Project Report: User Management System

Project Report: User Management System

1. Introduction

The User Management System is a web-based application developed using PHP and MySQL, designed to demonstrate the implementation of the Model-View-Controller (MVC) architecture, middleware for access control, user authentication, and a RESTful API for CRUD operations. The system enables users to register, log in, access a protected dashboard, and manage user data through a secure API. This report outlines the project's objectives, structure, implementation details, and security considerations.

2. Project Objectives

The primary objectives of the User Management System are to:

- Implement a modular application using the MVC pattern for clear separation of concerns.
- Restrict access to protected routes using middleware.
- Provide secure user authentication with registration, login, and logout functionality.
- Develop a RESTful API to perform Create, Read, Update, and Delete (CRUD) operations on user data.
- Ensure robust security through input validation, password hashing, and prepared database queries.

3. System Architecture

The system is built using the MVC architectural pattern, which organizes the codebase into three components:

- **Model**: Manages database interactions and business logic.
- **View**: Renders the user interface using HTML and PHP templates.
- **Controller**: Handles user requests, coordinates between the Model and View, and manages application flow.

The application uses PHP 7.4+ with PDO for database connectivity and MySQL as the database. Middleware enforces access control, and a RESTful API provides programmatic access to user data.

4. Project Structure

The project is organized into a clear directory structure to enhance maintainability and scalability. Below is the file structure with descriptions of each file's purpose:

File Structure

```
user-management/
├── api/
│   └── index.php
├── config/
│   └── database.php
├── controllers/
│   ├── UserController.php
│   └── DashboardController.php
├── middleware/
│   └── AuthMiddleware.php
├── models/
│   └── User.php
├── views/
│   ├── register.php
│   ├── login.php
│   ├── dashboard.php
│   └── 404.php
├── public/
│   └── index.php
└── .htacess
```

File Descriptions

1. `.htacess`:
   - **Location**: Root directory
   - **Purpose**: Configures Apache to route all requests through `public/index.php` for clean URLs and handles API routing.
   - **Key Functionality**: Rewrites URLs to enable MVC routing and directs API requests to `api/index.php`.
2. `public/index.php`:
   - **Location**: `public/`
   - **Purpose**: Serves as the application's entry point, handling routing for web requests.
   - **Key Functionality**: Parses the URL, loads the appropriate controller and method, and renders views or displays a 404 error for invalid routes.
3. `config/database.php`:
   - **Location**: `config/`
   - **Purpose**: Defines the database connection using PDO.
   - **Key Functionality**: Establishes a secure connection to the MySQL database (`user_management`) and handles connection errors.
4. `models/User.php`:
   - **Location**: `models/`
   - **Purpose**: Represents the User model, encapsulating database operations for user management.
   - **Key Functionality**: Includes methods for creating, reading, updating, and deleting users, using prepared statements to prevent SQL injection.
5. `controllers/UserController.php`:
   - **Location**: `controllers/`
   - **Purpose**: Manages user-related actions, including registration, login, and logout.
   - **Key Functionality**: Processes form submissions, validates inputs, interacts with the User model, and redirects users based on authentication status.
6. `controllers/DashboardController.php`:
   - **Location**: `controllers/`
   - **Purpose**: Handles requests for the protected dashboard page.
   - **Key Functionality**: Applies middleware to ensure only authenticated users can access the dashboard and renders the dashboard view.
7. `middleware/AuthMiddleware.php`:
   - **Location**: `middleware/`
   - **Purpose**: Implements access control for protected routes.
   - **Key Functionality**: Checks for an active user session and redirects unauthenticated users to the login page.
8. `views/register.php`:
   - **Location**: `views/`
   - **Purpose**: Provides the user interface for user registration.
   - **Key Functionality**: Displays a form for users to input their name, email, and password, with error messaging for failed registrations.
9. `views/login.php`:
   - **Location**: `views/`
   - **Purpose**: Provides the user interface for user login.
   - **Key Functionality**: Displays a form for users to input their email and password, with error messaging for invalid credentials.
10. `views/dashboard.php`:
    - **Location**: `views/`
    - **Purpose**: Displays a protected dashboard for authenticated users.
    - **Key Functionality**: Shows a welcome message and a logout link.
11. `views/404.php`:
    - **Location**: `views/`
    - **Purpose**: Displays an error page for invalid routes.
    - **Key Functionality**: Informs users that the requested page does not exist and provides a link to the login page.
12. `api/index.php`:
    - **Location**: `api/`
    - **Purpose**: Handles RESTful API requests for user management.
    - **Key Functionality**: Processes GET, POST, PUT, and DELETE requests to perform CRUD operations, returning JSON responses.

5. Implementation Details

5.1 Database Design

The system uses a single MySQL table, `users`, with the following schema:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **id**: Unique identifier for each user.
- **name**: User's full name.
- **email**: Unique email address for login.
- **password**: Hashed password for secure authentication.
- **created_at**: Timestamp of account creation.

5.2 MVC Implementation

- **Model** (`User.php`):
  - Connects to the database using PDO.
  - Provides methods for CRUD operations: `createUser()`, `getUserByEmail()`, `getAllUsers()`, `updateUser()`, and `deleteUser()`.
  - Uses prepared statements to ensure security.
- **View** (`register.php`, `login.php`, `dashboard.php`, `404.php`):
  - Renders HTML forms and pages for user interaction.
  - Displays error messages for invalid inputs or failed actions.
- **Controller** (`UserController.php`, `DashboardController.php`):
  - Handles HTTP requests and coordinates between the Model and View.
  - Validates inputs using `filter_input()` and hashes passwords using `password_hash()`.

5.3 Middleware

- Implemented in `AuthMiddleware.php`.
- Checks for an active session (`$_SESSION['user_id']`).
- Redirects unauthenticated users to the login page, ensuring the dashboard is accessible only to logged-in users.

5.4 Authentication

- **Registration**: Users submit their name, email, and password via `register.php`. The password is hashed using `password_hash()` and stored in the database.
- **Login**: Users enter their email and password via `login.php`. The system verifies credentials using `password_verify()` and starts a session for authenticated users.
- **Logout**: The `logout` method destroys the session and redirects to the login page.

5.5 RESTful API

- **Endpoints**:
  - `GET /api/users`: Retrieves all users.
  - `POST /api/users`: Creates a new user.
  - `PUT /api/users/{id}`: Updates a user's details.
  - `DELETE /api/users/{id}`: Deletes a user.
- **Implementation**:
  - Handled by `api/index.php`.
  - Processes requests based on HTTP methods (GET, POST, PUT, DELETE).
  - Returns JSON responses with appropriate HTTP status codes (e.g., 400 for invalid input, 405 for unsupported methods).
- **Security**: Input validation ensures only valid data is processed.

6. Security Considerations

The system incorporates several security measures:

- **Input Validation**: Uses `filter_input()` to sanitize form inputs and prevent malicious data.
- **Password Hashing**: Employs `password_hash()` with BCRYPT for secure password storage.
- **SQL Injection Prevention**: Uses PDO prepared statements to safely handle database queries.
- **Session Security**: Regenerates session IDs after login to prevent session fixation attacks.
- **API Security**: Validates API inputs to ensure data integrity.

7. Testing and Validation

The system was tested to ensure all requirements were met:

- **Authentication**: Registration, login, and logout functionality work as expected, with proper redirects and error handling.
- **Middleware**: Unauthenticated users are redirected to the login page when accessing the dashboard.
- **API**: All endpoints (`GET`, `POST`, `PUT`, `DELETE`) return correct JSON responses and handle errors appropriately.
- **Error Handling**: Invalid routes display the 404 page, and form errors are displayed to users.

8. Conclusion

The User Management System successfully demonstrates the application of MVC architecture, middleware, authentication, and a RESTful API using PHP and MySQL. The modular design ensures maintainability, while security measures protect user data and system integrity. The project meets all specified requirements and provides a foundation for further enhancements, such as improved UI styling, API authentication, or additional features like password recovery.

9. Recommendations for Future Work

- Enhance the user interface with CSS or a front-end framework (e.g., Bootstrap).
- Implement API authentication using API keys or JWT.
- Add password strength validation during registration.
- Introduce role-based access control to support different user types (e.g., admin, user).

Summary of File Structure and Descriptions

Below is a concise summary of the project's file structure and file descriptions, as requested:

File Structure

```
user-management/
├── api/
│   └── index.php
├── config/
│   └── database.php
├── controllers/
│   ├── UserController.php
│   └── DashboardController.php
├── middleware/
│   └── AuthMiddleware.php
├── models/
│   └── User.php
├── views/
│   ├── register.php
│   ├── login.php
│   ├── dashboard.php
│   └── 404.php
├── public/
│   └── index.php
└── .htacess
```

File Descriptions

File Path	Description
`.htacess`	Configures URL rewriting to route requests through `public/index.php` and `api/index.php`.
`public/index.php`	Application entry point; handles routing for web requests and loads controllers.
`config/database.php`	Defines PDO-based database connection to the `user_management` database.
`models/User.php`	Manages user-related database operations (CRUD) using PDO prepared statements.
`controllers/UserController.php`	Handles registration, login, and logout; processes form inputs and interacts with the User model.
`controllers/DashboardController.php`	Manages dashboard access, applying middleware to restrict to authenticated users.
`middleware/AuthMiddleware.php`	Enforces access control by checking session status and redirecting unauthenticated users.
`views/register.php`	HTML form for user registration, displaying errors if registration fails.
`views/login.php`	HTML form for user login, displaying errors for invalid credentials.
`views/dashboard.php`	Protected dashboard page for authenticated users, with a logout link.
`views/404.php`	Error page displayed for invalid routes, with a link to the login page.
`api/index.php`	Handles RESTful API requests for user CRUD operations, returning JSON responses.