SHUTTLER

Badminton Academy Management System

Production Readiness Plan

Document Version: 1.1
Created: February 2026
Last Updated: February 2026
App Version: 1.0.0+1
Target: App Store (iOS) + Google Play Store + Cloud Backend

Category	Status
Core App Features	~85% Complete
Security & Auth	~15% Complete — JWT MISSING
IDOR Protection	0% — CRITICAL GAP
Usage Capping	0% — Not Implemented
CI/CD Pipeline	0% — Not Configured
Cloud Infrastructure	0% — Running on localhost
Automated Testing	~5% Complete
Privacy & Legal	0% — No Privacy Policy
Overall Readiness	**~30%**

■ ***Critical Warning:*** *The app currently has NO JWT tokens and NO IDOR protection. Any user can access any other user's data by guessing an ID. DO NOT put real user data in production until Phase A security is complete.*

Production Readiness Plan

Document Version: 1.1
Created: February 2026
Last Updated: February 2026
Target: App Store (iOS) + Google Play Store (Android) + Cloud Backend Deployment
Current App Version: 1.0.0+1

Table of Contents

1. Project Overview

Shuttler is a multi-portal badminton academy management system designed to digitize the full workflow of a badminton academy — student enrollment, batch management, attendance tracking, fee collection, performance monitoring, BMI tracking, announcements, and scheduling.

User Roles

Role	Access Level	Portal
------	------------	--------

Role	Access Level	Portal
Owner/Admin	Full CRUD, manage everything	Owner Portal
Coach	View assigned batches, mark attendance, enter performance	Coach Portal
Student	Read-only, view own data	Student Portal

Core Modules

• Authentication & Authorization (Login, Signup, Forgot Password)
• Student Management
• Coach Management
• Batch / Season / Session Management
• Attendance Tracking (Student + Coach)
• Fee Management & Payments
• Performance Tracking
• BMI / Health Tracking
• Announcements & Notifications
• Calendar & Holiday Management
• Reports (Attendance, Fees, Performance) with PDF Export
• Tournament Management
• Video Resources
• Leave Requests
• Enquiry Management

2. Technology Stack

Backend

Component	Technology	Version	Status
-----------	-----------	---------	--------
Framework	FastAPI (Python)	0.104.1	■ In Use
Database	PostgreSQL	Latest	■ In Use
ORM	SQLAlchemy	2.0.23	■ In Use
Migrations	Alembic	1.13.0	■ Installed, ■ Not fully configured
Auth Library	python-jose + passlib[bcrypt]	3.3.0 / 1.7.4	■ Installed, ■ JWT not implemented
File Upload	python-multipart	0.0.6	■ In Use
Background Tasks	APScheduler	3.10.4	■ In Use
ASGI Server	Uvicorn	0.24.0	■ In Use
Push Notifications	Firebase Admin SDK	—	■ Not installed/configured
Rate Limiting	slowapi / custom middleware	—	■ Not implemented
Caching	Redis	—	■ Not implemented

Component	Technology	Version	Status
HTTPS/TLS	Nginx reverse proxy	—	■ Not configured

Frontend (Flutter)

Component	Technology	Version	Status
-----------	-----------	---------	--------
Framework	Flutter / Dart	SDK ^3.10.4	■ In Use
State Management	Riverpod + Code Gen	^2.4.0	■ In Use
HTTP Client	Dio	^5.3.0	■ In Use
Navigation	GoRouter	^12.0.0	■ In Use
Local Storage	SharedPreferences + Hive	^2.2.0	■ In Use
Push Notifications	Firebase Core + Messaging	^3.6.0 / ^15.1.3	■ Installed, ■■ Partial
Charts	FL Chart	^0.65.0	■ In Use
Calendar	Table Calendar	^3.0.9	■ In Use
PDF Export	pdf + printing	^3.10.0	■ In Use
Image Handling	image_picker + image_cropper	^1.0.4	■ In Use
Skeleton Loading	shimmer	3.0.0	■ In Use
Offline Support	connectivity_plus	^5.0.2	■ In Use
Deep Linking	—	—	■ Not configured
Crash Reporting	Firebase Crashlytics	—	■ Not installed
Analytics	Firebase Analytics	—	■ Not installed

---

3. Architecture Diagram

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ CURRENT ARCHITECTURE ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ■
■ Flutter App (iOS / Android) ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Owner Portal | Coach Portal | Student Portal ■ ■
■ ■ Riverpod State Management ■ ■
■ ■ Dio HTTP Client ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ FastAPI Backend (localhost:8000) ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ 100+ REST Endpoints ■ ■
■ ■ SQLAlchemy ORM ■ ■
■ ■ BCrypt Password Hashing ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ PostgreSQL Database (local) ■
■ 25 Tables, Connection Pool: 60 connections ■
■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ TARGET ARCHITECTURE ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ■
■ Flutter App (App Store / Play Store) ■
■ ■ ■
■ ▼ ■
■ CDN (CloudFront / Cloudflare) ■
■ ■ ■
■ ▼ ■
■ Load Balancer (Nginx / AWS ALB) ■
■ ■ ■ ■
■ ▼ ▼ ■
■ FastAPI (Instance 1) FastAPI (Instance 2) [Horizontal] ■
■ ■ ■
■ ■■■■ Redis Cache ■
■ ■■■■ PostgreSQL (AWS RDS / Railway) ■
■ ■■■■ Firebase Admin (FCM Push Notifications) ■
■ ■■■■ S3 / Cloudflare R2 (File Storage) ■
■ ■■■■ Monitoring (Sentry / Datadog) ■
■ ■
■ CI/CD: GitHub Actions ■■■ Docker ■■■ Cloud Deploy ■
■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

4. COMPLETED Features

4.1 Authentication & Authorization

■ Role-based login (Owner, Coach, Student)
■ Signup with role selection
■ Forgot password flow (reset token)
■ BCrypt password hashing (server-side)
■ Role-based screen access control (Owner-only, Coach-only, Student read-only)
■ Local credential storage (SharedPreferences)
■ Logout functionality

4.2 Owner Portal — Fully Implemented Screens

■ Owner Dashboard (stats cards, active batches, quick actions)
■ Student Management (create, edit, soft delete, status toggle active/inactive)
■ Coach Management (registration, invitation system, assignment)
■ Batch Management (create, edit, active/inactive filtering, session association, capacity, timing, operating days)
■ Session / Season Management (full CRUD, associate batches to sessions)
■ Attendance Tracking (batch-wise, student-wise, mark attendance, reports)
■ Fee Management (create fees, record payments, overdue tracking)
■ Performance Tracking (skill ratings per student, trend charts)
■ BMI Tracking (health records, trend charts)
■ Reports (Attendance / Fee / Performance — PDF export)

- 🟩 Announcement Management (create, edit, delete, target audience: all / coaches / students)
- 🟩 Calendar Management (events, holidays, Canadian public holidays pre-populated)
- 🟩 Leave Request Management (view, approve/reject coach leave)
- 🟩 Academy Setup & Details (name, address, contact, logo)
- 🟩 Notification Center (view notifications)
- 🟩 Profile Photo Upload (owner)
- 🟩 Settings Screen
- 🟩 More Menu (help, legal, about)

4.3 Coach Portal — Fully Implemented Screens

- 🟩 Coach Dashboard (stats, student count, batch list)
- 🟩 My Batches (view assigned batches, student lists)
- 🟩 Student List (all my students, search)
- 🟩 Attendance Marking (daily batch-wise attendance)
- 🟩 Schedule View (calendar-based with table_calendar, batch operating days, holidays)
- 🟩 Announcements (view-only)
- 🟩 Leave Request Submission (submit, view status)
- 🟩 Profile Management (edit details, photo upload)
- 🟩 Reports (attendance summary)
- 🟩 Video Resources Management (upload / view training videos for students)
- 🟩 Calendar View (view-only, events and holidays)
- 🟩 More Menu

4.4 Student Portal — Fully Implemented Screens

- 🟩 Student Dashboard (stats, batch summary)
- 🟩 Profile Management (edit own profile, photo upload, guided completion flow)
- 🟩 Attendance View (history, percentage calculation)
- 🟩 Fee View (payment history, pending amounts, overdue status)
- 🟩 Performance View (skill ratings, trend charts by skill)
- 🟩 BMI Tracking (health history, progress charts)
- 🟩 Announcements (view-only)
- 🟩 Schedule View (batch schedule)
- 🟩 Calendar View (events and holidays — read-only)
- 🟩 Tournament View (upcoming and past, search, filter by upcoming/past)
- 🟩 Settings Screen
- 🟩 More Menu

4.5 Cross-Portal / Infrastructure Features

- 🟩 Dark Neumorphic UI Theme (consistent design system across all portals)
- 🟩 Skeleton Loading States (shimmer placeholders for all list screens)
- 🟩 Offline Detection (connectivity_plus, queued requests)
- 🟩 Push Notification Infrastructure (Firebase FCM integrated, token storage)
- 🟩 Student Invitation System (WhatsApp / SMS / Email invitation links with tokens)
- 🟩 Coach Invitation System (WhatsApp-based with invite tokens)
- 🟩 Batch Capacity Management (capacity tracking per batch)
- 🟩 Image Upload (profile photos, UUID-based unique filenames)
- 🟩 Image Cropping (in-app cropping before upload)
- 🟩 PDF Report Generation and Export
- 🟩 Canadian Public Holidays (pre-populated in calendar utility)

🟩 Soft Delete for Students (preserves history, marks as deleted)

🟩 Status Management (active/inactive for students, coaches, batches)

4.6 Backend — Fully Implemented

🟩 FastAPI backend (2,126+ lines, 100+ REST endpoints)

🟩 PostgreSQL database with 25 tables

🟩 Connection pooling (20 base + 40 overflow = 60 max connections)

🟩 SQLAlchemy 2.0 ORM with all models

🟩 Alembic installed (migration tool)

🟩 File upload endpoint with UUID-based naming

🟩 Static file serving (/uploads/)

🟩 Background scheduler (APScheduler — cleanup of inactive records >2 years)

🟩 CORS enabled

🟩 Seed data scripts (seed_data.py, seed_full_data.py)

🟩 Database schema registry documentation

🟩 Migration scripts (session schema, fee schema, coach attendance)

🟩 Pydantic v2 data validation (request/response models)

🟩 Swagger UI documentation (auto-generated at /docs)

🟩 Soft delete implementation (students preserved on delete)

🟩 Invitation token system (students and coaches)

🟩 Enquiry management endpoints

---

5. PENDING Features (App-Level)

These are features identified in the project documents that are not yet implemented.

5.1 HIGH PRIORITY (Required Before Launch)

5.1.1 Multiple Coach Assignment per Batch

- **Status: 🟥 Not Implemented**
- **What**: Allow multiple coaches to be assigned to a single batch (currently only 1 coach)
- **Backend Work**:
  - A `batch_coaches` junction table already exists in DB (`BatchCoachDB` model is present)
  - Enable the many-to-many relationship in batch creation/update endpoints
  - Remove dependency on `assigned_coach_id` single field
- **Frontend Work**:
  - Update `Batch` model to include `List<int> assignedCoachIds`
  - Update batch creation/edit form with multi-select coach picker
  - Update batch card UI to display multiple coaches
  - Update `batch_service.dart` API calls

5.1.2 Partial Payment Status for Fees

- **Status: 🟥 Not Implemented**
- **What**: Add "Partially Paid" status when total paid > 0 but < total amount
- **Backend Work** (`Backend/main.py`, line ~2174):
  - Update `calculate_fee_status()` function to return `'partial'` when `0 < total_paid < amount`
- **Frontend Work**:
  - Add `partial` to `Fee` model status enum
  - Add "Partially Paid" badge/color in fees UI

  - Update fee status filter chips to include Partial

5.1.3 Payment Method Standardization

- **Status: 🟥 Not Implemented**
- **What**: Restrict payment methods to Cash and Card only (remove UPI, Bank Transfer)
- **Frontend Work**:
  - Update `add_payment_dialog.dart` (lines 230–253) — remove UPI and Bank Transfer chips
  - Update `record_payment_dialog.dart` — same restriction
- **Backend Work**: Add validation for allowed payment methods

5.1.4 Session-wise Reports

- **Status: 🟧🟧 Partially Implemented (date-range reports exist; session filter missing)**
- **What**: Generate attendance, fee, and performance reports filtered by session
- **Backend Work**:
  - Add `/reports/attendance/session/{session_id}` endpoint
  - Add `/reports/fees/session/{session_id}` endpoint
  - Add `/reports/performance/session/{session_id}` endpoint
  - Aggregate data within session's date range
- **Frontend Work**:
  - Add session picker/filter to `reports_screen.dart`
  - Show session-level summaries (total classes, revenue, performance averages)

5.1.5 Notification Triggers (Backend)

- **Status: 🟧🟧 Partially Implemented (fee notifications done; others missing)**
- **What**: Send push notifications on key events
- **Backend Work** (add to respective endpoints in `main.py`):
  - Attendance marked → notify student (present/absent confirmation)
  - Performance recorded → notify student (new performance entry)
  - BMI recorded → notify student (new BMI entry)
  - Announcement published → notify target audience (all / coaches / students)
  - Leave request approved/rejected → notify coach
  - Fee overdue → notify student (already partially done, verify cron job)
  - Report generated → notify owner

5.2 MEDIUM PRIORITY

5.2.1 Performance Entry Completion Status (Coach Portal)

- **Status: 🟥 Not Implemented**
- **What**: Show coaches which students have/haven't had performance recorded in a session
- **Backend**: Add completion status tracking in performance records
- **Frontend**: Create `coach_performance_screen.dart` with checklist of students per batch

5.2.2 Student Batch Capacity Visibility

- **Status**: ■ Needs Verification
- **What**: Ensure batch capacity (total slots) is hidden from students; they should only see batch name, timing, fees, coach, location, operating days
- **Frontend**: Audit `student_batches_screen.dart` and remove capacity display

5.3 LOW PRIORITY (Post-Launch)

5.3.1 Video Library (Student Portal)

- **Status: 🟥 Not Implemented**

- **What**: A video library for training videos and form-check videos assigned to specific students or all
- **Backend**: `VideoResourceDB` and `VideoTargetDB` models exist — expose full CRUD endpoints
- **Frontend**: Create `student_video_library_screen.dart` with categorized video browser and in-app player

5.3.2 Digital Waiver System

- **Status: 🟥 Not Implemented**
- **What**: Digital consent/waiver form for students to sign before joining a batch
- **Backend**: New `WaiverDB` and `WaiverSubmissionDB` tables, endpoints
- **Frontend**: Waiver management screen (owner), waiver signing screen (student)

5.3.3 Orphaned Database Table Cleanup

- **Status**: ⬛ Not Cleaned
- **What**: `requests` table exists in PostgreSQL with no corresponding model in code
- **Action**: Either create a model and use it, or drop the table

---

6. PENDING: Security & Authentication

This is the most critical area for production. Almost all items below are required before launch.

6.1 JWT Authentication (CRITICAL — NOT IMPLEMENTED)

- **Status: 🟥 Not Implemented — python-jose installed but NOT used**
- **Current**: Dummy/no token; credentials stored in SharedPreferences as plaintext
- **Required**:
  - Implement JWT token generation on login (`python-jose` is already installed)
  - Access Token: short-lived (15–60 minutes)
  - Refresh Token: long-lived (7–30 days), stored in secure HTTP-only cookie or secure storage
  - All protected endpoints must validate Bearer token
  - Token revocation list (blacklist) for logout
  - Auto-refresh token on expiry (frontend interceptor with Dio)
- **Backend Changes** (`main.py`):

```
POST /auth/login → returns {access_token, refresh_token, user_data}
POST /auth/refresh → returns {access_token}
POST /auth/logout → adds token to blacklist
GET /auth/me → returns current user profile
```

- **Frontend Changes**:
  - Store tokens in `flutter_secure_storage` (NOT SharedPreferences — SharedPreferences is NOT encrypted)
  - Add Dio interceptor to attach Bearer token to every request
  - Add Dio interceptor to auto-refresh on 401 response
  - Clear tokens and redirect to login on refresh token expiry

6.2 Secure Token Storage (CRITICAL)

- **Status: 🟥 Not Implemented**
- **Current**: Using `shared_preferences` — unencrypted, accessible on rooted devices
- **Required**: Replace with `flutter_secure_storage` for all sensitive data:
  - Auth tokens (access + refresh)
  - User credentials (if "Remember Me" is used)
  - FCM tokens
- **Package to Add**: `flutter_secure_storage: ^9.2.2`

6.3 HTTPS / TLS (CRITICAL)

- **Status: 🟥 Not Implemented — currently HTTP on localhost**
- **Required**:
  - Deploy backend behind Nginx reverse proxy with SSL/TLS termination
  - Use Let's Encrypt for free SSL certificates (via Certbot)
  - Or use cloud provider's managed HTTPS (AWS ALB, Railway, Render)
  - Enforce HTTPS-only (redirect HTTP → HTTPS)
  - Enable HSTS (HTTP Strict Transport Security) headers
- **Flutter**:
  - Update API base URL from `http://` to `https://`
  - Ensure no `http://` URLs are hardcoded
  - Certificate pinning (optional but recommended for banking/sensitive apps)

6.4 API Authorization (CRITICAL)

- **Status: 🟧🟧 Partially Implemented (role checks in frontend only)**
- **Current**: Role enforcement only happens in Flutter; backend endpoints are not role-protected
- **Required**: ALL backend endpoints must verify role before responding:
  - Owner-only: student CRUD, coach CRUD, batch management, fee collection, report generation
  - Coach-only: marking attendance, entering performance
  - Student-only: view own profile, own attendance, own fees, own performance
  - **FastAPI dependency injection** for role checks:

```
def require_owner(current_user = Depends(get_current_user)):
if current_user.role != "owner":
raise HTTPException(status_code=403, detail="Forbidden")
```

6.5 Input Validation & Sanitization (CRITICAL)

- **Status**: ■■ Partial (Pydantic validates types; no injection protection)
- **Required**:
  - Validate all text inputs (length limits, allowed characters)
  - Sanitize file uploads (validate MIME type, not just extension)
  - Restrict file upload types (images only: JPEG, PNG, WebP)
  - Limit file size (e.g., 5 MB max per image)
  - Protect against SQL injection (SQLAlchemy ORM handles this, but verify raw queries if any)
  - Protect against XSS in announcement/notification text fields
  - Phone number format validation
  - Email format validation
  - Date range validation (start_date < end_date)

6.6 Rate Limiting (CRITICAL)

- **Status: 🟥 Not Implemented**
- **Required**:
  - Add rate limiting middleware to FastAPI using `slowapi` or custom middleware
  - Login endpoint: max 5 attempts per IP per 15 minutes (prevent brute force)
  - Forgot password: max 3 requests per email per hour
  - General API: max 100 requests per user per minute
  - File upload: max 10 uploads per user per hour
  - Return HTTP 429 (Too Many Requests) with `Retry-After` header
- **Package to Add**: `slowapi==0.1.9`

6.7 Password Security

- **Status**: ■■ BCrypt hashing is implemented; policy not enforced

- **Required**:
  - Enforce minimum password length (at least 8 characters)
  - Enforce complexity (uppercase, lowercase, digit, special character) or use passphrase
  - Enforce maximum length (72 bytes — BCrypt limit, already noted in code)
  - Add password strength indicator in signup screen
  - Implement secure forgot password flow (one-time token, time-limited: 15 minutes)
  - Prevent reuse of last 3 passwords (optional)

6.8 CORS Configuration (IMPORTANT)

- **Status**: ■■ CORS enabled with wildcard (`*`) — not safe for production
- **Required**:
  - Restrict CORS to specific allowed origins (your production domain only)
  - Do NOT allow `*` in production
  - Specify allowed methods and headers explicitly

```
app.add_middleware(
CORSMiddleware,
allow_origins=["https://yourapp.com"], # NOT "*"
allow_methods=["GET", "POST", "PUT", "DELETE"],
allow_headers=["Authorization", "Content-Type"],
)
```

6.9 Security Headers

- **Status: 🟥 Not Implemented**
- **Required** (via Nginx or FastAPI middleware):
  - `Strict-Transport-Security: max-age=31536000; includeSubDomains`
  - `X-Content-Type-Options: nosniff`
  - `X-Frame-Options: DENY`
  - `X-XSS-Protection: 1; mode=block`
  - `Content-Security-Policy: default-src 'self'`
  - `Referrer-Policy: no-referrer`

6.10 Secrets Management (CRITICAL)

- **Status**: ■ Exposed in .env (not committed but risk exists)
- **Current**: `.env` file with database credentials stored locally
- **Required**:
  - NEVER commit `.env` to Git (verify `.gitignore` includes `.env`)
  - Use environment variables injected by cloud provider (Railway, AWS Secrets Manager, etc.)
  - Rotate all secrets (DB password, JWT secret key) before production
  - Use a strong, random SECRET_KEY (minimum 256-bit entropy)
  - Separate secrets per environment (development, staging, production)

6.11 Data Encryption

- **Status: 🟥 Not Implemented**
- **Required**:
  - Encrypt sensitive fields at rest (e.g., student guardian phone, address)
  - Database-level encryption (PostgreSQL with pgcrypto, or cloud-managed encryption)
  - File storage encryption (S3 server-side encryption)
  - Enable SSL for PostgreSQL connections (`sslmode=require` in DATABASE_URL)

7. PENDING: IDOR & Business Logic Security

7.1 Insecure Direct Object Reference (IDOR) — CRITICAL

- **Status**: ■ Not Protected
- **What it is**: A student can change the `student_id` in a request URL and access ANOTHER student's attendance, fees, performance, or BMI data. This is a critical privacy vulnerability.
- **Current Vulnerable Endpoints** (examples):

```
GET /attendance/student/42/ ← student 99 can call this with id=42
GET /fees/student/42/ ← same issue
GET /performance/student/42/ ← same issue
GET /bmi/student/42/ ← same issue
```

- **Required Fixes** (ALL endpoints must enforce ownership):

```
# Example fix for student attendance endpoint
@app.get("/attendance/student/{student_id}/")
async def get_student_attendance(student_id: int, current_user = Depends(get_current_user)):
# Students can only see their own data
if current_user.role == "student" and current_user.id != student_id:
raise HTTPException(status_code=403, detail="Access denied")
# Coaches can only see data for students in their assigned batches
if current_user.role == "coach":
if not is_student_in_coach_batches(student_id, current_user.id, db):
raise HTTPException(status_code=403, detail="Access denied")
```

- **Affected Resources**: students, coaches, attendance, fees, payments, performance, BMI, notifications, profile photos, leave requests
- **Action**: Audit every GET/PUT/DELETE endpoint for ownership enforcement

7.2 Mass Assignment Vulnerability

- **Status**: ■ Risk Exists
- **What it is**: If a Pydantic request model accepts `role`, `status`, or `id` as writable fields, a malicious user can send `{"role": "owner"}` and escalate privileges.
- **Required**:
  - Audit ALL Pydantic request schemas — remove `id`, `role`, `status`, `created_at`, `is_deleted`, `fcm_token` from user-facing update schemas
  - Use separate Pydantic schemas for create vs update vs response
  - Example:

```
class StudentUpdate(BaseModel):
name: Optional[str]
phone: Optional[str]
# NOT: role, status, is_deleted, id
```

7.3 Broken Object Level Authorization (BOLA) for Coaches

- **Status**: ■ Not Protected
- **What it is**: A coach could access or modify students/batches that are NOT assigned to them
- **Required**:
  - Before a coach marks attendance, verify the batch is assigned to that coach
  - Before a coach records performance, verify the student is in their batch
  - Before a coach updates a student record, verify access rights
  - Create a reusable `verify_coach_batch_access(coach_id, batch_id, db)` utility function

7.4 Password Reset Token Security

- **Status**: ■■ Reset flow exists; token security unclear
- **Required**:

- Password reset tokens must be:
  - Cryptographically random (use `secrets.token_urlsafe(32)`)
  - Single-use only (invalidated after first use)
  - Time-limited (expire after 15 minutes)
  - Stored as a hash in the database (not plaintext)
- Invalidate ALL active sessions on password reset
- Rate-limit password reset requests (3 per email per hour)

7.5 Concurrent Session Control

- **Status: 🟥 Not Implemented**
- **What**: Currently, multiple devices can be logged in simultaneously with no control
- **Required**:
  - Track active sessions per user
  - Owner/coach should be able to see and revoke active sessions
  - On password change, revoke all other sessions
  - Option to "log out all devices" in settings

7.6 Path Traversal in File Uploads

- **Status**: ■ Not Protected
- **What it is**: A malicious filename like `../../etc/passwd` or `../main.py` in an upload request can overwrite server files
- **Required**:
  - Sanitize all uploaded filenames: strip directory components, strip special characters
  - Generate server-side UUID filename (seems partially done — verify this is enforced for ALL uploads)
  - Store uploaded files in an isolated directory with no execute permissions
  - Example fix:

```
import os, uuid
safe_filename = f"{uuid.uuid4()}{os.path.splitext(original_name)[1].lower()}"
```

  - Validate file extension AND MIME type (magic bytes check)

7.7 Account Enumeration

- **Status**: ■ Vulnerable
- **What it is**: Login errors like "Email not found" vs "Wrong password" allow attackers to enumerate valid emails
- **Required**:
  - Return the SAME error message for both cases: "Invalid email or password"
  - Same HTTP status code (401) for both
  - Same response time (use constant-time comparison to avoid timing attacks)

---

8. PENDING: Usage Capping & Quotas

This section covers limiting resource usage to prevent abuse, runaway costs, and ensuring fair use if the system grows into a multi-academy SaaS platform.

8.1 API Usage Limits Per Academy (IMPORTANT)

- **Status**: ■ No limits
- **What**: Without limits, one academy or one script could exhaust server resources for all users
- **Required**:
  - Per-academy request quota: max 10,000 API calls per day
  - Burst allowance: max 200 requests per minute per academy

    - Track usage in Redis counter (per `owner_id` or `academy_id`)
    - Return HTTP 429 with `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset` headers
    - Notify owner via email/push when 80% of daily limit is reached

8.2 Storage Quota Per Academy

- **Status**: ■ No limits — unlimited file uploads

- **What**: Without storage caps, one academy could upload terabytes of images/videos and bankrupt the cloud bill
- **Required**:
    - Define max storage per academy (e.g., 5 GB per academy for Starter, 50 GB for Pro)
    - Track cumulative upload size per academy in database
    - Reject uploads when quota is exceeded (HTTP 413 Payload Too Large with message)
    - Show storage usage dashboard to owner (used / total)
    - Per-file size limit: 5 MB for images, 500 MB for videos
    - Total file count limit per academy (e.g., 10,000 files)

8.3 Student / Coach Count Limits

- **Status**: ■ No limits

- **What**: If tiered SaaS pricing, academy size determines which plan they need
- **Required** (if SaaS model):
    - Starter Plan: max 50 students, 3 coaches, 10 batches
    - Pro Plan: max 300 students, 20 coaches, unlimited batches
    - Enterprise: unlimited
    - Enforce limits in student/coach creation endpoints
    - Return HTTP 403 with plan upgrade message when limit exceeded
    - Show usage indicators in owner dashboard: "45/50 students used"

8.4 Notification Rate Limiting

- **Status**: ■ No limits

- **What**: Firebase FCM has rate limits. Sending too many notifications can get your FCM project blocked. Also, excessive notifications annoy users.
- **Required**:
    - Max push notifications per student per day: 10
    - Max announcement pushes per owner per hour: 5
    - Batch notifications: use FCM batch send instead of per-user sends
    - Unsubscribe/mute option per notification type
    - FCM quota monitoring (track send counts)

8.5 PDF Export Rate Limiting

- **Status**: ■ No limits

- **What**: PDF generation is CPU/memory intensive. Without limits, one user can queue hundreds of reports and bring down the server.
- **Required**:
    - Max PDF exports per owner per hour: 20
    - Max concurrent PDF generation jobs: 5
    - Queue PDF jobs with a background worker (FastAPI BackgroundTasks or Celery)
    - Return job status endpoint instead of blocking

8.6 File Upload Rate Limiting

- **Status**: ■ No limits

- **What**: Without limits, an attacker can fill up disk storage quickly
- **Required**:
  - Max uploads per user per hour: 20
  - Max total upload size per request: 5 MB
  - Validate MIME type before storing
  - Store in cloud (S3) not local disk (eliminates server disk fill risk)

8.7 Invitation Token Expiry & Usage Limits

- **Status**: ■■ Tokens exist; expiry/single-use unclear
- **Required**:
  - Invitation tokens must expire after 7 days
  - Each token is single-use (invalidate after successful signup)
  - Max active invitations per batch: 100
  - Invalidate tokens when student is removed from batch

---

9. PENDING: Database & Data Layer

7.1 Alembic Migrations (CRITICAL)

- **Status**: ■ Alembic installed but NOT properly configured — manual SQL scripts are used instead
- **Required**:
  - Initialize Alembic properly: `alembic init alembic`
  - Configure `alembic.ini` and `env.py` with the database URL
  - Convert all manual migration scripts to Alembic migration files
  - Create an initial migration from existing models
  - All future schema changes go through Alembic, not manual SQL
  - Alembic migrations run automatically in CI/CD on deploy
  - Never manually alter production database schema

7.2 Database Backups (CRITICAL)

- **Status**: ■ No backup strategy
- **Required**:
  - Automated daily backups (PostgreSQL `pg_dump`)
  - Point-in-time recovery (PITR) enabled on cloud database
  - Backup retention: 30 days minimum
  - Backup to separate storage (S3 or equivalent)
  - Test restore procedure monthly
  - On cloud: use AWS RDS automated backups or Railway's built-in backup

7.3 Database Indexing (IMPORTANT)

- **Status**: ■ No custom indexes beyond primary keys and foreign keys
- **Required** — Add indexes on frequently queried columns:

```sql
CREATE INDEX idx_students_status ON students(status);
CREATE INDEX idx_attendance_batch_date ON attendance(batch_id, date);
CREATE INDEX idx_attendance_student_date ON attendance(student_id, date);
CREATE INDEX idx_fees_student_status ON fees(student_id, status);
CREATE INDEX idx_notifications_user ON notifications(user_id, user_type, is_read);
CREATE INDEX idx_batches_session ON batches(session_id, status);
CREATE INDEX idx_performance_student ON performance(student_id, date);
CREATE INDEX idx_bmi_student ON bmi_records(student_id, date);
```

- Add these as Alembic migrations

7.4 Database Connection Management

- **Status**: ■ Connection pool configured (60 connections max)
- **Required**:
  - Verify connection pool settings are appropriate for production load
  - Add connection pool monitoring (log when pool is exhausted)
  - Add database health check endpoint: `GET /health/db`
  - Configure `connect_args={"connect_timeout": 10}` for connection timeout
  - Use `pool_pre_ping=True` (already may be set) to detect stale connections

7.5 Database SSL

- **Status**: ■ No SSL in DATABASE_URL
- **Required**:
  - Add `?sslmode=require` to PostgreSQL connection string in production
  - For AWS RDS: use the provided SSL certificate bundle
  - Example: `postgresql://user:pass@host:5432/db?sslmode=require`

7.6 Data Archiving / Retention Policy

- **Status**: ■■ Background scheduler deletes inactive records >2 years
- **Required**:
  - Define and document data retention policy
  - Archive (don't delete) old records to a separate archive table before deletion
  - Ensure compliance with any applicable privacy laws (right to erasure)
  - Verify the APScheduler cleanup job is working correctly
  - Add admin endpoint to manually trigger cleanup

7.7 Orphaned Table Cleanup

- **Status**: ■ `requests` table exists with no model
- **Required**:
  - Investigate if `requests` table is used or can be dropped
  - Create a migration to either add model support or drop the table

---

10. PENDING: API Layer

8.1 API Versioning (IMPORTANT)

- **Status**: ■ No versioning — all endpoints at `/`
- **Required**:
  - Version the API: `/api/v1/students`, `/api/v1/batches`, etc.
  - Allows future breaking changes without affecting existing clients
  - Use FastAPI router prefixing:

```
from fastapi import APIRouter
router = APIRouter(prefix="/api/v1")
```

8.2 Pagination (IMPORTANT)

- **Status**: ■ No pagination — all list endpoints return all records
- **Required** — Add pagination to all list endpoints:

  - Query params: `?page=1&limit=20` or `?skip=0&limit=20`
  - Response format:

```
{
"data": [...],
"total": 150,
"page": 1,
"limit": 20,
"pages": 8
}
```

  - Critical for performance when data grows (100s of students, 1000s of attendance records)

8.3 Request/Response Standardization (IMPORTANT)

- **Status**: ■■ Inconsistent response formats across endpoints
- **Required**:
  - Standardize all responses:

```
{
"success": true,
"data": {...},
"message": "Student created successfully",
"error": null
}
```

  - Consistent error responses:

```
{
"success": false,
"data": null,
"message": "Validation error",
"error": {"field": "email", "detail": "Invalid email format"}
}
```

8.4 API Documentation Enhancement

- **Status**: ■■ Auto-generated Swagger exists but not detailed
- **Required**:
  - Add detailed descriptions to all endpoints (tags, summaries, descriptions)
  - Document all request/response schemas with examples
  - Document authentication requirements per endpoint
  - Disable Swagger UI in production (or password-protect it):

```
app = FastAPI(docs_url=None if IS_PRODUCTION else "/docs")
```

8.5 File Upload Security

- **Status**: ■■ Accepts files but no type/size validation
- **Required**:
  - Validate MIME type (not just extension): `image/jpeg`, `image/png`, `image/webp`
  - Enforce file size limit (5 MB max)
  - Scan for malicious content (basic magic bytes check)
  - Store files in cloud storage (S3/Cloudflare R2) not local disk
  - Serve files via CDN, not directly from backend
  - Generate signed/expiring URLs for file access

8.6 Long-Running Tasks (Background Jobs)

- **Status**: ■ PDF report generation is synchronous (blocks API)
- **Required**:
  - Move PDF generation to background task (FastAPI BackgroundTasks or Celery)

- Return job ID immediately, notify user when report is ready (via push notification)
- Implement job status endpoint: `GET /reports/status/{job_id}`

11. PENDING: Caching Strategy

9.1 Redis Cache (IMPORTANT)

- **Status**: ■ No caching at all — every request hits the database
- **Required**:
    - Deploy Redis instance (Redis Cloud free tier, or Railway Redis)
    - Cache frequently read, rarely changed data:
        - Active batches list: TTL 5 minutes
        - Student list: TTL 2 minutes
        - Coach list: TTL 5 minutes
        - Calendar events: TTL 1 hour
        - Canadian holidays: TTL 24 hours (or indefinite)
        - Academy details: TTL 1 hour
    - Cache invalidation: clear relevant cache keys on write operations
    - **Package to Add**: `redis==5.0.1`, `fastapi-cache2==0.2.1`

9.2 Flutter-Side Caching (Local Cache)

- **Status**: ■■ SharedPreferences and Hive are installed but caching strategy unclear
- **Required**:
    - Cache API responses in Hive with TTL
    - Show cached data while refreshing (stale-while-revalidate pattern)
    - Offline-first for read operations (show cached data when offline)
    - Cache invalidation on data write (invalidate relevant keys after create/update/delete)
    - Clear all cached data on logout

9.3 HTTP-Level Caching

- **Status**: ■ No Cache-Control headers in API responses
- **Required**:
    - Add appropriate `Cache-Control` headers to GET endpoints
    - Static content (logos, assets): `Cache-Control: max-age=86400`
    - API responses: `Cache-Control: no-store` for sensitive data
    - Serve uploaded images with far-future expiry headers via CDN

12. PENDING: Dependency & Supply Chain Security

12.1 Dependency Vulnerability Scanning (CRITICAL)

- **Status**: ■ No scanning in place — packages have not been audited
- **What**: Outdated or compromised packages are a major source of production breaches (Log4Shell, node-ipc, etc.)
- **Backend (Python)**:
    - Run `pip-audit` to detect CVEs in all Python dependencies
    - Run `safety check` as an alternative scanner
    - Integrate into CI/CD: fail the pipeline if high/critical CVEs found

  - **Install**: `pip install pip-audit`
  - **Run**: `pip-audit -r requirements.txt`
- **Flutter (Dart)**:
  - Run `flutter pub outdated` to list outdated packages
  - Check pub.dev for security advisories on key packages
  - Keep all packages updated, especially dio, firebase, and image libraries
- **GitHub**:
  - Enable GitHub Dependabot for automatic dependency update PRs
  - Create `.github/dependabot.yml`:

```
updates:
- package-ecosystem: "pip"
directory: "/Backend"
schedule: {interval: "weekly"}
- package-ecosystem: "pub"
directory: "/Flutter_Frontend/Badminton"
schedule: {interval: "weekly"}
```

12.2 Secret Scanning in Git History (CRITICAL)

- **Status**: ■ Not checked — .env files may have been committed in the past
- **What**: If database passwords, JWT secrets, or API keys were ever committed to Git, they are permanently in the history even if deleted later
- **Required**:
  - Run `git-secrets` or `truffleHog` to scan the entire Git history:

```
pip install truffleHog
truffleHog --regex --entropy=True .
```

  - Enable GitHub Secret Scanning (Settings → Security → Secret scanning)
  - If any secrets found in history: rotate ALL affected credentials immediately
  - Add `git-secrets` pre-commit hook to prevent future secret commits:

```
git secrets --install
git secrets --register-aws
```

  - Add to `.gitignore`: `.env`, `.env.prod`, `*.pem`, `*.p12`, `google-services.json`, `GoogleService-Info.plist`

12.3 Third-Party SDK Security

- **Status**: ■ Not assessed
- **What**: Firebase, Dio, Riverpod, and other SDKs have network access. Verify each one:
- **Required**:
  - Audit which packages make network calls
  - Review Firebase data collection settings (disable analytics data sharing if not needed)
  - Verify Dio has no insecure defaults (`badCertificateCallback` must NOT return true in production)
  - Review image_cropper and image_picker — ensure no temp file leakage
  - Check that hive_flutter local database is NOT storing sensitive data unencrypted

12.4 Lock File Management

- **Status**: ■■ pubspec.lock exists; requirements.txt has exact versions
- **Required**:
  - Commit `pubspec.lock` to Git (ensures reproducible builds)
  - Commit `requirements.txt` with pinned exact versions (already done — good)
  - Do not use version ranges (^ or >=) in production — pin exact versions for stability
  - Use `pip install --require-hashes -r requirements.txt` in production Docker image for hash verification

13. PENDING: Audit Trail & Activity Logging

13.1 User Activity Audit Log (IMPORTANT)

- **Status**: ■ No audit trail

- **What**: For a production academy management system, you need to know who did what and when — especially for financial operations (fee collection, payment recording)
- **Required**:
  - Create an `audit_logs` table in PostgreSQL:

```
CREATE TABLE audit_logs (
id SERIAL PRIMARY KEY,
user_id INTEGER NOT NULL,
user_role VARCHAR(20) NOT NULL, -- owner, coach, student
action VARCHAR(100) NOT NULL, -- CREATE_STUDENT, RECORD_PAYMENT, DELETE_BATCH
resource_type VARCHAR(50), -- student, fee, batch
resource_id INTEGER,
old_values JSONB, -- before state
new_values JSONB, -- after state
ip_address VARCHAR(45),
timestamp TIMESTAMPTZ DEFAULT NOW()
);
```

  - Log ALL critical operations:
    - Student created, updated, deleted
    - Fee payment recorded
    - Attendance marked (batch-level)
    - Coach assigned/removed from batch
    - Leave request approved/rejected
    - Announcement created/deleted
    - Password changed
    - Login / logout events
    - Failed login attempts

13.2 Financial Audit Trail (CRITICAL)

- **Status**: ■ Not implemented

- **What**: Every fee payment must have a non-repudiable record of who collected it, when, and how
- **Required**:
  - Fee payments already record `collected_by` — extend to include:
    - IP address of the device that recorded the payment
    - Timestamp (already exists)
    - Whether the record was modified and by whom
  - Prevent deletion of fee payment records (only soft-cancel with reason)
  - Generate payment receipt PDF on demand
  - Lock payment records after 24 hours (no edits)

13.3 Login Activity Tracking

- **Status**: ■ Not implemented

- **Required**:
  - Track every login: user, timestamp, IP address, device/OS
  - Track failed login attempts per account
  - Auto-lock account after 10 consecutive failed logins (notify owner via email)
  - Allow owners to view login history of their coaches/students
  - Store for minimum 90 days

14. PENDING: Mobile App Hardening

14.1 Code Obfuscation (IMPORTANT for Release)

• **Status: 🟥 Not configured**

- **What**: Release builds without obfuscation allow attackers to decompile the app and extract API URLs, secrets, or business logic
- **Required** (for both Android and iOS release builds):

```
flutter build apk --release --obfuscate --split-debug-info=build/symbols/
flutter build appbundle --release --obfuscate --split-debug-info=build/symbols/
flutter build ipa --release --obfuscate --split-debug-info=build/symbols/
```

  - Store the `symbols/` directory securely — needed for crash symbolication
  - Upload symbols to Firebase Crashlytics

14.2 Root / Jailbreak Detection (IMPORTANT)

• **Status**: ■ Not implemented

- **What**: On a rooted/jailbroken device, security protections (flutter_secure_storage encryption, certificate pinning) can be bypassed
- **Required**:
  - Detect rooted Android or jailbroken iOS devices
  - Show warning to user (do not silently block — could lock out legitimate users)
  - Option to refuse app operation on compromised devices (for sensitive roles: Owner)
  - **Package to Add**: `flutter_jailbreak_detection: ^1.8.0` or `root_detection: ^2.0.0`

14.3 Screenshot & Screen Recording Prevention (IMPORTANT)

• **Status**: ■ Not implemented

- **What**: Screens showing fee data, personal student information, guardian phone numbers, and performance records should prevent screenshots
- **Required**:
  - Enable Flutter secure flag for sensitive screens:

```
// In your app's initState for sensitive screens
FlutterWindowManager.addFlags(FlutterWindowManager.FLAG_SECURE);
```

  - OR apply globally for the entire app
  - iOS: screenshots cannot be fully prevented, but FLAG_SECURE prevents screen recording
  - **Package to Add**: `flutter_windowmanager: ^0.2.0` (Android)

14.4 Certificate Pinning (RECOMMENDED)

• **Status**: ■ Not implemented

- **What**: Without certificate pinning, a man-in-the-middle attack (using a custom root CA) can intercept all API traffic — including credentials and personal data
- **Required**:
  - Pin your backend server's SSL certificate public key in the app
  - Use Dio's `BadCertificateCallback` and `trustedCertificates` for pinning
  - Update pinned certificate before it expires (keep backup pin for rotation)
  - Note: Not recommended for apps not yet in production — implement after you have a stable certificate
  - **Implementation**:

```
(dio.httpClientAdapter as IOHttpClientAdapter).createHttpClient = () {
final client = HttpClient();
client.badCertificateCallback = (cert, host, port) => false; // Reject bad certs
// Add pinned certificate bytes here
return client;
};
```

14.5 Anti-Tampering / Reverse Engineering Protection

- **Status**: ■ Not implemented
- **What**: Attackers can decompile the APK to extract API endpoints, modify business logic, or create fake clients
- **Required**:
  - Enable ProGuard/R8 for Android release (already partially configured in Flutter releases)
  - Do NOT hardcode secrets (API keys, JWT secrets) in Flutter app code
  - Use environment variables / dart-define for build-time config:

```
flutter build apk --dart-define=API_URL=https://api.shuttler.app
```

  - Consider Flutter's `--obfuscate` flag (see 14.1)

14.6 Biometric Authentication (NICE TO HAVE)

- **Status**: ■ Not implemented
- **What**: Allow owners and coaches to unlock the app with Face ID / fingerprint instead of re-entering password
- **Required**:
  - Add biometric auth as optional second factor on app re-open (after background)
  - Store auth token securely; use biometric to unlock access to token
  - **Package to Add**: `local_auth: ^2.2.0`

14.7 App Transport Security (iOS ATS)

- **Status: ■ Not configured for HTTPS**
- **What**: iOS requires all network requests to use HTTPS (App Transport Security). Any HTTP calls will be blocked.
- **Required**:
  - Ensure ALL API calls use HTTPS before iOS submission
  - Remove any `NSAllowsArbitraryLoads: true` exceptions from `Info.plist`
  - Test all network calls with ATS enforced (do NOT use `NSExceptionDomains` to whitelist your own domain — use HTTPS properly)

---

15. PENDING: CI/CD Pipeline

10.1 GitHub Actions Workflow (CRITICAL)

- **Status**: ■ No CI/CD configured
- **Required**: Create `.github/workflows/` directory with:

`backend-ci.yml` **— Backend CI**

```
Trigger: on push to main and on pull requests

Jobs:
1. lint: Run ruff/flake8 Python linter
2. test: Run pytest test suite
3. security-scan: Run bandit (Python security linter)
4. build: Build Docker image and push to registry
5. deploy-staging: Deploy to staging environment
6. deploy-prod: Deploy to production (on merge to main, with approval gate)
```

`flutter-ci.yml` **— Flutter CI**

```
Trigger: on push to main and on pull requests

Jobs:
1. analyze: Run flutter analyze (Dart static analysis)
2. test: Run flutter test (unit + widget tests)
3. build-android: Build Android APK/AAB (release)
4. build-ios: Build iOS IPA (release)
5. deploy-android: Upload to Google Play Internal Track
6. deploy-ios: Upload to TestFlight
```

10.2 Environments (CRITICAL)

• **Status**: ■ Only one environment (local development)

• **Required** — Three environments:

Environment	Purpose	Backend URL
-------------	---------	-------------
Development	Local development, feature branches	http://localhost:8000
Staging	Integration testing, QA	https://api-staging.shuttler.app
Production	Live app	https://api.shuttler.app

• Use Flutter environment files (`.env.dev`, `.env.staging`, `.env.prod` with `flutter_dotenv`)

• Never deploy directly to production without passing staging

10.3 Docker Containerization (IMPORTANT)

• **Status**: ■ No Docker configuration

• **Required**:

  • Create `Backend/Dockerfile` for FastAPI app

  • Create `docker-compose.yml` for local development (app + postgres + redis)

  • Multi-stage Docker build (smaller production image)

  • Health check in Dockerfile: `HEALTHCHECK CMD curl -f http://localhost:8000/health || exit 1`

10.4 Infrastructure as Code

• **Status**: ■ No IaC

• **Optional but Recommended**:

  • Terraform or Pulumi for cloud resource provisioning

  • Or use Railway/Render (managed platforms that handle IaC for you)

  • Document all cloud resources and their configurations

10.5 Semantic Versioning & Release Management

• **Status**: ■ App version is 1.0.0+1 (hardcoded in pubspec.yaml)

• **Required**:

  • Auto-increment build number in CI/CD pipeline

  • Use semantic versioning (MAJOR.MINOR.PATCH)

  • Create Git tags for each release

  • Maintain a `CHANGELOG.md`

  • Use release branches strategy (main, develop, feature branches)

---

16. PENDING: Testing

11.1 Backend Testing (CRITICAL)

- **Status**: ■ No tests exist (pytest is in requirements but commented out)
- **Required**:
  - Unit tests for business logic (fee calculation, status calculation, invitation token generation)
  - Integration tests for all API endpoints (using `httpx` TestClient)
  - Database tests with test database (not production)
  - Minimum 70% code coverage target
  - Test authentication flows (login, token refresh, unauthorized access)
  - Test role-based access (owner actions rejected for coach/student roles)
  - **Packages to Add**: `pytest==7.4.3`, `httpx==0.25.2`, `pytest-asyncio==0.21.1`

11.2 Flutter Testing (CRITICAL)

- **Status**: ■ Only boilerplate test exists (`widget_test.dart`)
- **Required**:
  - Unit tests for all service classes (auth, fee, student, batch, etc.)
  - Unit tests for all provider logic
  - Widget tests for key screens (login, dashboard, forms)
  - Integration tests for critical user flows:
    - Login → Dashboard → Mark Attendance
    - Login → Add Student → View Student
    - Login → Record Fee Payment → View Updated Status
  - Mock HTTP responses using `mockito` or `mocktail`
  - **Packages to Add**: `mockito: ^5.4.4`, `mocktail: ^1.0.3`, `flutter_test` (already included)

11.3 End-to-End (E2E) Testing

- **Status**: ■ Not implemented
- **Required** (post-launch):
  - Flutter Integration Tests using `integration_test` package
  - Test on real devices (iOS and Android) before each release
  - Automate in CI/CD on physical device farms (Firebase Test Lab or BrowserStack)

11.4 Performance Testing

- **Status**: ■ Not implemented
- **Required**:
  - Load test backend API with `locust` or `k6`
  - Test with realistic data volume (500 students, 50 coaches, 100 batches, 10,000 attendance records)
  - Identify and fix N+1 query problems
  - Target: API responses < 300ms at p95 under normal load

11.5 Security Testing

- **Status**: ■ Not done
- **Required**:
  - Run `bandit` (Python security linter) on backend code
  - Run `OWASP ZAP` or `Burp Suite` for API vulnerability scanning
  - Test for SQL injection on all input fields
  - Test for authentication bypass
  - Test for privilege escalation (student accessing owner endpoints)
  - Test rate limiting is working correctly

17. PENDING: Monitoring, Logging & Alerting

12.1 Structured Logging (CRITICAL)

- **Status**: ■ No structured logging — only print statements
- **Required**:
  - Replace all `print()` with Python's `logging` module
  - Use structured JSON logging for production
  - Log levels: DEBUG (dev), INFO (staging/prod), WARNING, ERROR, CRITICAL
  - Log key events:
    - Every API request (method, path, status code, duration)
    - Authentication attempts (success and failure)
    - Errors and exceptions (with stack traces)
    - Background job execution results
  - **Package to Add**: `loguru==0.7.2` (excellent structured logging for Python)

12.2 Error Tracking (CRITICAL)

- **Status**: ■ No error tracking
- **Required**:
  - Integrate Sentry for both backend and Flutter
  - Backend: `sentry-sdk[fastapi]` — captures every unhandled exception with full context
  - Flutter: `sentry_flutter` — captures crashes, UI errors, slow frames
  - Configure error grouping and assignment
  - Set up alert rules for new errors
  - **Packages**:
    - Backend: `sentry-sdk[fastapi]==1.38.0`
    - Flutter: `sentry_flutter: ^8.4.0`

12.3 Crash Reporting (CRITICAL for Mobile)

- **Status**: ■ No crash reporting
- **Required**:
  - Firebase Crashlytics (already have Firebase in project — just add crashlytics)
  - Captures all Flutter crashes automatically
  - Links crash reports to user sessions
  - **Package to Add**: `firebase_crashlytics: ^4.1.3`

12.4 Performance Monitoring

- **Status**: ■ No APM (Application Performance Monitoring)
- **Required**:
  - Firebase Performance Monitoring for Flutter app (measures screen load times, network latency)
  - Backend APM: Sentry Performance or Datadog (traces slow API calls)
  - Database query monitoring: log slow queries (PostgreSQL `log_min_duration_statement = 1000`)
  - **Package to Add**: `firebase_performance: ^0.10.0+6`

12.5 Health Check Endpoints

- **Status**: ■ No health endpoints
- **Required**:

```
GET /health → {"status": "ok", "timestamp": "..."}
GET /health/db → {"status": "ok", "latency_ms": 5}
GET /health/redis → {"status": "ok"} (when Redis is added)
GET /health/detailed → All component statuses (owner-only or internal)
```

12.6 Uptime Monitoring

- **Status: 🟥 Not configured**
- **Required**:
  - UptimeRobot (free tier) or Better Uptime for health check monitoring
  - Alert via email/Slack when backend goes down
  - Target uptime SLA: 99.5% (production)

12.7 Analytics

- **Status**: ■ No analytics
- **Required**:
  - Firebase Analytics (already have Firebase — just enable)
  - Track key events: login, screen views, fee payment recorded, attendance marked, report generated
  - Understand user behavior and feature usage
  - **Package to Add**: `firebase_analytics: ^11.3.3`

---

18. PENDING: Infrastructure & Cloud Deployment

13.1 Cloud Provider Selection

- **Status**: ■ Running on localhost only
- **Recommended Options**:

Option	Backend	Database	Cost	Ease
--------	---------	----------	------	------
Railway.app	■ FastAPI	■ PostgreSQL	$5-20/month	Very Easy
Render.com	■ FastAPI	■ PostgreSQL	$7-25/month	Easy
AWS (EC2 + RDS)	■ FastAPI	■ PostgreSQL	$30-100/month	Complex
Google Cloud Run	■ FastAPI	■ Cloud SQL	$10-50/month	Medium
Fly.io	■ FastAPI	■ PostgreSQL	$5-15/month	Easy

Recommendation: Start with Railway.app or Render.com for simplicity, migrate to AWS when scale demands it.

13.2 File Storage Migration

- **Status**: ■ Files stored on local disk (will be lost on server restart)
- **Required**:
  - Migrate file uploads to cloud object storage:
    - AWS S3 (most common)
    - Cloudflare R2 (cheaper, S3-compatible)
    - Google Cloud Storage
  - Update upload endpoint to store to cloud instead of local disk
  - Serve files via CDN for performance
  - Update Flutter to load images from CDN URLs

- **Package to Add**: `boto3==1.34.0` (for AWS S3)

13.3 Load Balancing & Scaling

- **Status**: ■ Single instance only
- **Required** (after initial launch):
  - Horizontal scaling: multiple FastAPI instances behind load balancer
  - Stateless API design (no server-side sessions — JWTs enable this)
  - Connection pooling via PgBouncer (for very high traffic)
  - Auto-scaling rules based on CPU/memory usage

13.4 Domain & DNS

- **Status**: ■ No domain configured
- **Required**:
  - Register a domain name for the API (e.g., `api.shuttler.app`)
  - Configure DNS records pointing to backend server
  - Configure SSL certificate for domain
  - Update Flutter API base URL constant

13.5 Environment Configuration for Flutter

- **Status**: ■■ flutter_dotenv installed but configuration unclear
- **Required**:
  - Separate `.env.dev`, `.env.staging`, `.env.prod` files
  - Load appropriate env file based on build mode
  - Never commit `.env.prod` to Git
  - API base URL, Firebase config, and feature flags in env files

---

19. PENDING: Mobile App Store Requirements

14.1 Google Play Store (Android)

- **Status**: ■ Not prepared
- **Required**:
  - **App Signing**: Generate release keystore, configure in `android/app/build.gradle`
  - **Build Type**: Configure release build (minification, obfuscation): `flutter build appbundle --release`
  - **ProGuard Rules**: Add rules to prevent code stripping for Flutter plugins
  - **Metadata**:
    - App name, short description (80 chars), full description (4000 chars)
    - App icon (512x512 PNG)
    - Feature graphic (1024x500 PNG)
    - Screenshots (minimum 2, up to 8) for phone and tablet
    - Categorization: Education or Sports
  - **Privacy Policy URL**: Required — host a privacy policy page
  - **Data Safety Section**: Declare what data is collected (user data, photos, location if any)
  - **Content Rating**: Fill IARC questionnaire
  - **Target SDK**: Must target Android API 34+ (current requirement)
  - **App Bundle**: Submit `.aab` format (not `.apk`) for Play Store
  - **Permissions**: Review AndroidManifest.xml — justify all requested permissions
  - **Firebase**: Update `google-services.json` with production credentials

14.2 Apple App Store (iOS)

- **Status**: ■ Not prepared
- **Required**:
  - **Apple Developer Account**: Paid ($99/year)
  - **App ID & Bundle ID**: Register `com.yourcompany.shuttler` in Apple Developer Portal
  - **Provisioning Profile**: Distribution certificate + provisioning profile
  - **Build**: `flutter build ipa --release`
  - **Xcode Configuration**:
    - Set deployment target (iOS 13.0 minimum recommended)
    - Update `Info.plist` for required permissions (camera, photo library access)
    - Configure push notification entitlement
    - Configure `GoogleService-Info.plist` with production Firebase config
  - **Metadata** (in App Store Connect):
    - App name, subtitle, description, keywords
    - App icon (1024x1024 PNG)
    - Screenshots for all required device sizes (iPhone 6.7", 6.5", iPad)
    - Privacy Policy URL (required)
    - App Privacy Nutrition Labels (what data is collected and how used)
    - Age Rating questionnaire
  - **Review Guidelines Compliance**:
    - No placeholder content
    - All features must function (no "coming soon" placeholders)
    - Login credentials for reviewer (create test account)
    - In-app purchases declared (if any)

14.3 Deep Linking (IMPORTANT)

- **Status: 🟥 Not configured**
- **Required**:
  - Student invitation links must open the app (not a browser)
  - Android: App Links (Digital Asset Links file at `/.well-known/assetlinks.json`)
  - iOS: Universal Links (`/.well-known/apple-app-site-association` file)
  - Configure in GoRouter to handle deep link paths
  - **Package to Add**: Built into `go_router` — configure link scheme

14.4 App Permissions

- **Status**: ■■ Some permissions configured, not all justified
- **Required**:
  - Camera permission (for profile photo via camera)
  - Photo library permission (for profile photo from gallery)
  - Notification permission (for FCM push notifications)
  - Network access (always required)
  - Request permissions at the right time (not on app launch — request when feature is used)
  - Handle permission denied gracefully with explanation
  - **Package to Add**: `permission_handler: ^11.3.1`

14.5 App Icon & Splash Screen

- **Status**: ■■ Default Flutter icon (blue logo); no custom splash screen
- **Required**:
  - Custom app icon (1024x1024 master, adaptive icon for Android)

    - Custom splash screen with brand colors and logo
    - **Packages to Add**:
        - `flutter_launcher_icons: ^0.13.1`
        - `flutter_native_splash: ^2.4.0`

14.6 App Review Preparation

- **Status**: ■ Not prepared
- **Required**:
    - Create demo / reviewer account for App Store review (Apple requires working login)
    - Document all features for reviewer notes
    - Ensure app works without network (show appropriate empty/offline states)
    - Ensure all required permissions are justified and used

---

20. PENDING: Performance Optimization

15.1 Backend Query Optimization

- **Status**: ■ Multiple N+1 query risks identified
- **Required**:
    - Use SQLAlchemy `joinedload()` / `selectinload()` for relationships instead of lazy loading
    - Add database indexes (see section 7.3)
    - Cache frequently read data in Redis
    - Paginate all list endpoints (see section 8.2)
    - Profile slow queries with `EXPLAIN ANALYZE` in PostgreSQL
    - Add query timeout (30 seconds max)

15.2 Image Optimization

- **Status**: ■ Full-size images served without optimization
- **Required**:
    - Resize images on upload (thumbnail + full size)
    - Convert to WebP format for 30-50% size reduction
    - Serve via CDN with compression (gzip/brotli)
    - Use `cached_network_image` (already installed) for Flutter-side caching
    - Lazy load images in list views

15.3 Flutter App Performance

- **Status**: ■■ Skeleton screens implemented; other optimizations needed
- **Required**:
    - Run `flutter analyze` and fix all warnings
    - Profile with Flutter DevTools — identify jank/slow frames
    - Minimize rebuilds: use `const` constructors where possible
    - Use `ListView.builder` (not `ListView`) for all long lists
    - Implement `keepAlive` for tab screens to avoid re-fetching
    - Reduce app size: run `flutter build apk --split-per-abi`
    - Enable tree shaking and code obfuscation in release builds:

```
flutter build apk --release --obfuscate --split-debug-info=symbols/
```

15.4 Startup Time Optimization

- **Status**: ■ Not measured/optimized
- **Required**:
  - Profile cold start time (target < 2 seconds)
  - Defer non-critical initializations to after first frame
  - Minimize `main.dart` initialization blocking operations
  - Use deferred loading for rarely-used features

---

21. PENDING: Error Handling & Resilience

16.1 Global Exception Handler (Backend)

- **Status**: ■ No global exception handler — unhandled exceptions return 500 with stack traces
- **Required**:

```
@app.exception_handler(Exception)
async def global_exception_handler(request, exc):
# Log the error
logger.error(f"Unhandled exception: {exc}", exc_info=True)
# Return safe response (no stack trace in production)
return JSONResponse(
status_code=500,
content={"success": False, "message": "Internal server error"}
)
```

  - Never expose stack traces to clients in production

16.2 Flutter Error Handling

- **Status**: ■■ Some error handling exists; not systematic
- **Required**:
  - Global `FlutterError.onError` handler (log and report to Sentry)
  - Global `PlatformDispatcher.instance.onError` handler
  - Handle all Dio errors uniformly (network, timeout, auth, server errors)
  - Show user-friendly error messages (not technical details)
  - Retry mechanism for transient network errors (with exponential backoff)
  - Graceful degradation when offline

16.3 API Timeout Configuration

- **Status**: ■ No timeouts configured in Dio
- **Required**:
  - Set connection timeout: 30 seconds
  - Set receive timeout: 60 seconds (longer for file uploads)
  - Set send timeout: 30 seconds
  - Handle timeout errors gracefully in UI

16.4 Circuit Breaker Pattern

- **Status**: ■ Not implemented
- **Optional** (for production resilience):
  - After N consecutive failures, stop making API calls and return cached/fallback data
  - Auto-recover after a cooldown period

---

22. PENDING: Privacy, Compliance & Legal

17.1 Privacy Policy (CRITICAL — Required for App Stores)

- **Status**: ■ No privacy policy exists
- **Required**:
  - Write a comprehensive privacy policy covering:
    - What personal data is collected (name, phone, email, date of birth, guardian info, photos)
    - Why it's collected (academy management purposes)
    - How it's stored and protected
    - Who has access (owner, coaches as appropriate)
    - How long data is retained
    - User rights (access, correction, deletion)
    - How to contact for privacy requests
  - Host it at a public URL (e.g., `https://shuttler.app/privacy`)
  - Reference in app (Settings → Privacy Policy link)
  - Required before App Store / Play Store submission

17.2 Terms of Service

- **Status**: ■ No ToS exists
- **Required**:
  - Write Terms of Service for academy owners and students
  - Host at public URL (e.g., `https://shuttler.app/terms`)
  - Reference in signup flow and settings

17.3 PIPEDA / Privacy Law Compliance (Canada)

- **Status**: ■ Not assessed (app is for Canadian badminton academy)
- **Required** (for Canadian market):
  - Comply with Canada's Personal Information Protection and Electronic Documents Act (PIPEDA)
  - Obtain explicit consent before collecting personal data
  - Provide right to access and correct personal information
  - Right to withdrawal of consent (account deletion)
  - Notify users within 72 hours of a data breach
  - Consider provincial laws (PIPA in Alberta/BC if applicable)

17.4 COPPA / Child Protection

- **Status**: ■ Not assessed
- **Required**:
  - Academy may have underage students (badminton is popular with youth)
  - Do NOT allow children under 13 to create accounts directly
  - Require parental/guardian consent for minors
  - Guardian information is already collected — ensure it's used properly
  - Review App Store age rating settings

17.5 GDPR (If EU Users)

- **Status**: ■ Not assessed
- **Required** (if any EU users):
  - Data processing consent
  - Right to erasure (full account deletion that removes all personal data)
  - Data portability (export user's own data)

- Data Protection Officer appointment

17.6 Account Deletion

- **Status**: ■ No account deletion feature
- **Required** (App Store requires this since 2022 for new apps with accounts):
  - Implement account deletion in app settings
  - Deletion must remove or anonymize all personal data
  - Cannot just soft-delete — must offer true data deletion
  - Backend endpoint: `DELETE /users/me/account`

---

23. PENDING: Notifications (FCM)

18.1 Firebase Backend Admin SDK

- **Status**: ■ `firebase-admin` not installed (commented out in requirements.txt)
- **Required**:
  - Install: `firebase-admin==6.3.0`
  - Configure service account credentials (from Firebase Console)
  - Implement `send_push_notification(fcm_token, title, body, data)` utility function

18.2 Notification Triggers to Implement

- **Status**: ■■ Fee notifications done; others pending
- **Required** (add to respective backend endpoints):

Event	Trigger	Recipient
-------	---------	-----------
Attendance marked	After attendance POST	Student (present/absent)
Performance recorded	After performance POST	Student
BMI recorded	After BMI POST	Student
Announcement published	After announcement POST	Target audience
Leave approved/rejected	After leave status update	Coach
Fee overdue	Daily cron job	Student
Fee payment received	After fee payment POST	Student
Invitation sent	After invitation POST	Invited person (WhatsApp link)
Report generated	After report generation	Owner

18.3 Notification Preferences

- **Status**: ■ No user preference controls
- **Required**:
  - Allow users to control which notifications they receive
  - Settings screen: toggle for each notification type
  - Store preferences in user profile
  - Respect preferences in backend trigger logic

18.4 In-App Notification Center

- **Status**: ■ Notification screen exists
- **Required**:
  - Verify notification read/unread status works correctly
  - Add notification badge count on home screen icon
  - Mark all as read functionality
  - Notification tap action navigates to relevant screen

---

24. PENDING: Accessibility

19.1 Screen Reader Support

- **Status**: ■ Not tested/configured
- **Required**:
  - Add `Semantics` widgets to custom components
  - Ensure all interactive elements have meaningful labels
  - Test with TalkBack (Android) and VoiceOver (iOS)
  - Add `excludeSemantics: true` to purely decorative elements

19.2 Text Scaling

- **Status**: ■ Not tested
- **Required**:
  - Test app with system font size set to largest setting
  - Ensure no text overflow or clipped content
  - Use `MediaQuery.textScaleFactor` where needed

19.3 Color Contrast

- **Status**: ■■ Dark neumorphic theme — contrast may be insufficient
- **Required**:
  - Check text contrast ratios meet WCAG 2.1 AA standard (4.5:1 for normal text)
  - Run colors through contrast checker
  - Adjust colors that don't meet minimum contrast

---

25. PENDING: Documentation

20.1 API Documentation

- **Status**: ■■ Swagger auto-generated; not comprehensive
- **Required**:
  - Document all endpoints with request/response examples
  - Document authentication requirements
  - Document role permissions per endpoint
  - Create Postman collection for API testing
  - Consider hosting API docs separately (Redoc or GitBook)

20.2 Developer Onboarding

- **Status**: ■■ Partial (README.md and SETUP.md exist)
- **Required**:

- Complete local setup guide (backend + Flutter + PostgreSQL)
- Common troubleshooting guide
- Architecture decision records (ADRs) for major decisions
- Git branching strategy documentation
- Code style guide

20.3 User Manual

- **Status**: ■ No end-user documentation
- **Required**:
  - In-app help/tour for first-time users
  - FAQ section
  - How-to guides for key features (marking attendance, recording payments, etc.)

20.4 Operations Runbook

- **Status**: ■ None
- **Required**:
  - How to deploy new backend version
  - How to run database migrations in production
  - How to scale the system
  - How to restore from backup
  - Incident response procedures (what to do when app goes down)

21. Production Launch Checklist

Phase A: Security & Auth (Must Do Before Any Public Access)

🟥 Implement JWT authentication (access + refresh tokens)
🟥 Switch to `flutter_secure_storage` from SharedPreferences
🟥 Implement role-based authorization on ALL backend endpoints
🟥 Fix IDOR: enforce ownership checks on all student/coach/fee/attendance/performance/BMI endpoints
🟥 Fix mass assignment: audit all Pydantic update schemas (remove role, id, status from writable fields)
🟥 Fix account enumeration: return same error message for wrong email and wrong password
🟥 Fix path traversal: enforce server-generated UUID filenames on ALL file uploads
🟥 Configure HTTPS/TLS for backend
🟥 Fix CORS to restrict to specific origins (not *)
🟥 Add rate limiting (login: 5/15min, forgot password: 3/hour, general API: 100/min)
🟥 Add API usage capping per academy (10,000 calls/day)
🟥 Input validation and file upload security (MIME type + magic bytes + size limit)
🟥 Secrets management (no .env in git, use cloud secrets, rotate all credentials)
🟥 Add security headers via Nginx
🟥 Scan Git history for leaked secrets (truffleHog)
🟥 Run pip-audit on backend dependencies

Phase B: Core Features (Must Do Before Beta)

🟥 Multiple coach assignment per batch
🟥 Partial payment status
🟥 Session-wise reports
🟥 All notification triggers (attendance, performance, BMI, announcements)

- 🟥 Payment method restriction (Cash/Card only)
- 🟥 Performance entry completion status tracker

Phase C: Infrastructure (Must Do Before Launch)

- 🟥 Configure Alembic migrations properly (convert all manual SQL scripts)
- 🟥 Add database indexes (attendance, fees, performance, notifications)
- 🟥 Deploy backend to cloud (Railway/Render/AWS)
- 🟥 Set up PostgreSQL on cloud (with SSL, automated backups, PITR)
- 🟥 Migrate file storage to S3/Cloudflare R2 (remove local disk dependency)
- 🟥 Set up Redis cache (batch/student lists, academy details)
- 🟥 Configure 3 environments (dev/staging/prod) with separate .env files
- 🟥 Domain name + SSL certificate
- 🟥 Health check endpoints (/health, /health/db)
- 🟥 CI/CD pipeline (GitHub Actions: lint → test → build → deploy)
- 🟥 Docker containerization (Dockerfile + docker-compose)
- 🟥 Set up storage quota tracking per academy
- 🟥 Enable GitHub Dependabot for automatic dependency updates

Phase D: Testing (Must Do Before Launch)

- 🟥 Backend unit and integration tests (minimum 70% coverage)
- 🟥 Flutter unit and widget tests
- 🟥 API security testing (role bypass, SQL injection)
- 🟥 Performance testing (load test)
- 🟥 Test on multiple Android devices and iOS versions
- 🟥 Full E2E flow testing for all 3 user roles

Phase E: Mobile App Hardening (Must Do Before Submission)

- 🟥 Enable code obfuscation in release builds (--obfuscate --split-debug-info)
- 🟥 Add root/jailbreak detection warning (flutter_jailbreak_detection)
- 🟥 Enable screenshot prevention on sensitive screens (flutter_windowmanager)
- 🟥 Remove any NSAllowsArbitraryLoads from Info.plist; enforce HTTPS (ATS)
- 🟥 Verify no secrets hardcoded in Flutter source code (use dart-define)

Phase F: Mobile App Store (Must Do Before Submission)

- 🟥 Write and host Privacy Policy (required: name, contact, data collected, retention)
- 🟥 Write and host Terms of Service
- 🟥 Implement account deletion feature (App Store policy since 2022)
- 🟥 App icon and splash screen (custom branded)
- 🟥 Deep linking for invitation system (Universal Links / App Links)
- 🟥 Request permissions at right time with explanation (permission_handler)
- 🟥 App store metadata (screenshots, description, keywords, ratings)
- 🟥 Android release signing configuration (keystore + build.gradle)
- 🟥 iOS provisioning profile and signing (Apple Developer account)
- 🟥 Firebase Crashlytics integration (crash reporting)
- 🟥 Firebase Analytics integration (usage insights)
- 🟥 TestFlight beta testing (iOS — minimum 2 weeks before submission)
- 🟥 Google Play Internal Testing (Android — minimum 1 week)
- 🟥 Create demo/reviewer account for App Store review team

Phase G: Monitoring (Must Do After Launch)

- 🟥 Sentry error tracking (backend + Flutter)
- 🟥 Firebase Crashlytics activated
- 🟥 Firebase Performance monitoring activated
- 🟥 Uptime monitoring (UptimeRobot)
- 🟥 Database slow query logging
- 🟥 Structured logging in backend
- 🟥 Alerting for critical errors and downtime

Phase H: Post-Launch Features

- 🟥 Video library for students
- 🟥 Digital waiver system
- 🟥 Advanced analytics dashboard for owner
- 🟥 Multi-language support (if needed)
- 🟥 Dark/Light mode toggle (currently only dark)
- 🟥 Bulk attendance import (CSV)
- 🟥 Automated fee reminders (configurable schedule)

---

22. Priority Roadmap

Sprint 1 — Security Foundation (Weeks 1–2)

Goal: Make the app secure enough for real users

Task	Effort	Priority
------	--------	----------
Implement JWT auth (backend)	High	CRITICAL
flutter_secure_storage migration	Low	CRITICAL
Role-based authorization on all endpoints	Medium	CRITICAL
Rate limiting on auth endpoints	Low	CRITICAL
Input validation and sanitization	Medium	CRITICAL
CORS restriction	Very Low	CRITICAL

Sprint 2 — Pending App Features (Weeks 3–4)

Goal: Complete all remaining app-level features

Task	Effort	Priority
------	--------	----------
Multiple coach assignment	Medium	HIGH
Partial payment status	Low	HIGH
Payment method restriction (Cash/Card)	Very Low	HIGH
Session-wise reports	Medium	HIGH
All notification triggers (FCM)	Medium	HIGH
Performance entry completion status	Low	MEDIUM

Sprint 3 — Infrastructure & Testing (Weeks 5–6)

Goal: Production-ready infrastructure

Task	Effort	Priority
------	--------	----------
Alembic migrations setup	Medium	CRITICAL
Cloud deployment (Railway/Render)	Medium	CRITICAL
S3 file storage migration	Medium	HIGH
Redis cache setup	Medium	HIGH
Backend tests (unit + integration)	High	CRITICAL
Flutter tests (unit + widget)	Medium	HIGH
CI/CD pipeline (GitHub Actions)	Medium	CRITICAL
Docker containerization	Low	HIGH

Sprint 4 — App Store Preparation (Weeks 7–8)

Goal: Ready for App Store submission

Task	Effort	Priority
------	--------	----------
Privacy Policy + Terms of Service	Medium	CRITICAL
Account deletion feature	Medium	CRITICAL
App icon + splash screen	Low	HIGH
Deep linking for invitations	Low	HIGH
Permission handler integration	Low	HIGH
App store metadata & screenshots	Medium	CRITICAL
Firebase Crashlytics	Very Low	HIGH
Firebase Analytics	Very Low	MEDIUM
Release signing (Android + iOS)	Medium	CRITICAL
TestFlight / Internal testing beta	Low	CRITICAL

Sprint 5 — Launch & Post-Launch (Weeks 9–10)

Goal: Live in production with monitoring

Task	Effort	Priority
------	--------	----------
Sentry error tracking	Very Low	CRITICAL
Uptime monitoring	Very Low	HIGH
Database backup verification	Low	CRITICAL
Load testing	Medium	HIGH
App Store submission (iOS)	Medium	CRITICAL

Task	Effort	Priority
Play Store submission (Android)	Medium	CRITICAL
Video library feature	High	LOW
Digital waiver system	High	LOW

Summary

Category	Status	Critical Gaps
----------	--------	--------------
Core App Features	~85% Complete	Multiple coach, partial payment, session reports
Security & Authentication	~15% Complete	JWT missing, no role enforcement on backend
IDOR & Business Logic Security	0% Complete	Any user can access any other user's data
Usage Capping & Quotas	0% Complete	No storage limits, no API caps
Database & Migrations	~60% Complete	Alembic not configured, no indexes, no backup
API Layer Quality	~40% Complete	No versioning, no pagination, no standardization
Caching	0% Complete	Every request hits PostgreSQL
Dependency & Supply Chain Security	0% Complete	No vulnerability scanning, no secret scanning
Audit Trail & Activity Logging	0% Complete	No record of who changed what
Mobile App Hardening	0% Complete	No obfuscation, no root detection, no cert pinning
CI/CD Pipeline	0% Complete	Manual deployments only
Automated Testing	~5% Complete	No backend tests, no Flutter tests
Monitoring & Logging	~5% Complete	No error tracking, no crash reporting
Cloud Infrastructure	0% Complete	Running on localhost
App Store Readiness	~10% Complete	No icon, no privacy policy, no signing
Performance Optimization	~30% Complete	N+1 queries, no pagination
Privacy & Legal	0% Complete	No privacy policy, no ToS, no PIPEDA assessment
Overall Production Readiness	**~30%**	Security holes are the biggest risk

Estimated effort to production: 8–10 weeks with 2 developers working full-time, or 16–20 weeks solo.

> ***Critical Note****: The biggest risk for production is the complete absence of JWT authentication and IDOR protection. As it stands, ANY authenticated user (even a student) can call any API endpoint and access ANY other user's data simply by guessing an ID. This must be the first thing fixed before any real user data is stored.*

Document maintained by: Development Team

Last Updated: February 2026

Next Review: After Sprint 1 completion