2023

# *Internet Technologies Semester Project*

A FORUM SYSTEM

HUGO BAPTISTA

# *Table of Contents*

## *Requirements*

For this semester's laboratory projects, we were asked to make a website according to the following requirements:

- 15 different content windows
- Technologies used: HTML, JavaScript and PHP
- An attractive design

## *My approach*

I have thus, chosen to make a forum system. In order to make this project possible I chose the following technology stack:

- Front-end: HTML, SCSS (CSS), JavaScript
- Back-end: PHP, MySQL, JavaScript (Node.JS)

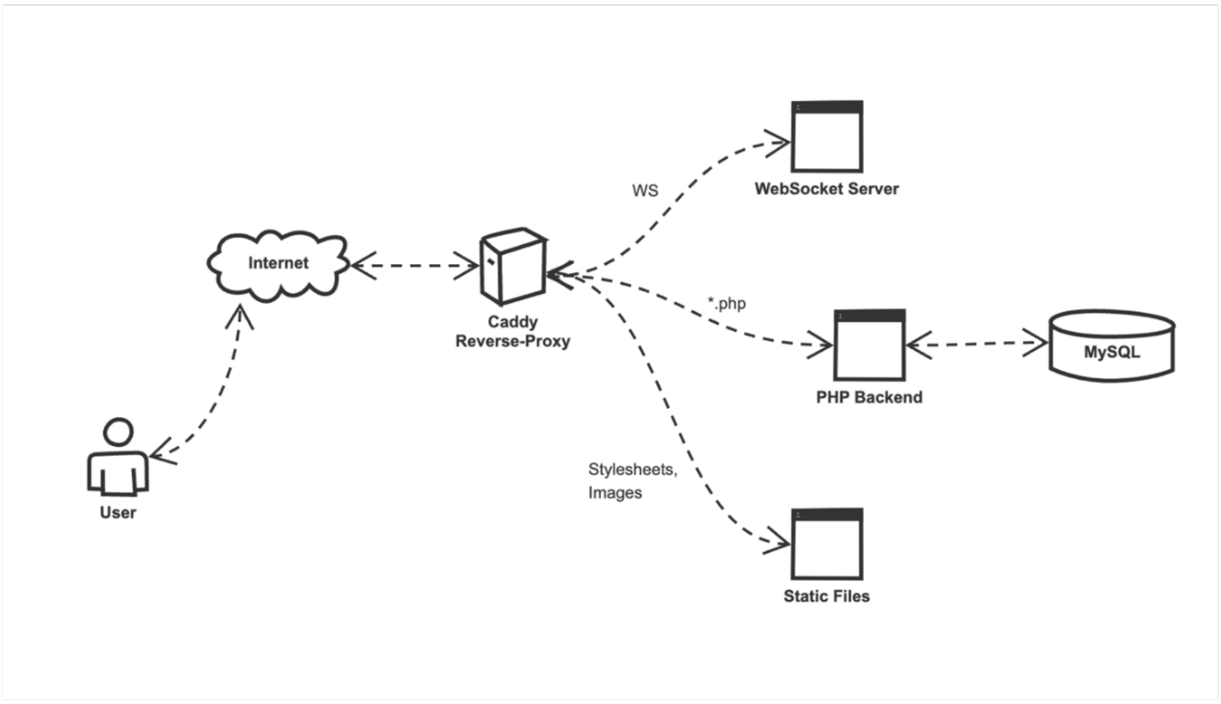
Since the website is in essence a Content Management System, it has an unlimited number of pages, however many different functionalities have been implemented to cover the 15 pages requirements: Homepage, Login, Register, Settings (My account, All posts, All comments, All users, Site settings), Live-Chat, About, Post and Unauthorized modules.

In this report we will first go over the architecture of the project and then we will review some important components of the project. Finally, we will talk about deployment and security issues.

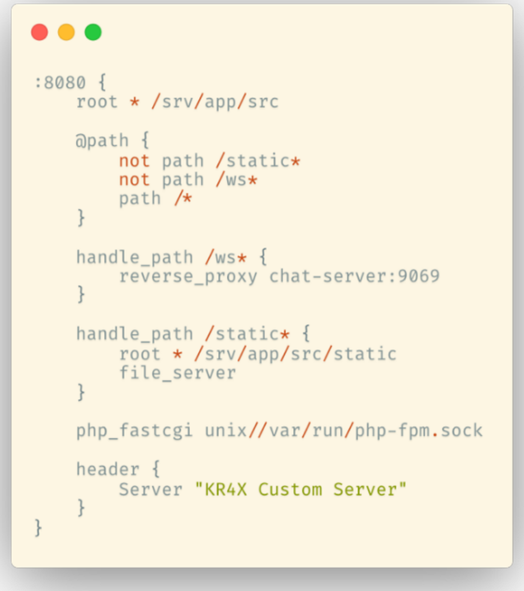
## *Project Architecture*

### *A semi-monolithic application*

In its core, PHP apps are monolithic, however for this project, we also needed a WebSocket server for the live-chat module, which was made using Node.JS. The following diagram shows how all the various services are accessed by a client.



Most apps would use Apache or NGINX, but since I am more experienced using Caddy 2, this is what I used for the reverse proxy. It provides automatic SSL certificate/HTTPS and is very easy to configure (actual configuration below).

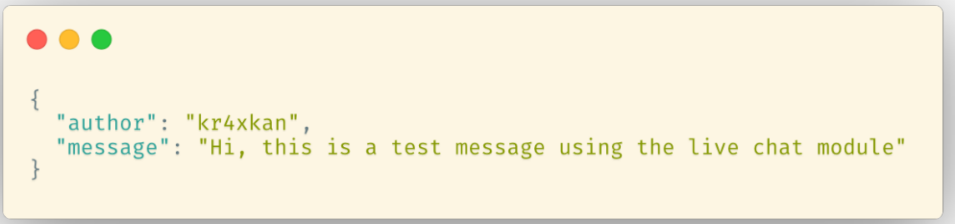


```
:8080 {
    root * /srv/app/src

    @path {
        not path /static*
        not path /ws*
        path /*
    }

    handle_path /ws* {
        reverse_proxy chat-server:9069
    }

    handle_path /static* {
        root * /srv/app/src/static
        file_server
    }

    php_fastcgi unix//var/run/php-fpm.sock

    header {
        Server "KR4X Custom Server"
    }
}
```

## *WebSocket Server*

In order to provide a live-chat module to websites user we need a server to relay the messages that are sent. To comply with the JavaScript requirement, I decided to make the server using Node.JS and the ws library.

One could have made this module possible using only PHP and MySQL, however every single POST and GET requests would lead to an extremely slow and expensive system overall. On the other hand, the WebSocket protocol enables us to communicate with multiple clients at the same time. It is event driven so there is no need for users to constantly make fetch requests for new messages. Below is a sample message sent through the WebSocket pipe.



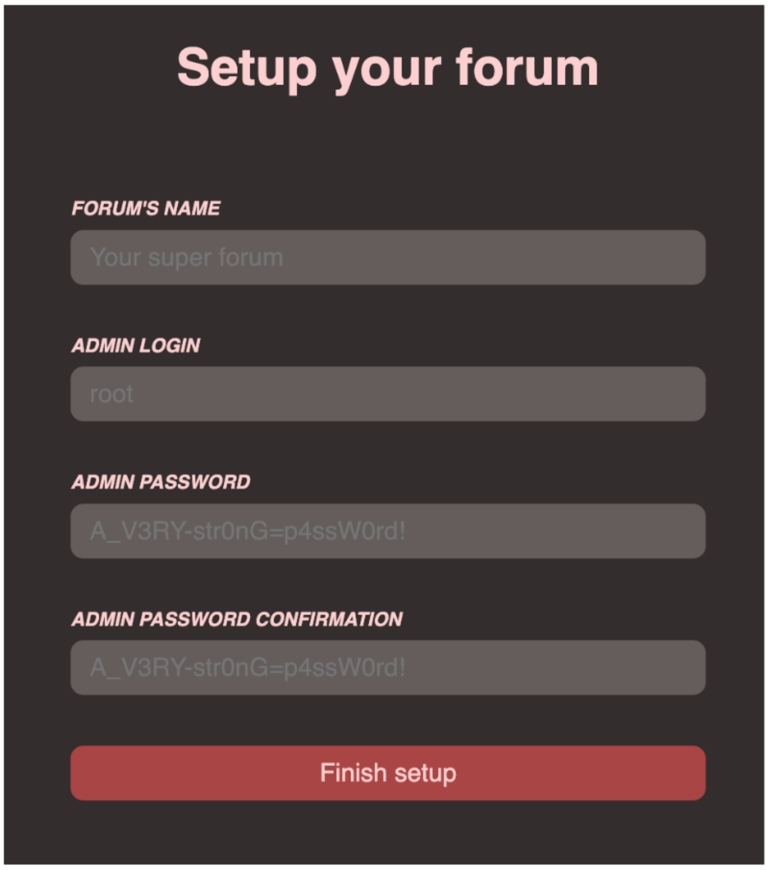
```
{
  "author": "kr4xkan",
  "message": "Hi, this is a test message using the live chat module"
}
```

Messages are sent using JSON, this is not the most efficient form of communication, but the faster to implement for this project. To improve this communication, we could serialize the message manually to save some network bandwidth. Also, we can see that the messages contain the author, this is because connections to the WebSocket server are not authenticated, this poses several security threats which will be discussed in the later part of this report.

## *The most important modules*

### *Setup*

The Forum is not only a simple forum, it’s a system that enables anyone to setup their own forums. There is a quickstart bash script that will build and run the docker containers. On first launch the MySQL database is initialized with all the tables and the only page accessible is <website >/setup.php.



Setup your forum

FORUM'S NAME

Your super forum

ADMIN LOGIN

root

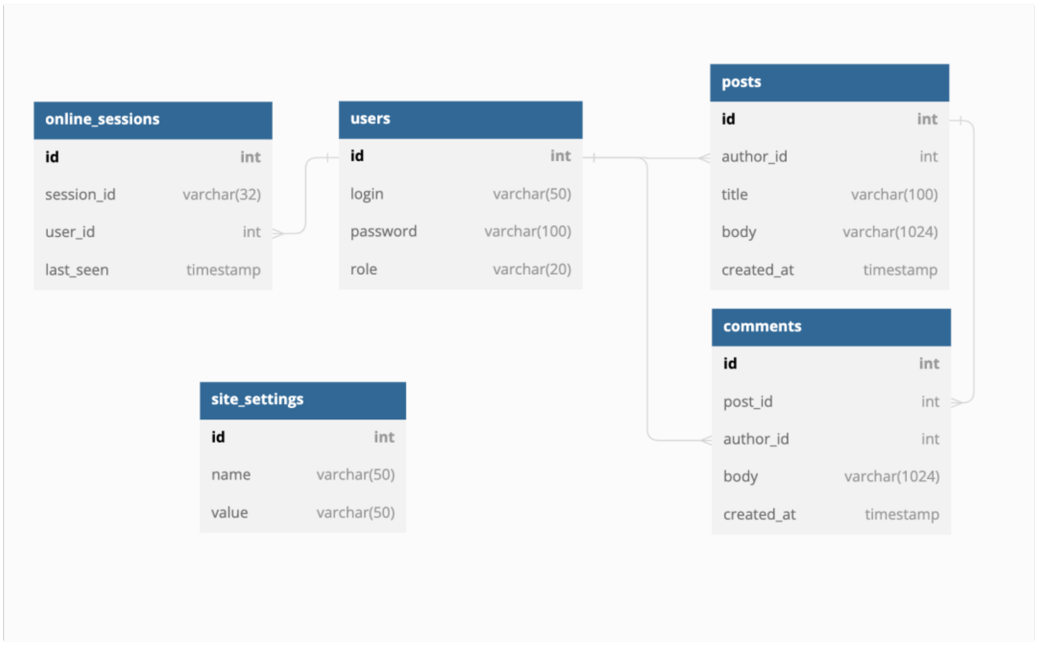
ADMIN PASSWORD

A_V3RY-str0nG=p4ssW0rd!

ADMIN PASSWORD CONFIRMATION

A_V3RY-str0nG=p4ssW0rd!

## *MySQL Tables*



| online_sessions | |
|---|---|
| **id** | **int** |
| session_id | varchar(32) |
| user_id | int |
| last_seen | timestamp |

| users | |
|---|---|
| **id** | **int** |
| login | varchar(50) |
| password | varchar(100) |
| role | varchar(20) |

| posts | |
|---|---|
| **id** | **int** |
| author_id | int |
| title | varchar(100) |
| body | varchar(1024) |
| created_at | timestamp |

| comments | |
|---|---|
| **id** | **int** |
| post_id | int |
| author_id | int |
| body | varchar(1024) |
| created_at | timestamp |

| site_settings | |
|---|---|
| **id** | **int** |
| name | varchar(50) |
| value | varchar(50) |

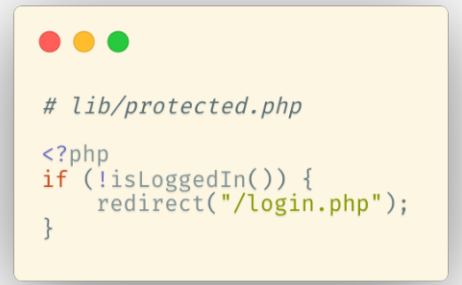
Site Settings is simply a key value pair store for settings such as the website title or if the website has been setup yet.  User role can be one of the following: user, moderator, admin.

Users can create post and comments, Moderators can delete any post and comments and the Administrator can delete users, make them moderator and manage website settings.

Online Sessions is used to find out of many users are currently connected to the website. The session_id is the PHPSESSID of a visitor which can be linked to an authenticated account.

### Protected pages

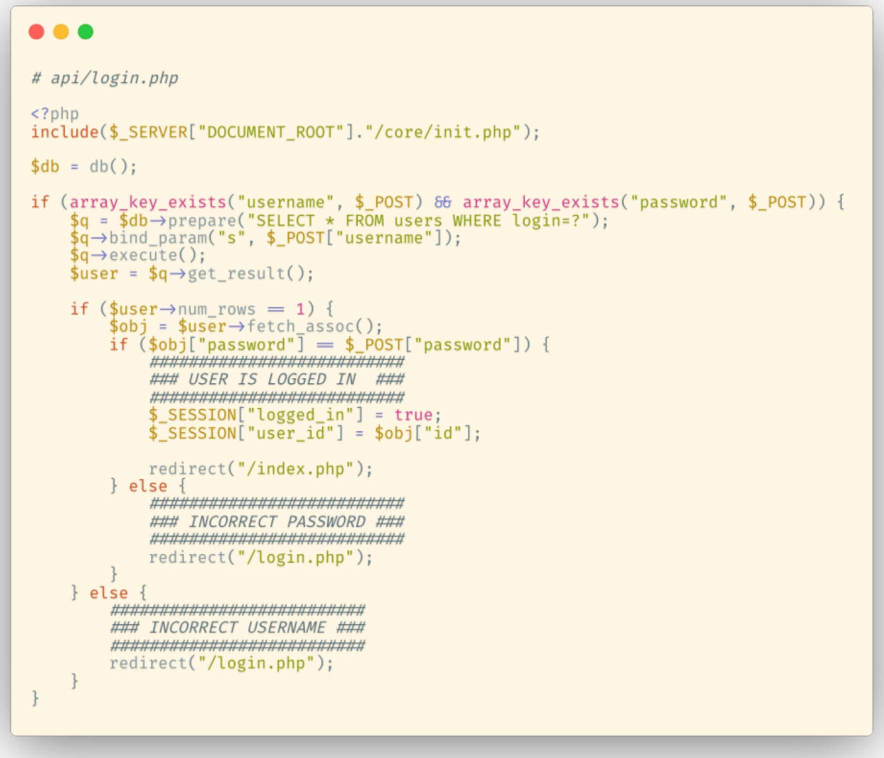
We do not want unauthenticated users to access our private forum. For this purpose, we use ‘lib/protected.php’:



```php
# lib/protected.php

<?php
if (!isLoggedIn()) {
    redirect("/login.php");
}
```

When we want a specific page to be protected (only available to authenticated users) we just have to include this file at the very top of the php file.

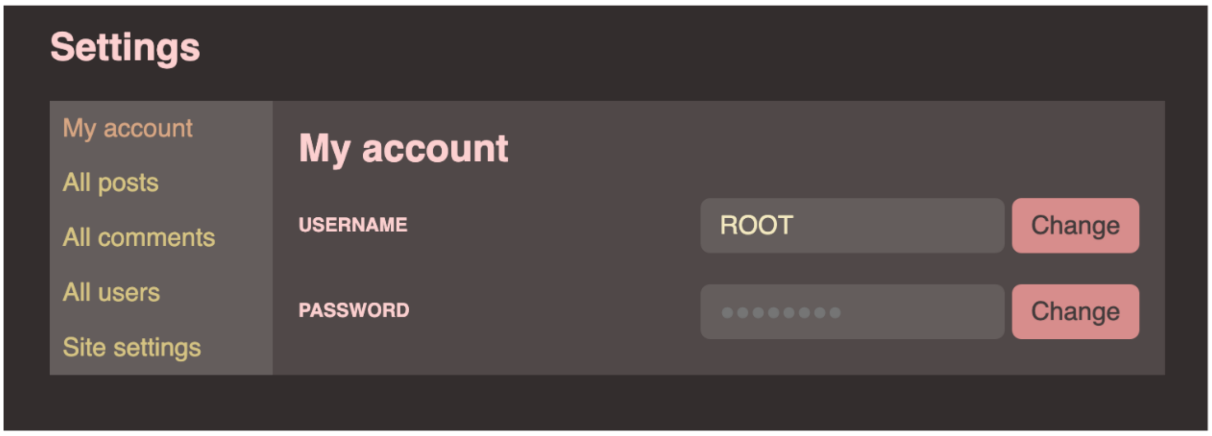
This works because every request has a PHP Session. In this session we can store information such as ‘logged_in’ and ‘user_id’. On login we set the variables accordingly:



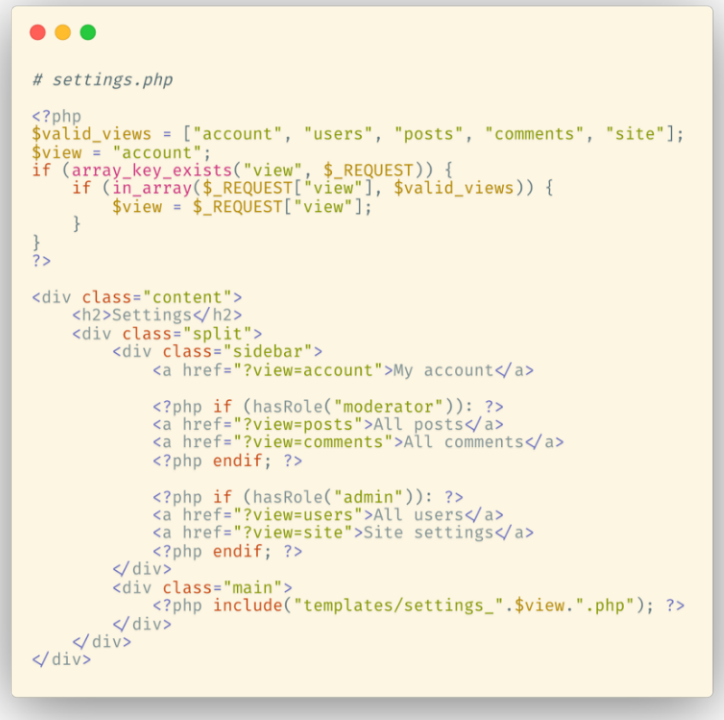
```php
# api/login.php

<?php
include($_SERVER["DOCUMENT_ROOT"]."/core/init.php");

$db = db();

if (array_key_exists("username", $_POST) && array_key_exists("password", $_POST)) {
    $q = $db->prepare("SELECT * FROM users WHERE login=?");
    $q->bind_param("s", $_POST["username"]);
    $q->execute();
    $user = $q->get_result();

    if ($user->num_rows == 1) {
        $obj = $user->fetch_assoc();
        if ($obj["password"] == $_POST["password"]) {
            ##########################
            ### USER IS LOGGED IN  ###
            ##########################
            $_SESSION["logged_in"] = true;
            $_SESSION["user_id"] = $obj["id"];

            redirect("/index.php");
        } else {
            ##########################
            ### INCORRECT PASSWORD ###
            ##########################
            redirect("/login.php");
        }
    } else {
        ##########################
        ### INCORRECT USERNAME ###
        ##########################
        redirect("/login.php");
    }
}
```

## *Settings*

I wanted to have a single page for the settings but with a center control panel. To achieve this result:



In order to make this possible I have used the power of request parameters:



```
# settings.php

<?php
$valid_views = ["account", "users", "posts", "comments", "site"];
$view = "account";
if (array_key_exists("view", $_REQUEST)) {
    if (in_array($_REQUEST["view"], $valid_views)) {
        $view = $_REQUEST["view"];
    }
}
?>

<div class="content">
    <h2>Settings</h2>
    <div class="split">
        <div class="sidebar">
            <a href="?view=account">My account</a>

            <?php if (hasRole("moderator")): ?>
            <a href="?view=posts">All posts</a>
            <a href="?view=comments">All comments</a>
            <?php endif; ?>

            <?php if (hasRole("admin")): ?>
            <a href="?view=users">All users</a>
            <a href="?view=site">Site settings</a>
            <?php endif; ?>
        </div>
        <div class="main">
            <?php include("templates/settings_".$view.".php"); ?>
        </div>
    </div>
</div>
```

We filter incorrect values using a validation filter. Then we use the $view variable to import a file from the templates folder which contains the php files using this filename format: "settings_<view>.php".

This enables us to make dynamic includes while still being secure because we filter bad inputs. All these files contain the inner windows and logic such as handling delete and update requests.

## ***Security concerns***

Writing a complex system from scratch without using any library is very valuable experience. I already had knowledge about SQL Injections, Cross-Site Scripting, LFI and other vulnerabilities which especially impact PHP applications. I believe the website does not contain any SQL Injectable field, but I have not done a full security analysis.

However, I know that the website is easily Cross-Site Scriptable, that means that html elements can be sent by a client and be executed by any visitors of the site. This is a huge security risk, because let's say an attacker could inject malicious JavaScript code or nefarious images on the website. Posts and comments are sanitized but usernames are not.

Concerning the WebSocket server, right now it can be accessed freely without any authentication process. A better way to handle this communication process would be to first authenticate the user in a GET request and then upgrading the connection to the WebSocket protocol only if the authentication was successful. This would not only protect a bit more our server, but this would also prevent other users from impersonating other users on the live chat. Because currently, the author's name is sent by the client.

And finally, passwords are stored using md5 hashing algorithm. A proper authentication algorithm would be to use at the very least BCRYPT with some salting strategy. Because this project was not designed to go into production, I have made the choice to go the easier route using md5, but it is important to note that this is a weak hashing algorithm.

## *Conclusion*

This project gave me some very valuable knowledge and skills on technologies I don't typically use. I have been using Node.JS for years to make my backends and using PHP has sometimes been quite challenging.

The most important part of this whole project for me was to create everything from scratch without any libraries. This gave me much more insight about what happens under the hood. In the future I will be able to use PHP frameworks more deeply because I now understand the abstraction made.

## *Source Code*

The full repository is available on my GitHub (https://github.com/kr4xkan) and has been uploaded to Moodle.

## *Reference*

Since no library have been used, I mostly used the documentations from PHP and Caddy:

- PHP Documentation : https://www.php.net/manual/en/
- Caddy Documentation : https://caddyserver.com/docs/caddyfile

My difficulties with PHP were mostly resolved using the Stackoverflow community. And as for the inspiration of a forum works, I used Xenforo, a CRM/Forum system.

- Xenforo: https://xenforo.com/
- Stackoverflow: https://stackoverflow.com/