WiFinder

Report

Nicvre-3 9012053196

Contents

1 User problem **2**

2 Design & Architecture **3**
2.1 LAMP Server 3
2.2 Database & Google maps API 3

3 API Design & Usage **4**
3.1 Design 4
3.2 Usage 4
3.3 API Structure 7
3.4 API Calls 11
3.4.1 adduser 11
3.4.2 addwifi 11
3.4.3 deletewifi 12
3.4.4 login 12
3.4.5 logout 12
3.5 Security 13
3.5.1 SQL injections 13
3.5.2 Hashing 13

4 Technologies **14**

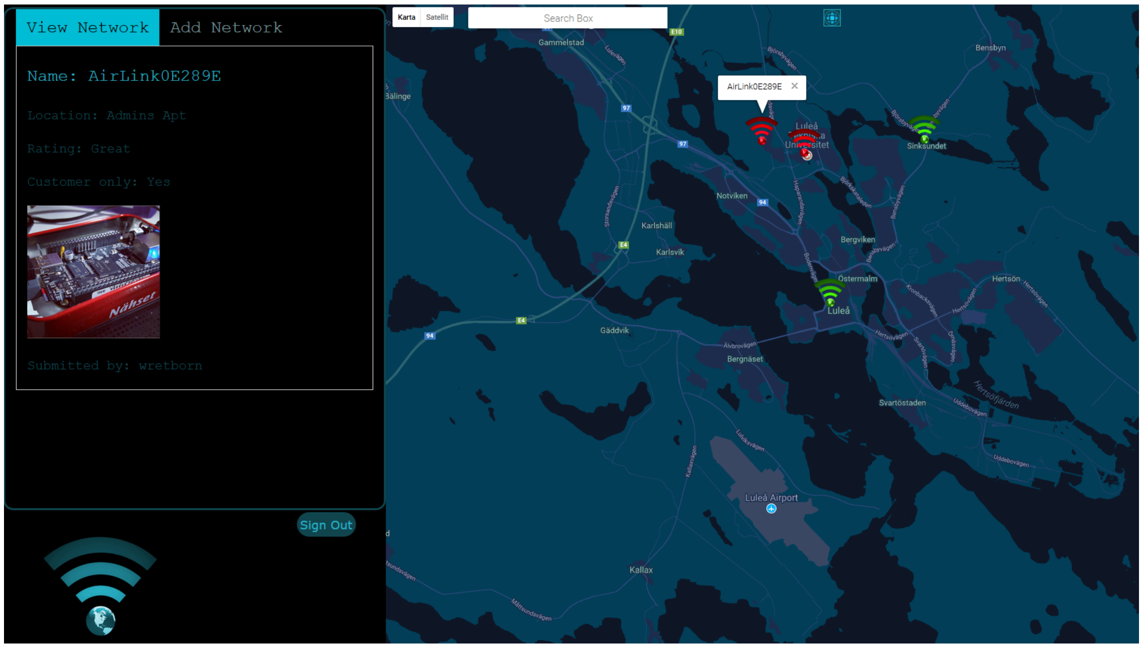
5 Discussion and evaluation **15**
5.1 Future work 15

1 User problem

Imagine being in a foreign city and in need of an internet connection. Before your trip you go to WiFinder and look for free wifi in the vicinity of where you are staying and using your smartphone to screenshot some of the locations in order to find them off-line.

Whenever you're out and about and need to find wifi then WiFinder can help you do that. The application itself can be customized for other needs, such as mapping mineral deposits or making a map of available camp sites in a nature reserve.

The application comes with launch instructions and can be set up by anyone with basic knowledge of web developing.



Name: AirLink0E289E

Location: Admins Apt

Rating: Great

Customer only: Yes

Submitted by: wretborn

2 Design & Architecture

2.1 LAMP Server

The website is hosted on a BeagleBone Black, which is a single-circuit computer similar to the popular Raspberry pi. It runs Jessie Debian 8 OS and is connected to the network via an Ethernet cable and made publicly available through a free DNS service and rerouting.

The BeagleBone is running a LAMP(Linux, Apache2, MySQL and PHP) server. Apache2 is the world most used web server and is responsible for hosting the site. MySQL is a relational database and stores the data server side. PHP is a popular script language for communicating front end to back end and vice versa.

This, together with the database GUI phpMyAdmin and you are left with a set of tools that are powerful enough to create a dynamic, modern website.

2.2 Database & Google maps API

The database consists of two tables, user and markers. The user table contains information about users registered on the site but there is also the option of using a google acount.

The marker table is used by google maps. The google maps API specified the design of the table 'markers' and also the addition of the file 'phpsqlajax_genxml3.php' which generates an XML document of the table 'markers'. The google maps API then uses the xml document to generate the markers on the map.

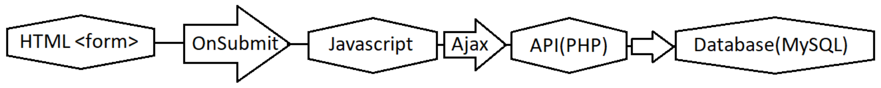
Modifying the database, the 'phpsqlajax_genxml3.php' document and 'map.js' allows the user to add or remove columns in the marker table. There are certain additions made to the google maps API, geolocating, adding markers and the color scheme are all customized.

3 API Design & Usage

3.1 Design

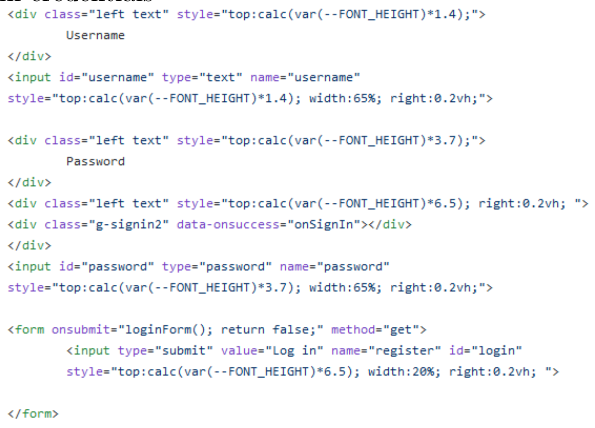
The API is written in PHP, it is a Rest API but it is very bare-bone. The information flow starts at a HTML form, in this form, the user enters information. Some information are in the form, but hidden, such as the latitude and longitude coordinates of the placed marker.

When the user clicks on the submit button, the form sends its data to a Javascript function. This functions formats the input and forms an ajax request. The ajax requests send the input in the form of a json object to the API. The API then forms a query and depending on the form received, either gets data from the database or posts data to the database.



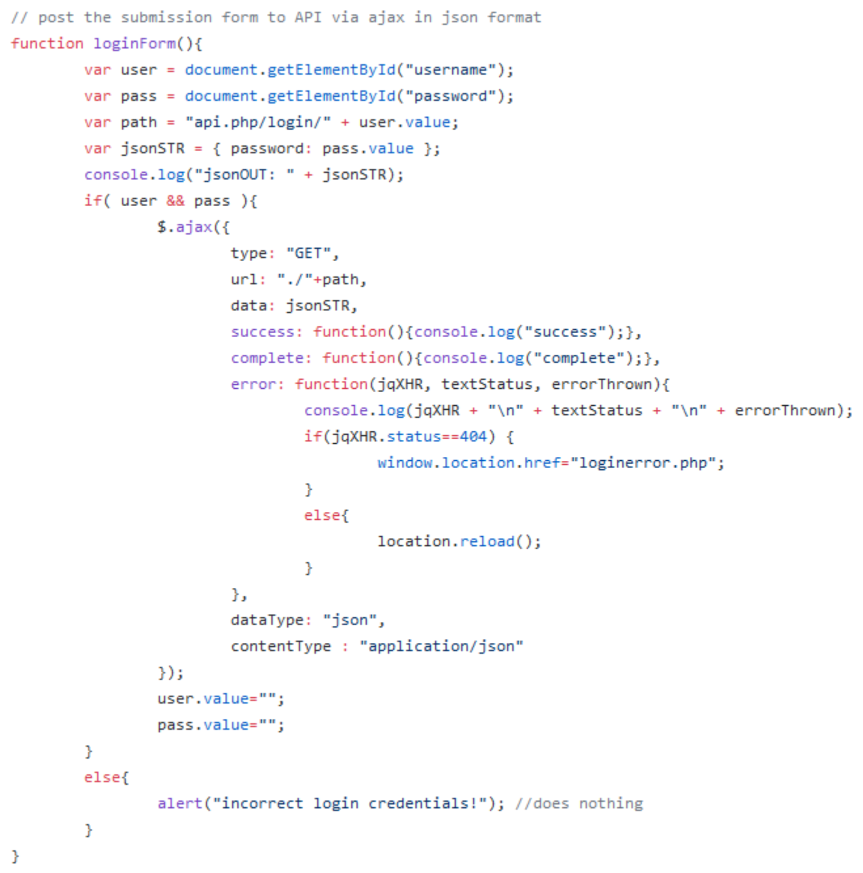
3.2 Usage

In order to explain the usage of the API, we will use an example of the log in form to better understand how queries are created. First we look at at the HTML form where the user inputs log in credentials



```
<div class="left text" style="top:calc(var(--FONT_HEIGHT)*1.4);">
        Username
</div>
<input id="username" type="text" name="username"
style="top:calc(var(--FONT_HEIGHT)*1.4); width:65%; right:0.2vh;">

<div class="left text" style="top:calc(var(--FONT_HEIGHT)*3.7);">
        Password
</div>
<div class="left text" style="top:calc(var(--FONT_HEIGHT)*6.5); right:0.2vh; ">
<div class="g-signin2" data-onsuccess="onSignIn"></div>
</div>
<input id="password" type="password" name="password"
style="top:calc(var(--FONT_HEIGHT)*3.7); width:65%; right:0.2vh;">

<form onsubmit="loginForm(); return false;" method="get">
        <input type="submit" value="Log in" name="register" id="login"
        style="top:calc(var(--FONT_HEIGHT)*6.5); width:20%; right:0.2vh; ">

</form>
```

on submitting, the form sends its data to 'loginForm();'

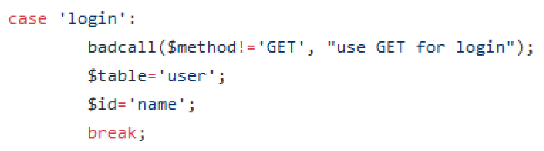


```
// post the submission form to API via ajax in json format
function loginForm(){
	var user = document.getElementById("username");
	var pass = document.getElementById("password");
	var path = "api.php/login/" + user.value;
	var jsonSTR = { password: pass.value };
	console.log("jsonOUT: " + jsonSTR);
	if( user && pass ){
		$.ajax({
			type: "GET",
			url: "./"+path,
			data: jsonSTR,
			success: function(){console.log("success");},
			complete: function(){console.log("complete");},
			error: function(jqXHR, textStatus, errorThrown){
				console.log(jqXHR + "\n" + textStatus + "\n" + errorThrown);
				if(jqXHR.status==404) {
					window.location.href="loginerror.php";
				}
				else{
					location.reload();
				}
			},
			dataType: "json",
			contentType : "application/json"
		});
		user.value="";
		pass.value="";
	}
	else{
		alert("incorrect login credentials!"); //does nothing
	}
}
```

The loginForm creates user name and password variables, it creates a path, in this case to the log in API function and on the end of the path is the name of the user trying to log in.

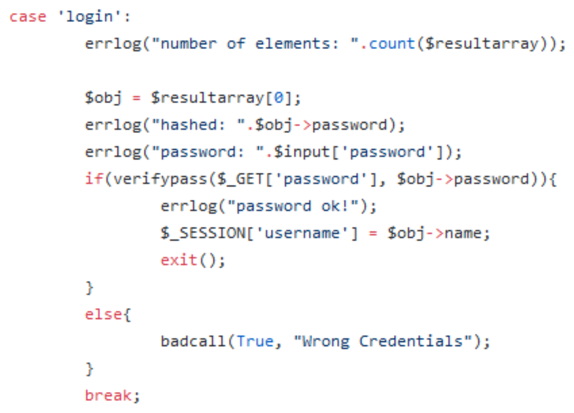
The function then creates a JSON object out of the password. If both user name and password has data entered, the function forms an Ajax method. The method specifies the type, in this case a 'GET', the url which was specified in the path variable and the data package, here a JSON object containing the password.

It checks if the response is 404(A response sent by the API when entering the wrong credentials), if so, the user is relocated to an error message. If successful, the following code is executed:



```
case 'login':
	badcall($method!='GET', "use GET for login");
	$table='user';
	$id='name';
	break;
```

The case 'login' is triggered because of the URL path taken by the ajax method. It checks the call method, which for log in is 'GET', if another method is used, such as 'POST', the statement fails. If not, the table of interest is defined as 'user' and the column of interest as 'name'. The next image shows the API comparing the password of the input with the password stored in the database. If they match, the php session is set and the user is logged in. Upon logging in, the user is given access to the map and all the member functionality.

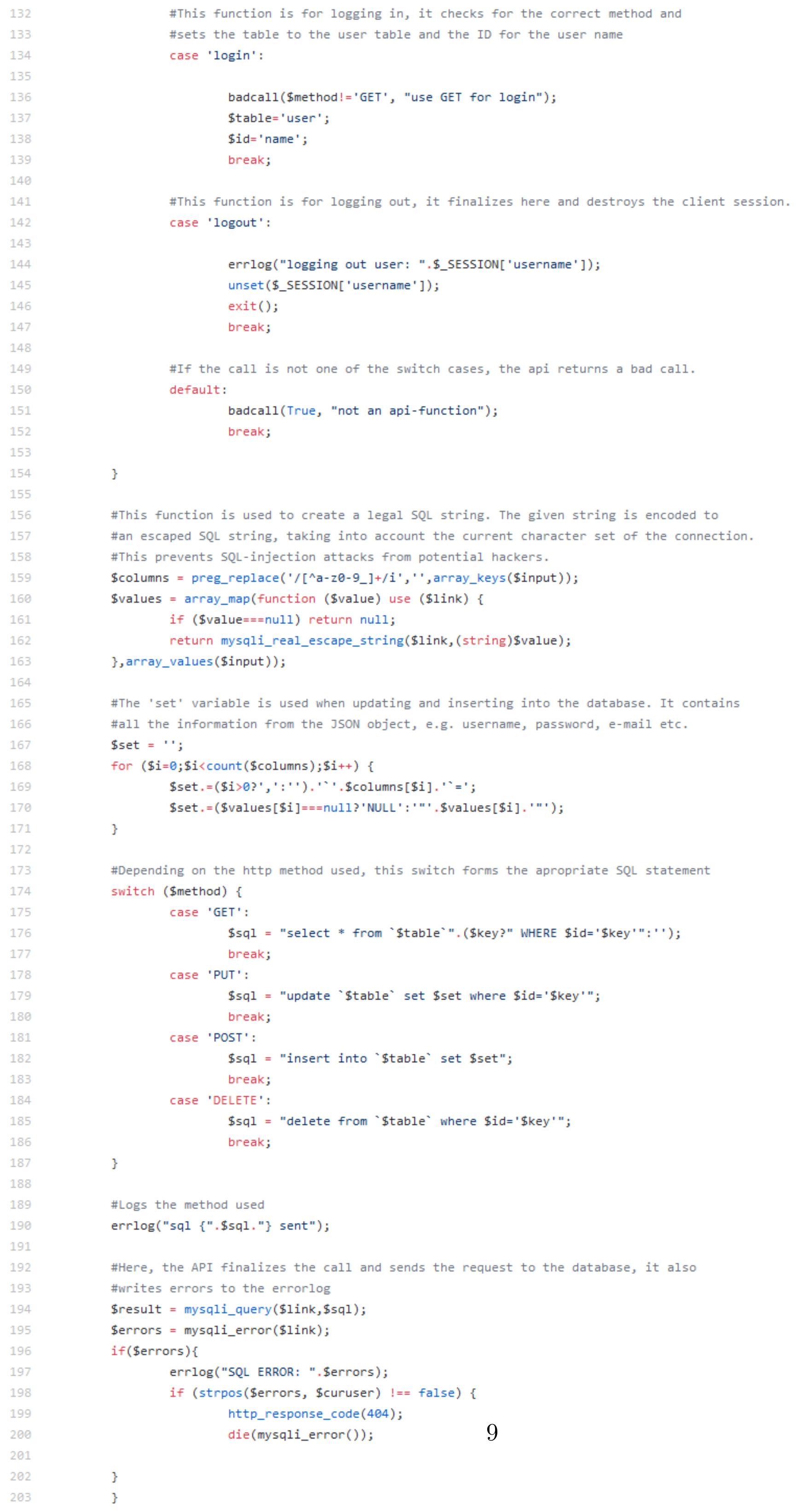


```
case 'login':
        errlog("number of elements: ".count($resultarray));

        $obj = $resultarray[0];
        errlog("hashed: ".$obj->password);
        errlog("password: ".$input['password']);
        if(verifypass($_GET['password'], $obj->password)){
                errlog("password ok!");
                $_SESSION['username'] = $obj->name;
                exit();
        }
        else{
                badcall(True, "Wrong Credentials");
        }
        break;
```

3.3 API Structure

Below is the API in full detail, the comments above the functions describe in detail how the API operates to form its queries.

```php
<?php

        #session_start() creates a session or resumes the current one based on a session identifier
        #passed via a GET or POST request,or passed via a cookie.
        session_start();

?>

<?php

        #Sets the charset to UTF8, this speciefies how characters are to be used within the API.
        #It also starts an error log that saves any errors recorded in a .txt file on the server.
        mysql_set_charset("UTF8");
        ini_set("allow_url_fopen", true);
        ini_set("log_errors", 1);
        ini_set("error_log", "./error.log");

        #Default id to go through database
        $id='id';

        #Function for writing errors to error log
        function errlog($message){
                error_log($message."\n", 3, "./scrap2.log");
        }

        #Badcall function, returns a 404 http response code(not found), if wrong method is used.
        function badcall($isbad, $message){
                if($isbad){
                        errlog($message);
                        http_response_code(404);
                        die($message);
                }
        }

        #Function for hashing(encrypting) the users password
        function hashpass($password){
                return password_hash($password, PASSWORD_DEFAULT);
        }

        #Function for verifying password when signing in
        function verifypass($password, $hashed_password){
                return password_verify($password, $hashed_password);
        }

?>

<?php

        #Writes to the errorlog and includes the database information file
        errlog("STARTLOG");
        errlog("--------------------------------------------------");
        require("phpsqlajax_dbinfo.php");

        #Get the HTTP method(POST, GET, PUT or DELETE)
        $method = $_SERVER['REQUEST_METHOD'];

        #Post information to errorlog
        errlog($_SERVER['PHP_SELF']." received ".$method." request");
        #Sets the path to a variable, e.g. adduser, addwifi etc
        $request = explode('/', trim($_SERVER['PATH_INFO'],'/'));

        #Assigns the JSON object to a php-variable and decodes it into the right format.
        $input = file_get_contents('php://input');
        $input = json_decode($input,true);

        #Creates a conenction to the database using variables found in the phpsqlajax_dbinfo.php file
        $link = mysqli_connect('localhost', $username, $password, $database);
        mysqli_set_charset($link,'utf8');

        #Retrieve table and key from the path remove from variable $request
        $apicall = preg_replace('/[^a-z0-9_]+/i','',array_shift($request));
        $key = array_shift($request);

        #Uses apicall to determine witch function was called.
        switch($apicall){

                #This function is for adding a user. It checks for the correct method, sets the table
                #to user and hashes the password.
                case 'adduser':

                        badcall($method!='POST', "use POST for adduser");
                        $table='user';
                        $curuser = $input['name'];

                        if($input['password']){
                                $input['password'] = hashpass($input['password']);
                        }
                        break;

                #This function is for adding wifi networks. It checks for the correct method and
                #sets the table to markers
                case 'addwifi':

                        badcall($method!='POST', "use POST for addwifi");
                        $table='markers';
                        break;

                #This function is for deleting a wifi network. It sets the table to markers and the id to the picurl.
                #Wifi networks are distinguished by their key, which is a combination of the latitude and longitude coordinates
                #saved in a string. It then sets the method to delete.
                case 'deletewifi':

                        badcall($method!='POST', "use POST for deletewifi");

                        if(($_SESSION['username'] == $input['user']) or ($_SESSION['username'] == 'admin')){
                                $table='markers';
                                $id='picurl';
                                $key= $input['picurl'];
                                $method = 'DELETE';
                                break;
                        }

                        else{
                                badcall(True, "Only your own networks can be removed");
                                break;
                        }

                #This functions adds a picture to a wifi network. It includes the upload.processor.php
                #used to upload files to the server.
                case 'wifipic':

                        errlog("POST picname: ".$_POST['picname']);

                        if(isset($_FILE['file']))
                                errlog("FILE is set!");

                        include('upload.processor.php');
                        exit();
                        break;
```



```php
        #This function is for logging in, it checks for the correct method and
        #sets the table to the user table and the ID for the user name
        case 'login':

            badcall($method!='GET', "use GET for login");
            $table='user';
            $id='name';
            break;

        #This function is for logging out, it finalizes here and destroys the client session.
        case 'logout':

            errlog("logging out user: ".$_SESSION['username']);
            unset($_SESSION['username']);
            exit();
            break;

        #If the call is not one of the switch cases, the api returns a bad call.
        default:
            badcall(True, "not an api-function");
            break;

    }

    #This function is used to create a legal SQL string. The given string is encoded to
    #an escaped SQL string, taking into account the current character set of the connection.
    #This prevents SQL-injection attacks from potential hackers.
    $columns = preg_replace('/[^a-z0-9_]+/i','',array_keys($input));
    $values = array_map(function ($value) use ($link) {
        if ($value===null) return null;
        return mysqli_real_escape_string($link,(string)$value);
    },array_values($input));

    #The 'set' variable is used when updating and inserting into the database. It contains
    #all the information from the JSON object, e.g. username, password, e-mail etc.
    $set = '';
    for ($i=0;$i<count($columns);$i++) {
        $set.=($i>0?',':'').'`'.$columns[$i].'`=';
        $set.=($values[$i]===null?'NULL':'"'.$values[$i].'"');
    }

    #Depending on the http method used, this switch forms the apropriate SQL statement
    switch ($method) {
        case 'GET':
            $sql = "select * from `$table`".($key?" WHERE $id='$key'":'');
            break;
        case 'PUT':
            $sql = "update `$table` set $set where $id='$key'";
            break;
        case 'POST':
            $sql = "insert into `$table` set $set";
            break;
        case 'DELETE':
            $sql = "delete from `$table` where $id='$key'";
            break;
    }

    #Logs the method used
    errlog("sql {".$sql."} sent");

    #Here, the API finalizes the call and sends the request to the database, it also
    #writes errors to the errorlog
    $result = mysqli_query($link,$sql);
    $errors = mysqli_error($link);
    if($errors){
        errlog("SQL ERROR: ".$errors);
        if (strpos($errors, $curuser) !== false) {
            http_response_code(404);
            die(mysqli_error());

    }
    }
```

```php
	#If the SQL statement fails, kills the connection
	if (!$result) {
		errlog("sql no result");
		http_response_code(404);
		die(mysqli_error());
	}

	#This code is for documentation only. If SQL statement is succesful, returns a text with
	#succesful data input. If not, writes it to error log.
	$resultarray = array();
	if ($method == 'GET') {
		if (!$key) echo '[';
		for ($i=0;$i<mysqli_num_rows($result);$i++) {
			$obj = mysqli_fetch_object($result);    // append all results to array
			$resultarray[$i] = $obj;
			echo ($i>0?',':'').json_encode($obj);
		}
		if (!$key)
			echo ']';
	} elseif ($method == 'POST') {
		echo '{ "success":true, "data":[ { "id":'.mysqli_insert_id($link).' }]}';
		errlog('post info: '.'{ "success":true, "data":[ { "id":'.mysqli_insert_id($link).' }]}');
	} else {
		echo mysqli_affected_rows($link);
		errlog("affected rows: ".mysqli_affected_rows($link) );
	}

	#The API is done and closes the link
	mysqli_close($link);

	#This switch case is for changes to backend without the need for a SQL statement.
	#An example of this is the login function, it just sets the session, no need for SQL!
	switch($apicall){
		case 'adduser':

			break;
		case 'addwifi':

			break;

		case 'deletewifi':

			break;
		case 'login':
			errlog("number of elements: ".count($resultarray));

			$obj = $resultarray[0];
			errlog("hashed: ".$obj->password);
			errlog("password: ".$input['password']);
			if(verifypass($_GET['password'], $obj->password)){
				errlog("password ok!");
				$_SESSION['username'] = $obj->name;
				exit();
			}
			else{
				badcall(True, "Wrong Credentials");
			}
			break;
		default:
			badcall(True, "reached end without api-function");
			break;

	}

	#Ends error log segment
	errlog("--------------------------------------------------------");
	errlog("ENDLOG");

?>
```

3.4 API Calls

3.4.1 adduser

Adds a user via a json object

Method
POST

Sample Call
$.ajax({
url: ”api.php/adduser”,
dataType: ”json”,
type : ”POST”,
success : function(r)
});

3.4.2 addwifi

Adds a WiFi via a json object

Method
POST

Sample Call
$.ajax({
url: ”api.php/addwifi”,
dataType: ”json”,
type : ”POST”,
success : function(r)
});

3.4.3 deletewifi

Allows user to delete wifi locations they added. The user named ”Admin” can delete all wifi locations

Method
POST

Sample Call
```
$.ajax({
url: ”api.php/deletewifi”,
dataType: ”json”,
type : ”POST”,
success : function(r)
});
```

3.4.4 login

Logs in a user

Method
GET

Sample Call
```
$.ajax({
url: ”api.php/login”,
dataType: ”json”,
type : ”GET”,
success : function(r)
});
```

3.4.5 logout

Logs out the current user

Method
POST

Sample Call
```
$.ajax({
url: ”api.php/logout”,
dataType: ”json”,
type : ”POST”,
success : function(r)
});
```

3.5 Security

There are two security measures taken when hosting this website. It is protected against SQL-injections and it protects the password of users via hashing.

3.5.1 SQL injections

SQL injections is one of the most popular way to hack a database. It allows the hacker to either destroy the database all together or save sensitive information. This is one of the main reasons why website encourage you to use a unique password, if you use the same password for all your accounts and the website in questions also stores your e-mail, then the hacker has access to all your accounts, on every website.

The way SQL injections work is by feeding input in the form of SQL statements instead of the information that is intended. An example of this being, instead of entering a user name and a password when you create an account, you enter a SQL statement, which in turn can alter the database. WiFinder is protected against these injections through the function mysql_real_escape_string. This partitions the input and protects against hackers inputting SQL statements. Is it a perfect fix? No, there is still the threat of injections from more rigorous hackers. A way to improve the security of the site is to use prepared statements.

3.5.2 Hashing

WiFinder hashes all passwords when they are stored. Upon comparing password(e.g. logging in a user), it compares them in the php script and never openly writes out the password. Hashed passwords have one vulnerability and that is brute force password breaking via a rainbow table. A way to improve security here would be to salt the password, which adds another layer of encryption. If a password is both hashed and salted but decrypted via brute force attempts, it gives no other information on how the remaining passwords in the database are to be decrypted, making the brute force option nonviable.

4 Technologies

There are different technologies being used in the frontend. HTML5 is used for geolocating the current user. This, however, is a functionality currently not working at the website. All popular web browsers require a secure connection in the form of HTTPS in order to use the functionality, however, it is implemented and works if hosted on a secure site.

Javascript is used to generate the map, the dynamic buttons as well as using information from both front end and back end. It's implemented in the graphical part when, for example, hiding HTML buttons depending on the user of the current network being viewed. It's also used when communicating with the API in the form of ajax posts.

The server is hosted on a home network and so the ISP frequently changes the outgoing IP address. To combat this, a 'no-ip' domain was registered and an IP address listener was installed on the BeagleBone. The listener automatically redirects traffic to the 'no-ip' domain, giving the network the functionality of a static addressed IP. The graphical design of the buttons, menus and highlighting was created with CSS

5 Discussion and evaluation

This project has been, for me, the most rewarding, inspiring and creative course I've taken during my studies. I'm quite new to programming, webdesign and network but I feel much more comfortable with it after this course.

The website lacks certain features that I would like to have implemented if I had more time, such as users rating other users networks, a facebook log in and most of all, a much more modern design. It would be interesting for the next project to use completely different architecture and technologies in order to get something to compare e.g mysql with.

The project has required work with all the integral parts of web hosting. Setting up the BeagleBone with the correct software, configuring my router and making sure all parts are communicating as they should have been very educating and interesting.

5.1 Future work

Make the site more adaptable and easier to set up and customize and have a general, clean design so the client can customize it depending on how the client want it to look. I would also like to learn more on how to make it scalable with android and other devices, browsers etc. I would also like to understand the security flaws in the application and learn about why it's vulnerable and how to prevent it.