Documentation

Project Scope & First Setup

My goal was to make a clean, working subscription system along with proper login, database handling, and logic that makes sense. The application truly has four of the main tables:

• ***Users*** store their name, email, and password hash. It does this, in fact.

• ***subscriptions*** – holds services like "Netflix." It also holds price and duration.

• ***user_subscriptions*** – it links users to services they are using, including start/end dates

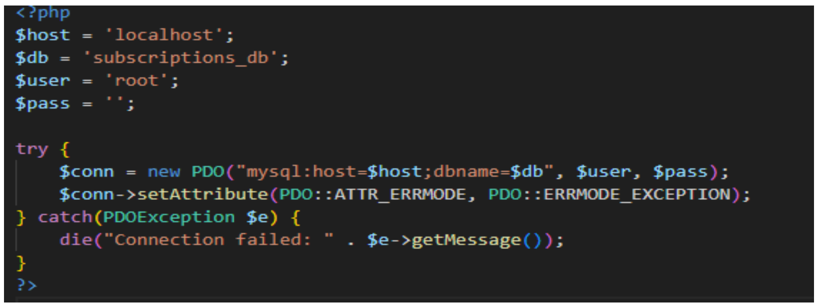
• payments – logs of all renewals from or new subscriptions with a timestamp

I truly started by first getting my environment quite ready—with XAMPP properly running, the project folder duly set up in ***htdocs,*** and also created was the database ***subscriptions_db***. I used ***schema.sql*** to fabricate each of the tables and inserted several dummy data like Spotify and Xbox Game Pass using ***sample_data.sql***.

I ran into several issues at the start. Git failed to work within the VS Code terminal right away, as well as I inadvertently tried pushing from the wrong folder. I had to actually backtrack, fix up the remote origin, and also restart all of the repo setup. But even after that time, affairs were quite smooth.

1. Database Connection Setup

The first actual file I coded was ***db_connect.php***. This file sets up the connection to the MySQL database using PDO. I used try/catch here so if anything fails (like if MySQL isn't running), it gives a clear error message.



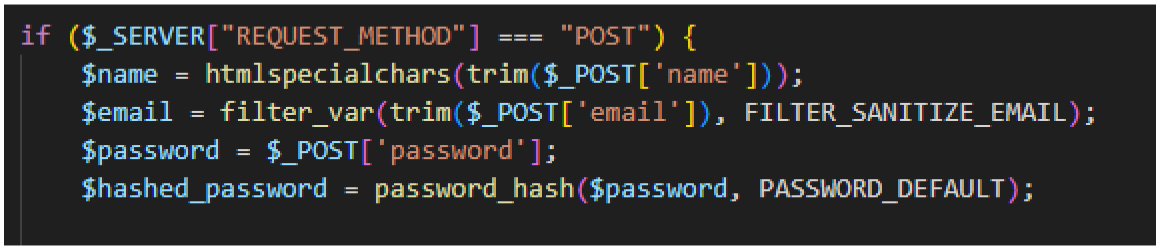
```php
<?php
$host = 'localhost';
$db = 'subscriptions_db';
$user = 'root';
$pass = '';

try {
    $conn = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(PDOException $e) {
    die("Connection failed: " . $e->getMessage());
}
?>
```

I included this in every other file using require ***db_connect.php***, so I never had to rewrite the connection logic.

2. User Registration

After the DB connection, I worked on letting users register. The form sends data to ***register.php***, where I sanitize it, hash the password, and insert the user into the users table.

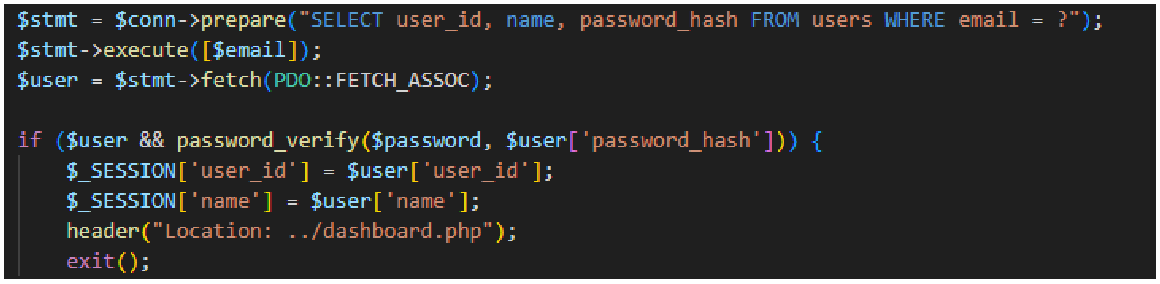


```
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $name = htmlspecialchars(trim($_POST['name']));
    $email = filter_var(trim($_POST['email']), FILTER_SANITIZE_EMAIL);
    $password = $_POST['password'];
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

I tested it by filling out the form and checking the users table in phpMyAdmin. Worked fine after a small bug where I forgot to name one of the inputs in the form.

3. Login & Session Handling

Then I built ***login.php***. It checks if the email exists, and if the password matches using ***password_verify()***. If it works, it start a session and redirect the user to the dashboard.

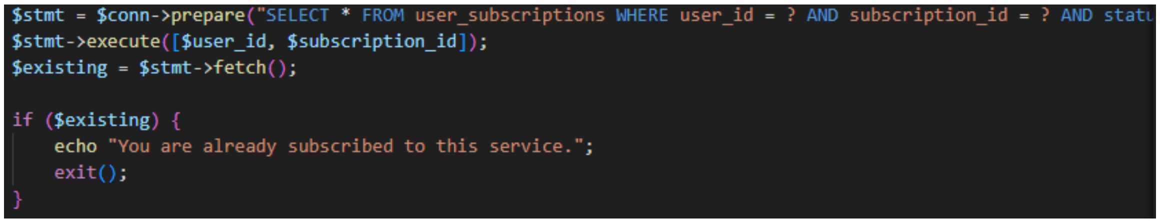


```
$stmt = $conn->prepare("SELECT user_id, name, password_hash FROM users WHERE email = ?");
$stmt->execute([$email]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);

if ($user && password_verify($password, $user['password_hash'])) {
    $_SESSION['user_id'] = $user['user_id'];
    $_SESSION['name'] = $user['name'];
    header("Location: ../dashboard.php");
    exit();
```

4. Viewing Subscriptions on the Dashboard

The dashboard pulls all available subscriptions from the database and shows them with a “Subscribe” button. This is just a simple SELECT query.



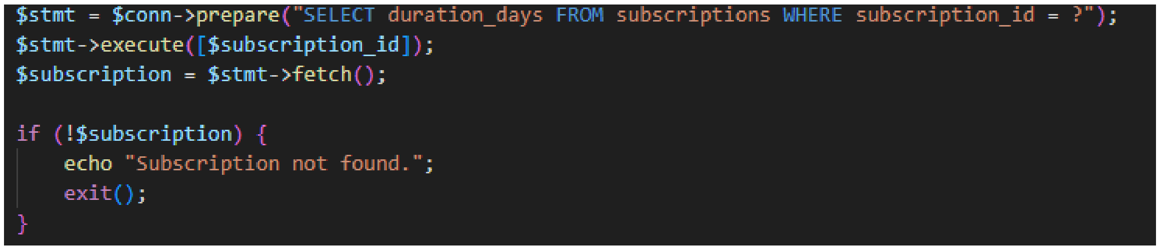
```
$stmt = $conn->prepare("SELECT * FROM user_subscriptions WHERE user_id = ? AND subscription_id = ? AND statu
$stmt->execute([$user_id, $subscription_id]);
$existing = $stmt->fetch();

if ($existing) {
    echo "You are already subscribed to this service.";
    exit();
}
```

Each one shows name, price, and duration. Clicking the button posts the ***subscription_id*** to ***subscribe.php.***

5. Subscribing to a Plan

In ***subscribe.php***, I check if the user already has an active sub to that plan. If not, I calculate the end date and insert it.



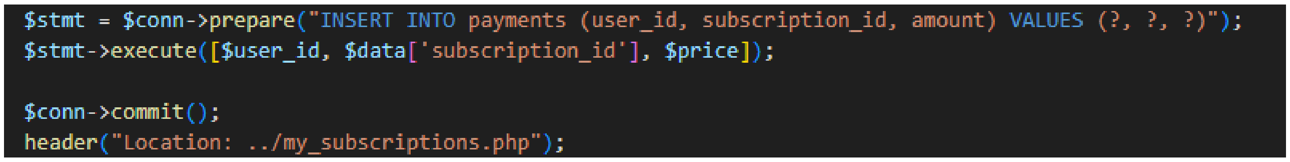
```php
$stmt = $conn->prepare("SELECT duration_days FROM subscriptions WHERE subscription_id = ?");
$stmt->execute([$subscription_id]);
$subscription = $stmt->fetch();

if (!$subscription) {
    echo "Subscription not found.";
    exit();
}
```

At first it didn't work, turned out I forgot to set method="POST" on the form, so the script wasn't receiving anything. Fixed that and it started inserting correctly.

6. Renewals (Using Transactions)

Renewals were next. This is in ***renew.php*** and it uses a transaction so if the update OR the payment log fails, it rolls everything back.



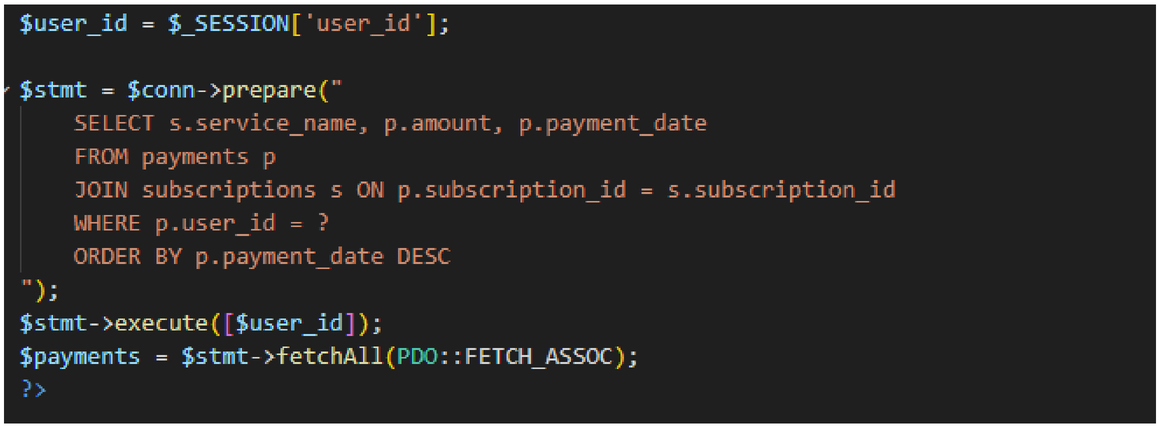
```php
$stmt = $conn->prepare("INSERT INTO payments (user_id, subscription_id, amount) VALUES (?, ?, ?)");
$stmt->execute([$user_id, $data['subscription_id'], $price]);

$conn->commit();
header("Location: ../my_subscriptions.php");
```

I tested it by forcing an error mid-way and confirming nothing got saved if something failed.

7. Payment History with SQL JOINs

Finally, I made ***payments.php*** so users can view their payment history. This uses a JOIN to show service names instead of just subscription IDs.



```php
$user_id = $_SESSION['user_id'];

$stmt = $conn->prepare("
    SELECT s.service_name, p.amount, p.payment_date
    FROM payments p
    JOIN subscriptions s ON p.subscription_id = s.subscription_id
    WHERE p.user_id = ?
    ORDER BY p.payment_date DESC
");
$stmt->execute([$user_id]);
$payments = $stmt->fetchAll(PDO::FETCH_ASSOC);
?>
```

It displays the service, amount, and date for each payment.