzung im Kontext mit RESTful Webservices und bildet die Grundlage für den nächsten Schritt in dem ein RESTful Webservice in Java erstellt wird.

6.1 Generieren der Klassen

Um diesen Schritt überhaupt beginnen zu können, wurden aus den XML Schemata, mit Hilfe von JAXB, unsere Klassen generiert. Diese Klassen beinhalten Funktionen wie get und set um an Inhalte zu gelangen oder Inhalte zu erzeugen.

6.2 Marshalling/Unmarshalling

Zum un-/marshalling wurde eine eigene Klasse geschrieben, damit in anderen Klassen nur noch die Methoden aufgerufen werden müssen.

6.3 Implementierung von GET, POST, DELETE

6.3.1 GET

Code Auszug Get. Besonderheit ist, dass hier die Nachfrage nach allen oder einem bestimmten Benutzer zusammengefasst wurde: Wenn der Name==null, also leer ist, werden alle Benutzer angezeigt, sonst wird der Benutzer mit dem gewünschten Namen zurückgegeben.

```

@GET @Path("benutzer") @Produces("text/xml")
public String benutzerXml(@QueryParam("name") String name){

    XMLMarshaller xmlm = new XMLMarshaller();
    xmlm.readBenutzer();
    java.io.StringWriter sw = new StringWriter();
    try {
        if (name != null)
            xmlm.m.marshal(xmlm.getBenutzer(name), sw);
        if (name == null)
            xmlm.m.marshal(xmlm.benutzerliste, sw);
    } catch (JAXBException e) {
        e.printStackTrace();
    }

    return sw.toString();
}

```

6.3.2 POST

Code Auszug POST. Hier werden alle Benutzereigenschaften übergeben und der Benutzerliste hinzugefügt. Auf die Angabe der Wunschliste wurde verzichtet, da eine variable Anzahl an Wünschen möglich ist, darum werden diese erst später hinzugefügt.

```

@POST @Path("benutzer")
public void postBenutzer(@QueryParam("name") String name,
    @QueryParam("password") String password,
    @QueryParam("email") String email,
    @QueryParam("stadt") String stadt,
    @QueryParam("plz") String plz,
    @QueryParam("strasse") String strasse,
    @QueryParam("hausnr") String hausnr,
    @QueryParam("bild") String bild){
    XMLMarshaller xmlm = new XMLMarshaller();
    xmlm.readBenutzer();

    Benutzer b=new Benutzer();
    b.setName(name);
    b.setPassword(password);
    b.setEmail(email);

    Benutzer.Wohnort w=new Benutzer.Wohnort();
    w.setStadt(stadt);
    w.setPLZ(plz);
    w.setStrasse(strasse);
    w.setHausNr(hausnr);

    b.setWohnort(w);
    b.setBild(bild);

    xmlm.benutzerliste.getBenutzer().add(b);
    xmlm.writeBenutzer();
}

```

6.3.3 DELETE

Code Auszug DELETE. Die Liste wird geholt und der Benutzer mit dem Namen gelöscht. Danach wird die Liste aktualisiert.

```

@DELETE @Path("benutzer/{name}")
public void deleteBenutzer(@PathParam("name") String name){
    XMLMarshaller xmlm = new XMLMarshaller();
    xmlm.readBenutzer();
    xmlm.benutzerliste.getBenutzer().remove(xmlm.getBenutzer(name));
    xmlm.writeBenutzer();
}

```

6.4 Query und PathParams

Im folgenden werden die Begriffe Query-/PathParams erklärt und Beispiele für die einzelnen Params.

QueryParams

QueryParams werden um Bereich der Query-Strings mitgereicht und dienen als Filter.

```
public String objekteXml(@QueryParam("name") String name){
```

PathParams

PathParams fügen Identifikationsnummern der Ressource an den Pfad (in diesem Fall den Namen des Benutzers)

```
public void deleteBenutzer(@PathParam("name") String name){
```

7 XMPP Server und Client

Das nächste Kapitel wendet sich der Entwicklung des XMPP-Client und Servers zu.

7.1 Topics und Publisher Subscriber

Als erstes wird festgelegt, welche Topics benötigt werden und wer jeweils Publisher und Subscriber ist. Desweiteren wird angegeben, welche Daten übertragen werden sollen.

Topic	Subscriber	Publisher	Daten	Beschreibung
Verleihobjekt	Benutzer der das Objekt verleihen will	Benutzer der das Objekt leihen will	Objekt.Name	Sobald ein Benutzer ein Objekt zum Verleih anbietet, wird ein neues Topic erstellt und der Benutzer automatisch zum Subscriber. Wenn jemand das Objekt leiht, wird der Benutzer benachrichtigt
Verleihvorgang	Benutzer der das Objekt leihen will	Benutzer der das Objekt verleihen will	Objekt.Name	Sobald man eine Anfrage zur Ausleihung eines Objektes gestellt hat, wird ein neues Topic erstellt und der Benutzer automatisch zum Subscriber. Wenn der Verleiher den Verleih bestätigt oder ablehnt, wird der Leihverleiher benachrichtigt
Freundesanfrage	Benutzer	Benutzer der eine Freundschaftsanfrage schickt	Benutzer.Name	Für jeden Benutzer gibt es ein Topic für seine Freundschaftsanfragen, dessen Subscriber er selber ist. Wenn ein anderer Benutzer eine Freundschaftsanfrage stellt, wird der Benutzer benachrichtigt

Freundesanfragevorgang	Benutzer der eine Freundschaftsanfrage geschickt hat	Benutzer der die Freundschaftsanfrage bestätigt	Benutzer.Name	Sobald ein Benutzer eine Freundschaftsanfrage schickt, wird ein neues Topic erstellt und der Benutzer automatisch zum Subscriber. Wenn der andere Benutzer die Freundschaftsanfrage bestätigt wird der Benutzer benachrichtigt.
Wunschliste	Alle Freunde des Benutzers	Benutzer der etwas zu seiner Wunschliste hinzufügt	Objekt.Name	Für jeden Benutzer gibt es ein Topic für seine Wunschliste, dessen Subscriber alle seine Freunde sind. Sobald der Benutzer etwas zu seiner Wunschliste hinzufügt werden alle Freunde benachrichtigt.
Wunschliste Treffer	Benutzer der etwas auf seiner Wunschliste hat	Benutzer der ein Objekt der Wunschliste zum Verleih anbietet	Objekt.Name	Der Benutzer ist Subscriber seiner eigenen Wunschliste, wenn jemand ein Objekt zum Verleih anbietet, welches auf der Wunschliste ist, wird der Benutzer benachrichtigt.

7.2 XMPP Server

Als nächstes wird ein XMPP Server mit Hilfe von Openfire eingerichtet. Dazu wird die Anwendung gestartet und ein Admin-Profil angelegt. Ist der Server nun erstellt kann man mehrere Benutzer hinzufügen und ihnen Rechte zuweisen. Ob der Server auch wirklich funktionstüchtig ist wurde mit einem kleinen Messenger Programms namens Pidgin überprüft.

7.3 XMPP Client

Der erstellte XMPP Client soll es ermöglichen Leafs zu abonnieren, Nachrichten zu empfangen und zu veröffentlichen. Außerdem sollen Nutzdaten übertragen werden und letztendlich sollen die Eigenschaften der Leafs angezeigt werden. Dazu wurde ein Testuser angelegt, welcher ein Topic erstellt, dieses abonniert und schließlich Daten veröffentlicht. Die Anwendung kann in der Klasse `smack.JabberSmackAPI` getestet werden.

8 Client Entwicklung

Abschließend wird ein Interface mit Hilfe von Java Swing erstellt. Mit Hilfe dieses Interfaces soll es möglich sein die Funktionen der App zu testen. Bei dem hier entstandenen Interface handelt es sich nicht um eine finale Version wie sie auf den Markt kommen würde. Es handelt sich hierbei nur um ein einfaches Interface was nur zu Testzwecken entwickelt wurde.

8.1 Interface

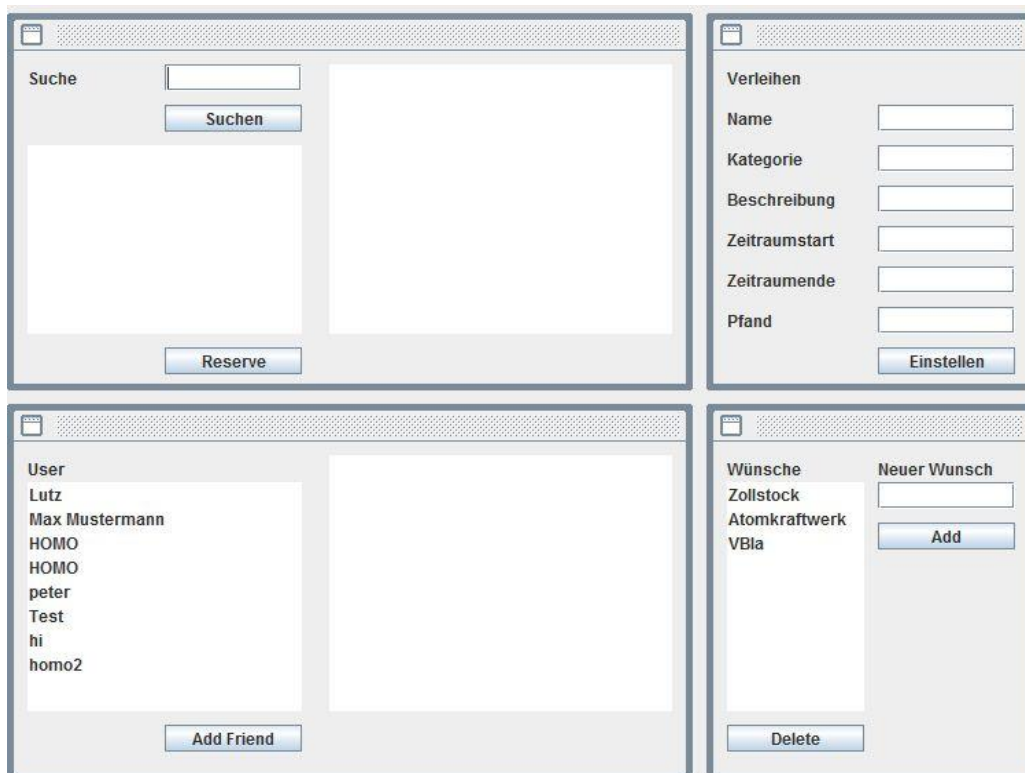
Beim Start des Programms wird der Benutzer aufgefordert, seinen Benutzernamen und Passwort einzugeben. Alternativ kann ein neuer Account erstellt werden. Sobald der Benutzer eingeloggt ist wird er zur Hauptanwendung weitergeleitet.



Name:

Password:

Die Hauptanwendung besteht aus mehreren Unterfenstern in welchen die verschiedenen Funktionen getestet werden können:



The image shows four sub-windows of the application:

- Suchen:** A search window with a text input field, a "Suchen" button, a large empty list area, and a "Reserve" button at the bottom.
- Verleihen:** A window for borrowing items with fields for Name, Kategorie, Beschreibung, Zeitraumstart, Zeitraumende, and Pfand. It includes an "Einstellen" button at the bottom.
- User:** A window showing a list of users: Lutz, Max Mustermann, HOMO, HOMO, peter, Test, hi, homo2. It has an "Add Friend" button at the bottom.
- Wünsche:** A window for wishes with a list of items: Zollstock, Atomkraftwerk, VBla. It includes a "Neuer Wunsch" section with an input field and an "Add" button, and a "Delete" button at the bottom.

Links oben ist das Suchfenster zu sehen. Hier kann der gewünschte Name eingegeben und mit einem Klick auf "Suchen" alle Objekte mit diesem Namen angezeigt werden. Wenn man nun ein bestimmtes Objekt auswählt, werden weitere Informationen angezeigt und man kann das Objekt reservieren.

Rechts oben befindet sich das Ausleihfenster in dem man die Daten des Objektes eingeben und das Objekt schließlich zum Verleih freigeben kann.

Links unten befindet sich eine Auflistung aller Benutzer. Ein Funktionsloser Button steht zur Verfügung um später Freundeslisten zu implementieren.

Rechts unten kann man seine Wunschliste einsehen und bearbeiten.