Marcus Mallia

PHP Task 3

Technical Document

1. Introduction

Overview of the Project

SpeakeasySounds is a social media platform designed for music enthusiasts to share their musical experiences, discover new music, and connect with other users. The platform allows users to create and share posts, comment on posts, follow other users, and receive notifications about activities related to their account. The aim of SpeakeasySounds is to provide a dedicated space for music lovers to interact, engage, and explore new musical content in a vibrant and dynamic online community.

Purpose of the Document

This technical document outlines the development process of SpeakeasySounds, a PHP and MySQL-based web application. The document provides a detailed explanation of the database setup, data manipulation techniques, local virtual server setup, dynamic web application techniques, and the comprehensive testing strategy used to ensure the functionality and completeness of the project. This document is intended to serve as a guide for developers, detailing the technical aspects and methodologies employed in the creation of SpeakeasySounds.

2. Database Setup

Database Planning

Before diving into the actual coding, I started with the ideation phase, where I brainstormed the key functionalities and features of SpeakeasySounds. This led to the creation of a sitemap and a UML (Unified Modeling Language) diagram. These diagrams helped in visualizing the overall structure of the application and the relationships between different components. This initial planning phase ensured that the database would be well-structured and capable of supporting all the necessary features.

Database Schema

For the database, I chose phpMyAdmin as the relational database management system. The database schema includes several tables designed to store various types of data. Here's a detailed overview of the main tables and their purposes:

- Users Table
    - Purpose: To store user information such as username, email, password, profile picture, bio, and the date of account creation.
    - Key Columns: user_id (Primary Key), username, email, password, profile_picture, bio, created_at.
- Posts Table
    - Purpose: To store user-generated posts, including the title, content, any associated link, and the creation date.
    - Key Columns: post_id (Primary Key), user_id (Foreign Key), title, content, link, created_at.
- Comments Table
    - Purpose: To store comments made on posts by users, including the comment content and the date it was created.
    - Key Columns: comment_id (Primary Key), post_id (Foreign Key), user_id (Foreign Key), content, created_at.

- Tags Table
    - Purpose: To store tags that can be associated with posts for categorization and search purposes.
    - Key Columns: tag_id (Primary Key), name.
- Post_Tags Table
    - Purpose: To create a many-to-many relationship between posts and tags.
    - Key Columns: post_id (Foreign Key), tag_id (Foreign Key).
- Likes Table
    - Purpose: To store information about which users have liked which posts.
    - Key Columns: like_id (Primary Key), user_id (Foreign Key), post_id (Foreign Key).
- Followers Table
    - Purpose: To track followers and followees, showing the relationship between users.
    - Key Columns: follower_id (Primary Key), user_id (Foreign Key), follower_user_id (Foreign Key).
- Notifications Table
    - Purpose: To store notifications for users, including the notification message and its read status.
    - Key Columns: notification_id (Primary Key), user_id (Foreign Key), message, is_read, created_at.
- Activity Status Table
    - Purpose: To track the online status of users and their last activity.
    - Key Columns: activity_id (Primary Key), user_id (Foreign Key), last_activity, online_status.
- Links Table
    - Purpose: To store links shared by users in their posts or profiles.
    - Key Columns: link_id (Primary Key), user_id (Foreign Key), post_id (Foreign Key), url, description.

Design Considerations

When designing the database schema, I ensured that relationships between tables were clearly defined using foreign keys. This ensures data integrity and enforces the connections between different data types. For instance:

- A post must be linked to a user, ensuring that every post has an author.
- A comment must be linked to both a user and a post, ensuring that comments are associated with the correct post and user.
- The many-to-many relationship between posts and tags is managed by the post_tags table, allowing each post to have multiple tags and each tag to be associated with multiple posts.

Data Relationships

To handle the various relationships:

- One-to-Many Relationships: These are used where one entity can be associated with multiple other entities. For example, one user can have multiple posts and comments.
- Many-to-Many Relationships: These are used where multiple entities can be related to multiple other entities. For example, a post can have multiple tags, and a tag can be associated with multiple posts.

Overall, starting with the sitemap and UML diagrams provided a clear roadmap for setting up the database, ensuring that it was well-organized and capable of supporting the features and functionalities of SpeakeasySounds

3. Techniques for Data Manipulation

In building SpeakeasySounds, several techniques were employed to manipulate data within the database. This involved a combination of SQL queries and PHP scripts to handle CRUD (Create, Read, Update, Delete) operations. Here's a detailed explanation of the techniques used:

Data Insertion (Create)

- Creating new records in the database is a fundamental operation. For SpeakeasySounds, data insertion is done through forms that users interact with. For example:
- User Registration: When a new user signs up, their information is collected via a registration form. The data is then inserted into the Users table.
  php
- Creating Posts: When a user creates a new post, the post details are inserted into the Posts table.
  php

Data Retrieval (Read)

- Reading data from the database involves querying tables to fetch the required information. This is used extensively throughout SpeakeasySounds to display user profiles, posts, comments, etc.
- Fetching User Information: User profiles are displayed by retrieving user data from the Users table.
  php
- Fetching Posts: The feed and individual post pages display posts by querying the Posts table.
  php

Data Update

- Updating records in the database allows for modifications to existing data. This is crucial for features like editing profiles, updating posts, and marking notifications as read.
- Updating User Profiles: Users can update their profiles, which involves updating records in the Users table.
  php
- Editing Posts: Users can edit their posts, which updates the Posts table.
  php

Data Deletion

- Deleting records is used to remove unwanted data from the database. This includes deleting posts, comments, and user accounts.
- Deleting Posts: Users can delete their posts, which removes records from the Posts table.
  php
- Deleting User Accounts: Users can delete their accounts, which involves removing their data from multiple tables.
  php

Using AJAX for Asynchronous Updates

- To enhance the user experience, AJAX (Asynchronous JavaScript and XML) is used to update parts of the web page without reloading the entire page. This is particularly useful for actions like following/unfollowing users and liking posts.
- Follow/Unfollow Users: The follow button triggers an AJAX request that updates the Followers table without refreshing the page.javascript
- Liking Posts: Similar to following, liking a post sends an AJAX request to update the Likes table.
  javascript

These techniques collectively ensure that data is efficiently managed and manipulated within the SpeakeasySounds application, providing a seamless and responsive user experience.

4. Setting Up a Local Virtual Server

Setting up a local virtual server is a crucial step in developing a web application like SpeakeasySounds. It allows you to test your application in an environment that closely mimics a live server, ensuring that your code works correctly before deploying it to a production server. Here's a detailed explanation of how I set up a local virtual server for SpeakeasySounds:

For this project, I chose XAMPP as my local development environment.

Installation Process

1. Download XAMPP:
    - I downloaded the latest version of XAMPP from the official website: XAMPP. As my previous version wasn't working well.
2. Install XAMPP:
    - The installation process is straightforward, involving a series of prompts where I selected the components I needed, mainly Apache and MySQL.

Configuring the Server

1. Start Apache and MySQL:
    - After installing XAMPP, I opened the XAMPP Control Panel and started the Apache and MySQL services. This sets up the local server and database server.
2. Configuring Apache:
    - The default settings for Apache worked well for my development needs. However, if needed, I could configure settings like the port number or document root by editing the httpd.conf file located in the apache/conf directory.

3. Configuring MySQL:
    - Similarly, the default MySQL settings were sufficient. Configuration changes, if needed, could be made in the my.ini file located in the mysql/bin directory.

Setting Up the Database

1. Access phpMyAdmin:
    - XAMPP includes phpMyAdmin, a web-based interface for managing MySQL databases. I accessed phpMyAdmin by navigating to http://localhost/phpmyadmin in my web browser.
2. Create the Database:
    - In phpMyAdmin, I created a new database for SpeakeasySounds. I named the database speakeasysounds.
3. Import the Database Schema:
    - I used the SQL file (speakeasysounds.sql) that contains the database schema and initial data. In phpMyAdmin, I selected the speakeasysounds database and used the import feature to upload and execute the SQL file.

Configuring the Application

1. Database Configuration:
    - In my application, I created a config.php file that contains the database connection settings. This file is included in all scripts that need to interact with the database.
2. Setting Up the Project Directory:
    - I placed my project files in the htdocs directory within the XAMPP installation folder. This allows Apache to serve my application when I navigate to http://localhost/MarcusMallia-PHPsynoptic in my web browser.

Testing the Local Server

1. Running the Application:
   - With the server and database configured, I opened my web browser and navigated to http://localhost/MarcusMallia-PHPsynoptic to ensure the application was running correctly.

By setting up a local virtual server using XAMPP, I was able to develop and test SpeakeasySounds in an environment similar to a live server. This setup ensured that any issues could be identified and resolved before deploying the application to a production server.

5. Techniques Used to Build a Dynamic Web Application

Building SpeakeasySounds involved various techniques to create a dynamic and interactive web application. Below are the key techniques and methods used:

Server-Side Scripting with PHP

PHP was the primary language used for server-side scripting. It handles requests, interacts with the database, processes forms, and dynamically generates HTML content.

Form Handling and Validation:

PHP scripts were used to handle form submissions for login, signup, post creation, comments, and other interactions. For example, in the login.php file, PHP validates user input, checks credentials against the database, and starts a session for authenticated users.

Session Management:

Sessions in PHP were used to manage user authentication and maintain user state across different pages. For instance, when a user logs in, their session data is stored and checked on subsequent pages to ensure they are logged in.

CRUD Operations:

PHP scripts performed Create, Read, Update, and Delete (CRUD) operations on the database.
For instance, create_post.php allows users to add new posts, update_post.php allows users to edit existing posts, and delete_post.php handles post deletions.

Client-Side Scripting with JavaScript

JavaScript enhanced the interactivity of the application, making it more responsive and user-friendly.

Form Validation:

JavaScript was used for client-side form validation to provide immediate feedback to users. The validation.js file contains scripts to check input fields before submission.

AJAX for Dynamic Content:

AJAX (Asynchronous JavaScript and XML) was used to update parts of the web page without reloading the entire page.
For instance, the follow.js script uses AJAX to handle follow/unfollow actions dynamically, updating the button state without a full page reload.

Interactive Elements:

JavaScript added interactivity, such as displaying error messages, handling button clicks, and toggling visibility of elements.

Database Interactions

Interaction with the MySQL database was a crucial part of building SpeakeasySounds.

Prepared Statements:

To enhance security and prevent SQL injection, prepared statements were used for database queries. This was particularly important for handling user input.php

Relational Data Handling:

The database schema was designed with proper relationships using foreign keys. This ensured data integrity and consistency.
For example, the Comments table references both Posts and Users tables, maintaining the relationship between comments, posts, and users.

Dynamic Content Generation

The application dynamically generates content based on user actions and database queries.

Template Inclusion:

PHP includes templates for consistent header, footer, and other common elements across pages. This modular approach simplifies maintenance and updates.php

User-Specific Content:

Pages like the user profile and feed dynamically display content based on the logged-in user. SQL queries fetch user-specific data, which is then rendered in the HTML.
For example, the profile.php script fetches posts, follower counts, and other details for the logged-in user.

Responsive Design with CSS

CSS ensured that SpeakeasySounds was visually appealing and usable on various devices.

Flexbox and Grid Layouts:

Flexbox and Grid CSS properties were used to create responsive layouts that adapt to different screen sizes.
For instance, the sidebar and main content area were designed using flexbox to ensure they adjust properly on smaller screens.

Styling:

Consistent styling across the application was achieved using a central style.css file. This included colors, fonts, button styles, and other UI elements.

Security Considerations

Input Sanitization:

User inputs were sanitized to prevent malicious data from being processed. Functions like htmlspecialchars were used to escape special characters in user input.php

Passwords were hashed using PHP's password_hash function before storing them in the database. During login, password_verify was used to check hashed passwords.

CSRF Protection:

Cross-Site Request Forgery (CSRF) tokens were implemented in forms to ensure that form submissions came from legitimate sources.

By combining these techniques, SpeakeasySounds was built to be a dynamic, secure, and user-friendly web application.

6. Test Cases

Below is a detailed IPO (Input-Process-Output) chart for the key functionalities of SpeakeasySounds. Each test case includes the test action, expected output, actual output, pass/fail status, and any relevant notes.

Test Case	Input (Test Action)	Process	Expected Output	Actual Output	Pass/Fail	Notes
Login (Valid Credentials)	Enter valid email and password	Verify credentials in the database	Redirect to index.php	Redirect to index.php	Pass	Ensure the user is registered and activated
Login (Invalid Credentials)	Enter invalid email and/or password	Verify credentials in the database	Display "Invalid credentials" message	Display "Invalid credentials" message	Pass	Test for both incorrect email and password
Signup (Valid Data)	Enter valid username, email, password	Insert new user into the database	Redirect to login.php with success message	Redirect to login.php with success message	Pass	Ensure email is unique and password meets criteria

Signup (Invalid Data)	Enter invalid or incomplete data	Attempt to insert new user into the database	Display error message for each invalid input	Display error message for each invalid input	Pass	Test for missing fields, invalid email format, and weak password
Create Post (Valid Data)	Enter valid title, content, optional link and tags	Insert new post into the database	Display new post on user profile and feed	Display new post on user profile and feed	Pass	Ensure user is logged in
Create Post (Invalid Data)	Enter invalid or incomplete data	Attempt to insert new post into the database	Display error message for each invalid input	Display error message for each invalid input	Pass	Test for missing title or content
Edit Post (Valid Data)	Modify title, content, optional link and tags	Update post in the database	Display updated post on user profile and feed	Display updated post on user profile and feed	Pass	Ensure post belongs to the logged-in user
Edit Post (Invalid Data)	Enter invalid or incomplete data	Attempt to update post in the database	Display error message for each invalid input	Display error message for each invalid input	Pass	Test for empty title or content
Delete Post	Click delete button for a post	Remove post from the database	Post is removed from user profile and feed	Post is removed from user profile and feed	Pass	Ensure post belongs to the logged-in user

Follow User	Click follow button on a user profile	Insert follow relationship into the database	Update button to "Unfollow" and increase follower count	Update button to "Unfollow" and increase follower count	Pass	Ensure user is logged in and not already following
Unfollow User	Click unfollow button on a user profile	Remove follow relationship from the database	Update button to "Follow" and decrease follower count	Update button to "Follow" and decrease follower count	Pass	Ensure user is logged in and currently following
Like Post	Click like button on a post	Insert like into the database	Update button to "Unlike" and increase like count	Update button to "Unlike" and increase like count	Pass	Ensure user is logged in and has not liked the post yet
Unlike Post	Click unlike button on a post	Remove like from the database	Update button to "Like" and decrease like count	Update button to "Like" and decrease like count	Pass	Ensure user is logged in and has already liked the post
Comment on Post (Valid Data)	Enter valid comment content	Insert comment into the database	Display new comment below the post	Display new comment below the post	Pass	Ensure user is logged in
Comment on Post (Invalid Data)	Enter invalid or empty comment	Attempt to insert comment into the database	Display error message for invalid input	Display error message for invalid input	Pass	Test for empty comment content

Edit Comment (Valid Data)	Modify comment content	Update comment in the database	Display updated comment below the post	Display updated comment below the post	Pass	Ensure comment belongs to the logged-in user
Edit Comment (Invalid Data)	Enter invalid or empty comment	Attempt to update comment in the database	Display error message for invalid input	Display error message for invalid input	Pass	Test for empty comment content
Delete Comment	Click delete button for a comment	Remove comment from the database	Comment is removed from the post	Comment is removed from the post	Pass	Ensure comment belongs to the logged-in user
Update Profile (Valid Data)	Enter valid username, email, password, profile picture, bio	Update user details in the database	Display updated profile details	Display updated profile details	Pass	Ensure email is unique and password meets criteria
Update Profile (Invalid Data)	Enter invalid or incomplete data	Attempt to update user details in the database	Display error message for each invalid input	Display error message for each invalid input	Pass	Test for invalid email format and weak password

Delete User Account	Confirm account deletion	Remove user and all related data from the database	User account is deleted and redirected to login page	User account is deleted and redirected to login page	Pass	Ensure all related data (posts, comments, likes, followers) are removed

7. Conclusion

Developing SpeakeasySounds has been an enriching experience, blending several key aspects of web development and database management. This project has allowed me to practically apply various concepts learned during my studies, such as relational database design, CRUD operations, session management, and dynamic web application development.

Building SpeakeasySounds has been a significant milestone in my journey as a web developer. It has provided a solid foundation in both backend and frontend development, equipping me with the skills and knowledge needed to tackle more complex projects in the future. The project's success is a testament to the importance of careful planning, continuous learning, and perseverance in the face of challenges.