Technical Document

1. Database Setup

The web application's database, named php_reverserecords, was established using MySQL. The setup process involved the following steps:

- Designing the Schema: Following the UML diagram from Task 1, the database schema included essential tables like users and products. Each table was carefully structured with fields such as UserID, Username, Password for users, and corresponding fields for other tables like products.
- Creating the Database: The database was created using phpMyAdmin, a user-friendly interface for managing MySQL databases.
- Defining Tables: Tables were defined as per the schema, with appropriate fields, data types, and constraints.
- Setting Constraints: Constraints such as primary keys for uniqueness and foreign keys for relationships were set up according to the UML diagram.

2. Data Manipulation Techniques

Data manipulation within the php_reverserecords database was conducted via PHP scripts, facilitating interaction with MySQL. The approach employed can be detailed as follows:

- CRUD Operations: The application incorporated Create, Read, Update, and Delete operations. These operations were fundamental in managing data within the database, such as adding new users or products, retrieving product details for display, updating user information, and deleting products from the catalogue.
- Usage of Prepared Statements: To enhance security and prevent SQL injection, the application employed prepared statements in PHP scripts. This practice was particularly evident in scripts like Account_Registration.php for user registration, Product_Management.php for product operations, and login.php for user authentication. By preparing SQL statements on the server and then binding parameters, the application minimised the risk of SQL injection.

- Data Sanitization and Validation: Prior to database insertion or updating, input data underwent sanitization and validation processes. This was crucial to ensure data integrity and security. For instance, user input from forms in Account_Registration.php and Product_Management.php was sanitised to escape potentially harmful characters and validated to adhere to the expected format and data type.

3. Virtual Server Setup for Local Development

Setting up a virtual server locally was a crucial step in the development of the web application. This allowed me to simulate a live server environment on my own device, enabling efficient development and testing. Here's an overview of how I set up the virtual server:

-
- Selection of XAMPP: For the local server setup, I chose XAMPP, which is a popular software package that includes Apache (the web server) and MySQL (the database server), along with PHP support. XAMPP is user-friendly and ideal for PHP application development.
- Installing XAMPP: The installation process was straightforward:
    - Downloaded XAMPP from the official website.
    - Ran the installer and selected Apache and MySQL components, as these were essential for my web application.
    - Followed the on-screen instructions to complete the installation.
- Starting Server Services:
    - I used the XAMPP Control Panel, an easy-to-use interface, to manage the server services.
    - Started the Apache service for the web server functionality and MySQL for the database management.
    - Ensured both services were running without issues, as indicated by the control panel.
- Testing the Setup:
    - To test the web server, I accessed http://localhost in a web browser. This directed the browser to my local server, and a successful loading of the XAMPP dashboard confirmed that Apache was functioning correctly.
    - For the database, I accessed phpMyAdmin, a web-based management tool included with XAMPP, by navigating to http://localhost/phpmyadmin. This allowed me to manage the MySQL database directly from the browser.

- Developing and Testing:
    - With the server running, I could develop and test the web application in real-time.
    - PHP scripts could be executed, and interactions with the MySQL database could be tested locally, closely simulating a live server environment.

4. Building a Dynamic Web Application

The dynamic nature of the web application was primarily achieved through the integration of PHP with MySQL and effective use of session management. Here's an overview of these aspects:

- PHP and MySQL Integration:
    - PHP scripts were used for server-side logic, directly interacting with the MySQL database for tasks such as retrieving, inserting, updating, and deleting data. This was fundamental for features like user registration and product management.
    - The interaction between PHP and MySQL meant that changes in the database, like new user sign-ups or product updates, were immediately reflected in the application.
- Session Management:
    - PHP sessions played a crucial role in maintaining user states across the application. This was essential for user authentication, allowing the application to recognize and track logged-in users as they navigated through different pages.
    - Sessions were used to store user information securely, enabling a personalised experience for each user.
- Dynamic Content Rendering:
    - The application dynamically generated HTML content based on the data from the database. This approach ensured that the content displayed to the user, such as product lists, was always up-to-date with the latest database information.

5. IPO Chart

Feature	Input	Process	Expected Output	Result
Account Registration	User's name, email, desired password.	Check if email already exists, hash password, create new user account.	User account creation confirmation, User ID.	Pass
Login	User's email and password.	Verify email and compare hashed password with database.	Authentication status, session initiation for logged-in user.	Pass
Edit Account Details	User's updated name, email, possibly password, and User ID for verification.	Authenticate User ID, update account details in database.	Account update confirmation.	Fail
Create New Product	Product name, description, price, and stock quantity details from staff.	Validate input details, insert new product record into database.	New product creation confirmation, Product ID.	Pass
Edit Product	Existing Product ID, updated product details.	Authenticate staff user, validate changes, update product record in database.	Product update confirmation.	Fail
‘Delete’ Product	Product ID to be deleted.	Authenticate staff user, set product record to inactive or remove from database.	Product deletion confirmation.	Fail
Create Order	User ID, selected product IDs with quantities,	Validate products against inventory, calculate total	Order confirmation, Order ID.	Fail

	shipping details.	cost, create new order record.		
Edit Order	Order ID, changes to product quantities or shipping details.	Authenticate user or staff, validate changes, update order record.	Order update confirmation.	Fail
Cancel Order	Order ID.	Authenticate user, set order status to 'Cancelled' in database.	Order cancellation confirmation.	Fail
Add Product to Wishlist	User ID, Product ID.	Check if product already in wishlist, add product to wishlist in database.	Wishlist update confirmation.	Fail
Remove Product from Wishlist	User ID, Product ID.	Authenticate user, remove product from wishlist in database.	Wishlist update confirmation.	Fail
Add Product to Shopping Cart	User ID, Product ID, and quantity.	Validate quantity against inventory, add to or update shopping cart in database.	Shopping cart update confirmation.	Fail
Modify Product Quantity in Shopping Cart	User ID, Product ID, new quantity.	Validate new quantity, update product quantity in shopping cart.	Shopping cart update confirmation.	Fail
Remove Product from Shopping Cart	User ID, Product ID.	Authenticate user, remove product from shopping cart in database.	Shopping cart update confirmation.	Fail
Create Review	User ID, Product ID, review text, rating score.	Validate input, create new review record linked to product.	Review submission confirmation.	