12.5 Class 12 Reinforcement Exercises

Exercise 12-1: Fixing The Chess Board

I messed up. In the chess game we've been building, I had you put the white pieces at the top in ranks 7 and 8, and the black pieces at the bottom in ranks 1 and 2. That's backwards! In chess, white pieces always start at the bottom (ranks 1 and 2) and black pieces at the top (ranks 7 and 8). Because of this, the queens and kings are also swapped - the white queen should be on a light square, and the black queen on a dark square but that's not how it is right now. Let's fix this mistake.

1. Copy `public/exercises/9-chess/` to a new folder named `public/exercises/12-chess/` in your local repository.
2. Open `public/exercises/12-chess/play.php` and change "white" at the top to "black" and "black" at the bottom to "white".
3. Open the `ChessBoard` class in `public/exercises/12-chess/src/Domain/ChessBoard.php` and update the initial piece placement in the `setupBoard()` method so that white pieces are on ranks 1 and 2, and black pieces are on ranks 7 and 8 - note that the board is built from top to bottom so elements 6 and 7 represent ranks 2 and 1 respectively while elements 0 and 1 represent ranks 8 and 7 respectively.
4. Open the `Pawn` class in `public/exercises/12-chess/src/Domain/Pawn.php` and update the `isValidMove()` method so that white pawns move "up" the board (decreasing rank number) and black pawns move "down" the board (increasing rank number). This will involve swapping the second and third operands in the ternary operator that sets the `$direction` variable as well as the one that sets the `$startRank` variable.
5. Test your changes by running `public/exercises/12-chess/index.php` and start a new game. Verify that the pieces are in the correct starting positions and that pawns move in the correct direction.

Exercise 12-2: Adding CSRF Protection

In this exercise, you'll add CSRF protection to your chess game by implementing CSRF tokens in the move submission form.

1. Open `public/exercises/12-chess/play.php` and start a session at the top of the file by adding `session_start();`.
2. Generate a CSRF token if one doesn't already exist in the session. You can do this by adding the following code after starting the session:

```php
if (empty($_SESSION['csrfToken'])) {
    $_SESSION['csrfToken'] = bin2hex(random_bytes(32));
}
$csrfToken = $_SESSION['csrfToken'];
```

3. Add a hidden input field to the move submission form that includes the CSRF token. Locate the form in `play.php` and add the following line inside the `<form>` tags:

```
<input type="hidden" name="csrfToken" value="<?= htmlspecialchars($csrfToken) ?>">
```

4. Before processing the move submission, verify that the CSRF token from the form matches the one stored in the session. You can do this by adding the following code before handling the move logic:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['start'],
$_POST['end'])) {
    if (!isset($_POST['csrfToken']) || !hash_equals($_SESSION['csrfToken'],
$_POST['csrfToken'])) {
        die('CSRF token validation failed.');
    }
    $chessBoard->movePiece($_POST['start'], $_POST['end']);
}
```

5. Test your changes by running `public/exercises/12-chess/index.php`, starting a new game, and making moves. Verify that moves are processed correctly and that if you tamper with the CSRF token in the form (e.g., by changing its value in the browser's developer tools), the submission is rejected with a CSRF token validation error.
6. (Optional) For added security, consider regenerating the CSRF token after each successful move submission to prevent reuse. You can do this by adding the following line after a successful move:

```
$_SESSION['csrfToken'] = bin2hex(random_bytes(32));
```

Exercise 12-3: Adding Authentication

In this exercise, you'll add basic authentication to your chess game by implementing a login system.

1. Create a new file `public/exercises/12-chess/login.php` and add the following PHP code:

```
<?php
session_start();

require __DIR__ . '/autoload.php';
use App\Infra\{PlayerRepository, GameRepository};

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username1 = $_POST['username1'];
    $password1 = $_POST['password1'];
    $username2 = $_POST['username2'];
    $password2 = $_POST['password2'];

    $playerRepo = new PlayerRepository();

    $player1 = $playerRepo->findByUsername($username1);
    $player2 = $playerRepo->findByUsername($username2);
    if ($player1 && password_verify($password1, $player1->passwordHash) &&
        $player2 && password_verify($password2, $player2->passwordHash)) {
        $_SESSION['player1'] = $player1->id;
        $_SESSION['player2'] = $player2->id;
        header('Location: index.php');
        exit;
    } else {
        $error = 'Invalid username or password.';
    }
}
?>

<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <?php if (isset($error)): ?>
        <p><?= $error ?></p>
    <?php endif; ?>
    <form method="post">
        <label for="username1">Player 1 Username:</label>
        <input type="text" name="username1" id="username1" required><br><br>

        <label for="password1">Player 1 Password:</label>
        <input type="password" name="password1" id="password1" required><br><br>

        <label for="username2">Player 2 Username:</label>
        <input type="text" name="username2" id="username2" required><br><br>

        <label for="password2">Player 2 Password:</label>
        <input type="password" name="password2" id="password2" required><br><br>

        <input type="submit" value="Login">
    </form>
</body>
</html>
```

2. Modify `public/exercises/12-chess/index.php` to check for authenticated users before allowing access to the game. Add the following code at the top of the file:

```
session_start();
if (!isset($_SESSION['player1']) || !isset($_SESSION['player2'])) {
    header('Location: login.php');
    exit;
}
```

3. Add a new method `findByUsername` to the `PlayerRepository` class in `public/exercises/12-chess/src/Infra/PlayerRepository.php` to retrieve a player by their username. The method should be based on the existing `findById` method.
4. Add a `passwordHash` property to the `Player` class and update your queries in `PlayerRepository.php` to load the `password_hash` column into the `passwordHash` property.
5. Add a `create` method to `PlayerRepository` to allow creating new players with hashed passwords. You can use `password_hash()` to hash the passwords before storing them in the database. For example:

```
public function create(string $username, string $password, string $email, string
$fullName): Player {
    $hash = password_hash($password, PASSWORD_DEFAULT);
    $stmt = Database::get()->prepare('INSERT INTO players (username,
password_hash, email, full_name) VALUES (:username, :password_hash, :email,
:full_name)');
    $stmt->execute([':username' => $username, ':password_hash' => $hash, ':email'
=> $email, ':full_name' => $fullName]);
    return $this->findById(Database::get()->lastInsertId());
}
```

6. Test your changes by running `public/exercises/12-chess/index.php`. You should be redirected to the login page if you are not authenticated. After logging in with valid credentials, you should be able to access the chess game.
7. Implement a logout feature by creating a `logout.php` file that destroys the session and redirects to the login page. Add the following code to `logout.php`:

```
<?php
session_start();
session_unset();
session_destroy();
header('Location: login.php');
exit;
```

8. Add a logout links to `index.php`, `start.php`, and `play.php` that points to `logout.php` so users can log out of the game.

12-4: Add Registration and Upgrade Password Hashing

On your own, create a registration page named `register.php` that allows new players to sign up with a username, password, email, and full name. Verify that no user with the same username or email address already exists and that the password meets basic security requirements that you set. Use the `PlayerRepository::create()` method to store the new player in the database with the hashed password. Additionally, implement logic to upgrade existing password hashes when users log in, if their current hash is using an older algorithm. You'll do this in `login.php` after verifying the password but before setting the session. Use `password_needs_rehash()` to check if the hash needs to be updated, and if so, generate a new hash with `password_hash()` and update the database record accordingly using the password that they entered and that you have already verified (the user doesn't need to be notified - it's irrelevant to them and they've already given you what you need to complete the task). You may want to implement a new method in the `PlayerRepository` class to handle the password hash upgrade. Test your registration and login functionality thoroughly.

12-5: Adding Authorization

In this exercise, you'll add basic authorization to your chess game by ensuring that games are only accessible to the players involved.

1. Open `public/exercises/12-chess/src/Infra/GameRepository.php` and add a new method `findOngoingGamesByPlayers` that retrieves a list of ongoing games involving the two logged in players. For example:

```
public function findOngoingGamesByPlayers(int $player1Id, int $player2Id): array {
    $stmt = Database::get()->prepare('SELECT * FROM games WHERE (white_player_id = ? OR black_player_id = ?) AND (white_player_id = ? OR black_player_id = ?) AND result = "ongoing"');
    $stmt->execute([$player1Id, $player1Id, $player2Id, $player2Id]);
    return $stmt->fetchAll();
}
```

2. Modify `public/exercises/12-chess/index.php` to use the above method in place of fetching all ongoing games. Ensure that only games involving the logged-in players are retrieved by passing in the player IDs from the `$_SESSION` superglobal.
3. Modify `public/exercises/12-chess/play.php` to check that the game being accessed belongs to the logged-in players. After loading `$gameMeta` and confirming that the game exists, add the following check:

```
if (($gameMeta['white_player_id'] !== $_SESSION['player1'] && $gameMeta['black_player_id'] !== $_SESSION['player1']) ||
    ($gameMeta['white_player_id'] !== $_SESSION['player2'] && $gameMeta['black_player_id'] !== $_SESSION['player2'])) {
    die('Unauthorized access to this game.');
}
```

4. Modify `public/exercises/12-chess/start.php` so that instead of allowing any two players to start a game, it only allows the logged-in players to start a game against each other. You'll need to ensure that the session is started at the top. Then use the player IDs from the `$_SESSION` superglobal and simply ask which is white and which is black (displaying the player usernames as before). You can use radio buttons, a dropdown, or another method for this selection.
5. Test your changes by running `public/exercises/12-chess/index.php`. Try starting a new game and also verify that only ongoing games involving the logged-in players are displayed. Also try attempting to access a game not involving the logged-in players and make sure it results in the "Unauthorized access" message.