DPlay API – Technical Specification

Your Activity Booking Partner

Environment: cPanel / Shared Hosting
Auth Model: Firebase Auth + JWT Bridge
Payment: Razorpay

Generated developer documentation – for internal use

1. Overview

This document describes the DPlay backend API used for activity and sports court booking. The API is built in PHP using Slim 4 and MySQL, deployed on a cPanel / shared hosting environment. It exposes REST endpoints for authentication, venue discovery, slot availability, booking management, and (future) Razorpay payment capture.

Authentication is handled using Firebase Authentication on the client side. The DPlay API validates Firebase ID tokens and issues internal JWT access tokens for fast authorization checks. This pattern is called a Firebase Auth + JWT Bridge.

This document contains:

- High level architecture description
- Authentication and security model
- Full API endpoint list with parameters and example payloads
- Booking logic (slots, discounts, capacity, cancellation)
- Error handling contract
- Mobile developer quick guide

2. System Architecture

2.1 Components

- Client Apps
  – Mobile app (Android / iOS)
  – Web frontend (future)
- Identity Provider
  – Firebase Authentication (email/password, OTP, social logins)
- Backend API
  – PHP 8+, Slim 4, running on Apache via cPanel
  – MySQL database (dplays_site)
- Payment Gateway
  – Razorpay (order creation, payment capture, webhook verification)

2.2 Request Flow (Authentication Bridge)

1) User authenticates with Firebase from the mobile app using email/password or another supported provider.
2) The mobile app receives a Firebase ID token.
3) The mobile app calls DPlay API endpoint POST /api/v1/auth/firebase-login with the Firebase ID token in the body or header.
4) DPlay validates the Firebase ID token using Firebase Admin SDK or REST API.
5) If valid, DPlay creates or updates a local user in the `users` table and issues an internal JWT access token and refresh token.
6) For all subsequent API calls, the client sends:
   Authorization: Bearer <internal JWT access token>
   The API no longer needs to talk to Firebase on each request.

2.3 Environments

- Development
  – Base URL: http://localhost/dplay-api/public
- Production
  – Base URL: https://api.dplay.in (example)
  – Deployed on cPanel, using /public as document root for the API project.

3. Authentication and Security

3.1 Firebase Login Endpoint (Bridge)

Endpoint:
POST /api/v1/auth/firebase-login

Headers:
Content-Type: application/json

Body:
```
{
  "firebase_id_token": "<FirebaseIDToken>"
}
```

Behaviour:
- Validate the Firebase ID token.
- Extract user ID, email, display name, photo from Firebase.
- If user does not exist in `users` table, create a new record.
- Issue internal JWT access token and refresh token.

Sample Success Response:
```
{
  "success": true,
  "user": {
    "id": 12,
    "email": "user@example.com",
    "name": "Test User"
  },
  "tokens": {
    "access_token": "<JWT>",
    "refresh_token": "<REFRESH>",
    "expires_in": 900
  }
}
```

3.2 Existing Auth Endpoints

- POST /api/v1/auth/login
  – Can be used for legacy email/password login stored in MySQL.
- POST /api/v1/auth/register
  – Optional if registration is done through Firebase only.
- POST /api/v1/auth/refresh
  – Uses stored refresh tokens in auth_tokens table to rotate access tokens.
- GET /api/v1/auth/me
  – Requires Authorization header and returns the current user profile.

3.3 Authorization Header

Most protected endpoints require:

Authorization: Bearer <access_token>

If the token is invalid or expired, the API responds with HTTP 401 and a JSON error payload:

```
{
  "success": false,
  "message": "Invalid or expired token"
}
```

3.4 Token Expiry and Refresh

- Access token lifetime: 15 minutes (configurable via JWT_TTL env variable).
- Refresh token lifetime: 30 days (configurable via JWT_REFRESH_TTL env variable).
- Refresh flow:
  - Client calls POST /api/v1/auth/refresh with a valid refresh_token.
  - API validates the refresh_token against auth_tokens table.
  - On success, it issues a new access_token and refresh_token, and deletes the old one.

4. Discover Module

4.1 List Venues

Endpoint:
GET /api/v1/discover/venues

Query Parameters:
- page (optional, default 1)
- per_page (optional, default 10, max 50)
- city (optional, filter by city name)
- sport (optional, substring match in sport_type)

Sample Request:
GET /api/v1/discover/venues?page=1&per_page=10&city=Vadodara&sport=Pickleball

Sample Response:

```
{
  "success": true,
  "data": [
    {
      "id": 3,
      "name": "DropShot Pickleball Arena",
      "city": "Vadodara",
      "state": "Gujarat",
      "small_des": "Premium pickleball venue",
      "photo": "venue-3.jpg",
      "seo_url": "dropshot-pickleball-arena-vadodara",
      "sport_type": "Pickleball"
    }
  ],
  "pagination": {
    "page": 1,
    "per_page": 10,
    "total": 1,
    "totalPages": 1
  }
}
```


4.2 Venue Details

Endpoint:
GET /api/v1/discover/venues/{id}

Description:
Returns venue profile, photos, rules, includes, and courts.

Sample Response (trimmed):

```
{
  "success": true,
  "data": {
    "venue": {
      "id": 3,
      "name": "DropShot Pickleball Arena",
      "address": "Some address",
      "city": "Vadodara",
      "state": "Gujarat",
      "about_venue": "..."
    },
    "photos": [
```

```
      { "id": 1, "image": "v3-1.jpg" }
    ],
    "includes": [
      { "id": 1, "name": "Drinking Water" }
    ],
    "rules": [
      { "id": 1, "name": "Non-marking shoes only" }
    ],
    "courts": [
      {
        "id": 10,
        "name": "Court 1",
        "sport_type": "Pickleball",
        "surface_type": "Synthetic",
        "noof_player": 4,
        "start_time": "06:00:00",
        "end_time": "23:00:00",
        "time_slot": "60"
      }
    ]
  }
}
```

5. Booking Module

5.1 Slot Availability

Endpoint:
GET /api/v1/bookings/slots

Query Parameters:
- court_id (required)
- date (required, YYYY-MM-DD)

Logic (matches legacy PHP web app):
- Determine weekday or weekend using date('N').
- For weekday:
  – weeekday_slot_price != 0
  – weeekday_slot_status = 1
  For weekend:
  – weeekend_slot_price != 0
  – weeekend_slot_status = 1
- If requested date is today, only show slots with start_time greater than current time.
- Exclude slots that are disabled in slot_disable for that court and date.
- Count confirmed bookings from user_court where payment_status IN (1,2).
  If booked_count >= noof_booking, the slot is fully booked.

Response fields per slot:
- slot_id – ID from court_slot
- time – slot_timing (e.g. "10:00 - 11:00")
- base_price – weekday or weekend price before discount
- final_price – price after discount_master is applied
- discount_per – percentage discount, if any
- discount_rs – flat discount, if any
- capacity – number of allowed bookings for the slot
- booked_count – confirmed bookings count
- is_disabled – true if recorded in slot_disable
- is_fully_booked – true if booked_count >= capacity
- is_available – true only if not disabled and not fully booked

5.2 Create Booking

Endpoint:
POST /api/v1/bookings

Headers:
Authorization: Bearer <access_token>
Content-Type: application/json

Body:
```
{
  "court_id": 10,
  "slot_id": 628,
  "date": "2025-12-01"
}
```

Validation:
- User must be authenticated.
- Slot must exist for the given court.
- Slot must not be disabled on this date.
- Confirmed bookings must be below capacity.
- User must not already have a booking for the same court/date/slot
  with payment_status IN (0,1,2).

On success:

- Price is recalculated using the same logic as availability.
- A new row in user_court is created with payment_status = 0 (pending/cart).
- A new order_no is assigned.

Sample Response:


```
{
  "success": true,
  "booking_id": 1024,
  "order_no": 10001,
  "court_id": 10,
  "venue_id": 3,
  "slot_id": 628,
  "court_date": "2025-12-01",
  "court_time": "10:00 - 11:00",
  "amount": 450,
  "status": "pending"
}
```

5.3 List Bookings

Endpoint:
GET /api/v1/bookings

Headers:
Authorization: Bearer <access_token>

Query Parameters:
- page (optional, default 1)
- per_page (optional, default 10, max 50)

Returns bookings for the authenticated user (or all if role=admin).

5.4 Booking Details

Endpoint:
GET /api/v1/bookings/{id}

Returns detailed information about a specific booking, including venue address and court details. Non-admin users can only access their own bookings.

5.5 Cancel Booking

Endpoint:
POST /api/v1/bookings/{id}/cancel

Rules:
- Only the owning user or an admin can cancel a booking.
- On success, payment_status is set to 3 (cancelled).
- Razorpay refund integration can be added later, invoking Razorpay APIs and updating payment_status and a dedicated refund table.

5.6 Payment Status Mapping

- 0 – Pending (cart, not paid yet)
- 1 – Confirmed / paid
- 2 – Payment success / verifying
- 3 – Cancelled by user or admin
- 4 – Refunded
- 5 – Rejected / failed

6. Razorpay Payment Flow (Planned)

6.1 High Level Flow

1) Client requests a booking using POST /api/v1/bookings.
2) API creates a pending booking in user_court and returns order_no and amount.
3) Client calls a new endpoint POST /api/v1/payments/create-order which:
   • Creates a Razorpay order via Razorpay API using the amount and order_no.
   • Returns Razorpay order_id to the client.
4) Client completes payment in the app using the Razorpay Checkout SDK.
5) Razorpay returns payment_id and order_id to the client.
6) Client calls POST /api/v1/payments/verify with:
   • razorpay_order_id
   • razorpay_payment_id
   • razorpay_signature
7) API verifies signature using Razorpay secret and, if valid, marks:
   • user_court.payment_status = 1 (confirmed)
   • Saves razorpay_payment_id and signature for audit.
8) Optionally, a Razorpay webhook is configured on server to confirm payment in case of client connectivity issues.

6.2 Security Considerations

• Never expose Razorpay secret key to the client.
• Always verify signature server-side before marking bookings as paid.
• Store payment logs in a dedicated table for reconciliation.

7. Mobile Developer Quick Guide

7.1 Login Flow

1) Use Firebase SDK to authenticate user (email/password or OTP).
2) Retrieve Firebase ID token from Firebase SDK.
3) Call POST /api/v1/auth/firebase-login with the ID token.
4) Store access_token and refresh_token securely on device.
5) Attach access_token as Authorization header for all further API calls.

7.2 Fetch Venues

- Call GET /api/v1/discover/venues
- Support filtering by city and sport.
- Cache venue list locally and refresh periodically.

7.3 Check Slot Availability

- When user selects date and court, call:
  GET /api/v1/bookings/slots?court_id={id}&date={YYYY-MM-DD}
- Render slot list showing:
  – final_price
  – availability (available / booked / disabled)

7.4 Create Booking

- When user picks a slot, call POST /api/v1/bookings.
- If payment is immediate, follow Razorpay flow.
- If payment is pay-at-venue (future option), mark booking as reserved and display time limit.

7.5 Token Refresh Handling

- If API returns 401 and message about expired token:
  – Call POST /api/v1/auth/refresh with refresh_token.
  – Replace stored access_token and refresh_token.
  – Retry original API call once.

7.6 Error Handling Best Practices

- Display user-friendly messages for 4xx errors.
- For 500 errors, show generic error and log details silently.
- Implement retry with backoff for transient network failures.

8. Error Handling Contract

8.1 Standard Error Format

All errors return JSON with at least these fields:

```
{
  "success": false,
  "message": "Human readable message"
}
```

Optionally, a `code` field can be added later for frontend mapping.

8.2 Common HTTP Status Codes

• 400 – Validation error (missing or invalid parameters)
• 401 – Unauthorized (missing or invalid token)
• 403 – Forbidden (user does not own the resource)
• 404 – Resource not found (e.g. venue, booking, slot)
• 409 – Conflict (slot already booked, duplicate booking attempt)
• 500 – Internal server error

9. Database Overview (High Level)

Key Tables:

• users
  – Stores local user information synchronized from Firebase (email, name, picture).

• venue_master
  – Venue details including address, city, state, about_venue, photos.

• court_master
  – Courts per venue, with sport_type, start_time, end_time, noof_player, time_slot.

• court_slot
  – Per-court slot configuration (slot_timing, weekday/weekend prices, capacity, commission percentages).

• slot_disable
  – Records disabled slots per court_time and date (maintenance, events).

• discount_master
  – Defines discounts for `courtbook` with percentage or flat discount per date range.

• user_court
  – Booking records (user, venue, court, date, time, price, slot_id, payment_status, order_no).

• auth_tokens
  – Refresh tokens for internal JWT access tokens.

10. Deployment Notes (cPanel)

• Place project under a subfolder and point document root to /public.
• Ensure `vendor` is installed via Composer and not web accessible.
• Configure .htaccess within public/ to route all requests to index.php.
• Use environment variables via .env and php dotenv library.
• Enable HTTPS and enforce it via cPanel or .htaccess for production.