Technical Requirement Document: HostelPulse v1

Version: 1.0
Date: February 26, 2026
Status: Draft
Related: PRD.md

1. System Architecture Overview

1.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                          CLIENT LAYER                            │
├──────────────────────────┬──────────────────────────────────────┤
│  Android Mobile App      │    Web Admin Dashboard               │
│  (React Native/Flutter)  │    (React/Vue.js)                    │
│  - Student               │    - Warden                          │
│  - Guard                 │                                      │
│  - Staff                 │                                      │
└──────────────┬───────────┴──────────────────┬───────────────────┘
               │                              │
               │         HTTPS (TLS 1.3)      │
               │                              │
┌──────────────▼──────────────────────────────▼───────────────────┐
│                    API GATEWAY / LOAD BALANCER                   │
│                    (Nginx / AWS ALB)                             │
└──────────────┬──────────────────────────────────────────────────┘
               │
┌──────────────▼──────────────────────────────────────────────────┐
│                      APPLICATION LAYER                           │
│  ┌────────────────────────────────────────────────────────────┐ │
│  │        REST API Server (FastAPI / Django REST)             │ │
│  │  - Authentication Service                                  │ │
│  │  - Student Management Service                              │ │
│  │  - Attendance Service                                      │ │
│  │  - Leave Management Service                                │ │
│  │  - Grievance Service                                       │ │
│  │  - Messaging Service                                       │ │
│  │  - Payment Tracking Service                                │ │
│  │  - Notification Service                                    │ │
│  └────────────────────────────────────────────────────────────┘ │
│                                                                   │
│  ┌────────────────────────────────────────────────────────────┐ │
│  │        WebSocket Server (Socket.io / Django Channels)      │ │
│  │  - Real-time messaging                                     │ │
│  │  - Live attendance updates                                 │ │
│  └────────────────────────────────────────────────────────────┘ │
└────┬─────────────────────────────┬───────────────────────────────┘
     │                             │
┌────▼──────────┐      ┌───────────▼────────────┐
│  CACHE LAYER  │      │  MESSAGE QUEUE          │
│  (Redis)      │      │  (Redis / RabbitMQ)     │
│               │      │  - Async tasks          │
│ - Sessions    │      │  - Notifications        │
│ - QR tokens   │      │  - File processing      │
│ - Rate limits │      └─────────────────────────┘
└───────────────┘
     │
┌────▼──────────────────────────────────────────────────────────┐
│                      DATA LAYER                                │
│  ┌───────────────────────┐    ┌───────────────────────────┐  │
│  │  PostgreSQL           │    │  File Storage             │  │
│  │  (Primary Database)   │    │  (AWS S3 / MinIO)         │  │
│  │                       │    │  - Documents              │  │
│  │  - User data          │    │  - Images                 │  │
│  │  - Attendance logs    │    │  - QR codes               │  │
│  │  - Leave requests     │    └───────────────────────────┘  │
│  │  - Grievances         │                                    │
│  │  - Messages           │                                    │
│  │  - Payments           │                                    │
│  └───────────────────────┘                                    │
└────────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────────┐
│                      EXTERNAL SERVICES                           │
│  - Email Provider (SendGrid / AWS SES)                          │
│  - SMS Gateway (Twilio / AWS SNS) [Future]                      │
│  - Push Notifications (Firebase Cloud Messaging)                │
│  - Geolocation APIs (Device native GPS)                         │
└─────────────────────────────────────────────────────────────────┘
```

1.2 Architecture Principles

1. **Monolithic Start, Modular Design**: Single deployable unit with service-oriented modules for easy future extraction
2. **Stateless API Layer**: All session state in Redis, enables horizontal scaling
3. **Database as Source of Truth**: No critical business logic in cache
4. **Async Processing**: Background jobs for non-critical operations (emails, notifications)
5. **Fail-Safe Defaults**: System degrades gracefully (attendance queue if offline)

1.3 Technology Stack Selection

Layer	Technology	Justification
Mobile	React Native	Single codebase, large ecosystem, faster development
Web	React + Vite	Modern, performant, great developer experience
Backend	FastAPI (Python 3.11+)	Fast, async support, auto-generated docs, type safety
Database	PostgreSQL 15+	ACID compliance, JSON support, robust, well-tested
Cache	Redis 7+	In-memory speed, pub/sub for real-time, simple
File Storage	AWS S3 / MinIO	Scalable, cost-effective, standard APIs
Queue	Redis (Celery)	Reuse Redis, simple setup for V1
WebSocket	Socket.io / FastAPI WebSockets	Real-time messaging and live updates
Web Server	Nginx	Reverse proxy, static files, load balancing

Alternative for smaller scale: Replace FastAPI + PostgreSQL + Redis with **Django + PostgreSQL + Redis** (batteries-included, admin panel, ORM, authentication built-in).

2. Frontend Responsibilities

2.1 Mobile App (React Native)

Core Responsibilities

- **UI/UX Rendering**: Role-based dashboards (Student, Guard, Staff)
- **Local State Management**: Redux/Zustand for app state
- **Authentication Flow**: Login, token storage (secure encrypted storage), auto-refresh
- **QR Code Scanning**: Camera access, barcode scanner library
- **Location Services**: GPS access, continuous location during attendance
- **File Upload**: Camera/gallery access, image compression before upload
- **Offline Queue**: Local storage for attendance scans when network unavailable
- **Push Notifications**: FCM integration, notification handling
- **Real-time Updates**: WebSocket client for messaging and live attendance

Key Libraries

```json
{
  "dependencies": {
    "react-native": "^0.73",
    "react-navigation": "^6.x",
    "@react-native-async-storage/async-storage": "Storage",
    "react-native-camera": "QR scanning",
    "react-native-geolocation-service": "GPS access",
    "react-native-image-picker": "File uploads",
    "axios": "HTTP client",
    "socket.io-client": "WebSocket",
    "@react-native-firebase/messaging": "Push notifications",
    "react-native-encrypted-storage": "Secure token storage",
    "zustand": "State management (lightweight)",
    "react-query": "Server state management"
  }
}
```

Offline Handling

- **Attendance Scans**: Store locally with timestamp, sync when online (server validates timestamp)
- **Failed Uploads**: Retry queue with exponential backoff
- **Read-Only Cache**: Display cached profile, room details when offline

2.2 Web Admin Dashboard (React)

Core Responsibilities

- **Admin UI**: Warden-focused interface (tables, forms, filters)
- **Data Tables**: Paginated lists (students, leaves, grievances, attendance)
- **Form Handling**: Student entry, leave approval, payment verification
- **File Preview**: View uploaded documents (PDF, images)
- **Real-time Messaging**: Frontdesk chat interface
- **Reports/Export**: CSV/PDF generation for attendance, payments
- **Responsive Design**: Works on desktop and tablets

Key Libraries

```json
{
  "dependencies": {
    "react": "^18.x",
    "react-router-dom": "Routing",
    "tanstack-table": "Data tables",
    "react-hook-form": "Form handling",
    "zod": "Validation",
    "axios": "HTTP client",
    "socket.io-client": "WebSocket",
    "zustand": "State management",
    "recharts": "Basic charts (attendance trends)",
    "tailwindcss": "Styling"
  }
}
```

2.3 Frontend Validation Rules

Field	Validation
Phone Numbers	10 digits, starts with 6-9
Registration Number	Alphanumeric, max 20 chars
File Upload	Max 50MB, types: PDF, JPG, PNG
Leave Dates	End date >= Start date, not in past (except retroactive with flag)
Passwords	Min 8 chars, 1 uppercase, 1 number, 1 special char

---

3. Backend Responsibilities

3.1 Service Modules

3.1.1 Authentication Service

```
Responsibilities:
- User registration (admin-created only)
- Login (email/registration_no + password)
- JWT token generation (access + refresh tokens)
- Role-based access control (Student, Warden, Guard, Staff)
- Password reset (OTP via email/SMS)
- Session management (token revocation)

Endpoints:
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh
POST   /api/v1/auth/forgot-password
POST   /api/v1/auth/reset-password
GET    /api/v1/auth/me
```

3.1.2 Student Management Service

```
Responsibilities:
- CRUD operations on student records (warden only)
- Room assignment and management
- Document upload and retrieval
- Profile view for students
- Student search and filtering

Endpoints:
POST   /api/v1/students
GET    /api/v1/students
GET    /api/v1/students/{id}
PUT    /api/v1/students/{id}
DELETE /api/v1/students/{id}
POST   /api/v1/students/{id}/documents
GET    /api/v1/students/{id}/documents
GET    /api/v1/students/me (student's own profile)
```

3.1.3 Attendance Service

```
Responsibilities:
- QR code generation (time-bound, encrypted)
- QR scan validation (token validity, location proximity)
- Attendance marking and logging
- Attendance reports and exports
- Manual correction (warden only)
- Offline scan queue processing

Endpoints:
POST   /api/v1/attendance/generate-qr
POST   /api/v1/attendance/scan
GET    /api/v1/attendance/sessions
GET    /api/v1/attendance/reports
POST   /api/v1/attendance/manual-correction
GET    /api/v1/attendance/live/{session_id} (WebSocket upgrade)
```

QR Token Structure:

```
{
  "session_id": "uuid",
  "block": "A",
  "year": 1,
  "guard_id": "uuid",
  "guard_location": {"lat": 0.0, "lng": 0.0},
  "valid_from": "ISO timestamp",
  "valid_until": "ISO timestamp",
  "signature": "HMAC-SHA256"
}
```

3.1.4 Leave Management Service

```
Responsibilities:
- Leave request submission
- Leave approval/rejection workflow
- Leave document storage
- Leave calendar view for guards
- Leave cancellation

Endpoints:
POST   /api/v1/leaves
GET    /api/v1/leaves
GET    /api/v1/leaves/{id}
PUT    /api/v1/leaves/{id}/approve
PUT    /api/v1/leaves/{id}/reject
DELETE /api/v1/leaves/{id}
GET    /api/v1/leaves/active (for guards)
```

3.1.5 Grievance Service

```
Responsibilities:
- Grievance submission with categorization
- Image upload
- Status tracking (Open, In Progress, Resolved)
- Priority-based filtering
- Resolution workflow

Endpoints:
POST   /api/v1/grievances
GET    /api/v1/grievances
GET    /api/v1/grievances/{id}
PUT    /api/v1/grievances/{id}
PUT    /api/v1/grievances/{id}/resolve
POST   /api/v1/grievances/{id}/images
```

3.1.6 Messaging Service

```
Responsibilities:
- Student-to-warden messaging (frontdesk)
- Message threading
- Real-time message delivery (WebSocket)
- Message history
- Rate limiting (10 messages/hour per student)

Endpoints:
POST   /api/v1/messages
GET    /api/v1/messages/threads
GET    /api/v1/messages/threads/{id}
WebSocket: /ws/messages
```

3.1.7 Payment Tracking Service

```
Responsibilities:
- Payment record management (warden creates)
- Payment verification
- Dues calculation
- Payment history for students

Endpoints:
POST   /api/v1/payments
GET    /api/v1/payments
GET    /api/v1/payments/student/{id}
PUT    /api/v1/payments/{id}/verify
```

3.1.8 Notification Service

```
Responsibilities:
- In-app notification creation
- Push notification dispatch (FCM)
- Email notifications (async via queue)
- Notification preferences

Endpoints:
GET    /api/v1/notifications
PUT    /api/v1/notifications/{id}/read
POST   /api/v1/notifications/mark-all-read
```

3.2 Business Logic (Backend Only)

- **Location Proximity Calculation**: Haversine formula, 2-meter threshold
- **QR Token Encryption**: AES-256 encryption, HMAC signature verification
- **Attendance Time Windows**: Configurable per year, enforced server-side
- **Rate Limiting**: Token bucket algorithm (Redis)
- **File Size Validation**: Server-side check (max 50MB)
- **Duplicate Detection**: Attendance scans (same student, same session)

3.3 Background Jobs (Celery Tasks)

```
Tasks:
- send_email(to, subject, body, attachments)
- send_push_notification(user_id, title, body)
- process_attendance_queue(offline_scans)
- cleanup_expired_qr_codes()
- generate_daily_attendance_report()
- send_leave_approval_notifications()
```

---

4. Database Schema Proposal

4.1 Entity Relationship Diagram

```
┌─────────────────┐         ┌──────────────────┐
│      users      │         │   student_docs   │
├─────────────────┤         ├──────────────────┤
│ id (PK)         │◄─┐      │ id (PK)          │
│ email           │  │      │ student_id (FK)  │
│ password_hash   │  │      │ doc_type         │
│ role            │  │      │ file_url         │
│ created_at      │  │      │ uploaded_at      │
└─────────────────┘  │      └──────────────────┘
                     │
                     │
                     │
┌─────────────────┐  │      ┌──────────────────┐
│    students     │  │      │      rooms       │
├─────────────────┤  │      ├──────────────────┤
│ id (PK)         │  │      │ id (PK)          │
│ user_id (FK)    ├──┘      │ room_no          │
│ registration_no │         │ block            │
│ name            │    ┌────┤ floor            │
│ phone           │    │    │ capacity         │
│ parent_phone    │    │    │ hostel_id (FK)   │
│ address         │    │    └──────────────────┘
│ year            │    │
│ semester        │    │
│ room_id (FK)    ├────┘
│ is_active       │
└─────────────────┘
        │
        │
        ├──────────────────────┐
        │                      │
        ▼                      ▼
┌──────────────────┐   ┌──────────────────┐
│  leave_requests  │   │    grievances    │
├──────────────────┤   ├──────────────────┤
│ id (PK)          │   │ id (PK)          │
│ student_id (FK)  │   │ student_id (FK)  │
│ leave_type       │   │ ticket_id (UQ)   │
│ from_date        │   │ category         │
│ to_date          │   │ sub_category     │
│ place_of_visit   │   │ priority         │
│ parent_phone     │   │ description      │
│ extra_details    │   │ status           │
│ document_url     │   │ resolved_at      │
│ status           │   │ created_at       │
│ approved_by (FK) │   └──────────────────┘
│ created_at       │            │
└──────────────────┘            │
                                ▼
                       ┌──────────────────┐
                       │ grievance_images │
                       ├──────────────────┤
                       │ id (PK)          │
                       │ grievance_id(FK) │
                       │ image_url        │
                       └──────────────────┘

┌──────────────────┐         ┌──────────────────┐
│attendance_session│         │ attendance_logs  │
├──────────────────┤         ├──────────────────┤
│ id (PK)          │◄────────┤ id (PK)          │
│ guard_id (FK)    │         │ session_id (FK)  │
│ block            │         │ student_id (FK)  │
│ year             │         │ scanned_at       │
│ qr_token         │         │ location_lat     │
│ valid_from       │         │ location_lng     │
│ valid_until      │         │ is_manual        │
│ guard_lat        │         └──────────────────┘
│ guard_lng        │
│ status           │
│ created_at       │
└──────────────────┘
```

```
┌──────────────────┐          ┌──────────────────┐
│    messages      │          │    payments      │
├──────────────────┤          ├──────────────────┤
│ id (PK)          │          │ id (PK)          │
│ thread_id        │          │ student_id (FK)  │
│ sender_id (FK)   │          │ amount           │
│ receiver_id (FK) │          │ payment_for      │
│ content          │          │ due_date         │
│ is_read          │          │ paid_at          │
│ sent_at          │          │ verified_by (FK) │
└──────────────────┘          │ status           │
                              │ created_at       │
                              └──────────────────┘

┌──────────────────┐
│  notifications   │
├──────────────────┤
│ id (PK)          │
│ user_id (FK)     │
│ type             │
│ title            │
│ body             │
│ data (JSON)      │
│ is_read          │
│ created_at       │
└──────────────────┘
```

4.2 Detailed Schema

users

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('student', 'warden', 'guard', 'staff')),
    is_active BOOLEAN DEFAULT TRUE,
    last_login TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);
```

students

```sql
CREATE TABLE students (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID UNIQUE REFERENCES users(id) ON DELETE CASCADE,
    registration_no VARCHAR(50) UNIQUE NOT NULL,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    parent_phone VARCHAR(15) NOT NULL,
    address TEXT,
    year INTEGER NOT NULL CHECK (year BETWEEN 1 AND 4),
    semester INTEGER NOT NULL CHECK (semester BETWEEN 1 AND 8),
    room_id UUID REFERENCES rooms(id),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_students_registration ON students(registration_no);
CREATE INDEX idx_students_room ON students(room_id);
```

rooms

```sql
CREATE TABLE rooms (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    room_no VARCHAR(20) NOT NULL,
    block VARCHAR(10) NOT NULL,
    floor INTEGER NOT NULL,
    capacity INTEGER NOT NULL DEFAULT 2,
    hostel_id UUID, -- Future: multi-hostel support
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(room_no, block)
);
CREATE INDEX idx_rooms_block ON rooms(block);
```

student_documents

```sql
CREATE TABLE student_documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID REFERENCES students(id) ON DELETE CASCADE,
    doc_type VARCHAR(50) NOT NULL, -- 'aadhaar', 'driving_license', 'pan', etc.
    file_url TEXT NOT NULL,
    file_size BIGINT,
    uploaded_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_student_docs_student ON student_documents(student_id);
```

leave_requests

```sql
CREATE TABLE leave_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID REFERENCES students(id) ON DELETE CASCADE,
    leave_type VARCHAR(50) NOT NULL,
    from_date DATE NOT NULL,
    to_date DATE NOT NULL,
    place_of_visit VARCHAR(255),
    parent_phone VARCHAR(15),
    extra_details TEXT,
    document_url TEXT, -- Mentor confirmation
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'approved', 'rejected', 'cancelled')),
    approved_by UUID REFERENCES users(id),
    approval_comments TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_leave_student ON leave_requests(student_id);
CREATE INDEX idx_leave_status ON leave_requests(status);
CREATE INDEX idx_leave_dates ON leave_requests(from_date, to_date);
```

grievances

```sql
CREATE TABLE grievances (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    ticket_id VARCHAR(20) UNIQUE NOT NULL, -- e.g., GRV-2026-00001
    student_id UUID REFERENCES students(id) ON DELETE CASCADE,
    category VARCHAR(50) NOT NULL,
    sub_category VARCHAR(50),
    priority VARCHAR(20) NOT NULL CHECK (priority IN ('high', 'medium', 'low')),
    description TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'open' CHECK (status IN ('open', 'in_progress', 'resolved')),
    resolved_by UUID REFERENCES users(id),
    resolution_comments TEXT,
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_grievance_student ON grievances(student_id);
CREATE INDEX idx_grievance_status ON grievances(status);
CREATE INDEX idx_grievance_priority ON grievances(priority);
CREATE INDEX idx_grievance_category ON grievances(category);
```

grievance_images

```sql
CREATE TABLE grievance_images (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    grievance_id UUID REFERENCES grievances(id) ON DELETE CASCADE,
    image_url TEXT NOT NULL,
    uploaded_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_grievance_images_grievance ON grievance_images(grievance_id);
```

attendance_sessions

```sql
CREATE TABLE attendance_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    guard_id UUID REFERENCES users(id),
    block VARCHAR(10) NOT NULL,
    year INTEGER NOT NULL,
    qr_token TEXT NOT NULL, -- Encrypted token
    valid_from TIMESTAMP NOT NULL,
    valid_until TIMESTAMP NOT NULL,
    guard_lat DECIMAL(10, 8),
    guard_lng DECIMAL(11, 8),
    status VARCHAR(20) DEFAULT 'active' CHECK (status IN ('active', 'closed')),
    created_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_attendance_session_guard ON attendance_sessions(guard_id);
CREATE INDEX idx_attendance_session_status ON attendance_sessions(status);
CREATE INDEX idx_attendance_session_validity ON attendance_sessions(valid_from, valid_until);
```

attendance_logs

```sql
CREATE TABLE attendance_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID REFERENCES attendance_sessions(id),
    student_id UUID REFERENCES students(id),
    scanned_at TIMESTAMP NOT NULL,
    student_lat DECIMAL(10, 8),
    student_lng DECIMAL(11, 8),
    distance_meters DECIMAL(6, 2),
    is_manual BOOLEAN DEFAULT FALSE, -- Manual correction by warden
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(session_id, student_id) -- Prevent duplicate scans
);
CREATE INDEX idx_attendance_logs_session ON attendance_logs(session_id);
CREATE INDEX idx_attendance_logs_student ON attendance_logs(student_id);
CREATE INDEX idx_attendance_logs_scanned ON attendance_logs(scanned_at);
```

messages

```sql
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    thread_id UUID NOT NULL, -- Group messages by conversation
    sender_id UUID REFERENCES users(id),
    receiver_id UUID REFERENCES users(id),
    content TEXT NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    sent_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_messages_thread ON messages(thread_id);
CREATE INDEX idx_messages_sender ON messages(sender_id);
CREATE INDEX idx_messages_receiver ON messages(receiver_id);
CREATE INDEX idx_messages_sent ON messages(sent_at DESC);
```

payments

```sql
CREATE TABLE payments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID REFERENCES students(id) ON DELETE CASCADE,
    amount DECIMAL(10, 2) NOT NULL,
    payment_for VARCHAR(100) NOT NULL, -- "Hostel Fee - Jan 2026"
    due_date DATE,
    paid_at TIMESTAMP,
    verified_by UUID REFERENCES users(id),
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'paid', 'overdue')),
    payment_mode VARCHAR(50), -- 'cash', 'online', 'cheque'
    transaction_ref VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_payments_student ON payments(student_id);
CREATE INDEX idx_payments_status ON payments(status);
CREATE INDEX idx_payments_due_date ON payments(due_date);
```

notifications

```
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL, -- 'leave_approved', 'grievance_resolved', etc.
    title VARCHAR(255) NOT NULL,
    body TEXT NOT NULL,
    data JSONB, -- Additional metadata
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);
CREATE INDEX idx_notifications_user ON notifications(user_id);
CREATE INDEX idx_notifications_read ON notifications(is_read);
CREATE INDEX idx_notifications_created ON notifications(created_at DESC);
```

4.3 Data Retention & Archival

Table	Retention	Archival Strategy
attendance_logs	2 years active	Move to cold storage after year-end
messages	1 year	Delete after 1 year, export on request
notifications	90 days	Auto-delete after 90 days
leave_requests	Indefinite	Active students only
grievances	Indefinite	Audit trail required
student_documents	2 years post-graduation	Compliance with data privacy laws

---

5. API Structure

5.1 API Design Principles

- **RESTful**: Resource-based URLs, standard HTTP methods
- **Versioned**: `/api/v1/` prefix for future compatibility
- **Consistent Response Format**:

```
{
  "success": true,
  "data": {...},
  "message": "Operation successful",
  "timestamp": "2026-02-26T10:30:00Z"
}
```

- **Error Format**:

```json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {"field": "phone", "error": "Must be 10 digits"}
    ]
  },
  "timestamp": "2026-02-26T10:30:00Z"
}
```

5.2 API Endpoints Summary

Authentication

```
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh
POST   /api/v1/auth/forgot-password
POST   /api/v1/auth/reset-password
GET    /api/v1/auth/me
```

Students

```
POST   /api/v1/students                    [Warden]
GET    /api/v1/students                    [Warden]
GET    /api/v1/students?block=A&year=1     [Warden] (with filters)
GET    /api/v1/students/{id}               [Warden, Self]
PUT    /api/v1/students/{id}               [Warden]
DELETE /api/v1/students/{id}               [Warden]
POST   /api/v1/students/{id}/documents     [Warden]
GET    /api/v1/students/{id}/documents     [Warden, Self]
GET    /api/v1/students/me                 [Student]
```

Attendance

```
POST   /api/v1/attendance/generate-qr      [Guard]
POST   /api/v1/attendance/scan             [Student]
GET    /api/v1/attendance/sessions         [Guard, Warden]
GET    /api/v1/attendance/sessions/{id}    [Guard, Warden]
PUT    /api/v1/attendance/sessions/{id}/close [Guard]
GET    /api/v1/attendance/reports          [Warden]
POST   /api/v1/attendance/manual-mark      [Warden]
WS     /ws/attendance/{session_id}         [Guard] (live updates)
```

Leaves

```
POST   /api/v1/leaves                      [Student]
GET    /api/v1/leaves                      [All] (filtered by role)
GET    /api/v1/leaves/{id}                 [Student, Warden, Guard]
PUT    /api/v1/leaves/{id}/approve         [Warden]
PUT    /api/v1/leaves/{id}/reject          [Warden]
DELETE /api/v1/leaves/{id}                 [Student] (cancel pending)
GET    /api/v1/leaves/active               [Guard]
```

Grievances

```
POST   /api/v1/grievances                 [Student]
GET    /api/v1/grievances                 [Student, Warden, Staff]
GET    /api/v1/grievances/{id}            [Student, Warden, Staff]
PUT    /api/v1/grievances/{id}/resolve    [Warden]
POST   /api/v1/grievances/{id}/images     [Student]
```

Messages

```
POST   /api/v1/messages                   [Student, Warden]
GET    /api/v1/messages/threads           [Student, Warden]
GET    /api/v1/messages/threads/{id}      [Student, Warden]
PUT    /api/v1/messages/{id}/read         [Student, Warden]
WS     /ws/messages                       [Student, Warden] (real-time)
```

Payments

```
POST   /api/v1/payments                   [Warden]
GET    /api/v1/payments/student/{id}      [Student, Warden]
PUT    /api/v1/payments/{id}/verify       [Warden]
GET    /api/v1/payments                   [Warden]
```

Notifications

```
GET    /api/v1/notifications              [All]
PUT    /api/v1/notifications/{id}/read    [All]
POST   /api/v1/notifications/read-all     [All]
```

Rooms

```
POST   /api/v1/rooms                      [Warden]
GET    /api/v1/rooms                      [Warden]
GET    /api/v1/rooms/{id}                 [Warden]
PUT    /api/v1/rooms/{id}                 [Warden]
```

5.3 Request/Response Examples

POST /api/v1/attendance/generate-qr

Request (Guard):

```
{
  "block": "A",
  "year": 1,
  "duration_minutes": 90
}
```

Response:

```
{
  "success": true,
  "data": {
    "session_id": "550e8400-e29b-41d4-a716-446655440000",
    "qr_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "qr_image_url": "https://storage.example.com/qr/session-550e8400.png",
    "valid_from": "2026-02-26T19:30:00Z",
    "valid_until": "2026-02-26T21:00:00Z",
    "block": "A",
    "year": 1
  }
}
```

POST /api/v1/attendance/scan

Request (Student):

```
{
  "qr_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "location": {
    "latitude": 28.6139,
    "longitude": 77.2090
  }
}
```

Response (Success):

```
{
  "success": true,
  "data": {
    "attendance_marked": true,
    "student_name": "John Doe",
    "room_no": "A-101",
    "scanned_at": "2026-02-26T19:45:32Z",
    "distance_meters": 1.2
  },
  "message": "Attendance marked successfully"
}
```

Response (Failure - Too far):

```
{
  "success": false,
  "error": {
    "code": "LOCATION_MISMATCH",
    "message": "You are too far from the guard's location",
    "details": {
      "distance_meters": 5.3,
      "max_allowed_meters": 2.0
    }
  }
}
```

5.4 Pagination

All list endpoints support pagination:

```
GET /api/v1/students?page=1&limit=20&sort_by=name&order=asc
```

Response:

```
{
  "success": true,
  "data": [...],
  "pagination": {
    "current_page": 1,
    "per_page": 20,
    "total_pages": 15,
    "total_items": 287
  }
}
```

5.5 Rate Limiting

Endpoint	Rate Limit
POST /api/v1/auth/login	5 requests/minute per IP
POST /api/v1/messages	10 requests/hour per user
POST /api/v1/attendance/scan	3 requests/minute per student
All other endpoints	100 requests/minute per user

---

6. Authentication Strategy

6.1 JWT-Based Authentication

Token Structure

```
{
  "access_token": {
    "user_id": "uuid",
    "email": "student@example.com",
    "role": "student",
    "exp": 1640000000,  // 15 minutes
    "iat": 1639999400
  },
  "refresh_token": {
    "user_id": "uuid",
    "token_id": "unique-refresh-token-id",
    "exp": 1672000000,  // 30 days
    "iat": 1639999400
  }
}
```

Token Lifecycle

1. **Login**: User provides credentials → Server validates → Issues access + refresh tokens
2. **API Request**: Client sends access token in `Authorization: Bearer <token>` header
3. **Token Expiry**: Access token expires after 15 minutes
4. **Refresh**: Client sends refresh token to `/api/v1/auth/refresh` → New access token issued
5. **Logout**: Refresh token blacklisted in Redis (TTL = 30 days)

Security Measures

- **Password Hashing**: bcrypt with cost factor 12
- **Token Signing**: HS256 algorithm with 256-bit secret key (rotate quarterly)
- **HTTPS Only**: All API traffic over TLS 1.3
- **Refresh Token Rotation**: New refresh token on each refresh (optional, V1.1)
- **Token Blacklist**: Redis set for revoked tokens

6.2 Role-Based Access Control (RBAC)

Permission Matrix

Resource	Student	Warden	Guard	Staff
Own Profile	Read	Read/Write All	Read	Read
Leave Requests	Create, Read Own	Read All, Approve	Read Approved	-
Grievances	Create, Read Own	Read All, Resolve	-	Read Assigned
Attendance	Scan QR	Read All, Manual Mark	Generate QR, Read	-
Payments	Read Own	Create, Read All, Verify	-	-
Messages	Send to Warden	Read All, Reply	-	-
Students	-	Full CRUD	-	-
Rooms	Read Own	Full CRUD	-	-

Implementation

```python
# Decorator example
@require_role(['warden'])
def approve_leave(leave_id):
    # Only wardens can approve

@require_role(['student', 'warden'])
@require_ownership('student_id')
def get_payment_history(student_id):
    # Students can only see their own, wardens can see all
```

6.3 Password Policy

- Minimum 8 characters
- At least 1 uppercase letter
- At least 1 number
- At least 1 special character
- Cannot be same as previous 3 passwords (Future)
- Mandatory reset every 180 days (Future, Warden only)

6.4 Password Reset Flow

1. User requests reset → Enters email/registration number
2. Server generates OTP (6 digits) → Stores in Redis (TTL 10 minutes)
3. OTP sent via email/SMS
4. User enters OTP + new password
5. Server validates OTP → Hashes new password → Updates database
6. All existing refresh tokens invalidated

---

7. Third-Party Dependencies

7.1 Critical Dependencies

Cloud Storage (AWS S3 / MinIO)

- **Purpose**: Document and image storage
- **Usage**: Student documents, leave confirmations, grievance images, QR codes
- **Configuration**:
  ```
  Bucket: hostelpulse-storage
  Region: ap-south-1 (Mumbai)
  Access: Private (signed URLs with 1-hour expiry)
  Lifecycle: Archive to Glacier after 1 year
  ```
- **Fallback**: Local filesystem storage (development/small scale)

Email Service (AWS SES / SendGrid)

- **Purpose**: Transactional emails (OTP, notifications, receipts)
- **Daily Limit**: 10,000 emails (AWS SES Free Tier: 62,000/month)
- **Templates**: Password reset, leave approval, payment confirmation
- **Fallback**: SMTP server (institutional email)

Push Notifications (Firebase Cloud Messaging)

- **Purpose**: Real-time alerts to mobile app
- **Events**: Leave approved/rejected, grievance resolved, new message
- **Free Tier**: Unlimited messages
- **Implementation**: Server sends to FCM, FCM delivers to devices

7.2 Optional Dependencies

SMS Gateway (Twilio / AWS SNS) - Future

- **Purpose**: OTP for password reset, urgent alerts
- **Cost**: $0.04/SMS (Twilio India)
- **V1 Status**: Email-only for OTP

Geolocation API - Not Required

- **Reason**: Using device native GPS (no external API needed)

Payment Gateway (Razorpay) - Future V1.1

- **Purpose**: Online hostel fee payment
- **Transaction Fee**: 2% + ₹0
- **V1 Status**: Manual payment tracking only

7.3 Development Tools

- **API Documentation**: Swagger/OpenAPI (auto-generated by FastAPI)
- **Logging**: Sentry (error tracking), CloudWatch/Datadog (application logs)
- **Monitoring**: Prometheus + Grafana (metrics), Uptime monitoring
- **CI/CD**: GitHub Actions / GitLab CI
- **Infrastructure**: Docker + Docker Compose (development), AWS ECS/EC2 (production)

7.4 Dependency Risk Mitigation

Dependency	Risk	Mitigation
AWS S3	Service outage	Local cache, queue uploads, retry logic
FCM	Delivery failure	In-app notification fallback
Email Service	Rate limiting	Queue emails, batch sending, backup SMTP
Redis	Cache loss	Rebuild from PostgreSQL, no critical data in cache only

---

8. Scalability Considerations

8.1 Current Scale (V1)

- **Users**: 500 students + 10 staff (510 total)
- **Requests**: ~5,000 API calls/day
- **Storage**: ~50GB (documents + images)
- **Database**: <10GB
- **Peak Load**: Attendance time (100 concurrent scans in 2 minutes)

8.2 Scaling Strategy (0-2,000 Users)

Vertical Scaling (Sufficient for V1)

```
Single Server Configuration:
- Application: 2 vCPU, 4GB RAM (t3.medium on AWS)
- Database: PostgreSQL RDS (db.t3.small - 2 vCPU, 2GB RAM)
- Redis: ElastiCache (cache.t3.micro - 1 vCPU, 0.5GB RAM)
- Total Cost: ~$150/month
```

Database Optimization

- **Indexing**: All foreign keys, search fields indexed
- **Query Optimization**: Use EXPLAIN ANALYZE, avoid N+1 queries
- **Connection Pooling**: Max 20 connections (PgBouncer)
- **Read Replicas**: Not needed for V1 (< 1000 QPS)

Caching Strategy

```
Cache Layers:
1. API Response Cache (Redis):
   - Student profiles: 1 hour TTL
   - Room details: 6 hours TTL
   - Attendance reports: 15 minutes TTL

2. Session Store (Redis):
   - JWT refresh tokens: 30 days TTL
   - QR session tokens: 2 hours TTL

3. Rate Limit Store (Redis):
   - Token bucket counters: 1 minute/1 hour TTL
```

File Storage

- **CDN**: CloudFront/CloudFlare for static QR images (low priority for V1)
- **Compression**: Image optimization before upload (client-side)
- **Lazy Loading**: Paginated document listings

8.3 Horizontal Scaling (2,000+ Users)

Application Layer

```
Load Balancer (Nginx/ALB)
     │
     ├──► App Server 1 (Docker Container)
     ├──► App Server 2 (Docker Container)
     └──► App Server N (Auto-scaling)

Auto-scaling triggers:
- CPU > 70% for 5 minutes → Add instance
- CPU < 30% for 10 minutes → Remove instance
- Min instances: 2
- Max instances: 10
```

Database Layer

```
PostgreSQL Primary (Writes)
     │
     ├──► Read Replica 1 (Reads - Reports)
     └──► Read Replica 2 (Reads - Dashboards)

Partitioning Strategy (when > 10GB):
- attendance_logs: Partition by month (Range partitioning)
- messages: Partition by thread_id (Hash partitioning)
```

WebSocket Scaling

```
Socket.io with Redis Adapter:
- Multiple WebSocket servers share state via Redis Pub/Sub
- Sticky sessions via load balancer (IP hash)
```

8.4 Performance Targets

Metric	V1 Target	V2 Target (2,000+ users)
API Response Time (p95)	<500ms	<300ms
QR Scan Processing	<2 seconds	<1 second
Database Query Time (p95)	<100ms	<50ms
Concurrent Users	100	500
File Upload (50MB)	<30 seconds	<15 seconds
WebSocket Latency	<200ms	<100ms

8.5 Monitoring & Alerting

Key Metrics

```
Application:
- Request rate (requests/second)
- Response time (p50, p95, p99)
- Error rate (5xx responses)
- Queue depth (Celery tasks)

Infrastructure:
- CPU utilization
- Memory usage
- Disk I/O
- Network throughput

Business:
- Active users (DAU/MAU)
- Attendance completion rate
- Leave approval time
- Grievance resolution time
```

Alerts

```
Critical (PagerDuty/SMS):
- API error rate > 5% for 5 minutes
- Database connection failures
- Disk space > 90%

Warning (Email/Slack):
- API response time p95 > 1 second for 10 minutes
- Queue depth > 1000 tasks for 15 minutes
- Failed email deliveries > 10% for 1 hour
```

8.6 Backup & Disaster Recovery

Database Backups

```
PostgreSQL:
- Automated daily backups (RDS automated backups)
- Point-in-time recovery (up to 7 days)
- Weekly manual snapshots (retained for 30 days)
- Backup storage: Cross-region replication
```

File Backups

```
S3:
- Versioning enabled (retain 5 versions)
- Cross-region replication (DR region)
- Lifecycle policy: Move to Glacier after 1 year
```

Recovery Time Objective (RTO) / Recovery Point Objective (RPO)

```
RTO: 4 hours (time to restore service)
RPO: 24 hours (maximum acceptable data loss)
```

9. Security Considerations

9.1 Application Security

Input Validation

- **SQL Injection**: Parameterized queries (SQLAlchemy ORM)
- **XSS**: Output encoding, Content Security Policy headers
- **CSRF**: CSRF tokens for state-changing operations (web dashboard)
- **File Upload**: Validate file type (magic bytes), scan for malware (ClamAV - future)
- **Rate Limiting**: Prevent brute force, DDoS

API Security

```
Headers:
- 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'
```

9.2 Data Security

Encryption

- **In-Transit**: TLS 1.3 for all API traffic
- **At-Rest**:
  - Database: Encrypted EBS volumes (AES-256)
  - S3: Server-side encryption (SSE-S3)
  - Passwords: bcrypt hashing (cost factor 12)

Sensitive Data Handling

- **PII**: Aadhaar, parent phone numbers → Encrypted columns (Future)
- **Documents**: Access via signed URLs (1-hour expiry), role-based access
- **Location Data**: Not stored long-term, used only for attendance verification

9.3 Compliance

Data Privacy

- **GDPR/DPDP Compliance**:
    - User consent for data collection
    - Right to access (API endpoint to export data)
    - Right to deletion (soft delete with anonymization)
    - Data retention policies (see Section 4.3)

Audit Logging

```
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY,
    user_id UUID,
    action VARCHAR(100), -- 'leave_approved', 'student_deleted', etc.
    resource_type VARCHAR(50),
    resource_id UUID,
    changes JSONB, -- Old and new values
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

---

10. Development & Deployment

10.1 Development Environment

```
# Local setup with Docker Compose
services:
  api:
    build: ./backend
    ports: ["8000:8000"]
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/hostelpulse
      - REDIS_URL=redis://redis:6379/0
    volumes: [".:/app"]
    depends_on: [db, redis]

  db:
    image: postgres:15
    environment:
      - POSTGRES_DB=hostelpulse
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
    volumes: ["postgres_data:/var/lib/postgresql/data"]

  redis:
    image: redis:7-alpine
    volumes: ["redis_data:/data"]

  celery:
    build: ./backend
    command: celery -A app.celery worker -l info
    depends_on: [redis, db]
```

10.2 CI/CD Pipeline

```
# .github/workflows/deploy.yml
name: Deploy
on:
  push:
    branches: [main]

jobs:
  test:
    - Run unit tests (pytest)
    - Run integration tests
    - Code coverage > 80%
    - Linting (ruff, black)

  build:
    - Build Docker image
    - Push to container registry

  deploy:
    - Deploy to staging (auto)
    - Run smoke tests
    - Deploy to production (manual approval)
    - Health check
```

10.3 Production Deployment

```
Infrastructure:
- App Server: AWS ECS Fargate (2 containers)
- Database: AWS RDS PostgreSQL (Multi-AZ)
- Cache: AWS ElastiCache Redis
- Storage: AWS S3 + CloudFront
- Load Balancer: AWS Application Load Balancer
- Monitoring: CloudWatch + Sentry
```

10.4 Environment Configuration

```
# Environment variables (.env)
DATABASE_URL=postgresql://user:pass@host:5432/dbname
REDIS_URL=redis://host:6379/0
SECRET_KEY=<256-bit-random-key>
JWT_ACCESS_TOKEN_EXPIRE_MINUTES=15
JWT_REFRESH_TOKEN_EXPIRE_DAYS=30
AWS_ACCESS_KEY_ID=<key>
AWS_SECRET_ACCESS_KEY=<secret>
AWS_S3_BUCKET=hostelpulse-storage
AWS_REGION=ap-south-1
SENDGRID_API_KEY=<key>
FCM_SERVER_KEY=<key>
SENTRY_DSN=<dsn>
ALLOWED_ORIGINS=https://admin.hostelpulse.com,hostelpulse://app
MAX_FILE_SIZE_MB=50
```

11. Testing Strategy

11.1 Test Coverage

```
Unit Tests (70% coverage):
- Business logic (attendance proximity calculation)
- Utility functions (haversine distance, QR token generation)
- Validators (phone number, dates)

Integration Tests (20% coverage):
- API endpoints (request/response)
- Database operations (CRUD)
- Authentication flow

End-to-End Tests (10% coverage):
- Critical user flows (attendance QR scan, leave approval)
- Mobile app (Detox/Appium - future)
```

11.2 Performance Testing

```
# Load testing with Locust
- Simulate 100 concurrent students scanning QR
- Target: <2 seconds response time, 0% errors
- Run before production deployment
```

---

12. Open Technical Questions

1. **Mobile Framework**: React Native vs Flutter? → **Recommendation: React Native** (larger talent pool, web code reuse)
2. **Backend Framework**: FastAPI vs Django? → **Recommendation: Django** (admin panel, batteries-included for V1)
3. **Hosting**: AWS vs DigitalOcean vs Self-hosted? → **Recommendation: AWS** (scalability, managed services)
4. **File Storage**: AWS S3 vs MinIO (self-hosted)? → **Recommendation: AWS S3** (reliability, no maintenance)
5. **Real-time**: WebSocket vs Server-Sent Events? → **Recommendation: WebSocket** (bidirectional, better mobile support)

---

13. Migration Path (Future)

V1 → V1.1 (Payment Gateway)

- Add Razorpay SDK to mobile app
- New API endpoints for payment initiation, webhook handling
- Database schema: Add `payment_gateway_txn_id`, `payment_gateway` columns

V1.1 → V2 (Multi-Hostel)

- Add `hostel_id` foreign key to all relevant tables
- Multi-tenancy: Shared schema with `hostel_id` filtering
- OR: Schema-per-hostel (PostgreSQL schemas)

V2 → V3 (Microservices - if needed)

- Extract attendance service (separate deployment)
- Extract messaging service (separate deployment)
- API Gateway for routing
- Distributed tracing (OpenTelemetry)

---

14. Risk Mitigation Summary

Risk	Impact	Mitigation	Owner
Database single point of failure	High	Multi-AZ RDS, automated backups, read replicas	DevOps
Location spoofing for attendance	High	Device fingerprinting, random audits	Backend Team
File storage costs exceed budget	Medium	Compression, lifecycle policies, usage monitoring	Backend Team
Third-party service downtime (FCM)	Medium	In-app notification fallback, retry queue	Backend Team
Peak load during attendance (100 concurrent scans)	Medium	Load testing, auto-scaling, caching	DevOps/Backend
Vendor lock-in (AWS)	Low	Use standard APIs (S3-compatible, PostgreSQL), IaC	DevOps

---

Document Owner: Backend Architect
Reviewers: CTO, Lead Developer, DevOps Engineer
Next Steps:

1. Review and approve technology choices
2. Set up development environment
3. Create database migration scripts
4. Implement authentication module (Week 1-2)
5. Build attendance service (Week 3-4)

Appendix: Glossary

- **JWT**: JSON Web Token
- **RBAC**: Role-Based Access Control
- **CDN**: Content Delivery Network
- **RTO**: Recovery Time Objective
- **RPO**: Recovery Point Objective
- **ORM**: Object-Relational Mapping
- **TTL**: Time To Live
- **QPS**: Queries Per Second
- **FCM**: Firebase Cloud Messaging