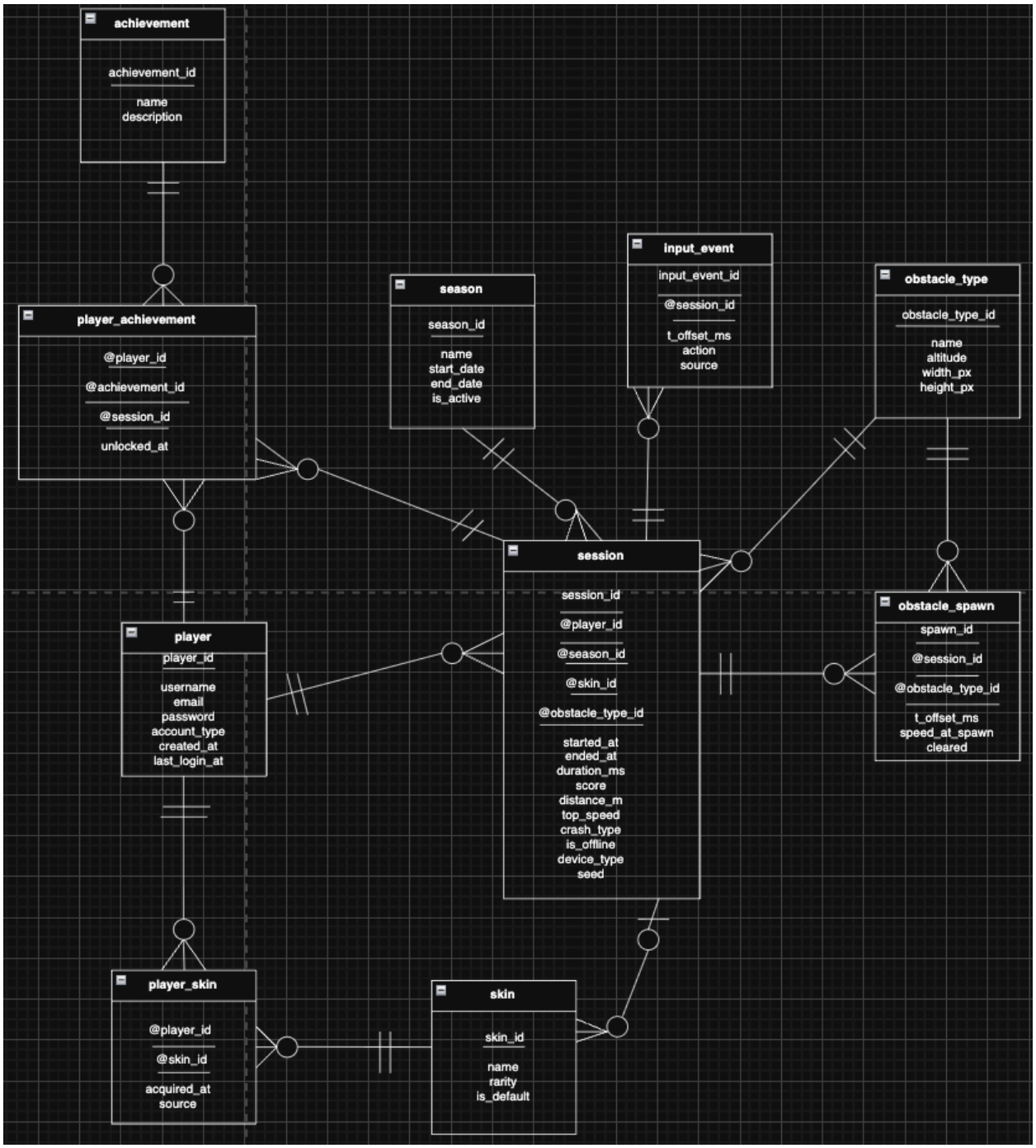
Kyle Jia, jj3708
Nikhil Diddee, nd2564
Junxiao Wang, jw8193

1) Database Design

1.a) Final ERD:



1.b) Final schema statements

Player(Player_id, Username, Email, Password, Account_type, Created_at, Last_login_at)

Season(Season_id , Name, Start_date, End_date, Is_active)

Skin(Skin_id, Name, Rarity, Is_default)

Player_skin(@Player_id, @Skin_id, Acquired_at, Source)

Session(Session_id, @Player_id, @Season_id, @Skin_id, @Obstacle_type_id, Started_at, Ended_at, Duration_ms, Score, Distance_m, Top_speed, Crash_type, Is_offline, Device_type, Seed)

Obstacle_type(Obstacle_type_id, Name, Altitude, Width_px, Height_px)

Obstacle_spawn(Spawn_id, @Session_id, @Obstacle_type_id, T_offset_ms, Speed_at_spawn, Cleared)

Input_event(Input_event_id, @Session_id, T_offset_ms, Action, Source)

Achievement(Achievement_id, Name, Description)

Player_achievement(@Player_id, @Achievement_id, @Session_id, Unlocked_at)

---

2) Database Programming

2.a) Where we host our database

For this project we host our database on **phpMyAdmin running locally on our machines**. phpMyAdmin is connected to a local MariaDB/MySQL server (Apache + PHP 8.x). The database we created in Milestones 1–3 is called **`slime_runner_db`** and contains all ten normalized tables for the Slime Runner endless-runner game (player, season, skin, player_skin, session, obstacle_type, obstacle_spawn, input_event, achievement, player_achievement).

We interact with this database only through our application when we test the game. We still use phpMyAdmin for administrative tasks such as running the initial schema from `COMMANDS.sql` and for backing up/exporting the data, as required in Milestone 3.

Although the DB server is local, it acts as a **shared central database** for the app: any browser that can reach the PHP server (for example, two different browsers on the same machine or two machines on the same network) is talking to the same `slime_runner_db` instance, so multiple users can use the system at the same time without interfering with each other.

2.b) Where we host our app

Our application is a **web-based PHP app**. We deploy it on the same Apache web server that runs phpMyAdmin. The PHP files are stored in a folder named, for example, `slime_runner/` inside the server's document root.

- **Frontend**: standard HTML/CSS/JavaScript pages that correspond to the wireframes we designed in Milestone 2 (Landing, Sign In, Register, Home, Skins, Achievements, Leaderboard, Run History, Session Detail, Game Over, Settings, etc.).

- **Backend**: PHP scripts that connect to the MySQL database using PDO and run SQL commands for all CRUD operations and for the advanced SQL features from Milestone 3.

Because it is a web app, we can easily test multiple concurrent users by opening the site in **different browsers or private windows at the same time** and logging in as different players. All of them are hitting the same PHP code and the same `slime_runner_db` instance.

2.c) How to deploy and run the project

Below are the full steps someone would need to follow to deploy and run our project on a fresh environment that has Apache, PHP, and MySQL/MariaDB installed (for example XAMPP or MAMP):

1. **Set up the database server**
    - Start Apache and MySQL/MariaDB.
    - Open phpMyAdmin in a browser (usually `http://localhost/phpmyadmin`).
    - In phpMyAdmin, open the SQL tab and run our `COMMANDS.sql` file. This script:
        - creates the `slime_runner_db` database,
        - creates all 10 tables with the appropriate constraints,
        - inserts at least 10 rows of realistic data into each table,

        - and defines the advanced stored procedure `sp_validate_session_crash` and the function `fn_player_owns_skin`.

2. **Create DB users and privileges**
    - Still in phpMyAdmin (or the MySQL console), create two accounts:
        - a **developer** account (e.g., `slime_dev`) with full privileges on `slime_runner_db` (used only by us for maintenance),
        - an **application** account (e.g., `slime_app`) with **limited privileges** such as `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `EXECUTE` on `slime_runner_db.*` but no `DROP` or `ALTER`.
    - Run the appropriate `GRANT` statements and `FLUSH PRIVILEGES`. (These statements are described later in the DB-security section of the report.)
3. **Deploy the PHP application**
    - Copy all the PHP source files and assets into a folder called `slime_runner/` under the Apache document root (for example `htdocs/slime_runner/`).
    - Open `config.php` in a text editor and set the database connection variables to match the application user created in the previous step (host, database name `slime_runner_db`, username `slime_app`, and its password).
    - Make sure PHP has PDO/MySQL enabled (this is on by default in most XAMPP/MAMP setups).
4. **Run the app**
    - In a browser, go to `http://localhost/slime_runner/login.php` (or `index.php` if we set a front controller).
    - From here the user can:
        - **Register** a new account (which inserts a row in `player` and a default skin in `player_skin`),

        - **Sign In** as an existing registered user (which authenticates against the `player` table and updates `last_login_at`),
        - or **Play as Guest** (which creates a temporary `GUEST` player entry in the database).

5. **Testing concurrent users**
    - Open two different browsers or a normal window + private window.
    - Log in as two different players, then start runs, view leaderboards, and check that leaderboards, histories, and achievements update correctly for each account while using the same shared database. This demonstrates that the app supports multiple users and returning users as required in the Milestone 4 spec.

2.d) How we incorporated the advanced SQL commands in the app

In Milestone 3 we implemented two advanced PL/SQL features:

1. A **stored procedure** `sp_validate_session_crash`
2. A **function** `fn_player_owns_skin`

Milestone 4 requires that we actually **use these PL/SQL features inside our app**, not just define them, so we integrated them into specific screens and workflows.

---

(1) Using `fn_player_owns_skin` in the Skins and Start-Run flows

Purpose of the function (from Milestone 3)
`fn_player_owns_skin(p_player_id, p_skin_id, p_at)` returns a Boolean indicating whether a player owns a given skin at a particular time. It treats any skin where `is_default = 1` as automatically owned by everyone and otherwise checks the `player_skin` table to see if the player acquired that skin on or before the timestamp `p_at`. This function directly enforces business rule **R4: Skin usage**, which says that a player can only use a skin they actually own.

Where we use it in the app

1. **Skins page (Skin selector UI)**
    - When a logged-in user opens the **Skins** screen, the PHP code runs queries to show two lists:
        - *Owned skins* (from `player_skin JOIN skin`),
        - *Locked skins* (skins not in `player_skin` for that player).

When the user clicks **"Use"** or **"Confirm"** to set a skin for the next run, the backend does **not** trust the UI alone. Instead, it calls:

```
SELECT fn_player_owns_skin(:player_id, :skin_id, NOW()) AS owns;
```

    -
    - If `owns` is true, we store that skin choice in the session and allow the player to proceed; otherwise we show an error message. This means the actual permission check is centralized in the database function rather than duplicated in PHP logic.

2. **Start-Run script (creating a new Session)**
    - When the user clicks **"Start Run"** from the Home screen, the app looks at the "currently selected skin" stored in the PHP session.
    - Before we insert a new row into the `session` table with that `skin_id`, we again call `fn_player_owns_skin` to make sure the player owns the skin as of the run's `started_at` time. If the function returns false, we fall back to the default skin.
    - This allows us to preserve integrity even if the application session is stale or if somebody tries to forge requests.

By wiring the skin selection through `fn_player_owns_skin`, we show that the function is actively used in the real gameplay flow and that it enforces the same rule across multiple parts of the app (Skins UI and Session creation).

---

(2) Using `sp_validate_session_crash` when a run ends

Purpose of the stored procedure (from Milestone 3)

`sp_validate_session_crash(p_session_id)` enforces the complex crash semantics described in our business rules. For a session that ends with `crash_type = 'COLLIDE'`, there must be **exactly one** obstacle spawn in `obstacle_spawn` with `cleared = 0`, and that spawn's `obstacle_type_id` must match the session's `obstacle_type_id`. For `QUIT` or `TIMEOUT` sessions, there must be **no uncleared spawns** at all. If any of these rules are violated, the procedure raises an error using `SIGNAL`.

Where we use it in the app

1. **End-Run script (saving a completed session)**
    - After the player finishes a run, the game sends the final stats (score, distance, top speed, crash type, and sometimes the crash obstacle type) to a PHP script responsible for closing the session.
    - The script updates the corresponding row in `session` by setting `ended_at`, `score`, `distance_m`, `top_speed`, `crash_type`, and `obstacle_type_id`.
    - Then it inserts detailed events into the `obstacle_spawn` and `input_event` tables for that session (in the real game this would come from the gameplay logic; for the Milestone we insert at least one spawn and one input event to demonstrate the structure).

Immediately after inserting these child rows, the script calls:

```
CALL sp_validate_session_crash(:session_id);
```

-
- If the crash data and obstacle spawns are consistent, the procedure returns normally and the app redirects the user to the **Game Over** or **Session Detail** page.
- If there is any inconsistency—for example, a COLLIDE crash with zero or two uncleared obstacles—the stored procedure raises an error. In that case, the PHP script can log the error or show a message.

By calling `sp_validate_session_crash` on every completed session, we delegate the **validation of complicated game rules** to the database layer. This makes the data in `session` and `obstacle_spawn` more trustworthy and keeps the application code simpler. It also demonstrates a realistic use of a stored procedure as required by the Milestone 3 and 4 specs.

---

3) Database Security at the Database Level

Database-level security in our project focuses on controlling which users (developers and the application itself) are permitted to access or modify the database. Since the Milestone 4 requirements emphasize separating developer privileges from end-user privileges, we implemented a clear access-control strategy on our MySQL server.

Our goal is to ensure that:

1. Developers have the necessary flexibility to maintain the system, update schema, and debug issues.

2. The running application (the PHP program) has **limited**, least-privilege access—only the permission needed to run the Slime Runner features, and nothing more.

3. End-users (players) **never** directly interact with the database. All their access is mediated through the app layer and the limited-privilege MySQL account.

3.a) Developer vs End-User Privileges

Developer Accounts

Developers (our team) need full control to:

- Create tables
- Modify schema
- Insert/edit/delete sample data
- Create or update stored procedures and functions
- Debug issues

For this, we created a MySQL account:

- **Username:** `slime_dev`

- **Privileges:** Full privileges (`ALL PRIVILEGES`) on the entire `slime_runner_db`
- **Scope:** Only accessible from localhost

Developers still primarily use phpMyAdmin for development work, but this account allows all schema-altering operations that normal users should not have.

Application Account (End Users Through the App)

End-users never access the DB directly. They interact only through the PHP application. The PHP app, in turn, uses a dedicated MySQL account:

- **Username:** `slime_app`
- **Privileges:** Only `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `EXECUTE`
- **Restrictions:**
    - No `DROP TABLE`
    - No `ALTER TABLE`
    - No `CREATE` privileges
    - No ability to modify the schema or structure of the database

This setup ensures that even if something goes wrong at the app layer, the database structure cannot be damaged since only developers have schema-modifying rights.

3.b) How We Set Up Access Control in MySQL

After creating the `slime_runner_db`, we ran `GRANT` and `REVOKE` commands in phpMyAdmin's SQL tab to enforce controlled access. (These are the commands required in Milestone 4.)

Developer Privileges

CREATE USER IF NOT EXISTS 'slime_dev'@'localhost'

IDENTIFIED BY 'StrongDevPass!';

GRANT ALL PRIVILEGES

ON slime_runner_db.*

TO 'slime_dev'@'localhost';

FLUSH PRIVILEGES;

Why: Developers require full privileges to maintain and update the database across Milestones 1–4.

Application Privileges (Used by our PHP App)

CREATE USER IF NOT EXISTS 'slime_app'@'localhost'

IDENTIFIED BY 'slime_app_password';

GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE

ON slime_runner_db.*

TO 'slime_app'@'localhost';

FLUSH PRIVILEGES;

Why:

- `SELECT` allows reading leaderboard, history, skins, achievements
- `INSERT` allows creating new sessions, new event logs, registering users
- `UPDATE` allows modifying session results, last_login_at, and profile info

- `DELETE` allows users to delete their own sessions in the Run History feature
- `EXECUTE` is required for our stored procedure and function from Milestone 3 (`sp_validate_session_crash`, `fn_player_owns_skin`).
- **No schema-level privileges** → the application can manipulate data but **cannot damage tables**.

Security Tightening (Revocation Example)

We also documented an example REVOKE command in case we want to further restrict privileges:

REVOKE DELETE

ON slime_runner_db.*

FROM 'slime_app'@'localhost';

This is not used in the final system because our app **needs Delete functionality** for the Run History page, but we included it as evidence that privileges can be granularly adjusted if needed.

3.c) Rationale for Privilege Design

This configuration follows the principle of **least privilege**, meaning each MySQL user only receives the permissions absolutely required to perform their role.

- The **developer account** supports ongoing maintenance without restrictions.
- The **application account** protects the schema from unauthorized changes and protects the system even if potential bugs exist in the PHP code.
- **End-users** interact only through the UI and never receive direct database credentials.

---

4) Database Security at the Application Level

At the application level, our project incorporates several layers of security to ensure that users interact with the database safely, that sensitive information (such as passwords) is protected, and that users can only access their own data. While database-level privilege control restricts what the *application account* can do inside MySQL, application-level security controls how the PHP code handles user authentication, validation, error handling, and data access paths. Together, both layers protect the entire Slime Runner system.

4.a) User Authentication and Session Management

Our application uses a standard login system to authenticate registered players. When a user signs in, the backend validates their credentials against the `player` table and records the login in PHP's session system.

How authentication works

1. The user enters an email and password on the Sign-In page.
2. The backend performs a parameterized query to find the matching registered user.
3. If the password is correct, the app initializes a PHP session and sets:
    - `$_SESSION['player_id']`
    - `$_SESSION['username']`
4. The user is then redirected to the **Home** page.

A simplified snippet from our `login.php`:

```
if ($user && password_verify($password, $user['password'])) {
   $_SESSION['player_id'] = $user['player_id'];
   $_SESSION['username']  = $user['username'];

   $upd = $pdo->prepare("UPDATE player SET last_login_at = NOW() WHERE player_id = ?");
   $upd->execute([$user['player_id']]);

   header('Location: home.php');
   exit;
}
```

This ensures that only authenticated users can access restricted pages.

Session protection

All protected pages begin with a check like:

```
require_once 'auth.php';
require_login();
```

The `require_login()` function redirects the user back to `login.php` if they are not authenticated.

This prevents:

- Unauthorized access to gameplay data
- Viewing another user's history or stats
- Manipulating URLs to skip login

4.b) Password Hashing

The application never stores plaintext passwords. When a user registers, the password is hashed using PHP's built-in `password_hash()` function, which applies a strong one-way hashing algorithm (bcrypt/argon2 depending on PHP version).

Snippet from `register.php`:

```
$hash = password_hash($password, PASSWORD_DEFAULT);

$insert = $pdo->prepare("
   INSERT INTO player (username, email, password, account_type, created_at)
   VALUES (?, ?, ?, 'REGISTERED', NOW())
");
$insert->execute([$username, $email, $hash]);
```

This ensures that even if the database were exposed, actual passwords cannot be recovered.

4.c) Access Control (Authorization)

Even after authentication, we further restrict data access. Users can only see or modify **their own data**, never other players' data.

Examples of authorization enforcement:

1. Run History

Every SELECT on the `session` table filters by the logged-in user:

WHERE player_id = :pid

2. Session Detail

A session's events are only shown if the session belongs to the current user:

```
$stmt = $pdo->prepare("
   SELECT * FROM session
   WHERE session_id = ? AND player_id = ?
");
```

3. Delete session

The delete button only works on the user's own sessions:

```
DELETE FROM session
WHERE session_id = ? AND player_id = ?
```

This prevents horizontal privilege escalation (e.g., player A trying to view/modify player B's data).

4.d) SQL Injection Prevention Using Prepared Statements

All SQL queries in the application use **prepared statements with placeholders**.

For example:

```
$stmt = $pdo->prepare("SELECT * FROM player WHERE email = ?");
$stmt->execute([$email]);
```

This protects against SQL injection attacks, which is an essential security measure for any database-driven web app.

4.e) Secure Use of Advanced SQL Features

1. Skin Ownership Check (Function)

When a user selects a skin, we do not trust the frontend.
We call:

SELECT fn_player_owns_skin(:player_id, :skin_id, NOW());

This ensures:

- The user truly owns the skin
- The ownership existed before the session started
- Guest/registered logic is enforced

This guards against tampering (e.g., modifying the request to equip a legendary skin).

2. Crash Validation (Stored Procedure)

When a run ends:

CALL sp_validate_session_crash(:session_id);

This ensures:

- COLLIDE crashes have exactly one uncleared obstacle
- Non-COLLIDE crashes have zero uncleared obstacles

Calling this from the app guarantees that session data cannot be submitted in an invalid state.

4.f) Controlled Execution Flow

To protect user data, the app sanitizes user flow by redirecting based on state:

- Unauthenticated users → redirected to `login.php`
- Guest players → restricted from editing account info
- Registered users → allowed to access Settings
- Attempting to access a resource without privilege → redirected to Home

This prevents users from manually navigating to pages they should not be able to access.

4.g) Display-Layer Protections

We sanitize all user-provided text before output using:

htmlspecialchars($value)

This prevents cross-site scripting (XSS) attacks, especially for usernames or messages.

---

5) php

5.1 config.php - database connection + session

```
<?php
// config.php
// Central DB connection using a restricted MySQL user (see GRANT section later)

$DB_HOST = 'localhost';
$DB_NAME = 'slime_runner_db';
$DB_USER = 'slime_app';         // create this user with limited privileges
$DB_PASS = 'slime_app_password';// change as needed

$dsn = "mysql:host=$DB_HOST;dbname=$DB_NAME;charset=utf8mb4";

$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
```

```
    PDO::ATTR_EMULATE_PREPARES   => false,
];

try {
    $pdo = new PDO($dsn, $DB_USER, $DB_PASS, $options);
} catch (PDOException $e) {
    exit("Database connection failed: " . htmlspecialchars($e->getMessage()));
}

if (session_status() === PHP_SESSION_NONE) {
    session_start();
}
?>
```

5.2 auth.php – helper functions

```
<?php
// auth.php
require_once 'config.php';

function current_user_id() {
    return $_SESSION['player_id'] ?? null;
}

function current_username() {
    return $_SESSION['username'] ?? null;
}

function require_login() {
    if (!current_user_id()) {
        header('Location: login.php');
        exit;
    }
}
?>
```

5.3 register.php – create REGISTERED players (Add)

```
<?php
require_once 'config.php';

$errors = [];
```

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = trim($_POST['username'] ?? '');
    $email    = trim($_POST['email'] ?? '');
    $password = $_POST['password'] ?? '';

    if ($username === '' || $email === '' || $password === '') {
        $errors[] = "All fields are required.";
    }

    if (empty($errors)) {
        // Check uniqueness
        $stmt = $pdo->prepare("SELECT 1 FROM player WHERE username = ? OR email = ?");
        $stmt->execute([$username, $email]);
        if ($stmt->fetch()) {
            $errors[] = "Username or email already in use.";
        } else {
            $hash = password_hash($password, PASSWORD_DEFAULT);

            $insert = $pdo->prepare("
                INSERT INTO player (username, email, password, account_type, created_at)
                VALUES (?, ?, ?, 'REGISTERED', NOW())
            ");
            $insert->execute([$username, $email, $hash]);
            $player_id = $pdo->lastInsertId();

            // Give default skin
            $defaultSkin = $pdo->query("SELECT skin_id FROM skin WHERE is_default = 1 LIMIT
1")->fetch();
            if ($defaultSkin) {
                $ps = $pdo->prepare("INSERT INTO player_skin (player_id, skin_id, acquired_at,
source)
                                     VALUES (?, ?, NOW(), 'DEFAULT')");
                $ps->execute([$player_id, $defaultSkin['skin_id']]);
            }

            $_SESSION['player_id'] = $player_id;
            $_SESSION['username']  = $username;
            header('Location: home.php');
            exit;
        }
    }
}
?>
<!DOCTYPE html>
```

```
<html>
<head><title>Register</title></head>
<body>
<h1>Create Account</h1>
<?php foreach ($errors as $e): ?>
<p style="color:red;"><?php echo htmlspecialchars($e); ?></p>
<?php endforeach; ?>

<form method="post">
   <label>Username <input name="username" /></label><br>
   <label>Email <input type="email" name="email" /></label><br>
   <label>Password <input type="password" name="password" /></label><br>
   <button type="submit">Create Account</button>
</form>

<p><a href="login.php">Back to Sign In</a></p>
</body>
</html>
```

5.4 login.php – returning users (Retrieve + Update last_login_at)

```
<?php
require_once 'config.php';

$errors = [];
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
   $email    = trim($_POST['email'] ?? '');
   $password = $_POST['password'] ?? '';

   $stmt = $pdo->prepare("SELECT player_id, username, password FROM player
                          WHERE email = ? AND account_type = 'REGISTERED'");
   $stmt->execute([$email]);
   $user = $stmt->fetch();

   if ($user && password_verify($password, $user['password'])) {
      $_SESSION['player_id'] = $user['player_id'];
      $_SESSION['username']  = $user['username'];

      $upd = $pdo->prepare("UPDATE player SET last_login_at = NOW() WHERE player_id =
?");
      $upd->execute([$user['player_id']]);

      header('Location: home.php');
```

```
        exit;
    } else {
        $errors[] = "Invalid email or password.";
    }
}
?>
<!DOCTYPE html>
<html>
<head><title>Sign In</title></head>
<body>
<h1>Sign In</h1>
<?php foreach ($errors as $e): ?>
<p style="color:red;"><?php echo htmlspecialchars($e); ?></p>
<?php endforeach; ?>

<form method="post">
    <label>Email <input type="email" name="email"></label><br>
    <label>Password <input type="password" name="password"></label><br>
    <button type="submit">Sign In</button>
</form>

<p><a href="guest.php">Continue as Guest</a></p>
<p>Don't have an account? <a href="register.php">Register</a></p>
</body>
</html>
```

5.5 guest.php – Play as guest (Add)

```
<?php
require_once 'config.php';

// create a guest account on demand
$randName = 'guest' . rand(1000, 9999);

$insert = $pdo->prepare("
    INSERT INTO player (username, email, password, account_type, created_at)
    VALUES (?, NULL, NULL, 'GUEST', NOW())
");
$insert->execute([$randName]);
$player_id = $pdo->lastInsertId();

// give default skin
```

```
$defaultSkin = $pdo->query("SELECT skin_id FROM skin WHERE is_default = 1 LIMIT
1")->fetch();
if ($defaultSkin) {
    $ps = $pdo->prepare("INSERT INTO player_skin (player_id, skin_id, acquired_at, source)
                         VALUES (?, ?, NOW(), 'DEFAULT')");
    $ps->execute([$player_id, $defaultSkin['skin_id']]);
}

$_SESSION['player_id'] = $player_id;
$_SESSION['username']  = $randName;

header('Location: home.php');
exit;
```

5.6 logout.php

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

5.7 home.php – dashboard + “Start Run”

```
<?php
require_once 'auth.php';
require_login();

// active season
$seasonStmt = $pdo->query("SELECT * FROM season WHERE is_active = 1 LIMIT 1");
$activeSeason = $seasonStmt->fetch();

// last run
$lastStmt = $pdo->prepare("
    SELECT * FROM session
    WHERE player_id = ?
    ORDER BY started_at DESC
    LIMIT 1
");
$lastStmt->execute([current_user_id()]);
$last = $lastStmt->fetch();
```

```
?>
<!DOCTYPE html>
<html>
<head><title>Home</title></head>
<body>
<p>Welcome back, <?php echo htmlspecialchars(current_username()); ?>!</p>

<?php if ($activeSeason): ?>
<p>Active Season: <?php echo htmlspecialchars($activeSeason['name']); ?></p>
<?php endif; ?>

<nav>
  <a href="start_session.php">Start Run</a> |
  <a href="skins.php">Skins</a> |
  <a href="achievements.php">Achievements</a> |
  <a href="leaderboard.php">Leaderboard</a> |
  <a href="history.php">Run History</a> |
  <a href="settings.php">Settings (optional)</a> |
  <a href="logout.php">Logout</a>
</nav>

<h2>Last Run</h2>
<?php if ($last): ?>
<ul>
  <li>Score: <?php echo (int)$last['score']; ?></li>
  <li>Distance: <?php echo (int)$last['distance_m']; ?> m</li>
  <li>Top Speed: <?php echo htmlspecialchars($last['top_speed']); ?></li>
  <li>Crash Type: <?php echo htmlspecialchars($last['crash_type']); ?></li>
</ul>
<?php else: ?>
<p>No runs yet.</p>
<?php endif; ?>

</body>
</html>
```

5.8 skins.php – choose skin (Retrieve + uses fn_player_owns_skin)

```
<?php
require_once 'auth.php';
require_login();

// which tab?
$tab = $_GET['tab'] ?? 'owned';
```

```
// owned skins
$ownedStmt = $pdo->prepare("
    SELECT s.skin_id, s.name, s.rarity, s.is_default, ps.acquired_at, ps.source
    FROM skin s
    JOIN player_skin ps ON ps.skin_id = s.skin_id
    WHERE ps.player_id = ?
");
$ownedStmt->execute([current_user_id()]);
$owned = $ownedStmt->fetchAll();

// locked (not owned) skins
$lockedStmt = $pdo->prepare("
    SELECT s.skin_id, s.name, s.rarity
    FROM skin s
    WHERE s.skin_id NOT IN (
        SELECT skin_id FROM player_skin WHERE player_id = ?
    )
");
$lockedStmt->execute([current_user_id()]);
$locked = $lockedStmt->fetchAll();

// handle selection
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $skin_id = (int)($_POST['skin_id'] ?? 0);

    // Use your function to check ownership at current time
    $checkStmt = $pdo->prepare("SELECT fn_player_owns_skin(?, ?, NOW()) AS owns");
    $checkStmt->execute([current_user_id(), $skin_id]);
    $row = $checkStmt->fetch();

    if ($row && $row['owns']) {
        $_SESSION['selected_skin_id'] = $skin_id;
        $_SESSION['flash'] = "Skin set for next run.";
    } else {
        $_SESSION['flash'] = "You do not own that skin.";
    }
    header('Location: skins.php');
    exit;
}
?>
<!DOCTYPE html>
<html>
<head><title>Skins</title></head>
```

```
<body>
<h1>Skins</h1>
<p><a href="home.php">Back to Home</a></p>

<?php if (!empty($_SESSION['flash'])): ?>
<p style="color:green;"><?php echo htmlspecialchars($_SESSION['flash']);
unset($_SESSION['flash']); ?></p>
<?php endif; ?>

<p>Current selected skin ID (session): <?php echo $_SESSION['selected_skin_id'] ?? 'default';
?></p>

<p>
   <a href="?tab=owned">Owned</a> |
   <a href="?tab=locked">Locked</a>
</p>

<?php if ($tab === 'owned'): ?>
<h2>Owned Skins</h2>
<ul>
<?php foreach ($owned as $s): ?>
   <li>
      <?php echo htmlspecialchars($s['name'] . " ({$s['rarity']})"); ?>
      <form method="post" style="display:inline;">
         <input type="hidden" name="skin_id" value="<?php echo (int)$s['skin_id']; ?>">
         <button type="submit">Use</button>
      </form>
   </li>
<?php endforeach; ?>
</ul>
<?php else: ?>
<h2>Locked Skins</h2>
<ul>
<?php foreach ($locked as $s): ?>
   <li><?php echo htmlspecialchars($s['name'] . " ({$s['rarity']}) - Locked"); ?></li>
<?php endforeach; ?>
</ul>
<?php endif; ?>

</body>
</html>
```

5.9 achievements.php – unlocked vs locked (Retrieve + filter-style toggle)

```
<?php
require_once 'auth.php';
require_login();

$tab = $_GET['tab'] ?? 'unlocked';

// unlocked
$unlockedStmt = $pdo->prepare("
   SELECT a.name, a.description, pa.unlocked_at, pa.session_id
   FROM achievement a
   JOIN player_achievement pa
     ON pa.achievement_id = a.achievement_id
   WHERE pa.player_id = ?
   ORDER BY pa.unlocked_at DESC
");
$unlockedStmt->execute([current_user_id()]);
$unlocked = $unlockedStmt->fetchAll();

// locked
$lockedStmt = $pdo->prepare("
   SELECT a.name, a.description
   FROM achievement a
   WHERE a.achievement_id NOT IN (
      SELECT achievement_id FROM player_achievement WHERE player_id = ?
   )
");
$lockedStmt->execute([current_user_id()]);
$locked = $lockedStmt->fetchAll();
?>
<!DOCTYPE html>
<html>
<head><title>Achievements</title></head>
<body>
<h1>Achievements</h1>
<p><a href="home.php">Back to Home</a></p>

<p>
   <a href="?tab=unlocked">Unlocked</a> |
   <a href="?tab=locked">Locked</a>
</p>
```

```
<?php if ($tab === 'unlocked'): ?>
<h2>Unlocked</h2>
<ul>
<?php foreach ($unlocked as $a): ?>
  <li>
    <strong><?php echo htmlspecialchars($a['name']); ?></strong> –
    <?php echo htmlspecialchars($a['description']); ?><br>
    Unlocked at: <?php echo htmlspecialchars($a['unlocked_at']); ?> –
    <a href="session_detail.php?id=<?php echo (int)$a['session_id']; ?>">View Session</a>
  </li>
<?php endforeach; ?>
</ul>
<?php else: ?>
<h2>Locked</h2>
<ul>
<?php foreach ($locked as $a): ?>
  <li>
    <strong><?php echo htmlspecialchars($a['name']); ?></strong> –
    <?php echo htmlspecialchars($a['description']); ?>
  </li>
<?php endforeach; ?>
</ul>
<?php endif; ?>

</body>
</html>
```

5.10 leaderboard.php – sort + season dropdown (Retrieve + Sort)

```
<?php
require_once 'auth.php';
require_login();

// seasons for dropdown
$seasonList = $pdo->query("SELECT season_id, name FROM season ORDER BY
start_date")->fetchAll();

// default to active season
$active = $pdo->query("SELECT season_id FROM season WHERE is_active = 1 LIMIT
1")->fetch();
$defaultSeasonId = $active['season_id'] ?? ($seasonList[0]['season_id'] ?? null);

$season_id = (int)($_GET['season_id'] ?? $defaultSeasonId);
```

```
$sort = $_GET['sort'] ?? 'score';

$allowedSorts = [
    'score'    => 's.score',
    'distance' => 's.distance_m',
    'speed'    => 's.top_speed',
    'date'     => 's.ended_at'
];
$orderBy = $allowedSorts[$sort] ?? $allowedSorts['score'];

$stmt = $pdo->prepare("
    SELECT s.session_id, p.username, s.score, s.distance_m, s.top_speed, s.ended_at
    FROM session s
    JOIN player p ON p.player_id = s.player_id
    WHERE s.season_id = ?
    ORDER BY $orderBy DESC
    LIMIT 20
");
$stmt->execute([$season_id]);
$rows = $stmt->fetchAll();

// current user's rank in this season
$rankStmt = $pdo->prepare("
    SELECT COUNT(*) + 1 AS rank, s2.score AS my_score
    FROM session s1
    JOIN session s2 ON s2.player_id = ? AND s2.season_id = ?
    WHERE s1.season_id = ? AND s1.score > s2.score
");
$rankStmt->execute([current_user_id(), $season_id, $season_id]);
$rankRow = $rankStmt->fetch();
?>
<!DOCTYPE html>
<html>
<head><title>Leaderboard</title></head>
<body>
<h1>Leaderboard</h1>
<p><a href="home.php">Back to Home</a></p>

<form method="get">
    <label>Season:
        <select name="season_id">
            <?php foreach ($seasonList as $s): ?>
                <option value="<?php echo (int)$s['season_id']; ?>"
                    <?php if ($s['season_id'] == $season_id) echo 'selected'; ?>>
```

```
                <?php echo htmlspecialchars($s['name']); ?>
              </option>
            <?php endforeach; ?>
        </select>
    </label>
    <label>Sort by:
        <select name="sort">
            <option value="score" <?php if ($sort==='score') echo 'selected'; ?>>Score</option>
            <option value="distance" <?php if ($sort==='distance') echo 'selected';
?>>Distance</option>
            <option value="speed" <?php if ($sort==='speed') echo 'selected'; ?>>Top
Speed</option>
            <option value="date" <?php if ($sort==='date') echo 'selected'; ?>>Date</option>
        </select>
    </label>
    <button type="submit">Apply</button>
</form>

<table border="1" cellpadding="4">
    <tr>
        <th>#</th><th>Player</th><th>Score</th><th>Distance</th><th>Top
Speed</th><th>Date</th>
    </tr>
    <?php $i = 1; foreach ($rows as $r): ?>
    <tr>
        <td><?php echo $i++; ?></td>
        <td><?php echo htmlspecialchars($r['username']); ?></td>
        <td><?php echo (int)$r['score']; ?></td>
        <td><?php echo (int)$r['distance_m']; ?></td>
        <td><?php echo htmlspecialchars($r['top_speed']); ?></td>
        <td><?php echo htmlspecialchars($r['ended_at']); ?></td>
    </tr>
    <?php endforeach; ?>
</table>

<?php if ($rankRow && $rankRow['my_score'] !== null): ?>
<p>Your approximate rank this season: #<?php echo (int)$rankRow['rank']; ?>
    | Best score used: <?php echo (int)$rankRow['my_score']; ?></p>
<?php else: ?>
<p>You haven't played in this season yet.</p>
<?php endif; ?>

</body>
</html>
```

5.11 history.php – filter + delete + CSV export (Retrieve, Filter, Delete)

```
<?php
require_once 'auth.php';
require_login();

// filters
$season_id  = isset($_GET['season_id']) ? (int)$_GET['season_id'] : null;
$crash_type = $_GET['crash_type'] ?? '';
$date_from  = $_GET['date_from'] ?? '';
$date_to    = $_GET['date_to'] ?? '';

// seasons for dropdown
$seasonList = $pdo->query("SELECT season_id, name FROM season ORDER BY
start_date")->fetchAll();

// handle deletion
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['delete_id'])) {
    $delete_id = (int)$_POST['delete_id'];

    // only allow deleting your own session
    $del = $pdo->prepare("DELETE FROM session WHERE session_id = ? AND player_id = ?");
    try {
        $del->execute([$delete_id, current_user_id()]);
        $_SESSION['flash'] = "Attempted to delete session #$delete_id (will fail if achievements
reference it).";
    } catch (PDOException $e) {
        $_SESSION['flash'] = "Delete failed (probably due to foreign key): " . $e->getMessage();
    }
    header('Location: history.php');
    exit;
}

// dynamic WHERE
$where = ["player_id = :pid"];
$params = ['pid' => current_user_id()];

if ($season_id) {
    $where[] = "season_id = :sid";
    $params['sid'] = $season_id;
}
if ($crash_type !== '') {
    $where[] = "crash_type = :ct";
```

```
    $params['ct'] = $crash_type;
}
if ($date_from !== '') {
    $where[] = "started_at >= :df";
    $params['df'] = $date_from . " 00:00:00";
}
if ($date_to !== '') {
    $where[] = "started_at <= :dt";
    $params['dt'] = $date_to . " 23:59:59";
}

$sql = "SELECT session_id, season_id, score, distance_m, top_speed, crash_type, started_at,
ended_at
        FROM session
        WHERE " . implode(' AND ', $where) . "
        ORDER BY started_at DESC";

$stmt = $pdo->prepare($sql);
$stmt->execute($params);
$sessions = $stmt->fetchAll();

// simple stats
$totalSessions = count($sessions);
$avgScore = $totalSessions ? array_sum(array_column($sessions, 'score')) / $totalSessions : 0;
$longestDistance = $totalSessions ? max(array_column($sessions, 'distance_m')) : 0;
?>
<!DOCTYPE html>
<html>
<head><title>Run History</title></head>
<body>
<h1>Run History</h1>
<p><a href="home.php">Back to Home</a></p>

<?php if (!empty($_SESSION['flash'])): ?>
<p style="color:blue;"><?php echo htmlspecialchars($_SESSION['flash']);
unset($_SESSION['flash']); ?></p>
<?php endif; ?>

<form method="get">
    <label>Season:
        <select name="season_id">
            <option value="">All</option>
            <?php foreach ($seasonList as $s): ?>
                <option value="<?php echo (int)$s['season_id']; ?>"
```

```
                <?php if ($season_id == $s['season_id']) echo 'selected'; ?>>
                <?php echo htmlspecialchars($s['name']); ?>
              </option>
            <?php endforeach; ?>
        </select>
    </label>
    <label>Crash Type:
        <select name="crash_type">
            <option value="">All</option>
            <option value="COLLIDE" <?php if ($crash_type==='COLLIDE') echo 'selected';
?>>COLLIDE</option>
            <option value="QUIT" <?php if ($crash_type==='QUIT') echo 'selected';
?>>QUIT</option>
            <option value="TIMEOUT" <?php if ($crash_type==='TIMEOUT') echo 'selected';
?>>TIMEOUT</option>
        </select>
    </label>
    <label>From: <input type="date" name="date_from" value="<?php echo
htmlspecialchars($date_from); ?>"></label>
    <label>To: <input type="date" name="date_to" value="<?php echo
htmlspecialchars($date_to); ?>"></label>
    <button type="submit">Filter</button>
</form>

<p><a href="export_history_csv.php?<?php echo http_build_query($_GET); ?>">Export as
CSV</a></p>

<table border="1" cellpadding="4">
    <tr>
        <th>ID</th><th>Season</th><th>Score</th><th>Distance</th><th>Top Speed</th>
        <th>Crash</th><th>Started</th><th>Ended</th><th>Actions</th>
    </tr>
    <?php foreach ($sessions as $s): ?>
    <tr>
        <td><?php echo (int)$s['session_id']; ?></td>
        <td><?php echo (int)$s['season_id']; ?></td>
        <td><?php echo (int)$s['score']; ?></td>
        <td><?php echo (int)$s['distance_m']; ?></td>
        <td><?php echo htmlspecialchars($s['top_speed']); ?></td>
        <td><?php echo htmlspecialchars($s['crash_type']); ?></td>
        <td><?php echo htmlspecialchars($s['started_at']); ?></td>
        <td><?php echo htmlspecialchars($s['ended_at']); ?></td>
        <td>
            <a href="session_detail.php?id=<?php echo (int)$s['session_id']; ?>">View</a>
```

```
            <form method="post" style="display:inline;" onsubmit="return confirm('Delete this
session?');">
                <input type="hidden" name="delete_id" value="<?php echo (int)$s['session_id']; ?>">
                <button type="submit">Delete</button>
            </form>
        </td>
    </tr>
    <?php endforeach; ?>
</table>

<h3>Stats</h3>
<ul>
    <li>Total Sessions: <?php echo $totalSessions; ?></li>
    <li>Average Score: <?php echo number_format($avgScore, 1); ?></li>
    <li>Longest Distance: <?php echo (int)$longestDistance; ?> m</li>
</ul>

</body>
</html>
```

5.12 export_history_csv.php – CSV export (Export requirement)

```
<?php
require_once 'auth.php';
require_login();

// reuse the same filter logic as in history.php but without deletion or stats
$season_id  = isset($_GET['season_id']) ? (int)$_GET['season_id'] : null;
$crash_type = $_GET['crash_type'] ?? '';
$date_from  = $_GET['date_from'] ?? '';
$date_to    = $_GET['date_to'] ?? '';

$where = ["player_id = :pid"];
$params = ['pid' => current_user_id()];

if ($season_id) {
    $where[] = "season_id = :sid";
    $params['sid'] = $season_id;
}
if ($crash_type !== '') {
    $where[] = "crash_type = :ct";
    $params['ct'] = $crash_type;
}
if ($date_from !== '') {
```

```
    $where[] = "started_at >= :df";
    $params['df'] = $date_from . " 00:00:00";
}
if ($date_to !== '') {
    $where[] = "started_at <= :dt";
    $params['dt'] = $date_to . " 23:59:59";
}

$sql = "SELECT session_id, season_id, score, distance_m, top_speed, crash_type, started_at,
ended_at
        FROM session
        WHERE " . implode(' AND ', $where) . "
        ORDER BY started_at DESC";

$stmt = $pdo->prepare($sql);
$stmt->execute($params);

header('Content-Type: text/csv');
header('Content-Disposition: attachment; filename="run_history.csv"');

$out = fopen('php://output', 'w');
fputcsv($out,
['session_id','season_id','score','distance_m','top_speed','crash_type','started_at','ended_at']);
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    fputcsv($out, $row);
}
fclose($out);
exit;
```

5.13 session_detail.php – event timeline (Retrieve, join Obstacle/Input tables)

```
<?php
require_once 'auth.php';
require_login();

$session_id = (int)($_GET['id'] ?? 0);

// make sure user owns this session (or relax if you want public view)
$stmt = $pdo->prepare("
    SELECT s.*, sk.name AS skin_name
    FROM session s
```

```
    LEFT JOIN skin sk ON sk.skin_id = s.skin_id
    WHERE s.session_id = ? AND s.player_id = ?
");
$stmt->execute([$session_id, current_user_id()]);
$session = $stmt->fetch();

if (!$session) {
    exit("Session not found or access denied.");
}

// obstacle spawns with type info
$obs = $pdo->prepare("
    SELECT os.t_offset_ms, os.speed_at_spawn, os.cleared,
           ot.name AS obstacle_name, ot.altitude
    FROM obstacle_spawn os
    JOIN obstacle_type ot ON ot.obstacle_type_id = os.obstacle_type_id
    WHERE os.session_id = ?
");
$obs->execute([$session_id]);
$obstacles = $obs->fetchAll();

// input events
$ie = $pdo->prepare("
    SELECT t_offset_ms, action, source
    FROM input_event
    WHERE session_id = ?
");
$ie->execute([$session_id]);
$inputs = $ie->fetchAll();

// merge into timeline
$events = [];
foreach ($obstacles as $o) {
    $events[] = [
        't' => (int)$o['t_offset_ms'],
        'text' => sprintf("Obstacle %s (%s) speed %.2f, cleared=%s",
            $o['obstacle_name'], $o['altitude'], $o['speed_at_spawn'],
            $o['cleared'] ? 'yes' : 'no')
    ];
}
foreach ($inputs as $i) {
    $events[] = [
        't' => (int)$i['t_offset_ms'],
        'text' => "Input: {$i['action']} via {$i['source']}"
```

```
    ];
}
usort($events, fn($a, $b) => $a['t'] <=> $b['t']);
?>
<!DOCTYPE html>
<html>
<head><title>Session Detail</title></head>
<body>
<h1>Session #<?php echo (int)$session_id; ?></h1>
<p><a href="history.php">Back to History</a></p>

<ul>
    <li>Score: <?php echo (int)$session['score']; ?></li>
    <li>Distance: <?php echo (int)$session['distance_m']; ?> m</li>
    <li>Top Speed: <?php echo htmlspecialchars($session['top_speed']); ?></li>
    <li>Crash Type: <?php echo htmlspecialchars($session['crash_type']); ?></li>
    <li>Skin: <?php echo htmlspecialchars($session['skin_name'] ?? 'None'); ?></li>
    <li>Device: <?php echo htmlspecialchars($session['device_type']); ?></li>
    <li>Start: <?php echo htmlspecialchars($session['started_at']); ?></li>
    <li>End: <?php echo htmlspecialchars($session['ended_at']); ?></li>
</ul>

<h2>Timeline</h2>
<ol>
    <li>0 ms – START</li>
    <?php foreach ($events as $e): ?>
        <li><?php echo (int)$e['t']; ?> ms – <?php echo htmlspecialchars($e['text']); ?></li>
    <?php endforeach; ?>
    <li><?php echo (int)$session['duration_ms']; ?> ms – END (<?php echo
htmlspecialchars($session['crash_type']); ?>)</li>
</ol>

</body>
</html>
```

1.14 start_session.php – create a new run (Add + uses fn_player_owns_skin)

```
<?php
require_once 'auth.php';
require_login();

// choose season
$active = $pdo->query("SELECT * FROM season WHERE is_active = 1 LIMIT 1")->fetch();
```

```
if (!$active) {
    exit("No active season configured.");
}
$season_id = $active['season_id'];

// choose skin: selected in session, otherwise default
$skin_id = $_SESSION['selected_skin_id'] ?? null;
if ($skin_id) {
    $check = $pdo->prepare("SELECT fn_player_owns_skin(?, ?, NOW()) AS owns");
    $check->execute([current_user_id(), $skin_id]);
    $row = $check->fetch();
    if (!$row || !$row['owns']) {
        $skin_id = null; // fallback
    }
}
if (!$skin_id) {
    $default = $pdo->query("SELECT skin_id FROM skin WHERE is_default = 1 LIMIT
1")->fetch();
    $skin_id = $default['skin_id'] ?? null;
}

// create session row
$seed = random_int(1000, 999999);
$ins = $pdo->prepare("
    INSERT INTO session (player_id, season_id, skin_id, obstacle_type_id,
                         started_at, ended_at, score, distance_m, top_speed,
                         crash_type, is_offline, device_type, seed)
    VALUES (?, ?, ?, NULL, NOW(), NULL, 0, 0, 0, NULL, 0, 'browser', ?)
");
$ins->execute([current_user_id(), $season_id, $skin_id, $seed]);
$session_id = $pdo->lastInsertId();

// store in PHP session for convenience
$_SESSION['current_session_id'] = $session_id;
?>
<!DOCTYPE html>
<html>
<head><title>Simulated Run</title></head>
<body>
<h1>New Run Started</h1>
<p>Session ID: <?php echo (int)$session_id; ?> | Season: <?php echo
htmlspecialchars($active['name']); ?></p>
<p>(For the DB project we use a simple form instead of the full canvas game.)</p>
```

```
<form method="post" action="end_session.php">
  <input type="hidden" name="session_id" value="<?php echo (int)$session_id; ?>">
  <label>Score: <input type="number" name="score" value="2000"></label><br>
  <label>Distance (m): <input type="number" name="distance_m" value="600"></label><br>
  <label>Top Speed: <input type="number" step="0.1" name="top_speed"
value="20.5"></label><br>
  <label>Crash Type:
    <select name="crash_type">
      <option value="QUIT">QUIT</option>
      <option value="TIMEOUT">TIMEOUT</option>
      <option value="COLLIDE">COLLIDE</option>
    </select>
  </label><br>
  <label>If COLLIDE, obstacle type:
    <select name="obstacle_type_id">
      <option value="">(auto or none)</option>
      <?php
      $ots = $pdo->query("SELECT obstacle_type_id, name FROM
obstacle_type")->fetchAll();
      foreach ($ots as $ot) {
        echo '<option value="'.(int)$ot['obstacle_type_id'].'">' .
          htmlspecialchars($ot['name']) . '</option>';
      }
      ?>
    </select>
  </label><br>
  <button type="submit">Finish Run</button>
</form>

</body>
</html>
```

1.15 end_session.php – finish run (Update + Insert into obstacle_spawn & input_event + CALL procedure)

```
<?php
require_once 'auth.php';
require_login();

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
  header('Location: home.php'); exit;
}

$session_id      = (int)($_POST['session_id'] ?? 0);
$score           = (int)($_POST['score'] ?? 0);
```

```
$distance_m       = (int)($_POST['distance_m'] ?? 0);
$top_speed        = (float)($_POST['top_speed'] ?? 0);
$crash_type       = $_POST['crash_type'] ?? 'QUIT';
$obstacle_type_id = $_POST['obstacle_type_id'] !== '' ? (int)$_POST['obstacle_type_id'] : null;

// verify session belongs to user
$chk = $pdo->prepare("SELECT * FROM session WHERE session_id = ? AND player_id = ?");
$chk->execute([$session_id, current_user_id()]);
if (!$chk->fetch()) {
    exit("Invalid session.");
}

// pick obstacle type if COLLIDE and not provided
if ($crash_type === 'COLLIDE' && !$obstacle_type_id) {
    $row = $pdo->query("SELECT obstacle_type_id FROM obstacle_type ORDER BY
obstacle_type_id LIMIT 1")->fetch();
    $obstacle_type_id = $row['obstacle_type_id'] ?? null;
}

// update session record
$upd = $pdo->prepare("
    UPDATE session
       SET ended_at       = NOW(),
           score          = ?,
           distance_m     = ?,
           top_speed      = ?,
           crash_type     = ?,
           obstacle_type_id = ?
     WHERE session_id     = ?
");
$upd->execute([$score, $distance_m, $top_speed, $crash_type, $obstacle_type_id,
$session_id]);

// Insert a couple of dummy events to demonstrate use of child tables.
// In a real game this would be recorded live by JavaScript.
$pdo->prepare("
    INSERT INTO obstacle_spawn (session_id, obstacle_type_id, t_offset_ms, speed_at_spawn,
cleared)
    VALUES (?, ?, 1000, ?, ?)
")->execute([
    $session_id,
    $obstacle_type_id ?? 1,
    max(10.0, $top_speed - 2),
    $crash_type === 'COLLIDE' ? 0 : 1
```

```
]);

$pdo->prepare("
    INSERT INTO input_event (session_id, t_offset_ms, action, source)
    VALUES (?, 900, 'JUMP', 'KEYBOARD')
")->execute([$session_id]);

// Call stored procedure to validate crash rules
$proc = $pdo->prepare("CALL sp_validate_session_crash(?)");
$proc->execute([$session_id]);

// optionally unlock a simple achievement for demo
if ($score >= 3000) {
    // find achievement id for 'Marathon Runner'
    $a = $pdo->prepare("SELECT achievement_id FROM achievement WHERE name =
'Marathon Runner' LIMIT 1");
    $a->execute();
    if ($aid = $a->fetchColumn()) {
        // insert ignore because primary key is (player_id, achievement_id)
        $ins = $pdo->prepare("
            INSERT IGNORE INTO player_achievement (player_id, achievement_id, session_id,
unlocked_at)
            VALUES (?, ?, ?, NOW())
        ");
        $ins->execute([current_user_id(), $aid, $session_id]);
    }
}

$_SESSION['flash'] = "Run saved! Session #$session_id.";
header("Location: session_detail.php?id=" . $session_id);
exit;
```

1.16 settings.php – example Update player info

```
<?php
require_once 'auth.php';
require_login();

$errors = [];
$success = '';

$stmt = $pdo->prepare("SELECT username, email, account_type, created_at FROM player
WHERE player_id = ?");
$stmt->execute([current_user_id()]);
$player = $stmt->fetch();
```

```
if (!$player) exit("Player not found.");

if ($_SERVER['REQUEST_METHOD'] === 'POST' && $player['account_type'] ===
'REGISTERED') {
    $username = trim($_POST['username'] ?? '');
    $email    = trim($_POST['email'] ?? '');

    if ($username === '' || $email === '') {
        $errors[] = "Username and email required.";
    } else {
        $upd = $pdo->prepare("UPDATE player SET username = ?, email = ? WHERE player_id =
?");
        try {
            $upd->execute([$username, $email, current_user_id()]);
            $_SESSION['username'] = $username;
            $success = "Profile updated.";
        } catch (PDOException $e) {
            $errors[] = "Update failed: " . $e->getMessage();
        }
    }
}
?>
<!DOCTYPE html>
<html>
<head><title>Settings</title></head>
<body>
<h1>Settings</h1>
<p><a href="home.php">Back to Home</a></p>
<?php foreach ($errors as $e): ?><p style="color:red;"><?php echo htmlspecialchars($e);
?></p><?php endforeach; ?>
<?php if ($success): ?><p style="color:green;"><?php echo htmlspecialchars($success);
?></p><?php endif; ?>

<form method="post">
    <p>Account type: <?php echo htmlspecialchars($player['account_type']); ?></p>
    <?php if ($player['account_type'] === 'REGISTERED'): ?>
        <label>Username <input name="username" value="<?php echo
htmlspecialchars($player['username']); ?>"></label><br>
        <label>Email <input type="email" name="email" value="<?php echo
htmlspecialchars($player['email']); ?>"></label><br>
        <button type="submit">Save Changes</button>
    <?php else: ?>
        <p>Guest accounts cannot change profile info.</p>
```

```
    <?php endif; ?>
</form>

</body>
</html>
```