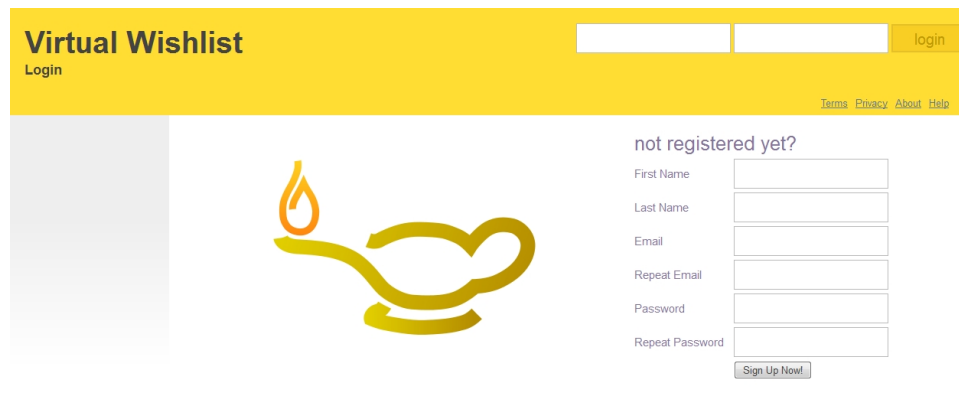


# Virtual Wishlist



**Virtual Wishlist**  
Login

[Terms](#) [Privacy](#) [About](#) [Help](#)

not registered yet?

First Name

Last Name

Email

Repeat Email

Password

Repeat Password

Figure 1: Welcome Page of the Virtual Wishlist

Group project carried out by

Josua Dietrich, Nils Braune and Tino Canziani

created 2011 at D-ITET, ETH Zürich

21.12.2011

# Contents

---

<b>1</b>	<b>Introduction and Background</b>	<b>3</b>
1.1	Idea . . . . .	3
1.2	Related Work . . . . .	3
1.3	Tools . . . . .	3
<b>2</b>	<b>Concept</b>	<b>5</b>
2.1	Core functions . . . . .	5
2.2	Website Layout . . . . .	5
2.3	Sessions & Profiles . . . . .	7
<b>3</b>	<b>Implementation</b>	<b>8</b>
3.1	HTML/Login/Session (Welcome.html) . . . . .	8
3.2	MySQL server: creating a database . . . . .	9
3.3	setupConnection.php . . . . .	11
3.4	\$.ajax(), getWishlists() . . . . .	12
3.5	Adding a profile picture . . . . .	14
<b>4</b>	<b>Results and Future Steps</b>	<b>16</b>
4.1	Results . . . . .	16
4.1.1	Welcome Page . . . . .	16
4.1.2	Profile Page . . . . .	16
4.1.3	my Wishes menu . . . . .	16
4.1.4	my Friends and Search Friends Menu . . . . .	16
4.1.5	Settings menu . . . . .	17
4.1.6	Friend Page . . . . .	17
4.1.7	Terms, Privacy, About, Help Page . . . . .	17
4.1.8	Process of fulfilling and getting wishes fulfilled . . . . .	17
4.1.9	Security . . . . .	17
4.2	Future steps . . . . .	18
<b>A</b>	<b>Server/PHP functions</b>	<b>20</b>
<b>B</b>	<b>Screenshot of the profile page</b>	<b>21</b>
<b>C</b>	<b>Setting up the Virtual Wishlist step by step</b>	<b>22</b>
<b>D</b>	<b>Endnotes</b>	<b>23</b>

# Abstract

---

In this group project we designed and implemented a virtual wish list as a social media service on the web. It enables its users to maintain a personal profile, create their own wish lists or opt to grant other friends their specific wishes. The use case of this virtual wish list will mostly tend to birthday and christmas events, social gatherings, leisure activities or work related events. For instance, a couple will be able to create a wish list for their wedding, making it accessible online to all the guests listed in their friend list. A guest may then opt to fulfill any of those wishes on the wish list, which will be marked thereafter and thus making sure that no second guest will bring the same gift.

The implementation is based on the web standards HTML, JavaScript, PHP and SQL. HTML describes the structure of the web page, JavaScript its dynamic interaction at run time, PHP enables the flow of information between user and database and SQL fetches or changes the entries in the database. The programming was done with an IDE named Aptana Studio and the implementation was built upon the very common XAMPP stack.

## Chapter 1

# Introduction and Background

---

### 1.1 Idea

The general concept is simple and clear: Let people create their own wish list on the web to share them with others. This should address two kinds of problems: being in a situation where you receive a gift you aren't very happy about because you didn't want it, and the situation of not being able to figure out or find a suitable gift for a person. Our project proposes a solution to circumvent both of these embarrassing situations.

It is important to note that a wish list should not necessarily have to be for a birthday party, it should be usable for an arbitrary event, such as a picnic. Attendees could then organize who brings what.

### 1.2 Related Work

Similar concepts already exist on the web, notably websites which let you manage wedding presents, such as [weddingwishlist.ie](http://weddingwishlist.ie), [weddingshop.com](http://weddingshop.com) or [marriagegiftlist.com](http://marriagegiftlist.com).

But we haven't come across a virtual wish list designed for an arbitrary kind of event and with the possibility to set up personal profiles. The use cases were limited exclusively to specific events.

Another site offering a similar service is [amazon.com](http://amazon.com). Yet it is bound to its own selling platform. And there is [facebook.com](http://facebook.com), as a social media service, which yet lacks the functionalities of wish lists.

The idea of facilitating the organization of events was also grabbed up by [doodle.com](http://doodle.com), yet addressing different kinds of problems, as it is a site which allows people to show their availability for different dates and times.

### 1.3 Tools

To manage our code, we used a CVS (Concurrent Versions System) called

**SVN**<sup>1</sup>

A SVN repository allows to store all code files on an external server such that all group members can update their local code (`svn update`) to the newest version or save it back onto the file server when finished (`svn commit`). Every change is tracked and can be reset at any time, while SVN automatically identifies conflicts in the code. The work was done on an IDE (Integrated Development Environment) called

**Aptana Studio 3**<sup>2</sup>

It is available as a standalone version as well as a eclipse plugin. The service is built upon a server called

**XAMPP**<sup>3</sup>

It bundles an **Apache server**<sup>4</sup> to display web pages with content, a **PHP environment**<sup>5</sup> for server side computing as well as a **MySQL database**<sup>6</sup> which stores our user data. You can run XAMPP locally for development purposes, but for deployment, there will be a significant physical distance between web server and a user surfing the web with a browser.

The PHP functions are called upon by JavaScript through the `$.ajax()` method. This method is part of a set of functions in the

**jQuery Library**<sup>7</sup>

which is freely available on the web and is used to simplify common tasks in JavaScript. This library also provides other useful functionality, such as the handling of events like mouse clicks as well as visual animation effects.

## Chapter 2

# Concept

---

The following section describes the evolution of an idea into an implementable concept.

### 2.1 Core functions

The core functions of the service consist of:

Client	Server (database)
Create, edit and delete profile	Store profiles
Create, edit and delete wish lists	Store wish lists
Create, edit and delete wishes	Store wishes
Add and delete friends	Store user relationships
View and/or fulfill own or friends' wishes	Store user-wishes and relationships
Login and logout	Start and end a session for a user

Table 2.1: Core Functions

As seen in the table 2.1 above, all actions a user proceeds requires an immediate action by the server. A list of necessary functions on the server side to handle the actions of the user can be found in Appendix A. Beside the core functionalities, a website layout is required, as discussed in the next section.

### 2.2 Website Layout

The layout of the site has to follow some requirements:

- The website must present itself in a simple manner, everyone should immediately be able to use it.
- Content like wishes or names should be loaded dynamically into the same page, instead of loading the whole page time and time again.
- The service should consist of a few single webpages, rather than many different ones.

The following page will show a diagram of the website flow 2.1. For a screenshot of the profile page, please see Appendix B.

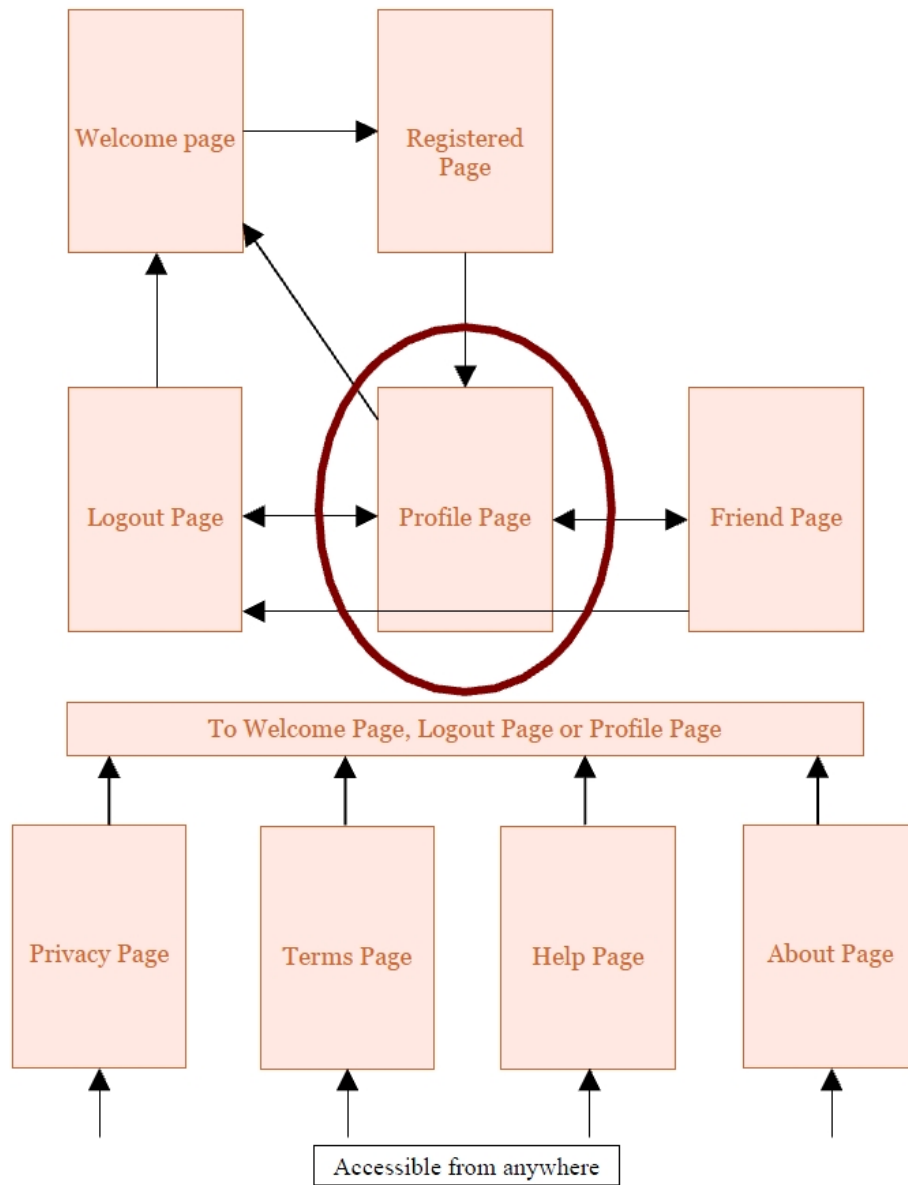


Figure 2.1: Websites Flow

**A short description of each page:**

**Welcome.html:** Welcomes the user and allows registering and logging in to the virtual wish list service

**Profile.html:** Shows the user's profile with his wishes, profile details and friends

**Friend.html:** Shows a friend's profile with all his wishes, personal information and friends

**Registered.html:** After a user registered, he will be redirected here

**notLoggedIn.html:** If no session exists or a user logged out unexpectedly, he will be redirected to this page

**Privacy.html:** Displays details on the privacy agreement (which has still to be written)

**Terms.html:** Displays terms and conditions (which also have to be written yet)

**Help.html:** Displays help

**About.html:** Displays about

## 2.3 Sessions & Profiles

To handle user data and address privacy concerns correctly, the data needs to be stored and linked on a per-profile basis. Also, the connections to several users at the same time need to be separated in sessions. With this, and by encrypting every connection, user data can be protected from illegitimate manipulation.



## Chapter 3

# Implementation

---

This chapter will explain the programming by highlighting and showing some code examples of some important functions. The general structure of any web page looks the following:

HTML	JavaScript	PHP	SQL
Defines and makes visible: inputfields, simple texts, pictures etc.	Calls PHP functions with AJAX and handles clicks on buttons etc.	Communicates with the database and loads the content into the site	Inserts data from the requests and stores data.

Table 3.1: Language Usage

When looking at the source code, these program files will be suffixed .html, .js, .php and .dml/.ddl respectively. HTML will include JavaScript functions, which call PHP code in files on the server, which then again connect to our database and call SQL procedures.

The starting page is called Welcome.html. If someone tries to access other web pages without logging in before, he will also be redirected to this page (as a result of his missing session).

### 3.1 HTML/Login/Session (Welcome.html)

In our case, HTML<sup>8</sup> merely defines the regions and containers where content is placed in. The properties and attributes of those are set in CSS<sup>9</sup> (cascading style sheets), which we will not examine further. Following figure shows the HTML passage for the login boxes:

```
1 <div id="loginBox">
2   <form method="post" style="top:2px,right:355px" action="login.php">
3     <input class="textform" id="loginEmail" type="text"
4       name="email" maxlength="40" onkeypress="ifEnterLogin(event,this.form)"/>
5     <input class="textform" id="loginPassword"
6       type="password" name="password" maxlength="30"
7       onkeypress="ifEnterLogin(event,this.form)"/>
8     <input id="login" type="button" name="login"
9       value="login" >
10  </form>
11  <div id="loginErrorMessage" class="errorMessage"
12    style="top:22px,right:355px"></div>
13 </div>
```

Figure 3.1: HTML Definition loginBox

It makes appear two text input fields in the top right corner. Line 1 defines a <div>, the common container for content in the HTML markup language. Line 2 adds a form, which has three inputs assigned to it, one for the email address (line 3), one for the password (line 5) and a last one being the log in button (line 8). The onkeypress event will call a JavaScript function, which will check if the user has pressed the 'enter' key, causing the system to commence a login attempt. Lines 11/12 define a region in which an error message will appear if the input is not valid. Clicking the input button will trigger the call of the login() JavaScript function. This will validate if the user exists in the database by calling a PHP function, again with the name login(). Furthermore, a session is initialised, as visible below:

```

1 function login(){
2   $getUser = "CALL get_user('".$REQUEST['email']."'");
3   $userResult = mysql_query($getUser);
4   if (mysql_num_rows($userResult) == 0 || mysql_num_rows($userResult) > 1) {
5     echo "error";
6   } else {
7     $user = mysql_fetch_array($userResult);
8     $passwordHash = hash("sha512", $REQUEST['password'].$user["salt"]);
9     if ($passwordHash == $user["password"]) {
10      session_start();
11      $_SESSION["salt"] = $user["salt"];
12      $_SESSION["passwordHash"] = $user["password"];
13      // take a hash of the PID and the login time, keeping the PID
14      // internally, not showing it to the user
15      $_SESSION["loginTime"] = time();
16      $_SESSION["PID"] = $user["PID"];
17      $_SESSION["hashPID"] = hash("sha256",
18        $_SESSION["PID"]*$_SESSION["loginTime"]);
19      setcookie("PID", $_SESSION["hashPID"], 0, "/", ".localhost", true);
20      // $expires = 0: expires at the end of session
21      $_SESSION["name"] = $user["name"];
22      $_SESSION["lastname"] = $user["lastname"];
23      $_SESSION["birthdate"] = $user["birthdate"];
24      $_SESSION["street"] = $user["street"];
25      $_SESSION["citycode"] = $user["citycode"];
26      $_SESSION["city"] = $user["city"];
27      $_SESSION["aboutme"] = $user["aboutme"];
28      $_SESSION["email"] = $user["email"];
29      $_SESSION["profileLoaded"] = true;
30      echo "success";
31    } else {
32      echo "error";
33    }
34  }
35 }

```

Figure 3.2: PHP Function login()

Line 2 produces a SQL command string containing the call of a procedure with the given email. Line 3 sends this command to the database server through the standard function `mysql_query()`. The fetched data is stored in `$result`. If the email does not exist or there are multiple database entries with the same email we throw an error message (lines 4-6). Otherways we create a password hash with the password the user typed in and the stored salt string and compare the result to the stored password hash (lines 7-8). If the password hashes match (line 9), an individual session is started for the user using the standard command `session_start()` (line 10), which allocates memory for his session and initializes it. Lines 11-30 store all the necessary data in this session and line 19 sets a hashed user ID cookie which is stored locally on the clients' browser for identification purposes. The user will now be able to access his profile and use all the functionalities with the given user ID cookie. When he decides to log out, the cookies are set to expire and the session is closed through the standard `session_unset()` method.

To track the login and logout of each user, we found a fitting set of core PHP functions implementing user sessions<sup>10</sup>. When the user logs in and requests his data, the information is stored into the servers' RAM in a predefined `$_SESSION` array assigned to that user. When he logs out, his session memory is cleared. Through the process of hashing and salt<sup>11</sup> sensitive data such as passwords and person IDs is protected from being compromised by external threats. The data is transmitted over an encrypted SSL connection with a (yet) unsigned certificate. Sessions expire even when the user forgets to log out after a certain time.

### 3.2 MySQL server: creating a database

Every database needs to be set up before use. Ours is called genie. It needs to know what to store and how its stored elements interact with each other or the outside world. The structure of a database is made by adding tables and linking them with each other. These are sophisticated, two dimensional arrays

which are able to track and control dependencies among each other. Each user will have different data associated to the tables named *person*, *photo*, *wishlist*, *wish*, *wishfulfiller*, *friend* and *friendrequest*. An example:

```

1 DROP TABLE IF EXISTS friendrequest;
2 create table friendrequest
3 (
4     FRID integer not null auto_increment,
5     friendrequest1 integer,
6     friendrequest2 integer,
7     primary key (FRID),
8     foreign key (friendrequest1)
9         references person(PID) ON DELETE cascade,
10    foreign key (friendrequest2)
11        references person(PID) ON DELETE cascade
12 );

```

Figure 3.3: SQL Table friendrequest

Line 1 asserts that tables are not defined twice. Line 2 includes the SQL demand to create a table. Lines 4-6 define the columns of our table. Lines 5/6 will tell us which user sent a friendrequest to which other user. Lines 7-11 make sure that the table entries are identified by their FRID (“friend request ID”) and that entries corresponding to a specific user (PID for “person ID”) are deleted when the user is deleted (lines 9/11).

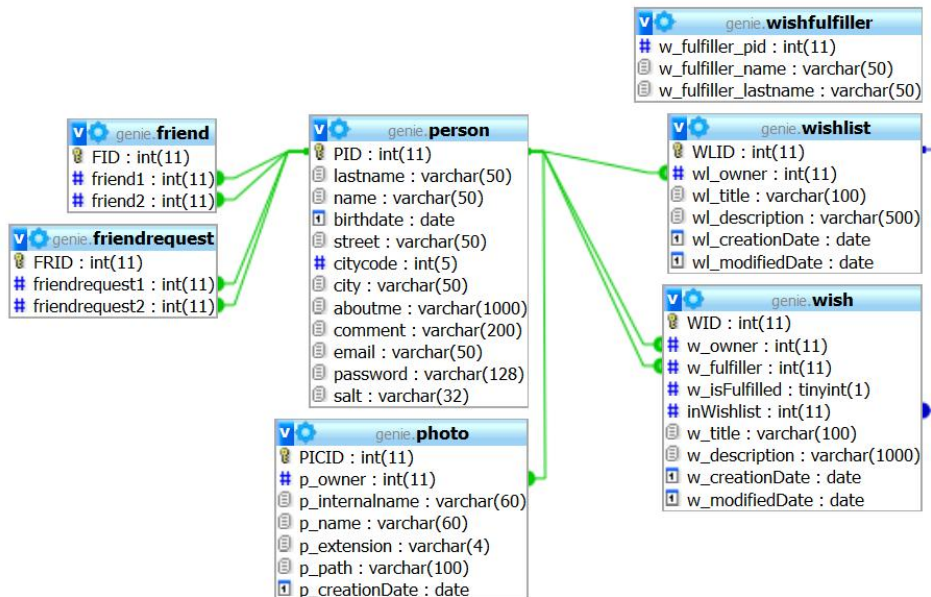


Figure 3.4: Database Structure

The figure 3.4 shows the structure of our database. As you can see, our data is stored in tables or so-called entities, every of them containing a key and several attributes. The tables are linked amongst each other meaning that an attribute in one table can reference a key in an other table. The attribute `genie.wish.w_fulfiller` for example depends on `genie.person.PID`. These links, also called foreign key constraints, are represented by the lines between the tables and show the structure among the different tables. Please note that `genie.wishfulfiller` is not a table, but a "view". It selectively takes data from the table `genie.person` and presents an interface to that data accessible from the outside, while the data showed by `genie.wishfulfiller` itself is not re-stored physically.

After setting up the tables, we need procedures (SQL statements similar to functions) to read and write on these tables. These procedures are called by PHP functions, which connect to the database. An example:

```

1 DROP PROCEDURE IF EXISTS genie.get_wishlists;
2
3 CREATE PROCEDURE genie.get_wishlists(v_pid integer)
4     SELECT wl.wlid, wl.wl_title, wl.wl_description, wl.wl_creationDate,
5     w.wid, w.w_title, w.w_description, w.w_creationDate
6     FROM wishlist wl
7     INNER JOIN wish w ON wl.wlid = w.inWishlist
8     WHERE w.w_owner = v_pid
9     ORDER BY wl.wl_title, w.w_title;

```

Figure 3.5: SQL Procedure `get_wishlists()`

Line 1 asserts that the procedure is not defined twice. Line 3 includes the SQL demand to create a procedure. We select data from the wishlist table (line 4-6) and data from the wish table (line 7) from our user (line 8), which is then sorted (line 9) and returned to the calling PHP function, which will relay the data to the user. For testing purposes, we wrote code fragments which insert example data into the database through predefined procedures. When setting up a new database, we execute following code segments in the given order: `importalltables.ddl`, `importallmethods.dml`, `importAllExampleData.dml`. The database is now ready. All we have to do is to set up a connection to it.

### 3.3 setupConnection.php

As stated before, PHP enables the communication between the clients' system and the server with the database. But before a connection is established through the `setupConnection.php` method, a check is performed to see if the user is still logged in by the `checkCookie()` function in `Profile.js`:

```

1 function checkCookie() {
2     var pid = contentFromCookie("PID");
3     if (!pid){
4         window.location = "notLoggedIn.html";
5         return false;
6     } else {
7         return pid;}}

```

Figure 3.6: Javascript Function `checkCookie()`

Line 2 takes the content from the cookies PID. If there is no content, the user will be redirected to the page "notLoggedIn.html" (line 4). After this short check, the connection is established:

```

1 function connectDatabase(){
2     $dbuser = "root";
3     $pw = "";
4     $server = "localhost";
5     $database = "genie";
6     $connection = mysql_connect($server,
7     $dbuser, $pw);
8     mysql_select_db($database, $connection);
9     return $connection;}

```

Figure 3.7: PHP Function `connectDatabase()`

Lines 1 and 2 save the username and password for the database (which in this example are set to root and none). The command `mysql_connect()` establishes the desired connection to the database server. Since we are working

with XAMPP locally, the server's address is simply "localhost". Line 8 includes a necessary step to select the database we want (here: "genie") since one server can have more than one database.

### 3.4 \$.ajax(), getWishlists()

One core functionality available to the user is looking at his wish lists. Doing so involves the contribution of most of the processes mentioned above. The following explanation on how a wishlist is loaded into the users Profile page wraps up the code. First we click on "My Wishes" in the Profile page. This will activate the following JavaScript event:

```

1 $("#myWishes").click(function() {
2   position.top = startPosition - 5*50;
3   $("#menu-arrow").animate({top: position.top}, 400);
4   showWishlists();
5   return false;});

```

Figure 3.8: Button for 'My Wishes'

When the user clicks on "My Wishes", which is the HTML element named myWishes, the functions above will be called. Line 1 declares the event and binds it to the clicking of the element myWishes. Lines 3 and 4 move a small triangle in the left menu bar as a visual effect. Line 5 will then call the main function showWishlists() seen below in Figure 3.9

```

1 function showWishlists() {
2   var pid = checkCookie(); // still logged in? check for Cookie
3   $.ajax({
4     url: "../server/getWishlists.php",
5     success: function(msg) {
6       if(msg == "not logged in"){
7         window.location = "notLoggedIn.html";
8       } else {
9         $("#content").empty()
10        .prepend(msg);
11        $("#addWishlist").click(function() {
12          // alert("hello"); // debug
13          addWishlist();
14          return false;
15        });
16        var onHover = function() {
17          $(this).children(".editWish").css("visibility", "visible");}
18        var onOut = function() {
19          $(this).children(".editWish").css("visibility", "hidden");}
20        $("#wishTable tr").hover(onHover, onOut);
21    });});

```

Figure 3.9: showWishlists()

Before continuing, we will check if the user is still logged in (line 2). If not, he will be directed back to the welcome page. After that, the \$.ajax() method is called. It comes from the jQuery library we included and has the job to make an AJAX request to a PHP function and send it the proper inputs. This routine takes the following parameters: The PHP script to be called (line 4) and what to do in case of a successful execution of the PHP script (line 5). If the user is still logged in, we fill the HTML container named content with the message send by the PHP script which contains our wish lists stored in the database, after emptying it beforehand (lines 9/10). Lines 11 addresses the element addWishlist, which is added our table in the PHP script and allows us to add a new wishlist (line 13) by triggering the addWishlist() function when addWishlist is pressed. We will now go one level deeper into the code, into the PHP.

```

1 function getWishlists() {
2   $getWishlists = "CALL get_wishlists(".$_SESSION["PID"].")";
3   $result = mysql_query($getWishlists);
4   $responseText = "<table>";
5   $currentWishlist = null;
6   if (mysql_num_rows($result) == 0) {
7     ... 4 Lines of code...
8   } else {
9     $currentWishlist = null;
10    while ($wishlistResultRow = mysql_fetch_assoc($result)) {
11      if (!$currentWishlist == $wishlistResultRow["wlid"]) {
12        if ($currentWishlist != null) {
13          $responseText.= "<tr><td class='wish'><a
14            id='addWishName".$currentWishlist."' class='wishEdit'
15            title='Add a new Wish to your Wishlist.'
16            onclick='addWish(\".$currentWishlist.\")' href='#'
17            linkindex=''>add new Wish</td>";
18          $responseText.= "<td><a id='addWishComment'".
19            $currentWishlist." class='wishEdit'></a></td><td><a
20            id='addWishSubmit'".
21            $currentWishlist."></a></td></tr>";
22          $responseText.= "<tr>&nbsp;</tr>";
23          $responseText.= "<tr id='".$wishlistResultRow["wlid"].">";
24          $responseText.= "<td class='wishlist' >".
25            $wishlistResultRow["wl_title"]."</td>";
26          $responseText.= "<td class='wishlist' >".
27            $wishlistResultRow["wl_description"]."</td>";
28          $responseText.= "<td class='wishlist' >".
29            $wishlistResultRow["wl_creationDate"]."</td>";
30          ... 2 Lines of code...
31          $responseText.= "</tr>";
32          $currentWishlist = $wishlistResultRow["wlid"];
33        } else if ($wishlistResultRow["wid"]) {
34          ... 7 Lines of code ...
35        }
36        $responseText.= "<tr><td class='wish'><a id='addWishName'".
37          $currentWishlist." class='wishEdit' title='Add a new Wish to
38          your Wishlist.' onclick='addWish(\".$currentWishlist.\")'
39          href='#' linkindex=''>add new Wish</td>";
40        $responseText.= "<td><a id='addWishComment'".$currentWishlist."
41          class='wishEdit'></a></td><td><a id='addWishSubmit'".
42          $currentWishlist."></a></td></tr>";
43        $responseText.= "<tr><td class='wish'><div id='addWishlistName'><a
44          id='addWishlist' title='Add a wishlist.'
45          onclick='addWishlist()' accesskey='' href='#'>add new
46          Wishlist</a></div></td>";
47        $responseText.= "<td><a id='addWishlistComment' class='wishEdit'></
48          a></td><td><a id='addWishlistSubmit'></a></td></tr>";
49        $responseText.= "</table>";
50      }
51      echo $responseText;}

```

Figure 3.10: Function `getWishlists()`

We therefore look at the PHP code of `getWishlists()` as presented in figure 3.12. Line 2 initialises the SQL command which will be sent to the server on line 3. The result is saved in `$result`. We are going to need 2 variables for the upcoming processing: `$responseText`, which will store the message going back to the Javascript file by the `echo` command right at the end (line 51) and `$currentWishlist`, which keeps track which wishlist we are looking at when loading the wishes, since one user can have several wishlists. Lines 6/7 handle a empty return my the SQL server, these 12 lines of code are omitted for this purpose and can be seen in the code. If `$result` is not empty we will work through every single wishlist and read it out first into `$wishResultRow` (while loop, line 10). For simplification purpose, we will only look at the posting of wishlist elements into `$result`, since the adding of wishes is similar. First is checked if there is a new wishlist or not. I yes, we continue to store that into `$currentWishlist`(line 11). Lines 13-21 add the inputs for adding a new wish. After that we first enter the wishlist ID into our `$responseText`(line23), then subsequently the title (lines 24/25), the description and creation date (lines

26-29). The next element added is an input in the form of a picture to delete a wishlist (omitted). This button is attributed an `onclick` event with the parameter being the current wishlist. The `onclick` function will be triggered, as the name suggests, when the user clicks on the picture. Line 34 would have shown a similar procedure for an “edit” button, which lets the user edit the wishlist. Before returning the resulting `$responseText` to our JavaScript function, the `$currentWishlist` variable is updated on line 35. Lines 36-48 add the functionality to add a new wish. So to summarize, the initial action will be fetching the data from the database, after which the data is processed and written in a structured manner into the variable called `$responseText`. This variable will be sent back to the JavaScript function (where it is referred to as `msg`) which will write it into the container called `content` in the Profile page. As mentioned, we have generated an input to delete a wishlist. The code executed when the “delete” picture is clicked is the following:

```

1 function deleteWishlist(wlid) {
2   var pid = checkCookie();
3   $.ajax({
4     url: "../server/deleteWishlist.php",
5     async: false,
6     data: ("wlid=" + wlid),
7     success: function(msg) {
8       if(msg == "not logged in!"){
9         window.location="notLoggedIn.html";
10      } else if(msg != "error"){
11        showWishlists();
12      } else {
13        alert(msg);
14      }
15    }
16  });
17 }

```

Figure 3.11: `deleteWishlist()`

As you might have noticed, the structure is similar to the JavaScript function `showWishlists()`. But here we have another extra parameter in our `$.ajax()` method: “`async: false`”. It will assure that no other JavaScript function will be executed until the Wishlist is deleted. AJAX will call `deleteWishlist.php`, which takes the wishlist ID `wlid` and sends a command to the SQL server to delete the corresponding wishlist (“CALL `delete_wishlist('.$_REQUEST['wlid']')`”).

### 3.5 Adding a profile picture

Because of scope reasons the `uploadNewPhoto.php` file contains a whole HTML form, namely the `UploadNewPhoto` form (line 1-12) to search for a file on the user’s computer, which is loaded (line 13) into an `iframe`. When the `Upload Photo` button gets pressed, the variable `$_POST` is set and the file written into the variable `$_FILES`. The figure 3.12 shows the structure of the process to upload the photo. To keep it simple we didn’t include the subfunctions, but their name explain their use.

If the `$_POST` variable is set and no `filesize-error` on the client side occurred we start the uploading process (line 15-16). We get the `$filename` and extension `$ext` (line 17-19) and use our previously defined routine `checkExtension()` to determine the validity of the extension. If it isn’t valid, an error message is returned and the uploading process ended (line 20-22). The same thing happens with the `filesize` using our predefined routine `checkFileSize` (line 24-26). We create an internal image name using a time stamp (line 27) and a path to store our photo in the filesystem (line 28). We copy the file from the variable `$_FILES` to our path using the PHP internal function `copy()`. If the attempt

```

1 $form = "<link rel='stylesheet' href='../css/general/style.css'>";
2 $form .= "<link rel='stylesheet' href='../css/Profile.css'>";
3 $form .= "<form name='uploadNewPhoto' method='post' enctype='multipart/form-data'
4 action='../server/uploadNewPhoto.php'><table><tbody>";
5 $form .= "<tr><td><input type='hidden' name='MAX_FILE_SIZE'
6 value='1048576' /></td></tr>";
7 $form .= "<tr><td style='width: 166px; vertical-align: middle'><span
8 class='label settings'>upload a new photo:</span></td>";
9 $form .= "<td><input class='textform settings' type='file' name='image'></td></tr>";
10 $form .= "<tr><td><input id='submitPhoto' name='Submit'
11 onSubmit='window.parent.upload(this);' type='submit' value='Upload
12 Photo'></td></tr>";
13 $form .= "</tbody></table></form>";
14 echo $form;
15 if(isset($_POST['Submit'])) {
16     if (!($_FILES['image']['error'] == UPLOAD_ERR_FORM_SIZE)) {
17         $filename = stripslashes($_FILES['image']['name']);
18         $ext = getExtension($filename);
19         $ext = strtolower($ext);
20         if(!checkExtension($ext)){
21             echo "<div class='errorMessage'>Unknown extension!</div>";
22             return 0;}
23         $size=filesize($_FILES['image']['tmp_name']);
24         if(!checkFileSize($size)){
25             echo "<div class='errorMessage'>You have exceeded the size limit!</div>";
26             return 0;}
27         $internal_image_name=time().'.'.$ext;
28         $path="../images/".$internal_image_name;
29         if (!copy($_FILES['image']['tmp_name'], $path)) {
30             echo "<div class='errorMessage'>Photo upload failed!</div>";
31             return 0;}
32         ... 11 lines of code ...
33         $original_image_name = getNameOnly($filename);
34         $date = date("Y.m.j");
35         if(!storePhotoDetails($PhotoOwner, $internal_image_name,
36 $original_image_name, $ext, $path, $date)){
37             $deleted = unlink($path);
38             echo "<div class='errorMessage'>Store new photo details failed!</div>";
39             return 0;
40         } ... 6 lines of code ...
41 }}

```

Figure 3.12: Code fragments from uploadNewPhoto.php

fails, an error message is returned and the uploading process ends (line 29-31). We ask the database for the details of the old photo (line 32, code not shown) in order to remove the old photo from the file system in a later step (line 40, code not shown). We prepare the new photo details for the database and overwrite the database-entry of the old photo (line 34-36). If the attempt fails, the new photo gets deleted from the filesystem, an error message is returned and the uploading process is interrupted. A handy additional feature would be a photo crop and resize/rescale function (to be used before putting the photo into the filesystem), but this has not been implemented yet.



# Results and Future Steps

---

## 4.1 Results

The following list is an overview of the implemented functionality. We concentrated on listing the the characteristic functionality for the Virual Wishlist itself and as well implemented functionality that cannot be seen directly by using the page but is running in the background as for example checking the photo size when uploading the photo.

### 4.1.1 Welcome Page

- Register form
  - check for empty fields which are not allowed
  - check matching of password and repeat password
  - check matching of email and repeat email
  - check whether the user that wants to register already exists or not
- login form
- enter key can be used in both forms as alternative to pressing the respective button.
- check if cookie exists and redirection to Profile.html if it does

### 4.1.2 Profile Page

- namebar with the user's name at the top of the page
- overview with the newest wishes from friends
- overview with wishes user wants to fulfill
- 'upload new photo' button when hovering over photo
- design: white arrow pointing on current menu item

### 4.1.3 my Wishes menu

- input sanitisation
- add wishlists and wishes
- edit and delete wishlists and wishes
- design: background color orange when hovering over names
- design: date, edit button and delete button have a fade-in/fade-out effect

### 4.1.4 my Friends and Search Friends Menu

- display friendlist
- display sent and recieved friendrequests
- delete friend-relation
- making, accepting and declining friendrequest
- click on names to get on other people's profile

#### 4.1.5 Settings menu

- change all user details except the email address, as it is used as an identifier
- change password
  - empty password fields not allowed when wanting to submit new password
  - check matching of new password and repeat new password
  - design: fade out effect for password notification
- upload a photo
  - check file type
  - check file size
  - unique internal name for photo
  - replace old photo
  - database entry for photo details
  - photo stored in file system
- delete user profile

#### 4.1.6 Friend Page

- view wishes, information and friends
- click to show intention to fulfill a wish
- undo intention to fulfill wish

#### 4.1.7 Terms, Privacy, About, Help Page

- Button to get back to Profile Page or Welcome Page

#### 4.1.8 Process of fulfilling and getting wishes fulfilled

- user can click 'fulfill wish' or undo it.
- a wish cannot be fulfilled by two users at the same time
- user doesn't see he gets a wish fulfilled
- user can mark his wish as fulfilled

#### 4.1.9 Security

- auto-logout when cookie not available for any reason on every page
- auto-logout when forgetting to logout
- sanitization of Input
- hashed password with salt in database and in cookie

## 4.2 Future steps

The next step now would be to go online. Nonetheless we consider it being a good idea to start programming the Virtual Wishlist from scratch again using a server side framework. The reason for that is to have a standardised environment where common tasks are greatly simplified.

In order to go online we intend to put emphasis on scalability and handling multiple server requests, data security, user friendliness of the page and design polishment. Regarding scalability and handling multiple server requests the database choice should be reevaluated. In contrast to a relational database as MySQL using an object-relational database such as PostgreSQL could be the preferred choice when it comes to a very large amount of user data and user profiles. It has been found that object-relational databases are given the first draw when handling millions of data points or data objects such as in GIS systems.

After going online, we proceed to gather users as a first task and making a business out of Virtual Wishlist as a second one. There are various opportunities to do so, ranging from setting up a Facebook landing page, press releases, mailings, to marketing campaigns and search engine optimisation. Having a fair amount of users would give us the opportunity to make a business by referring to relevant products and offers as soon as someone wants to fulfill his or her friend's wish.

# Acknowledgements

---

We would like to thank the following people for their contribution to the group project by providing conceptual advise, infrastructure and guidance.

Professor Bernhard Plattner, our supervisor, for allowing us to develop the project under his institute for technical infromatics and communication networks and intital advise.

Sacha Trifunovic, our assigned advisor, for his continuous guidance, constructive inputs and developing a work plan.

The members of the IT-Support-Group who set up and keep running a SVN repository to store and handle our code.

## Appendix A

# Server/PHP functions

---

The file name explains its use.

AcceptFriendRequest.php	GetReceivedFriendRequests.php
AddWish.php	GetSentFriendRequests.php
AddWishlist.php	GetSettings.php
CheckUser.php	GetUser.php
DeclineFriendRequest.php	GetUserlist.php
DeleteFriend.php	GetWishesToFulfill.php
DeleteFriendRequest.php	GetWishlists.php
DeleteProfile.php	Login.php
DeleteWish.php	Logout.php
DeleteWishlist.php	MakeFriendrequest.php
EditPassword	NewUser.php
EditUserdata.php	SetupConnection.php
EditWish.php	SetWishIsFulfilled.php
EditWishlist.php	SetWishNotFulfilled.php
FulfillWish.php	UndoFulfillWish.php
GetFriend.php	UpdateFriend.php
GetFriendFriendlist.php	UpdateSettings.php
GetFriendlist.php	UpdateWishesToFulfill.php
GetFriendPhoto.php	UpdateWishlists.php
GetFriendSettings.php	UploadNewPhoto.php
GetFriendWishlists.php	ViewFriend.php
GetNewestWishes.php	ViewFriendsFriend.php
GetPhoto.php	

## Appendix B

# Screenshot of the profile page

---

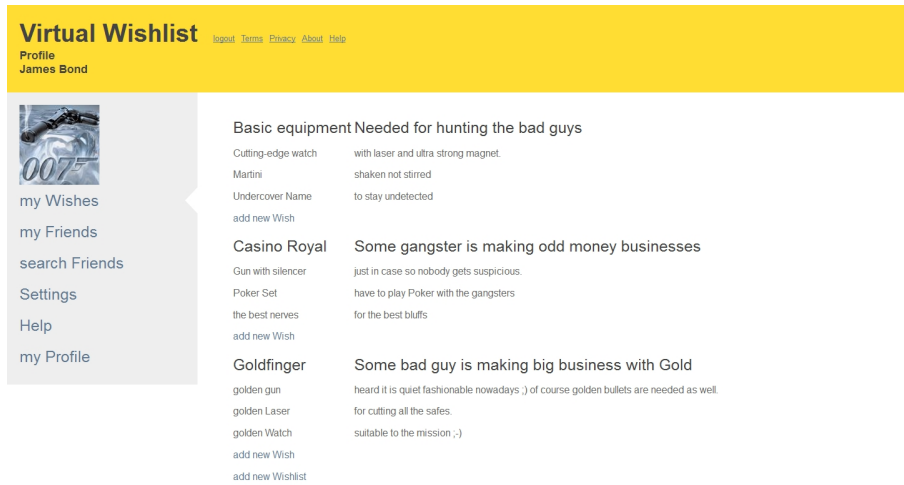


Figure B.1: Profile Page

## Appendix C

# Setting up the Virtual Wishlist step by step

---

In order to get our Virtual Wishlist running on a computer (running Windows), the following steps need to be accomplished:

- Install XAMPP from <http://xampp.com>
- Copy our Code to `../xampp/htdocs/`. Copy the entire folder 'Gruppenarbeit' to the named location.
- Follow the steps in `../Gruppenarbeit/SSL/HOWTO.txt` to configure SSL on your server
- Start XAMPP:
  - run 'setup\_xampp.bat' in `../xampp`. This needs to be executed only once after having installed xampp.
  - run 'xampp-control.exe' in `../xampp` and press the button 'Start' next to 'Apache' and 'MySQL'
- Set up the database:
  - replace `../xampp/phpmyadmin/config.inc.php` with `../xampp/htdocs/Gruppenarbeit/configuration/config.inc.php`.
  - open your browser and type 'localhost' into the adress bar,
  - open 'phpmyadmin' in the XAMPP menu
  - import the file 'setupDatabase.ddl' from '`../xampp/htdocs/Gruppenarbeit/My_SQL/example data & import files`' using the import tab.
- Open our Virtual Wishlist homepage by typing 'https://localhost/Gruppenarbeit/html/Welcome.html' into the browser's adress bar.
- You can now register a new user account or sign into an existing user account using the identities provided in the table C.1 below.

Email	Password
jamesbond@mi6.com	james
vindiesel@triplex.com	vin
johnnyenglish@topsecret.com	johnny
batman@gotham.com	batman
superman@metropolis.com	superman
spiderman@manhattan.com	spiderman
luke@xwingfighter.com	luke

Figure C.1: Predefined User Accounts

## Appendix D

# Endnotes

---

### Notes

<sup>1</sup>SVN was provided by the [isg.ee.ethz](http://isg.ee.ethz)

<sup>2</sup>Aptana Studio 3 was downloaded from <http://aptana.com/products/studio3/download>

<sup>3</sup>XAMPP was downloaded from <http://xampp.com>

<sup>4</sup>Apache server standard see on <http://apache.org>

<sup>5</sup>PHP see on <http://php.net>

<sup>6</sup>MySQL standard see on <http://mysql.com>

<sup>7</sup>The jQuery-library <http://jquery.com>

<sup>8</sup>For instructions and tutorials on the topic see <http://www.w3schools.com/>

<sup>9</sup>The Cascading Style Sheet Standard see on <http://www.w3.org/Style/CSS/>

<sup>10</sup>Usage of php-sessions see on <http://php.net/manual/en/book.session.php>

<sup>11</sup>Usage of salt see [http://de.wikipedia.org/wiki/Salt\\_\(Kryptologie\)](http://de.wikipedia.org/wiki/Salt_(Kryptologie))