How to Make an API with PHP for a Google Play Game Using Unity and C#

1 Introduction

This documentation provides a comprehensive guide on creating a PHP-based API for a Google Play game developed with Unity and C#. It covers the essential aspects of game development architecture, including authentication, server communication, and security. By following this guide, you will learn how to:

- Set up a PHP server to handle game data and user authentication.
- Integrate Google Sign-In for secure user authentication.
- Design and implement a RESTful API for client-server communication.
- Secure your API and protect user data.
- Manage game data with a MySQL database.
- Test and debug your game and API.
- Deploy your game to Google Play and your API to a live server.

1.1 Prerequisites

Before starting, ensure you have:

- Basic knowledge of PHP, Unity, and C#.
- Familiarity with web development concepts (HTTP, JSON, etc.).
- A development environment set up for Unity and PHP (e.g., XAMPP for local testing).
- Access to a MySQL database.
- A Google Developer account for Google Play Services integration.

2 Setting Up the Environment

2.1 PHP Server Setup

- For local development, install XAMPP or WAMP to set up a PHP environment.
- For production, choose a hosting service that supports PHP and MySQL (e.g., Hostinger, Bluehost).

2.2 Unity Setup

- Install Unity Hub and the latest version of Unity.
- Add the Android Build Support module.
- Install the Google Play Games Services plugin for Unity.

2.3 Google Play Services

- Create a project in the Google Play Console.
- Enable Google Play Games Services.
- Generate OAuth credentials for server-side authentication.

3 Authentication

3.1 Overview

Use Google Sign-In to authenticate users securely. The client (Unity) handles the sign-in process and sends a token to the server for verification.

3.2 Implementing Google Sign-In in Unity

Use the Google Play Games Services plugin. Configure it with your game's credentials and implement the sign-in logic:

```
using GooglePlayGames;
using GooglePlayGames.BasicApi;

public class AuthManager : MonoBehaviour
{
    void Start()
    {
        PlayGamesPlatform.Activate();
        Social.localUser.Authenticate((bool success) => {
            if (success) {
                Debug.Log("Authentication successful");
            } else {
                Debug.Log("Authentication failed");
            }
```

```
        });
    }
}
```

3.3 Verifying Google Sign-In on the Server

The client sends the ID token to the server, and PHP verifies it using Google's API:

```php
require 'vendor/autoload.php';
use Google\Client;

function verifyGoogleToken($idToken) {
    $client = new Client(['client_id' => 'YOUR_CLIENT_ID']);
    $payload = $client->verifyIdToken($idToken);
    if ($payload) {
        $userid = $payload['sub'];
        // Proceed with user authentication
    } else {
        // Invalid token
    }
}
```

4 API Structure

4.1 Designing Endpoints

Define RESTful endpoints such as:

- `/login`: Authenticate user and return a session token.
- `/get_player_data`: Retrieve player data.
- `/update_score`: Update player's score.

4.2 Handling Requests and Responses

Use JSON for data exchange and implement proper HTTP methods (POST for sensitive data, GET for retrieval).

4.3 Example API Call

Unity sends a POST request to `/login` with the Google ID token, and PHP returns a session token.

5 Security

5.1 HTTPS

Ensure your server uses HTTPS to encrypt data in transit.

5.2 Input Validation

Sanitize and validate all input data to prevent SQL injection and XSS attacks.

5.3 Data Protection

Use bcrypt or Argon2 to hash sensitive data (e.g., passwords, if applicable). Never store Google tokens in plain text.

5.4 API Security

- Implement rate limiting to prevent abuse.
- Use API keys or JWT for authentication.

6 Database Management

6.1 Setting Up MySQL

Create a database and design tables for users and game data.

6.2 Connecting PHP to MySQL

Use PDO for secure database connections:

```
$dsn = 'mysql:host=localhost;dbname=your_db';
$username = 'your_username';
$password = 'your_password';
$pdo = new PDO($dsn, $username, $password);
```

6.3 Best Practices

- Use prepared statements to prevent SQL injection.
- Regularly back up your database.

7 Game Architecture

7.1 Client-Server Model

The client (Unity) handles game logic and UI, while the server (PHP) manages data persistence and validation.

7.2 Handling Game States

Save game progress on the server and implement offline play with local caching and syncing.

8 Testing and Debugging

8.1 Tools

- Use Postman to test API endpoints.
- Unity's debugger for client-side issues.
- Xdebug for PHP debugging.

8.2 Logging

Implement logging on both client and server for error tracking.

9 Deployment

9.1 Deploying PHP API

Upload your PHP files to the hosting server and configure the database connection.

9.2 Publishing to Google Play

Build your Unity project for Android, upload the APK to the Google Play Console, and configure Google Play Services integration.

10 Conclusion

This documentation has provided a comprehensive guide to developing a PHP API for a Google Play game using Unity and C#. By following these steps, you can create a secure, scalable, and efficient game server that integrates seamlessly with Google Play Services.

For further learning, explore:

- Implementing leaderboards and achievements.
- Handling in-app purchases.
- Optimizing database queries for large-scale games.