Driving School Lesson Management Platform - Technical Specification

Version: 1.0
Date: October 7, 2025
Status: Draft

Table of Contents

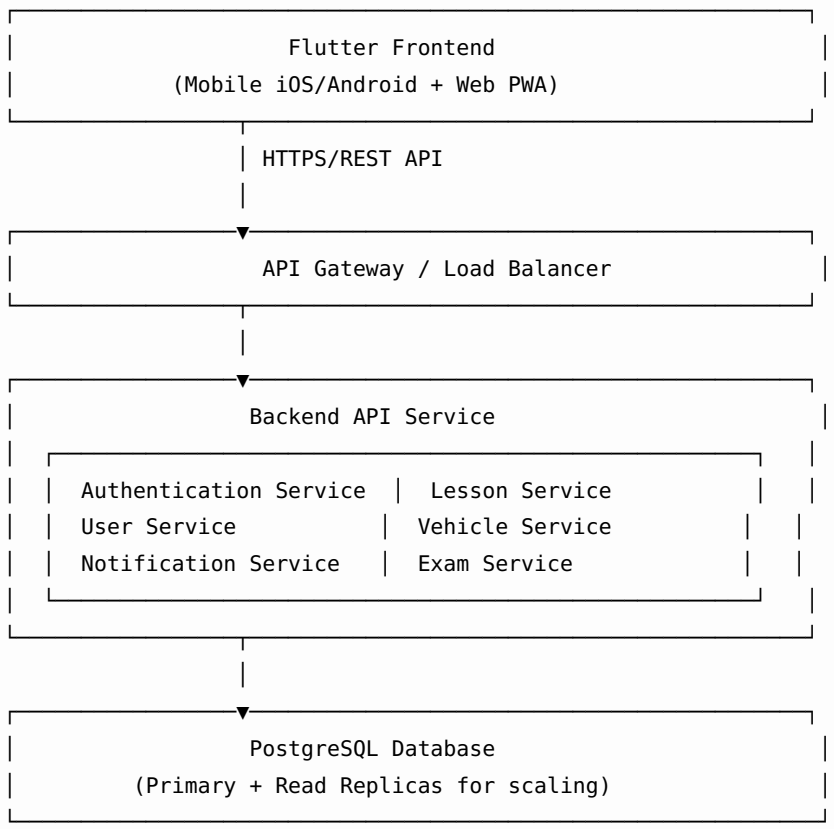
Executive Summary

This document specifies the technical requirements for a comprehensive driving school lesson management platform. The system manages students, instructors, lessons, vehicles, exams, and administrative workflows with role-based access control.

Key Features: - Multi-role user management (Students, Instructors, Super Admin) - Lesson scheduling with calendar views (daily/weekly/monthly) - Vehicle fleet management - Exam scheduling and tracking - Lesson request approval workflow - Real-time availability tracking - Email verification and notifications - Category and transmission type support

Technology Stack: - **Database:** PostgreSQL 14+ - **Backend:** REST API (Node.js/Express, Python/FastAPI, or similar) - **Frontend:** Flutter (Mobile & Web) - **Authentication:** JWT-based with refresh tokens - **Email Service:** SMTP or third-party (SendGrid, Mailgun)

System Architecture Overview

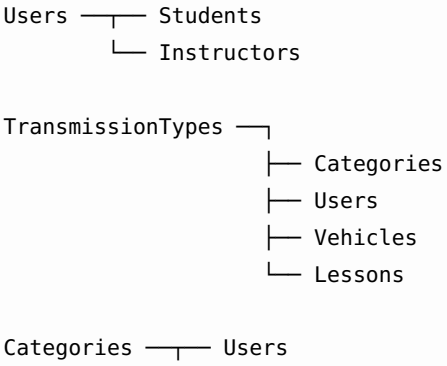


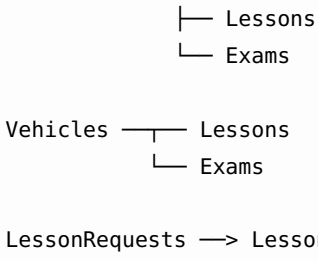
Database Schema

Overview

The database is designed for PostgreSQL with strong referential integrity, appropriate indexes for performance, and support for concurrent operations.

Schema Diagram





---

1. Core Tables

1.1 Users Table

Primary table for all user authentication and profile data.

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('student',
        'instructor', 'super_admin')),
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20),
    date_of_birth DATE,
    address TEXT,
    city VARCHAR(100),
    postal_code VARCHAR(20),
    profile_picture_url TEXT,

    -- Account status
    is_email_verified BOOLEAN DEFAULT FALSE,
    email_verification_token VARCHAR(255),
    email_verification_expires_at TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    is_approved BOOLEAN DEFAULT FALSE,

    -- Password reset
    password_reset_token VARCHAR(255),
    password_reset_expires_at TIMESTAMP,

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login_at TIMESTAMP,

    -- Indexes
    CONSTRAINT email_lowercase CHECK (email = LOWER(email))
);
```

```sql
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_users_is_approved ON users(is_approved);
CREATE INDEX idx_users_email_verification_token ON
        users(email_verification_token);
CREATE INDEX idx_users_password_reset_token ON
        users(password_reset_token);
```

1.2 Transmission Types Table

Defines available transmission types (Manual, Automatic, Semi-Automatic).

```sql
CREATE TABLE transmission_types (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    code VARCHAR(10) UNIQUE NOT NULL,
    description TEXT,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_transmission_types_code ON
        transmission_types(code);

-- Default data
INSERT INTO transmission_types (name, code, description) VALUES
    ('Manual', 'MT', 'Manual transmission vehicles'),
    ('Automatic', 'AT', 'Automatic transmission vehicles'),
    ('Semi-Automatic', 'SAT', 'Semi-automatic transmission
        vehicles');
```

1.3 Categories Table

Driving license categories (A, A1, B, BE, C, CE, D, etc.).

```sql
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(10) UNIQUE NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    description TEXT,
    transmission_type_id INTEGER REFERENCES transmission_types(id)
        ON DELETE RESTRICT,
    min_age INTEGER,
    requires_theory_exam BOOLEAN DEFAULT TRUE,
    requires_practical_exam BOOLEAN DEFAULT TRUE,
    min_lesson_hours INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
```

```
    display_order INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_categories_name ON categories(name);
CREATE INDEX idx_categories_transmission_type ON
        categories(transmission_type_id);
CREATE INDEX idx_categories_is_active ON categories(is_active);

-- Default data
INSERT INTO categories (name, full_name, description,
        transmission_type_id, min_age, min_lesson_hours,
        display_order) VALUES
    ('A', 'Motorcycle', 'Motorcycles and motor tricycles', 1, 24,
        20, 1),
    ('A1', 'Light Motorcycle', 'Light motorcycles up to 125cc', 1,
        16, 15, 2),
    ('B', 'Car', 'Passenger vehicles', 1, 18, 30, 3),
    ('B-AT', 'Car (Automatic)', 'Passenger vehicles with automatic
        transmission', 2, 18, 25, 4),
    ('BE', 'Car with Trailer', 'Car with trailer exceeding 750kg',
        1, 18, 10, 5),
    ('C', 'Truck', 'Vehicles exceeding 3,500kg', 1, 21, 40, 6),
    ('D', 'Bus', 'Vehicles with more than 8 passenger seats', 1, 24,
        50, 7);
```

1.4 Students Table

Extended profile data specific to students.

```
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL REFERENCES users(id) ON DELETE
        CASCADE,
    student_id_number VARCHAR(50) UNIQUE,
    category_id INTEGER REFERENCES categories(id) ON DELETE
        RESTRICT,
    transmission_type_id INTEGER REFERENCES transmission_types(id)
        ON DELETE RESTRICT,

    -- License information
    has_driving_license BOOLEAN DEFAULT FALSE,
    existing_license_number VARCHAR(100),
    existing_license_categories TEXT[], -- Array of categories
        already held

    -- Medical
    medical_certificate_url TEXT,
    medical_certificate_expiry DATE,

    -- Emergency contact
    emergency_contact_name VARCHAR(200),
```

```
    emergency_contact_phone VARCHAR(20),
    emergency_contact_relationship VARCHAR(50),

    -- Preferences
    preferred_instructor_id INTEGER REFERENCES instructors(id) ON
        DELETE SET NULL,
    preferred_lesson_time VARCHAR(20), -- 'morning', 'afternoon',
        'evening'

    -- Progress tracking
    theory_exam_passed BOOLEAN DEFAULT FALSE,
    practical_exam_passed BOOLEAN DEFAULT FALSE,
    enrollment_date DATE DEFAULT CURRENT_DATE,
    graduation_date DATE,

    -- Notes
    admin_notes TEXT,
    special_requirements TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_students_user_id ON students(user_id);
CREATE INDEX idx_students_category_id ON students(category_id);
CREATE INDEX idx_students_student_id_number ON
        students(student_id_number);
CREATE INDEX idx_students_preferred_instructor ON
        students(preferred_instructor_id);
```

1.5 Instructors Table

Extended profile data specific to instructors.

```
CREATE TABLE instructors (
    id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL REFERENCES users(id) ON DELETE
        CASCADE,
    instructor_id_number VARCHAR(50) UNIQUE,

    -- Certifications
    instructor_license_number VARCHAR(100) UNIQUE NOT NULL,
    instructor_license_expiry DATE NOT NULL,
    instructor_certificate_url TEXT,

    -- Qualified categories (array of category IDs they can teach)
    qualified_category_ids INTEGER[] NOT NULL DEFAULT '{}',
    qualified_transmission_type_ids INTEGER[] NOT NULL DEFAULT '{}',

    -- Work information
```

```
    hire_date DATE,
    employment_type VARCHAR(20) CHECK (employment_type IN
        ('full_time', 'part_time', 'contractor')),
    hourly_rate DECIMAL(10, 2),

    -- Availability
    default_working_hours JSONB, -- e.g., {"monday": {"start":
        "09:00", "end": "17:00"}, ...}
    max_lessons_per_day INTEGER DEFAULT 8,

    -- Performance metrics
    total_lessons_completed INTEGER DEFAULT 0,
    average_rating DECIMAL(3, 2) DEFAULT 0.00,
    total_students_trained INTEGER DEFAULT 0,
    pass_rate_percentage DECIMAL(5, 2) DEFAULT 0.00,

    -- Status
    is_available_for_booking BOOLEAN DEFAULT TRUE,

    -- Notes
    admin_notes TEXT,
    specializations TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_instructors_user_id ON instructors(user_id);
CREATE INDEX idx_instructors_instructor_id_number ON
        instructors(instructor_id_number);
CREATE INDEX idx_instructors_license_number ON
        instructors(instructor_license_number);
CREATE INDEX idx_instructors_qualified_categories ON instructors
        USING GIN(qualified_category_ids);
CREATE INDEX idx_instructors_is_available ON
        instructors(is_available_for_booking);
```

1.6 Vehicles Table

Fleet management for driving school vehicles.

```
CREATE TABLE vehicles (
    id SERIAL PRIMARY KEY,
    registration_number VARCHAR(50) UNIQUE NOT NULL,

    -- Vehicle details
    make VARCHAR(100) NOT NULL,
    model VARCHAR(100) NOT NULL,
    year INTEGER NOT NULL,
    color VARCHAR(50),
```

```
    vin VARCHAR(17) UNIQUE,

    -- Classification
    category_id INTEGER REFERENCES categories(id) ON DELETE
        RESTRICT,
    transmission_type_id INTEGER NOT NULL REFERENCES
        transmission_types(id) ON DELETE RESTRICT,

    -- Status
    status VARCHAR(20) DEFAULT 'available' CHECK (status IN
        ('available', 'in_use', 'maintenance', 'out_of_service')),
    is_active BOOLEAN DEFAULT TRUE,

    -- Maintenance
    last_service_date DATE,
    next_service_date DATE,
    last_service_mileage INTEGER,
    current_mileage INTEGER DEFAULT 0,
    service_interval_km INTEGER DEFAULT 10000,

    -- Insurance
    insurance_policy_number VARCHAR(100),
    insurance_expiry_date DATE,
    insurance_company VARCHAR(200),

    -- Features
    has_dual_controls BOOLEAN DEFAULT TRUE,
    has_dashcam BOOLEAN DEFAULT FALSE,
    fuel_type VARCHAR(20) CHECK (fuel_type IN ('petrol', 'diesel',
        'electric', 'hybrid')),

    -- Media
    vehicle_image_url TEXT,

    -- Notes
    admin_notes TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_vehicles_registration ON
        vehicles(registration_number);
CREATE INDEX idx_vehicles_category ON vehicles(category_id);
CREATE INDEX idx_vehicles_transmission_type ON
        vehicles(transmission_type_id);
CREATE INDEX idx_vehicles_status ON vehicles(status);
CREATE INDEX idx_vehicles_is_active ON vehicles(is_active);
CREATE INDEX idx_vehicles_next_service_date ON
        vehicles(next_service_date);
```

1.7 Lesson Requests Table

Handles student requests for lessons that require instructor/admin approval.

```
CREATE TABLE lesson_requests (
    id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE
        CASCADE,
    instructor_id INTEGER REFERENCES instructors(id) ON DELETE SET
        NULL,

    -- Request details
    requested_date DATE NOT NULL,
    requested_start_time TIME NOT NULL,
    requested_end_time TIME NOT NULL,
    lesson_type VARCHAR(20) NOT NULL CHECK (lesson_type IN
        ('driving', 'theory', 'exam')),
    category_id INTEGER REFERENCES categories(id) ON DELETE
        RESTRICT,

    -- Optional vehicle preference
    preferred_vehicle_id INTEGER REFERENCES vehicles(id) ON DELETE
        SET NULL,

    -- Status workflow
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN
        ('pending', 'approved', 'rejected', 'cancelled')),

    -- Approval tracking
    reviewed_by INTEGER REFERENCES users(id) ON DELETE SET NULL,
    reviewed_at TIMESTAMP,
    rejection_reason TEXT,

    -- Notes
    student_notes TEXT,
    admin_notes TEXT,

    -- Reference to created lesson (if approved)
    lesson_id INTEGER REFERENCES lessons(id) ON DELETE SET NULL,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_lesson_requests_student ON
        lesson_requests(student_id);
CREATE INDEX idx_lesson_requests_instructor ON
        lesson_requests(instructor_id);
CREATE INDEX idx_lesson_requests_status ON lesson_requests(status);
CREATE INDEX idx_lesson_requests_date ON
        lesson_requests(requested_date);
```

```sql
CREATE INDEX idx_lesson_requests_lesson_id ON
        lesson_requests(lesson_id);
```

1.8 Lessons Table

Core table for all scheduled lessons.

```sql
CREATE TABLE lessons (
    id SERIAL PRIMARY KEY,

    -- Participants
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE
        CASCADE,
    instructor_id INTEGER NOT NULL REFERENCES instructors(id) ON
        DELETE RESTRICT,
    vehicle_id INTEGER REFERENCES vehicles(id) ON DELETE RESTRICT,

    -- Schedule
    lesson_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    duration_minutes INTEGER GENERATED ALWAYS AS (
        EXTRACT(EPOCH FROM (end_time - start_time))/60
    ) STORED,

    -- Classification
    lesson_type VARCHAR(20) NOT NULL CHECK (lesson_type IN
        ('driving', 'theory', 'exam')),
    category_id INTEGER NOT NULL REFERENCES categories(id) ON DELETE
        RESTRICT,

    -- Status
    status VARCHAR(20) DEFAULT 'scheduled' CHECK (status IN (
        'scheduled', 'in_progress', 'completed', 'cancelled',
        'no_show'
    )),

    -- Completion details
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    start_mileage INTEGER,
    end_mileage INTEGER,

    -- Location
    pickup_location TEXT,
    dropoff_location TEXT,
    route_description TEXT,

    -- Evaluation (for driving lessons)
    instructor_rating INTEGER CHECK (instructor_rating BETWEEN 1 AND
        5),
```

```
    instructor_feedback TEXT,
    skills_practiced TEXT[],
    areas_for_improvement TEXT,
    student_performance_rating INTEGER CHECK
        (student_performance_rating BETWEEN 1 AND 5),

    -- Student feedback
    student_rating INTEGER CHECK (student_rating BETWEEN 1 AND 5),
    student_feedback TEXT,

    -- Cancellation
    cancelled_by INTEGER REFERENCES users(id) ON DELETE SET NULL,
    cancelled_at TIMESTAMP,
    cancellation_reason TEXT,

    -- Pricing
    lesson_price DECIMAL(10, 2),
    payment_status VARCHAR(20) DEFAULT 'pending' CHECK
        (payment_status IN (
        'pending', 'paid', 'refunded', 'waived'
    )),

    -- Notes
    admin_notes TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- Constraints
    CONSTRAINT lesson_time_valid CHECK (end_time > start_time),
    CONSTRAINT lesson_date_not_past CHECK (
        lesson_date >= CURRENT_DATE OR status != 'scheduled'
    )
);

CREATE INDEX idx_lessons_student ON lessons(student_id);
CREATE INDEX idx_lessons_instructor ON lessons(instructor_id);
CREATE INDEX idx_lessons_vehicle ON lessons(vehicle_id);
CREATE INDEX idx_lessons_date ON lessons(lesson_date);
CREATE INDEX idx_lessons_date_time ON lessons(lesson_date,
        start_time);
CREATE INDEX idx_lessons_status ON lessons(status);
CREATE INDEX idx_lessons_type ON lessons(lesson_type);
CREATE INDEX idx_lessons_category ON lessons(category_id);

-- Composite index for availability checking
CREATE INDEX idx_lessons_instructor_date_time ON
        lessons(instructor_id, lesson_date, start_time, end_time)
    WHERE status IN ('scheduled', 'in_progress');
```

```sql
CREATE INDEX idx_lessons_vehicle_date_time ON lessons(vehicle_id,
        lesson_date, start_time, end_time)
    WHERE status IN ('scheduled', 'in_progress');
```

1.9 Lesson Counters Table

Tracks lesson progress for each student per category.

```sql
CREATE TABLE lesson_counters (
    id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE
        CASCADE,
    category_id INTEGER NOT NULL REFERENCES categories(id) ON DELETE
        CASCADE,

    -- Counters by lesson type
    total_driving_lessons INTEGER DEFAULT 0,
    completed_driving_lessons INTEGER DEFAULT 0,
    total_theory_lessons INTEGER DEFAULT 0,
    completed_theory_lessons INTEGER DEFAULT 0,

    -- Hours tracking
    total_driving_hours DECIMAL(10, 2) DEFAULT 0.00,
    required_driving_hours DECIMAL(10, 2) DEFAULT 0.00,

    -- Exam attempts
    theory_exam_attempts INTEGER DEFAULT 0,
    practical_exam_attempts INTEGER DEFAULT 0,

    -- Progress percentage (0-100)
    progress_percentage DECIMAL(5, 2) DEFAULT 0.00,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(student_id, category_id)
);

CREATE INDEX idx_lesson_counters_student ON
        lesson_counters(student_id);
CREATE INDEX idx_lesson_counters_category ON
        lesson_counters(category_id);
```

1.10 Exams Table

Scheduled theory and practical exams.

```sql
CREATE TABLE exams (
    id SERIAL PRIMARY KEY,
```

```
    -- Exam details
    exam_type VARCHAR(20) NOT NULL CHECK (exam_type IN ('theory',
        'practical')),
    category_id INTEGER NOT NULL REFERENCES categories(id) ON DELETE
        RESTRICT,

    -- Schedule
    exam_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,

    -- Location
    exam_location TEXT NOT NULL,
    exam_center_name VARCHAR(200),

    -- For practical exams
    vehicle_id INTEGER REFERENCES vehicles(id) ON DELETE RESTRICT,
    examiner_id INTEGER REFERENCES instructors(id) ON DELETE SET
        NULL,

    -- Capacity
    max_students INTEGER DEFAULT 1,
    current_students_count INTEGER DEFAULT 0,

    -- Status
    status VARCHAR(20) DEFAULT 'scheduled' CHECK (status IN (
        'scheduled', 'in_progress', 'completed', 'cancelled'
    )),

    -- Notes
    special_instructions TEXT,
    admin_notes TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT exam_time_valid CHECK (end_time > start_time),
    CONSTRAINT exam_capacity_valid CHECK (current_students_count <=
        max_students)
);

CREATE INDEX idx_exams_type ON exams(exam_type);
CREATE INDEX idx_exams_category ON exams(category_id);
CREATE INDEX idx_exams_date ON exams(exam_date);
CREATE INDEX idx_exams_status ON exams(status);
CREATE INDEX idx_exams_vehicle ON exams(vehicle_id);
CREATE INDEX idx_exams_examiner ON exams(examiner_id);
```

1.11 Exam Registrations Table

Junction table linking students to exams with results.

```
CREATE TABLE exam_registrations (
    id SERIAL PRIMARY KEY,
    exam_id INTEGER NOT NULL REFERENCES exams(id) ON DELETE CASCADE,
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE
        CASCADE,

    -- Registration
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    registration_fee DECIMAL(10, 2),
    payment_status VARCHAR(20) DEFAULT 'pending' CHECK
        (payment_status IN (
        'pending', 'paid', 'refunded', 'waived'
    )),

    -- Status
    attendance_status VARCHAR(20) DEFAULT 'registered' CHECK
        (attendance_status IN (
        'registered', 'present', 'absent', 'cancelled'
    )),

    -- Results
    result VARCHAR(20) CHECK (result IN ('pass', 'fail',
        'pending')),
    score DECIMAL(5, 2), -- For theory exams
    examiner_comments TEXT,
    result_date DATE,

    -- Attempt number for this student and category
    attempt_number INTEGER DEFAULT 1,

    -- Certificate
    certificate_issued BOOLEAN DEFAULT FALSE,
    certificate_number VARCHAR(100),
    certificate_issue_date DATE,
    certificate_url TEXT,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(exam_id, student_id)
);

CREATE INDEX idx_exam_registrations_exam ON
        exam_registrations(exam_id);
CREATE INDEX idx_exam_registrations_student ON
        exam_registrations(student_id);
CREATE INDEX idx_exam_registrations_result ON
        exam_registrations(result);
CREATE INDEX idx_exam_registrations_attendance ON
        exam_registrations(attendance_status);
```

1.12 Payments Table

Financial transactions for lessons, exams, and other services.

```
CREATE TABLE payments (
    id SERIAL PRIMARY KEY,

    -- Payer
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    student_id INTEGER REFERENCES students(id) ON DELETE SET NULL,

    -- Payment details
    amount DECIMAL(10, 2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'USD',
    payment_method VARCHAR(50) CHECK (payment_method IN (
        'cash', 'credit_card', 'debit_card', 'bank_transfer',
        'online', 'mobile_payment'
    )),

    -- Payment type
    payment_type VARCHAR(50) NOT NULL CHECK (payment_type IN (
        'lesson_fee', 'exam_fee', 'registration_fee',
        'material_fee', 'late_fee', 'other'
    )),

    -- References
    lesson_id INTEGER REFERENCES lessons(id) ON DELETE SET NULL,
    exam_registration_id INTEGER REFERENCES exam_registrations(id)
        ON DELETE SET NULL,

    -- Status
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN (
        'pending', 'completed', 'failed', 'refunded', 'cancelled'
    )),

    -- Transaction details
    transaction_id VARCHAR(255) UNIQUE,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- Gateway information
    payment_gateway VARCHAR(100),
    gateway_response JSONB,

    -- Refund information
    refund_amount DECIMAL(10, 2),
    refund_date TIMESTAMP,
    refund_reason TEXT,

    -- Receipt
```

```
    receipt_number VARCHAR(100) UNIQUE,
    receipt_url TEXT,

    -- Notes
    description TEXT,
    admin_notes TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_payments_user ON payments(user_id);
CREATE INDEX idx_payments_student ON payments(student_id);
CREATE INDEX idx_payments_status ON payments(status);
CREATE INDEX idx_payments_transaction_date ON
        payments(transaction_date);
CREATE INDEX idx_payments_lesson ON payments(lesson_id);
CREATE INDEX idx_payments_exam ON payments(exam_registration_id);
CREATE INDEX idx_payments_receipt_number ON
        payments(receipt_number);
```

1.13 Notifications Table

System notifications for users.

```
CREATE TABLE notifications (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,

    -- Notification details
    type VARCHAR(50) NOT NULL CHECK (type IN (
        'lesson_scheduled', 'lesson_cancelled', 'lesson_reminder',
        'exam_scheduled', 'exam_result', 'payment_received',
        'approval_required', 'approval_granted',
        'approval_rejected',
        'system_announcement', 'instructor_assigned', 'other'
    )),

    title VARCHAR(200) NOT NULL,
    message TEXT NOT NULL,

    -- Related entities
    lesson_id INTEGER REFERENCES lessons(id) ON DELETE SET NULL,
    exam_id INTEGER REFERENCES exams(id) ON DELETE SET NULL,
    lesson_request_id INTEGER REFERENCES lesson_requests(id) ON
        DELETE SET NULL,

    -- Status
    is_read BOOLEAN DEFAULT FALSE,
```

```
    read_at TIMESTAMP,

    -- Delivery
    send_email BOOLEAN DEFAULT FALSE,
    email_sent BOOLEAN DEFAULT FALSE,
    email_sent_at TIMESTAMP,

    send_push BOOLEAN DEFAULT FALSE,
    push_sent BOOLEAN DEFAULT FALSE,
    push_sent_at TIMESTAMP,

    -- Priority
    priority VARCHAR(20) DEFAULT 'normal' CHECK (priority IN ('low',
        'normal', 'high', 'urgent')),

    -- Action link (deep link for mobile app)
    action_url TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP
);

CREATE INDEX idx_notifications_user ON notifications(user_id);
CREATE INDEX idx_notifications_is_read ON notifications(is_read);
CREATE INDEX idx_notifications_type ON notifications(type);
CREATE INDEX idx_notifications_created_at ON
        notifications(created_at);
CREATE INDEX idx_notifications_priority ON notifications(priority);
```

1.14 Audit Logs Table

Comprehensive audit trail for system actions.

```
CREATE TABLE audit_logs (
    id SERIAL PRIMARY KEY,

    -- Actor
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    user_email VARCHAR(255),
    user_role VARCHAR(20),

    -- Action details
    action VARCHAR(100) NOT NULL,
    entity_type VARCHAR(50) NOT NULL,
    entity_id INTEGER,

    -- Changes
    old_values JSONB,
```

```
    new_values JSONB,

    -- Request context
    ip_address INET,
    user_agent TEXT,
    request_method VARCHAR(10),
    request_path TEXT,

    -- Status
    status VARCHAR(20) CHECK (status IN ('success', 'failure',
        'error')),
    error_message TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_audit_logs_user ON audit_logs(user_id);
CREATE INDEX idx_audit_logs_entity ON audit_logs(entity_type,
        entity_id);
CREATE INDEX idx_audit_logs_action ON audit_logs(action);
CREATE INDEX idx_audit_logs_created_at ON audit_logs(created_at);
```

1.15 System Settings Table

Application-wide configuration.

```
CREATE TABLE system_settings (
    id SERIAL PRIMARY KEY,
    setting_key VARCHAR(100) UNIQUE NOT NULL,
    setting_value TEXT NOT NULL,
    setting_type VARCHAR(20) DEFAULT 'string' CHECK (setting_type IN
        (
        'string', 'integer', 'boolean', 'json', 'decimal'
    )),
    description TEXT,
    category VARCHAR(50),
    is_public BOOLEAN DEFAULT FALSE,
    updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_system_settings_key ON
        system_settings(setting_key);
CREATE INDEX idx_system_settings_category ON
        system_settings(category);

-- Default settings
INSERT INTO system_settings (setting_key, setting_value,
        setting_type, description, category, is_public) VALUES
    ('business_name', 'ABC Driving School', 'string', 'School name',
        'general', true),
```

```
    ('business_email', 'info@example.com', 'string', 'Primary
        contact email', 'general', true),
    ('business_phone', '+1234567890', 'string', 'Primary contact
        phone', 'general', true),
    ('default_lesson_duration', '60', 'integer', 'Default lesson
        duration in minutes', 'lessons', true),
    ('lesson_cancellation_hours', '24', 'integer', 'Minimum hours
        before lesson to cancel', 'lessons', true),
    ('auto_approve_lessons', 'false', 'boolean', 'Auto-approve
        lesson requests', 'lessons', false),
    ('max_daily_lessons_per_student', '3', 'integer', 'Maximum
        lessons per student per day', 'lessons', true),
    ('theory_exam_pass_score', '80', 'decimal', 'Minimum score to
        pass theory exam (%)', 'exams', true),
    ('practical_exam_max_attempts', '3', 'integer', 'Maximum
        practical exam attempts', 'exams', true),
    ('email_notifications_enabled', 'true', 'boolean', 'Enable email
        notifications', 'notifications', false),
    ('booking_advance_days', '30', 'integer', 'Days in advance for
        booking', 'lessons', true);
```

---

2. Database Triggers and Functions

2.1 Update Timestamp Trigger

Automatically update `updated_at` timestamp on record modification.

```
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ language 'plpgsql';

-- Apply to all tables with updated_at column
CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_students_updated_at BEFORE UPDATE ON students
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_instructors_updated_at BEFORE UPDATE ON
        instructors
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_vehicles_updated_at BEFORE UPDATE ON vehicles
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_lessons_updated_at BEFORE UPDATE ON lessons
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

```
CREATE TRIGGER update_lesson_requests_updated_at BEFORE UPDATE ON
        lesson_requests
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_lesson_counters_updated_at BEFORE UPDATE ON
        lesson_counters
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_exams_updated_at BEFORE UPDATE ON exams
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_exam_registrations_updated_at BEFORE UPDATE ON
        exam_registrations
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_payments_updated_at BEFORE UPDATE ON payments
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

2.2 Lesson Counter Update Trigger

Automatically update lesson counters when lessons are completed.

```
CREATE OR REPLACE FUNCTION update_lesson_counter()
RETURNS TRIGGER AS $$
BEGIN
    -- Only process when lesson status changes to completed
    IF NEW.status = 'completed' AND OLD.status != 'completed' THEN
        -- Update or insert lesson counter
        INSERT INTO lesson_counters (student_id, category_id,
        completed_driving_lessons, total_driving_hours)
        VALUES (
            NEW.student_id,
            NEW.category_id,
            CASE WHEN NEW.lesson_type = 'driving' THEN 1 ELSE 0 END,
            CASE WHEN NEW.lesson_type = 'driving' THEN
        NEW.duration_minutes / 60.0 ELSE 0 END
        )
        ON CONFLICT (student_id, category_id)
        DO UPDATE SET
            completed_driving_lessons =
        lesson_counters.completed_driving_lessons +
                CASE WHEN NEW.lesson_type = 'driving' THEN 1 ELSE 0
        END,
            completed_theory_lessons =
        lesson_counters.completed_theory_lessons +
                CASE WHEN NEW.lesson_type = 'theory' THEN 1 ELSE 0
        END,
            total_driving_hours =
        lesson_counters.total_driving_hours +
                CASE WHEN NEW.lesson_type = 'driving' THEN
        NEW.duration_minutes / 60.0 ELSE 0 END,
            updated_at = CURRENT_TIMESTAMP;
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_lesson_counter
AFTER UPDATE ON lessons
FOR EACH ROW
EXECUTE FUNCTION update_lesson_counter();
```

2.3 Exam Capacity Management Trigger

Manage exam student count automatically.

```
CREATE OR REPLACE FUNCTION manage_exam_capacity()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        -- Increment student count
        UPDATE exams
        SET current_students_count = current_students_count + 1
        WHERE id = NEW.exam_id;

        -- Check if exam is full
        IF (SELECT current_students_count FROM exams WHERE id =
        NEW.exam_id) >
           (SELECT max_students FROM exams WHERE id = NEW.exam_id)
        THEN
            RAISE EXCEPTION 'Exam is full';
        END IF;

    ELSIF TG_OP = 'DELETE' THEN
        -- Decrement student count
        UPDATE exams
        SET current_students_count = current_students_count - 1
        WHERE id = OLD.exam_id AND current_students_count > 0;

    ELSIF TG_OP = 'UPDATE' AND NEW.attendance_status = 'cancelled'
          AND OLD.attendance_status != 'cancelled' THEN
        -- Decrement when cancelled
        UPDATE exams
        SET current_students_count = current_students_count - 1
        WHERE id = NEW.exam_id AND current_students_count > 0;
    END IF;

    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;
```

```sql
CREATE TRIGGER trigger_manage_exam_capacity
AFTER INSERT OR UPDATE OR DELETE ON exam_registrations
FOR EACH ROW
EXECUTE FUNCTION manage_exam_capacity();
```

2.4 Conflict Detection Function

Check for scheduling conflicts for instructors, vehicles, and students.

```sql
CREATE OR REPLACE FUNCTION check_lesson_conflicts(
    p_instructor_id INTEGER,
    p_vehicle_id INTEGER,
    p_student_id INTEGER,
    p_lesson_date DATE,
    p_start_time TIME,
    p_end_time TIME,
    p_exclude_lesson_id INTEGER DEFAULT NULL
)
RETURNS TABLE(
    conflict_type VARCHAR,
    conflict_entity VARCHAR,
    conflicting_lesson_id INTEGER
) AS $$
BEGIN
    -- Check instructor conflicts
    RETURN QUERY
    SELECT
        'instructor'::VARCHAR,
        'Instructor already booked'::VARCHAR,
        id
    FROM lessons
    WHERE instructor_id = p_instructor_id
        AND lesson_date = p_lesson_date
        AND status IN ('scheduled', 'in_progress')
        AND (p_exclude_lesson_id IS NULL OR id !=
        p_exclude_lesson_id)
        AND (
            (start_time, end_time) OVERLAPS (p_start_time,
        p_end_time)
        );

    -- Check vehicle conflicts
    RETURN QUERY
    SELECT
        'vehicle'::VARCHAR,
        'Vehicle already booked'::VARCHAR,
        id
```

```
    FROM lessons
    WHERE vehicle_id = p_vehicle_id
        AND lesson_date = p_lesson_date
        AND status IN ('scheduled', 'in_progress')
        AND (p_exclude_lesson_id IS NULL OR id !=
        p_exclude_lesson_id)
        AND (
            (start_time, end_time) OVERLAPS (p_start_time,
        p_end_time)
        );

    -- Check student conflicts
    RETURN QUERY
    SELECT
        'student'::VARCHAR,
        'Student already has a lesson'::VARCHAR,
        id
    FROM lessons
    WHERE student_id = p_student_id
        AND lesson_date = p_lesson_date
        AND status IN ('scheduled', 'in_progress')
        AND (p_exclude_lesson_id IS NULL OR id !=
        p_exclude_lesson_id)
        AND (
            (start_time, end_time) OVERLAPS (p_start_time,
        p_end_time)
        );
END;
$$ LANGUAGE plpgsql;
```

---

3. Database Views

3.1 Student Progress View

Consolidated view of student progress.

```
CREATE OR REPLACE VIEW v_student_progress AS
SELECT
    s.id AS student_id,
    u.first_name,
    u.last_name,
    u.email,
    s.student_id_number,
    c.name AS category,
    c.full_name AS category_full_name,
    tt.name AS transmission_type,

    lc.completed_driving_lessons,
    lc.completed_theory_lessons,
```

```
    lc.total_driving_hours,
    lc.required_driving_hours,
    lc.progress_percentage,

    s.theory_exam_passed,
    s.practical_exam_passed,
    s.enrollment_date,
    s.graduation_date,

    CASE
        WHEN s.practical_exam_passed THEN 'Graduated'
        WHEN s.theory_exam_passed THEN 'Theory Passed - Practical
        Pending'
        WHEN lc.progress_percentage >= 100 THEN 'Ready for Theory
        Exam'
        WHEN lc.progress_percentage >= 50 THEN 'In Progress'
        ELSE 'Getting Started'
    END AS status,

    i.id AS instructor_id,
    iu.first_name AS instructor_first_name,
    iu.last_name AS instructor_last_name
FROM students s
JOIN users u ON s.user_id = u.id
LEFT JOIN categories c ON s.category_id = c.id
LEFT JOIN transmission_types tt ON s.transmission_type_id = tt.id
LEFT JOIN lesson_counters lc ON lc.student_id = s.id AND
        lc.category_id = s.category_id
LEFT JOIN instructors i ON s.preferred_instructor_id = i.id
LEFT JOIN users iu ON i.user_id = iu.id
WHERE u.is_active = TRUE;
```

3.2 Instructor Schedule View

Daily schedule for instructors.

```
CREATE OR REPLACE VIEW v_instructor_schedule AS
SELECT
    l.id AS lesson_id,
    l.lesson_date,
    l.start_time,
    l.end_time,
    l.lesson_type,
    l.status,

    i.id AS instructor_id,
    iu.first_name AS instructor_first_name,
    iu.last_name AS instructor_last_name,
```

```
    s.id AS student_id,
    su.first_name AS student_first_name,
    su.last_name AS student_last_name,
    su.phone_number AS student_phone,

    v.registration_number AS vehicle_registration,
    v.make AS vehicle_make,
    v.model AS vehicle_model,

    c.name AS category
FROM lessons l
JOIN instructors i ON l.instructor_id = i.id
JOIN users iu ON i.user_id = iu.id
JOIN students s ON l.student_id = s.id
JOIN users su ON s.user_id = su.id
LEFT JOIN vehicles v ON l.vehicle_id = v.id
LEFT JOIN categories c ON l.category_id = c.id
WHERE l.status IN ('scheduled', 'in_progress')
ORDER BY l.lesson_date, l.start_time;
```

3.3 Vehicle Availability View

Current vehicle status and availability.

```
CREATE OR REPLACE VIEW v_vehicle_availability AS
SELECT
    v.id,
    v.registration_number,
    v.make,
    v.model,
    v.status,
    c.name AS category,
    tt.name AS transmission_type,

    -- Current lesson (if in use)
    CASE
        WHEN v.status = 'in_use' THEN (
            SELECT CONCAT(su.first_name, ' ', su.last_name)
            FROM lessons l
            JOIN students s ON l.student_id = s.id
            JOIN users su ON s.user_id = su.id
            WHERE l.vehicle_id = v.id
                AND l.status = 'in_progress'
                AND l.lesson_date = CURRENT_DATE
            LIMIT 1
        )
        ELSE NULL
```

```
    END AS current_student,

    -- Next scheduled lesson
    (
        SELECT l.start_time
        FROM lessons l
        WHERE l.vehicle_id = v.id
            AND l.status = 'scheduled'
            AND l.lesson_date = CURRENT_DATE
            AND l.start_time > CURRENT_TIME
        ORDER BY l.start_time
        LIMIT 1
    ) AS next_available_time,

    v.current_mileage,
    v.next_service_date,
    CASE
        WHEN v.next_service_date < CURRENT_DATE THEN TRUE
        ELSE FALSE
    END AS service_overdue
FROM vehicles v
LEFT JOIN categories c ON v.category_id = c.id
LEFT JOIN transmission_types tt ON v.transmission_type_id = tt.id
WHERE v.is_active = TRUE;
```

REST API Specification

API Design Principles

1. **RESTful conventions**: Use standard HTTP methods (GET, POST, PUT, PATCH, DELETE)
2. **Resource-based URLs**: `/api/v1/resource` pattern
3. **JSON format**: All requests and responses in JSON
4. **HTTP status codes**: Proper use of 2xx, 4xx, 5xx codes
5. **Pagination**: Use `page` and `limit` query parameters
6. **Filtering**: Support query parameters for filtering
7. **Authentication**: JWT bearer tokens in Authorization header
8. **API Versioning**: Version in URL path (`/api/v1/`)

Base URL

```
Production: https://api.drivingschool.com/api/v1
Development: http://localhost:3000/api/v1
```

Standard Response Format

Success Response

```
{
  "success": true,
  "data": { },
  "message": "Operation successful",
  "timestamp": "2025-10-07T10:30:00Z"
}
```

Error Response

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {
        "field": "email",
        "message": "Invalid email format"
      }
    ]
  },
  "timestamp": "2025-10-07T10:30:00Z"
}
```

Paginated Response

```
{
  "success": true,
  "data": [],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8,
    "hasNext": true,
    "hasPrev": false
  },
  "timestamp": "2025-10-07T10:30:00Z"
}
```

Authentication Endpoints

1.1 Register User

POST `/auth/register`

Register a new user (student or instructor).

Request:

```
{
  "email": "john.doe@example.com",
  "password": "SecureP@ss123",
  "firstName": "John",
  "lastName": "Doe",
  "phoneNumber": "+1234567890",
  "dateOfBirth": "2000-05-15",
  "role": "student",
  "address": "123 Main St",
  "city": "Springfield",
  "postalCode": "12345",

  // Role-specific fields
  "studentDetails": {
    "categoryId": 3,
    "transmissionTypeId": 1,
    "hasDrivingLicense": false,
    "emergencyContactName": "Jane Doe",
    "emergencyContactPhone": "+1234567891",
    "emergencyContactRelationship": "Mother"
  },

  // OR for instructor
  "instructorDetails": {
    "instructorLicenseNumber": "INS-12345",
    "instructorLicenseExpiry": "2026-12-31",
    "qualifiedCategoryIds": [1, 3, 4],
    "qualifiedTransmissionTypeIds": [1, 2],
    "employmentType": "full_time"
  }
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "userId": 123,
    "email": "john.doe@example.com",
    "role": "student",
    "emailVerificationSent": true
  },
  "message": "Registration successful. Please verify your email."
```

```
}
```

1.2 Verify Email

POST `/auth/verify-email`

Verify email address using token sent via email.

Request:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Email verified successfully"
}
```

1.3 Login

POST `/auth/login`

Authenticate user and receive JWT tokens.

Request:

```
{
  "email": "john.doe@example.com",
  "password": "SecureP@ss123"
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "user": {
      "id": 123,
      "email": "john.doe@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "role": "student",
      "isEmailVerified": true,
      "isApproved": true
    },
    "tokens": {
      "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
```

```
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "expiresIn": 3600
  }
 }
}
```

1.4 Refresh Token

POST `/auth/refresh`

Get new access token using refresh token.

Request:

```
{
 "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Response: `200 OK`

```
{
 "success": true,
 "data": {
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expiresIn": 3600
 }
}
```

1.5 Forgot Password

POST `/auth/forgot-password`

Request password reset email.

Request:

```
{
 "email": "john.doe@example.com"
}
```

Response: `200 OK`

```
{
 "success": true,
 "message": "Password reset email sent"
}
```

1.6 Reset Password

POST `/auth/reset-password`

Reset password using token from email.

Request:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "newPassword": "NewSecureP@ss123"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Password reset successful"
}
```

1.7 Logout

POST `/auth/logout`

Invalidate current session.

Headers:

```
Authorization: Bearer {accessToken}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Logged out successfully"
}
```

Student Endpoints

Authentication Required: All endpoints require valid JWT token

2.1 Get Current Student Profile

GET `/students/me`

Get logged-in student's profile.

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": 45,
```

```
    "userId": 123,
    "studentIdNumber": "STU-2025-045",
    "firstName": "John",
    "lastName": "Doe",
    "email": "john.doe@example.com",
    "phoneNumber": "+1234567890",
    "dateOfBirth": "2000-05-15",
    "category": {
      "id": 3,
      "name": "B",
      "fullName": "Car"
    },
    "transmissionType": {
      "id": 1,
      "name": "Manual"
    },
    "progress": {
      "completedDrivingLessons": 15,
      "totalDrivingHours": 22.5,
      "requiredDrivingHours": 30.0,
      "progressPercentage": 75.0,
      "theoryExamPassed": true,
      "practicalExamPassed": false
    },
    "preferredInstructor": {
      "id": 12,
      "firstName": "Sarah",
      "lastName": "Johnson"
    }
  }
}
```

2.2 Update Student Profile

PATCH `/students/me`

Update logged-in student's profile.

Request:


```
{
  "phoneNumber": "+1234567899",
  "address": "456 Oak Avenue",
  "preferredLessonTime": "afternoon",
  "emergencyContactPhone": "+1234567892"
}
```


Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": 45,
    "phoneNumber": "+1234567899",
    "address": "456 Oak Avenue"
  },
  "message": "Profile updated successfully"
}
```

2.3 Get All Students (Admin/Instructor)

GET `/students`

Get list of all students with filtering and pagination.

Query Parameters: - `page` (integer): Page number (default: 1) - `limit` (integer): Items per page (default: 20) - `search` (string): Search by name or email - `categoryId` (integer): Filter by category - `isApproved` (boolean): Filter by approval status - `instructorId` (integer): Filter by preferred instructor

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 45,
      "studentIdNumber": "STU-2025-045",
      "firstName": "John",
      "lastName": "Doe",
      "email": "john.doe@example.com",
      "category": "B",
      "transmissionType": "Manual",
      "enrollmentDate": "2025-08-15",
      "progressPercentage": 75.0,
      "isApproved": true
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

2.4 Get Student by ID (Admin/Instructor)

GET `/students/:id`

Get detailed information about a specific student.

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "id": 45,
    "studentIdNumber": "STU-2025-045",
    "user": {
      "firstName": "John",
      "lastName": "Doe",
      "email": "john.doe@example.com",
      "phoneNumber": "+1234567890"
    },
    "category": {
      "id": 3,
      "name": "B"
    },
    "transmissionType": {
      "id": 1,
      "name": "Manual"
    },
    "progress": {
      "completedDrivingLessons": 15,
      "totalDrivingHours": 22.5
    },
    "upcomingLessons": [
      {
        "id": 234,
        "lessonDate": "2025-10-10",
        "startTime": "10:00",
        "endTime": "11:00",
        "lessonType": "driving"
      }
    ]
  }
}
```

2.5 Approve Student (Admin)

POST `/students/:id/approve`

Approve a pending student registration.

Response: `200 OK`

```
{
  "success": true,
  "message": "Student approved successfully"
}
```

2.6 Reject Student (Admin)

POST `/students/:id/reject`

Reject a pending student registration.

Request:

```
{
  "reason": "Incomplete documentation"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Student registration rejected"
}
```

Instructor Endpoints

3.1 Get Current Instructor Profile

GET `/instructors/me`

Get logged-in instructor's profile.

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": 12,
    "userId": 89,
    "instructorIdNumber": "INS-2024-012",
    "firstName": "Sarah",
    "lastName": "Johnson",
    "email": "sarah.j@example.com",
    "instructorLicenseNumber": "INS-12345",
    "instructorLicenseExpiry": "2026-12-31",
    "qualifiedCategories": [
      {"id": 1, "name": "A"},
```

```
      {"id": 3, "name": "B"}
    ],
    "qualifiedTransmissionTypes": [
      {"id": 1, "name": "Manual"},
      {"id": 2, "name": "Automatic"}
    ],
    "performance": {
      "totalLessonsCompleted": 245,
      "averageRating": 4.7,
      "totalStudentsTrained": 68,
      "passRatePercentage": 87.5
    },
    "isAvailableForBooking": true
  }
}
```

3.2 Update Instructor Profile

PATCH `/instructors/me`

Update logged-in instructor's profile.

Request:

```
{
  "phoneNumber": "+1234567899",
  "isAvailableForBooking": false,
  "defaultWorkingHours": {
    "monday": {"start": "09:00", "end": "17:00"},
    "tuesday": {"start": "09:00", "end": "17:00"}
  }
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Profile updated successfully"
}
```

3.3 Get All Instructors

GET `/instructors`

Get list of all instructors.

Query Parameters: - `page`, `limit`, `search` - `categoryId`: Filter by qualified category - `isAvailable`: Filter by availability - `isApproved`: Filter by approval status

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 12,
      "instructorIdNumber": "INS-2024-012",
      "firstName": "Sarah",
      "lastName": "Johnson",
      "qualifiedCategories": ["A", "B"],
      "averageRating": 4.7,
      "isAvailableForBooking": true
    }
  ],
  "pagination": {}
}
```

3.4 Get Instructor Schedule

GET `/instructors/:id/schedule`

Get instructor's schedule for a date range.

Query Parameters: - `startDate` (date): Start date (required) - `endDate` (date): End date (required)

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "instructorId": 12,
    "instructorName": "Sarah Johnson",
    "lessons": [
      {
        "id": 234,
        "lessonDate": "2025-10-10",
        "startTime": "10:00",
        "endTime": "11:00",
        "lessonType": "driving",
        "status": "scheduled",
        "student": {
          "id": 45,
          "name": "John Doe"
        },
        "vehicle": {
          "id": 5,
          "registration": "ABC-123"
```

```
        }
      }
    ]
  }
}
```

3.5 Get Instructor Availability

GET `/instructors/:id/availability`

Check instructor availability for a specific date.

Query Parameters: - `date` (date): Date to check (required)

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "date": "2025-10-10",
    "workingHours": {
      "start": "09:00",
      "end": "17:00"
    },
    "availableSlots": [
      {"start": "09:00", "end": "10:00"},
      {"start": "11:00", "end": "12:00"},
      {"start": "14:00", "end": "15:00"}
    ],
    "bookedSlots": [
      {
        "start": "10:00",
        "end": "11:00",
        "lessonId": 234,
        "studentName": "John Doe"
      }
    ]
  }
}
```

Lesson Endpoints

4.1 Create Lesson Request (Student)

POST `/lesson-requests`

Student creates a new lesson request.

Request:

```
{
  "instructorId": 12,
  "requestedDate": "2025-10-15",
  "requestedStartTime": "10:00",
  "requestedEndTime": "11:00",
  "lessonType": "driving",
  "categoryId": 3,
  "preferredVehicleId": 5,
  "studentNotes": "First time on highway"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "id": 567,
    "status": "pending",
    "requestedDate": "2025-10-15",
    "requestedStartTime": "10:00",
    "requestedEndTime": "11:00"
  },
  "message": "Lesson request submitted successfully"
}
```

4.2 Get Lesson Requests (Admin/Instructor)

GET `/lesson-requests`

Get all pending lesson requests.

Query Parameters: - `status`: Filter by status (pending, approved, rejected) - `studentId`: Filter by student - `instructorId`: Filter by instructor - `startDate`, `endDate`: Date range filter

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "id": 567,
      "student": {
        "id": 45,
        "name": "John Doe"
      },
      "instructor": {
```

```
        "id": 12,
        "name": "Sarah Johnson"
      },
      "requestedDate": "2025-10-15",
      "requestedStartTime": "10:00",
      "requestedEndTime": "11:00",
      "lessonType": "driving",
      "status": "pending",
      "createdAt": "2025-10-07T10:30:00Z"
    }
  ],
  "pagination": {}
}
```

4.3 Approve Lesson Request (Admin/Instructor)

POST `/lesson-requests/:id/approve`

Approve a lesson request and create a lesson.

Request:


```
{
  "vehicleId": 5,
  "adminNotes": "Assigned vehicle ABC-123"
}
```


Response: `200 OK`


```
{
  "success": true,
  "data": {
    "lessonRequestId": 567,
    "lessonId": 789,
    "status": "approved"
  },
  "message": "Lesson request approved and lesson created"
}
```

4.4 Reject Lesson Request (Admin/Instructor)

POST `/lesson-requests/:id/reject`

Reject a lesson request.

Request:


```
{
  "rejectionReason": "Instructor not available at requested time"
```

```
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Lesson request rejected"
}
```

4.5 Get Lessons

GET `/lessons`

Get lessons with filtering.

Query Parameters: - `studentId`: Filter by student - `instructorId`: Filter by instructor - `vehicleId`: Filter by vehicle - `startDate`, `endDate`: Date range - `status`: Filter by status - `lessonType`: Filter by type

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "id": 789,
      "lessonDate": "2025-10-15",
      "startTime": "10:00",
      "endTime": "11:00",
      "durationMinutes": 60,
      "lessonType": "driving",
      "status": "scheduled",
      "student": {
        "id": 45,
        "name": "John Doe"
      },
      "instructor": {
        "id": 12,
        "name": "Sarah Johnson"
      },
      "vehicle": {
        "id": 5,
        "registration": "ABC-123"
      },
      "category": "B"
    }
  ],
  "pagination": {}
}
```

4.6 Get Lesson by ID

GET `/lessons/:id`

Get detailed information about a specific lesson.

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "id": 789,
    "lessonDate": "2025-10-15",
    "startTime": "10:00",
    "endTime": "11:00",
    "lessonType": "driving",
    "status": "completed",
    "student": {
      "id": 45,
      "name": "John Doe",
      "phoneNumber": "+1234567890"
    },
    "instructor": {
      "id": 12,
      "name": "Sarah Johnson"
    },
    "vehicle": {
      "id": 5,
      "registration": "ABC-123",
      "make": "Toyota",
      "model": "Corolla"
    },
    "evaluation": {
      "instructorRating": 5,
      "instructorFeedback": "Excellent progress on highway driving",
      "studentPerformanceRating": 4,
      "skillsPracticed": ["highway_driving", "lane_changing",
        "merging"],
      "areasForImprovement": "More practice needed on parallel
        parking"
    },
    "mileage": {
      "start": 45000,
      "end": 45025
    }
  }
}
```

4.7 Update Lesson

PATCH `/lessons/:id`

Update lesson details (Admin/Instructor only).

Request:

```
{
  "startTime": "11:00",
  "endTime": "12:00",
  "vehicleId": 6,
  "pickupLocation": "123 Main St",
  "adminNotes": "Time changed due to instructor request"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Lesson updated successfully"
}
```

4.8 Start Lesson (Instructor)

POST `/lessons/:id/start`

Mark lesson as in progress and record start time.

Request:

```
{
  "startMileage": 45000
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": 789,
    "status": "in_progress",
    "startedAt": "2025-10-15T10:05:00Z"
  }
}
```

4.9 Complete Lesson (Instructor)

POST `/lessons/:id/complete`

Mark lesson as completed and provide evaluation.

Request:

```
{
  "endMileage": 45025,
  "instructorFeedback": "Excellent progress on highway driving",
  "studentPerformanceRating": 4,
  "skillsPracticed": ["highway_driving", "lane_changing"],
  "areasForImprovement": "More practice needed on parallel parking"
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "id": 789,
    "status": "completed",
    "completedAt": "2025-10-15T11:00:00Z"
  },
  "message": "Lesson completed successfully"
}
```

4.10 Cancel Lesson

POST `/lessons/:id/cancel`

Cancel a scheduled lesson.

Request:

```
{
  "cancellationReason": "Student illness"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Lesson cancelled successfully"
}
```

4.11 Rate Lesson (Student)

POST `/lessons/:id/rate`

Student provides rating and feedback for completed lesson.

Request:

```
{
  "instructorRating": 5,
  "studentFeedback": "Great instructor, very patient and clear"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Rating submitted successfully"
}
```

4.12 Check Scheduling Conflicts

POST `/lessons/check-conflicts`

Check for scheduling conflicts before creating a lesson.

Request:

```
{
  "instructorId": 12,
  "vehicleId": 5,
  "studentId": 45,
  "lessonDate": "2025-10-15",
  "startTime": "10:00",
  "endTime": "11:00"
}
```

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "hasConflicts": false,
    "conflicts": []
  }
}
```

Or if conflicts exist:

```
{
  "success": true,
  "data": {
    "hasConflicts": true,
    "conflicts": [
      {
        "conflictType": "instructor",
        "entity": "Instructor already booked",
        "conflictingLessonId": 788,
```

```
        "timeRange": "10:00 - 11:00"
      }
    ]
  }
}
```

Vehicle Endpoints

5.1 Get All Vehicles

GET `/vehicles`

Get list of all vehicles.

Query Parameters: - `categoryId`: Filter by category - `transmissionTypeId`: Filter by transmission type - `status`: Filter by status (available, in_use, maintenance) - `isActive`: Filter active vehicles

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 5,
      "registrationNumber": "ABC-123",
      "make": "Toyota",
      "model": "Corolla",
      "year": 2023,
      "color": "Silver",
      "category": "B",
      "transmissionType": "Manual",
      "status": "available",
      "currentMileage": 45000,
      "nextServiceDate": "2025-11-15",
      "hasDualControls": true
    }
  ],
  "pagination": {}
}
```

5.2 Get Vehicle by ID

GET `/vehicles/:id`

Get detailed information about a specific vehicle.

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "id": 5,
    "registrationNumber": "ABC-123",
    "make": "Toyota",
    "model": "Corolla",
    "year": 2023,
    "color": "Silver",
    "vin": "1HGBH41JXMN109186",
    "category": {
      "id": 3,
      "name": "B"
    },
    "transmissionType": {
      "id": 1,
      "name": "Manual"
    },
    "status": "available",
    "currentMileage": 45000,
    "maintenance": {
      "lastServiceDate": "2025-08-15",
      "nextServiceDate": "2025-11-15",
      "lastServiceMileage": 40000,
      "serviceIntervalKm": 10000
    },
    "insurance": {
      "policyNumber": "INS-987654",
      "expiryDate": "2026-06-30",
      "company": "ABC Insurance Co."
    },
    "features": {
      "hasDualControls": true,
      "hasDashcam": false,
      "fuelType": "petrol"
    }
  }
}
```

5.3 Create Vehicle (Admin)

POST `/vehicles`

Add a new vehicle to the fleet.

Request:

```
{
  "registrationNumber": "XYZ-789",
  "make": "Honda",
  "model": "Civic",
  "year": 2024,
  "color": "Blue",
  "vin": "2HGBH41JXMN109187",
  "categoryId": 3,
  "transmissionTypeId": 2,
  "fuelType": "petrol",
  "hasDualControls": true,
  "insurancePolicyNumber": "INS-123456",
  "insuranceExpiryDate": "2026-12-31",
  "insuranceCompany": "XYZ Insurance"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "id": 15,
    "registrationNumber": "XYZ-789"
  },
  "message": "Vehicle added successfully"
}
```

5.4 Update Vehicle (Admin)

PATCH `/vehicles/:id`

Update vehicle information.

Request:

```
{
  "currentMileage": 46000,
  "status": "maintenance",
  "nextServiceDate": "2025-12-15"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Vehicle updated successfully"
}
```

5.5 Delete Vehicle (Admin)

DELETE `/vehicles/:id`

Soft delete a vehicle (mark as inactive).

Response: `200 OK`

```
{
  "success": true,
  "message": "Vehicle deleted successfully"
}
```

5.6 Get Vehicle Availability

GET `/vehicles/:id/availability`

Check vehicle availability for a date range.

Query Parameters: - `startDate` (date): Start date - `endDate` (date): End date

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "vehicleId": 5,
    "vehicle": "ABC-123 - Toyota Corolla",
    "status": "available",
    "bookedSlots": [
      {
        "lessonId": 789,
        "date": "2025-10-15",
        "startTime": "10:00",
        "endTime": "11:00",
        "instructor": "Sarah Johnson"
      }
    ],
    "maintenanceScheduled": []
  }
}
```

Exam Endpoints

6.1 Get All Exams

GET `/exams`

Get list of all exams.

Query Parameters: - `examType`: Filter by type (theory, practical) - `categoryId`: Filter by category - `startDate`, `endDate`: Date range filter - `status`: Filter by status

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 25,
      "examType": "practical",
      "category": "B",
      "examDate": "2025-10-20",
      "startTime": "09:00",
      "endTime": "17:00",
      "examLocation": "Main Testing Center",
      "maxStudents": 12,
      "currentStudentsCount": 8,
      "availableSlots": 4,
      "status": "scheduled"
    }
  ],
  "pagination": {}
}
```

6.2 Get Exam by ID

GET `/exams/:id`

Get detailed information about a specific exam.

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "id": 25,
    "examType": "practical",
    "category": {
      "id": 3,
      "name": "B"
    },
    "examDate": "2025-10-20",
    "startTime": "09:00",
    "endTime": "17:00",
    "examLocation": "Main Testing Center",
    "examCenterName": "State Driving Test Center",
    "vehicle": {
```

```
      "id": 5,
      "registration": "ABC-123"
    },
    "examiner": {
      "id": 12,
      "name": "Sarah Johnson"
    },
    "capacity": {
      "max": 12,
      "current": 8,
      "available": 4
    },
    "registeredStudents": [
      {
        "id": 45,
        "name": "John Doe",
        "attemptNumber": 1,
        "registrationDate": "2025-10-01"
      }
    ]
  }
}
```

6.3 Create Exam (Admin)

POST `/exams`

Schedule a new exam.

Request:


```
{
  "examType": "practical",
  "categoryId": 3,
  "examDate": "2025-11-15",
  "startTime": "09:00",
  "endTime": "17:00",
  "examLocation": "Main Testing Center",
  "examCenterName": "State Driving Test Center",
  "vehicleId": 5,
  "examinerId": 12,
  "maxStudents": 12,
  "specialInstructions": "Bring original ID"
}
```


Response: `201 Created`


```
{
  "success": true,
```

```
  "data": {
    "id": 26,
    "examDate": "2025-11-15"
  },
  "message": "Exam scheduled successfully"
}
```

6.4 Register for Exam (Student)

POST `/exams/:id/register`

Student registers for an exam.

Request:

```
{
  "paymentMethod": "credit_card"
}
```

Response: `201 Created`

```
{
  "success": true,
  "data": {
    "examRegistrationId": 89,
    "examId": 25,
    "examDate": "2025-10-20",
    "registrationFee": 75.00,
    "paymentStatus": "pending"
  },
  "message": "Successfully registered for exam"
}
```

6.5 Cancel Exam Registration (Student)

DELETE `/exams/:id/register`

Student cancels their exam registration.

Response: `200 OK`

```
{
  "success": true,
  "message": "Exam registration cancelled successfully"
}
```

6.6 Record Exam Result (Admin/Examiner)

POST `/exam-registrations/:id/result`

Record the result of an exam for a student.

Request:

```
{
  "attendanceStatus": "present",
  "result": "pass",
  "score": 92.5,
  "examinerComments": "Excellent driving skills, confident on all
        maneuvers"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Exam result recorded successfully"
}
```

6.7 Get Student Exam History

GET `/students/:id/exams`

Get all exam attempts for a student.

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "id": 89,
      "examType": "theory",
      "category": "B",
      "examDate": "2025-09-15",
      "attemptNumber": 1,
      "result": "pass",
      "score": 92.5,
      "resultDate": "2025-09-15"
    },
    {
      "id": 95,
      "examType": "practical",
      "category": "B",
      "examDate": "2025-10-20",
      "attemptNumber": 1,
      "result": "pending"
    }
  ]
}
```

Category & Transmission Type Endpoints

7.1 Get All Categories

GET `/categories`

Get list of all driving categories.

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 3,
      "name": "B",
      "fullName": "Car",
      "description": "Passenger vehicles",
      "transmissionType": {
        "id": 1,
        "name": "Manual"
      },
      "minAge": 18,
      "requiresTheoryExam": true,
      "requiresPracticalExam": true,
      "minLessonHours": 30
    }
  ]
}
```

7.2 Get All Transmission Types

GET `/transmission-types`

Get list of all transmission types.

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "Manual",
      "code": "MT",
      "description": "Manual transmission vehicles"
    },
    {
```

```
      "id": 2,
      "name": "Automatic",
      "code": "AT",
      "description": "Automatic transmission vehicles"
    }
  ]
}
```

Dashboard & Analytics Endpoints

8.1 Get Student Dashboard

GET `/dashboard/student`

Get dashboard data for logged-in student.

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "progress": {
      "completedLessons": 15,
      "totalHours": 22.5,
      "requiredHours": 30.0,
      "progressPercentage": 75.0
    },
    "upcomingLessons": [
      {
        "id": 789,
        "date": "2025-10-15",
        "time": "10:00 - 11:00",
        "instructor": "Sarah Johnson",
        "type": "driving"
      }
    ],
    "nextExam": {
      "id": 25,
      "type": "practical",
      "date": "2025-10-20",
      "location": "Main Testing Center"
    },
    "recentActivity": [
      {
        "type": "lesson_completed",
        "date": "2025-10-05",
        "description": "Driving lesson completed"
      }
```

```
    ]
  }
}
```

8.2 Get Instructor Dashboard

GET `/dashboard/instructor`

Get dashboard data for logged-in instructor.

Response: `200 OK`

```
{
  "success": true,
  "data": {
    "todaysSchedule": [
      {
        "lessonId": 789,
        "time": "10:00 - 11:00",
        "student": "John Doe",
        "type": "driving",
        "vehicle": "ABC-123"
      }
    ],
    "weeklyStats": {
      "totalLessons": 18,
      "hoursWorked": 18,
      "cancelledLessons": 1
    },
    "pendingRequests": 3,
    "performance": {
      "averageRating": 4.7,
      "totalStudents": 68,
      "passRate": 87.5
    }
  }
}
```

8.3 Get Admin Dashboard

GET `/dashboard/admin`

Get comprehensive dashboard data for admin.

Response: `200 OK`

```
{
  "success": true,
```

```
  "data": {
    "overview": {
      "totalStudents": 150,
      "activeStudents": 120,
      "totalInstructors": 15,
      "activeInstructors": 12,
      "totalVehicles": 20,
      "availableVehicles": 16
    },
    "pendingApprovals": {
      "students": 5,
      "instructors": 2,
      "lessonRequests": 8
    },
    "todaysLessons": {
      "scheduled": 45,
      "completed": 12,
      "inProgress": 3,
      "cancelled": 2
    },
    "upcomingExams": [
      {
        "id": 25,
        "type": "practical",
        "date": "2025-10-20",
        "registeredStudents": 8,
        "capacity": 12
      }
    ],
    "recentActivity": [
      {
        "type": "new_student_registered",
        "timestamp": "2025-10-07T09:30:00Z",
        "description": "John Doe registered"
      }
    ]
  }
}
```

Notification Endpoints

9.1 Get User Notifications

GET `/notifications`

Get notifications for logged-in user.

Query Parameters: - `isRead`: Filter by read status - `type`: Filter by notification type - `limit`: Number of notifications to retrieve

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 456,
      "type": "lesson_scheduled",
      "title": "Lesson Scheduled",
      "message": "Your driving lesson has been scheduled for Oct 15,
        2025 at 10:00 AM",
      "isRead": false,
      "createdAt": "2025-10-07T10:00:00Z",
      "priority": "normal",
      "actionUrl": "/lessons/789"
    }
  ],
  "unreadCount": 5
}
```

9.2 Mark Notification as Read

PATCH `/notifications/:id/read`

Mark a notification as read.

Response: `200 OK`


```
{
  "success": true,
  "message": "Notification marked as read"
}
```

9.3 Mark All Notifications as Read

POST `/notifications/read-all`

Mark all notifications as read for the user.

Response: `200 OK`


```
{
  "success": true,
  "message": "All notifications marked as read"
}
```

Payment Endpoints

10.1 Get Payment History

GET `/payments`

Get payment history.

Query Parameters: - `studentId`: Filter by student (admin only) - `status`: Filter by status - `startDate`, `endDate`: Date range

Response: `200 OK`


```
{
  "success": true,
  "data": [
    {
      "id": 123,
      "amount": 50.00,
      "currency": "USD",
      "paymentMethod": "credit_card",
      "paymentType": "lesson_fee",
      "status": "completed",
      "transactionDate": "2025-10-05T10:30:00Z",
      "receiptNumber": "RCP-2025-123",
      "description": "Driving lesson payment"
    }
  ],
  "pagination": {}
}
```

10.2 Create Payment

POST `/payments`

Create a new payment record.

Request:


```
{
  "amount": 50.00,
  "paymentMethod": "credit_card",
  "paymentType": "lesson_fee",
  "lessonId": 789,
  "description": "Driving lesson payment"
}
```


Response: `201 Created`


```
{
```

```
  "success": true,
  "data": {
    "id": 124,
    "amount": 50.00,
    "status": "pending",
    "receiptNumber": "RCP-2025-124"
  },
  "message": "Payment initiated successfully"
}
```

Report Endpoints

11.1 Generate Student Progress Report

GET `/reports/student/:id/progress`

Generate detailed progress report for a student.

Response: `200 OK` (PDF or JSON)


```
{
  "success": true,
  "data": {
    "student": {
      "id": 45,
      "name": "John Doe",
      "category": "B"
    },
    "summary": {
      "enrollmentDate": "2025-08-15",
      "completedLessons": 15,
      "totalHours": 22.5,
      "progressPercentage": 75.0
    },
    "lessonHistory": [],
    "examHistory": [],
    "instructorFeedback": []
  },
  "reportUrl": "/downloads/reports/student-45-progress.pdf"
}
```

11.2 Generate Instructor Performance Report

GET `/reports/instructor/:id/performance`

Generate performance report for an instructor.

Query Parameters: - `startDate`, `endDate`: Date range

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "instructor": {
      "id": 12,
      "name": "Sarah Johnson"
    },
    "period": {
      "start": "2025-08-01",
      "end": "2025-10-31"
    },
    "metrics": {
      "totalLessons": 245,
      "totalHours": 245.0,
      "averageRating": 4.7,
      "studentsAssigned": 35,
      "completionRate": 94.3
    },
    "studentFeedback": []
  }
}
```

11.3 Generate Financial Report (Admin)

GET `/reports/financial`

Generate financial report.

Query Parameters: - `startDate`, `endDate`: Date range - `format`: json or pdf

Response: `200 OK`


```
{
  "success": true,
  "data": {
    "period": {
      "start": "2025-08-01",
      "end": "2025-10-31"
    },
    "summary": {
      "totalRevenue": 45000.00,
      "lessonRevenue": 35000.00,
      "examRevenue": 10000.00,
      "totalExpenses": 15000.00,
      "netIncome": 30000.00
    },
```

```
    "breakdown": {
      "byPaymentType": {},
      "byMonth": {}
    }
  }
}
```

System Settings Endpoints

12.1 Get System Settings (Admin)

GET `/settings`

Get all system settings.

Query Parameters: - `category`: Filter by category

Response: `200 OK`

```
{
  "success": true,
  "data": [
    {
      "key": "default_lesson_duration",
      "value": "60",
      "type": "integer",
      "description": "Default lesson duration in minutes",
      "category": "lessons"
    }
  ]
}
```

12.2 Update System Setting (Admin)

PATCH `/settings/:key`

Update a system setting.

Request:

```
{
  "value": "90"
}
```

Response: `200 OK`

```
{
  "success": true,
  "message": "Setting updated successfully"
```

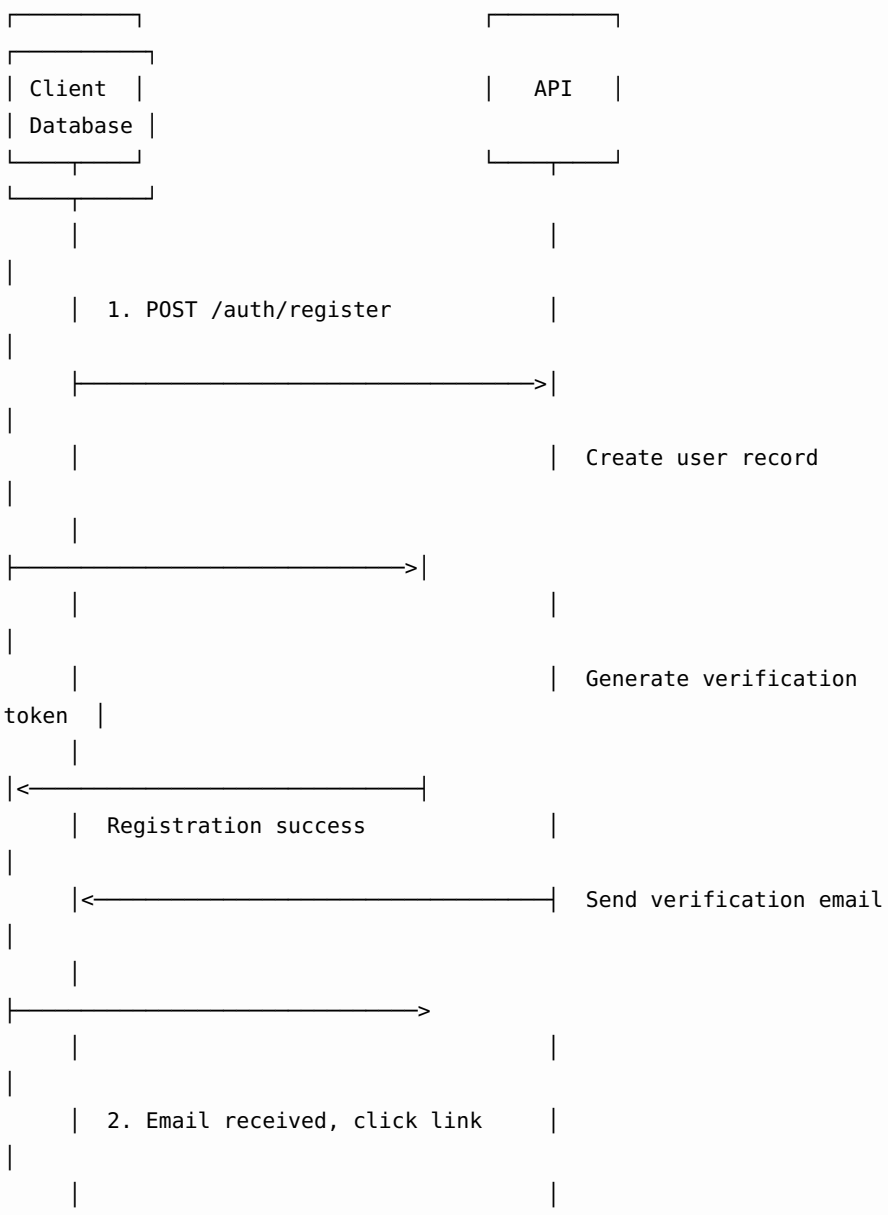
```
}
```

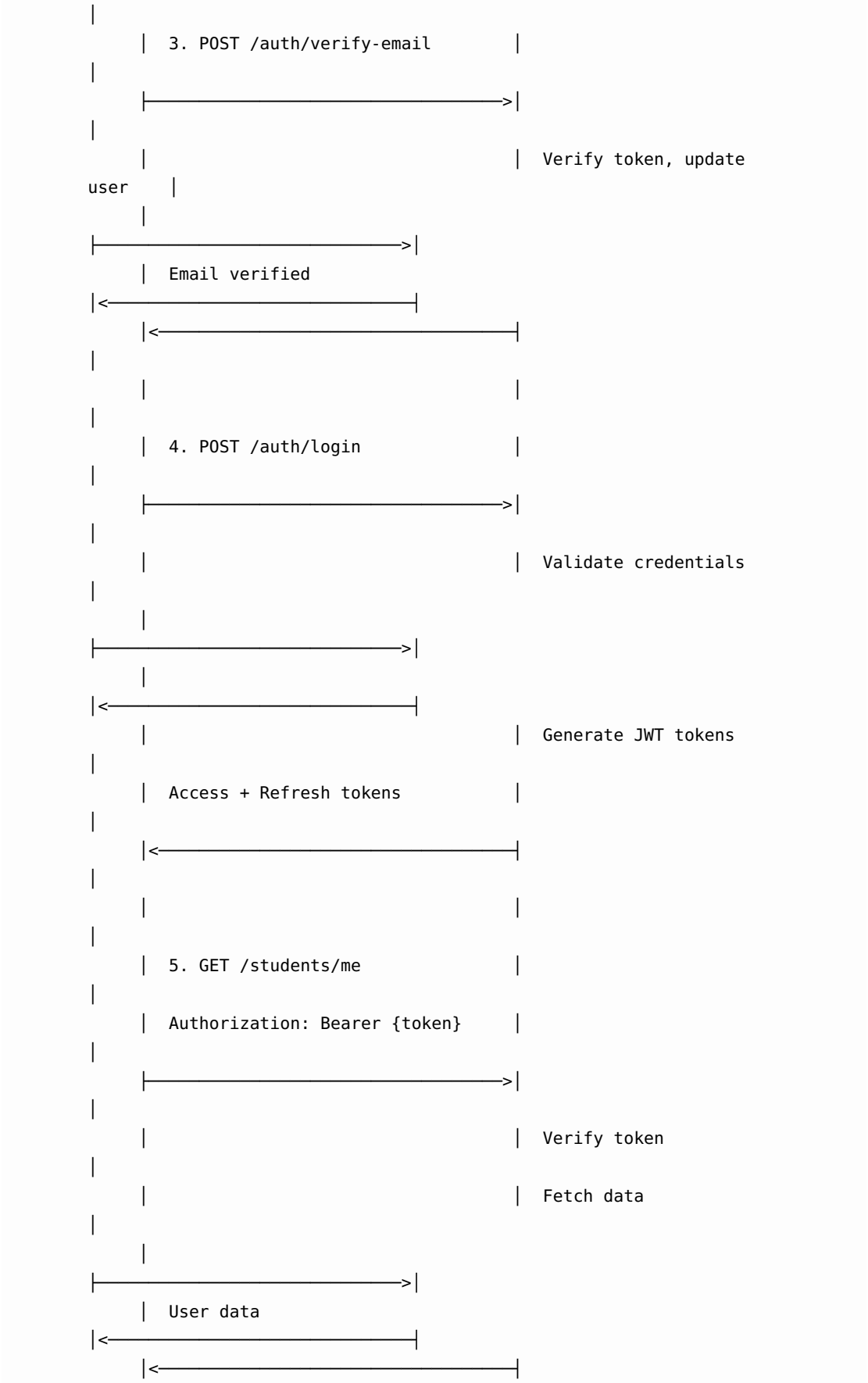
Authentication & Authorization

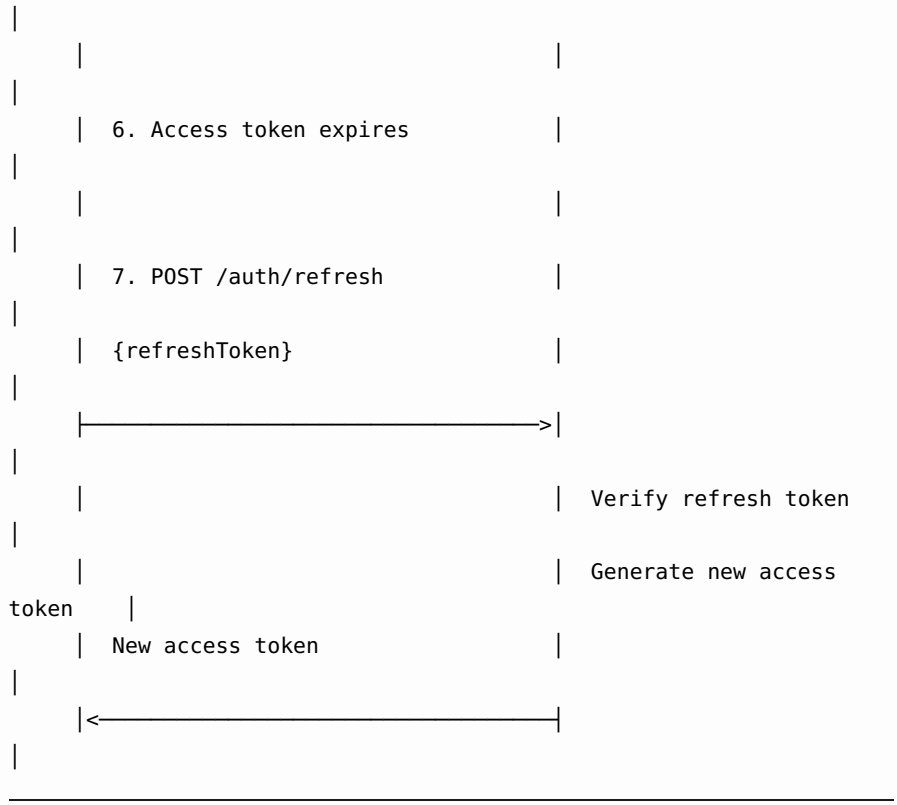
Overview

The platform uses JWT (JSON Web Tokens) for authentication with a two-token system: - **Access Token**: Short-lived (1 hour), used for API requests - **Refresh Token**: Long-lived (30 days), used to obtain new access tokens

Authentication Flow




3. POST /auth/verify-email
Verify token, update user
Email verified
4. POST /auth/login
Validate credentials
Generate JWT tokens
Access + Refresh tokens
5. GET /students/me
Authorization: Bearer {token}
Verify token
Fetch data
User data



Registration Flow with Role Selection

Student Registration

```
1. User visits registration page
2. Selects "I am a Student"
3. Fills form:
   - Email, password
   - Personal information (name, phone, DOB)
   - Address details
   - Category selection (A, A1, B, etc.)
   - Transmission type (Manual/Automatic)
   - Emergency contact information
   - Upload medical certificate (optional)
4. Submits registration
5. Account created with status: is_email_verified=false,
is_approved=false
6. Verification email sent
7. User clicks verification link
8. Email verified: is_email_verified=true
9. Admin reviews and approves: is_approved=true
10. User can now login and book lessons
```

Instructor Registration

```
1. User visits registration page
2. Selects "I am an Instructor"
3. Fills form:
   - Email, password
   - Personal information
   - Instructor license number and expiry
   - Qualified categories (checkboxes: A, B, C, etc.)
   - Qualified transmission types (checkboxes)
   - Upload instructor certificate
   - Employment preferences
4. Submits registration
5. Account created with status: is_email_verified=false,
is_approved=false
6. Verification email sent
7. User clicks verification link
8. Email verified: is_email_verified=true
9. Admin reviews credentials and approves: is_approved=true
10. User can now login and manage lessons
```

Super Admin Creation

Super Admins are not created through registration. They are: 1. Created directly in the database via migration script 2. Or promoted from existing users by another Super Admin 3. Cannot be created through the public API

Role-Based Access Control (RBAC)

Roles Overview

Role	Description	Can Access
Student	Enrolled student learning to drive	Own profile, lessons, exams, progress
Instructor	Driving instructor	Own profile, assigned lessons, student performance, schedule
Super Admin	System administrator	All resources, user management, system settings

---

Permission Matrix

Resource	Student	Instructor	Super Admin
Own Profile			
View	✓	✓	✓

Edit	✓	✓	✓
Delete	✗	✗	✓
Other Profiles			
View	✗	Students only	✓
Edit	✗	✗	✓
Approve	✗	✗	✓
Lessons			
View own	✓	✓	✓
View all	✗	Assigned only	✓
Request	✓	✗	✓
Create	✗	✗	✓
Update	✗	✗	✓
Cancel	Own only	✗	✓
Start/Complete	✗	Assigned only	✓
Rate	Own only	✗	✗
Lesson Requests			
Create	✓	✗	✓
View	Own only	Assigned only	✓
Approve/Reject	✗	Assigned only	✓
Exams			
View	✓	✓	✓
Register	✓	✗	✓
Create	✗	✗	✓
Record Results	✗	As examiner	✓
Vehicles			
View	✗	✓	✓
Create	✗	✗	✓
Update	✗	✗	✓
Delete	✗	✗	✓
Payments			
View own	✓	✗	✓
View all	✗	✗	✓
Create	✓	✗	✓
Reports			
Own progress	✓	✗	✓
Instructor performance	✗	Own only	✓
Financial	✗	✗	✓
System Settings			

View	✗	✗	✓
Update	✗	✗	✓

JWT Token Structure

Access Token Payload

```
{
  "userId": 123,
  "email": "john.doe@example.com",
  "role": "student",
  "studentId": 45,
  "isEmailVerified": true,
  "isApproved": true,
  "iat": 1696680000,
  "exp": 1696683600,
  "type": "access"
}
```

Refresh Token Payload

```
{
  "userId": 123,
  "email": "john.doe@example.com",
  "iat": 1696680000,
  "exp": 1699272000,
  "type": "refresh"
}
```

API Authentication Requirements

Public Endpoints (No Authentication)

- `POST /auth/register`
- `POST /auth/login`
- `POST /auth/verify-email`
- `POST /auth/forgot-password`
- `POST /auth/reset-password`
- `GET /categories`
- `GET /transmission-types`

Protected Endpoints (Authentication Required)

All other endpoints require a valid access token in the Authorization header:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Authorization Middleware Flow

```
// Pseudocode
function authenticateToken(req, res, next) {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'No token provided' });
  }

  try {
    const decoded = jwt.verify(token, JWT_SECRET);

    // Check if email is verified
    if (!decoded.isEmailVerified) {
      return res.status(403).json({ error: 'Email not verified' });
    }

    // Check if user is approved
    if (!decoded.isApproved) {
      return res.status(403).json({ error: 'Account pending
        approval' });
    }

    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ error: 'Invalid token' });
  }
}

function authorizeRole(...roles) {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Insufficient
        permissions' });
    }
    next();
  };
}

// Usage
app.get('/students', authenticateToken, authorizeRole('super_admin',
        'instructor'), getStudents);
app.get('/students/me', authenticateToken, authorizeRole('student'),
        getCurrentStudent);
```

Security Best Practices

Password Requirements

- Minimum 8 characters
- At least one uppercase letter
- At least one lowercase letter
- At least one number
- At least one special character

Password Storage

- Use bcrypt with salt rounds = 12
- Never store plain text passwords
- Implement rate limiting on login attempts (5 attempts per 15 minutes)

Token Security

- Access tokens expire in 1 hour
- Refresh tokens expire in 30 days
- Invalidate refresh tokens on logout
- Implement token blacklist for compromised tokens
- Use HTTPS only in production
- Set secure, httpOnly cookies for web clients

API Security

- Implement rate limiting (100 requests per 15 minutes per IP)
- Validate all input data
- Sanitize output to prevent XSS
- Use parameterized queries to prevent SQL injection
- Implement CORS with whitelist
- Log all authentication attempts
- Implement audit logging for sensitive operations

UI/UX Design Guidelines

Design Principles

1. **Simplicity**: Clean, uncluttered interfaces
2. **Consistency**: Uniform design patterns across all views
3. **Accessibility**: WCAG 2.1 AA compliance
4. **Responsiveness**: Mobile-first approach
5. **Feedback**: Clear feedback for all user actions

Color Scheme

Primary Colors

```
/* Lesson Type Colors */
--color-driving: #10B981;      /* Green - Driving lessons */
--color-theory: #3B82F6;       /* Blue - Theory lessons */
--color-exam: #F59E0B;         /* Orange - Exams */

/* Status Colors */
--color-scheduled: #8B5CF6;    /* Purple */
--color-completed: #10B981;    /* Green */
--color-cancelled: #EF4444;    /* Red */
--color-in-progress: #F59E0B;  /* Orange */
--color-pending: #F59E0B;      /* Orange */

/* UI Colors */
--color-primary: #2563EB;      /* Blue */
--color-secondary: #64748B;    /* Slate */
--color-success: #10B981;      /* Green */
--color-warning: #F59E0B;      /* Orange */
--color-error: #EF4444;        /* Red */
--color-info: #3B82F6;         /* Blue */

/* Neutrals */
--color-background: #F8FAFC;   /* Light gray */
--color-surface: #FFFFFF;      /* White */
--color-text-primary: #0F172A; /* Dark slate */
--color-text-secondary: #64748B; /* Slate */
--color-border: #E2E8F0;       /* Light slate */
```

Typography

```
/* Font Family */
--font-primary: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe
        UI', sans-serif;
--font-mono: 'JetBrains Mono', 'Courier New', monospace;

/* Font Sizes */
--text-xs: 0.75rem;    /* 12px */
--text-sm: 0.875rem;   /* 14px */
--text-base: 1rem;     /* 16px */
--text-lg: 1.125rem;   /* 18px */
--text-xl: 1.25rem;    /* 20px */
--text-2xl: 1.5rem;    /* 24px */
--text-3xl: 1.875rem;  /* 30px */
--text-4xl: 2.25rem;   /* 36px */
```

```
/* Font Weights */
--font-normal: 400;
--font-medium: 500;
--font-semibold: 600;
--font-bold: 700;
```

Spacing System

```
--space-1: 0.25rem;   /* 4px */
--space-2: 0.5rem;    /* 8px */
--space-3: 0.75rem;   /* 12px */
--space-4: 1rem;      /* 16px */
--space-5: 1.25rem;   /* 20px */
--space-6: 1.5rem;    /* 24px */
--space-8: 2rem;      /* 32px */
--space-10: 2.5rem;   /* 40px */
--space-12: 3rem;     /* 48px */
--space-16: 4rem;     /* 64px */
```

Layout Guidelines

Mobile Layout (< 768px)

- Single column layout
- Full-width components
- Bottom navigation bar
- Collapsible headers
- Swipeable calendar views
- Bottom sheets for modals

Tablet Layout (768px - 1024px)

- Two-column layout where appropriate
- Sidebar navigation (collapsible)
- Grid layouts for cards
- Side panels for details

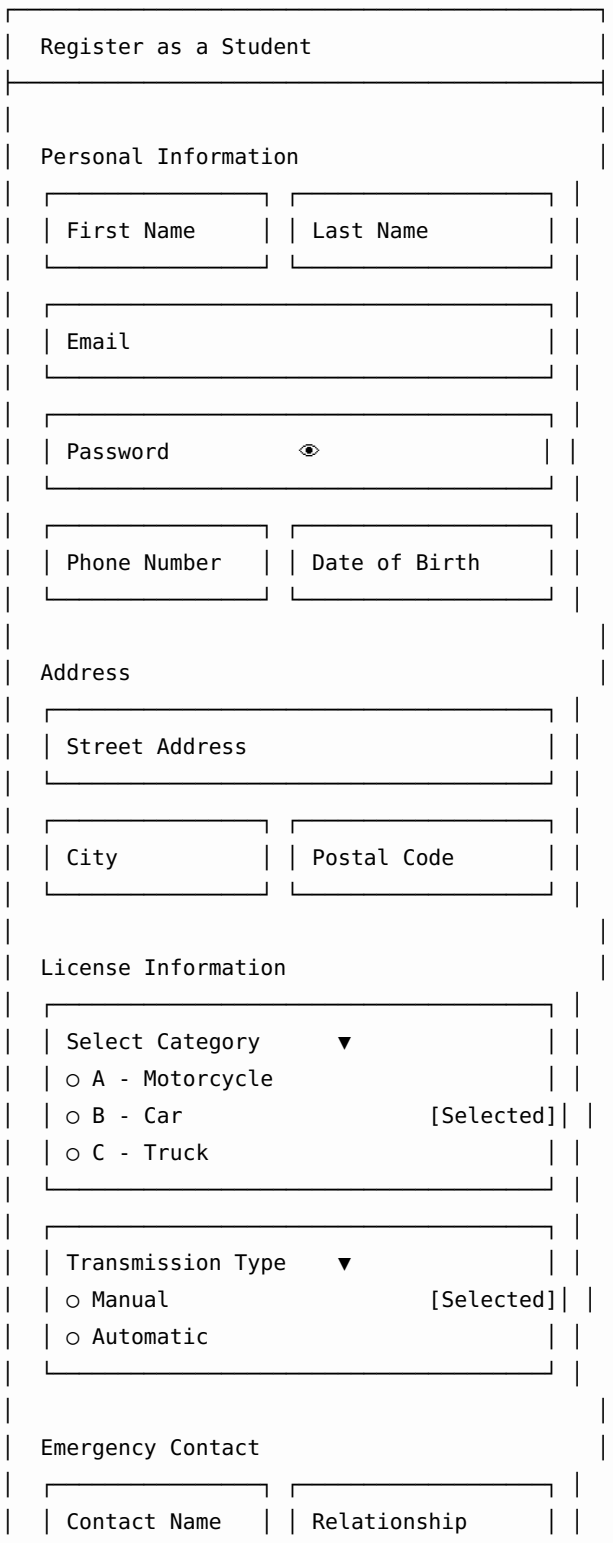
Desktop Layout (> 1024px)

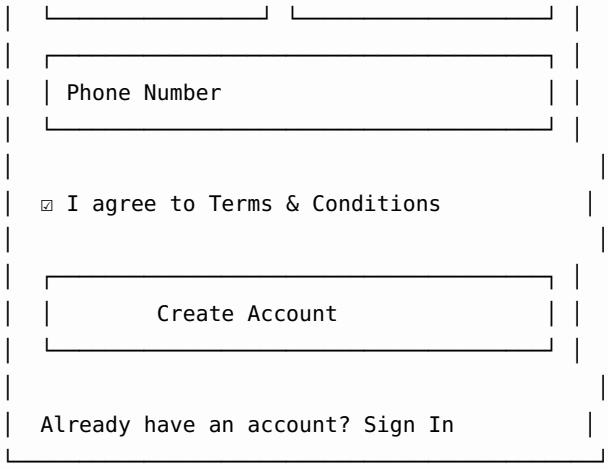
- Multi-column layouts
- Fixed sidebar navigation
- Expanded calendar views
- Modal dialogs for forms
- Split views (list + details)

Key Views & Components

1. Registration Forms

Student Registration





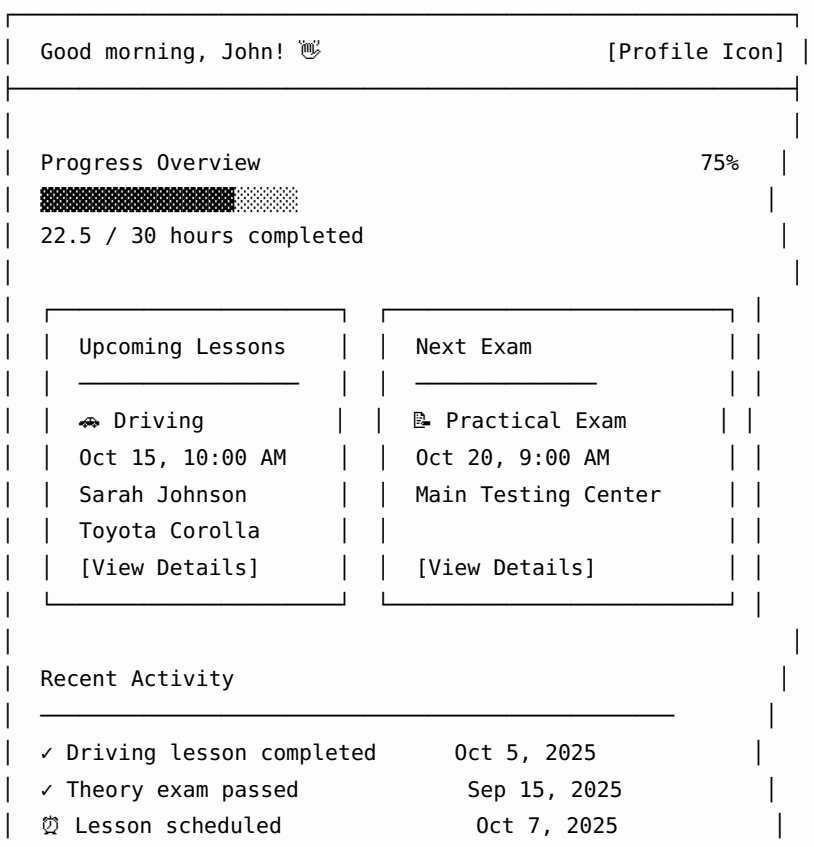
Instructor Registration

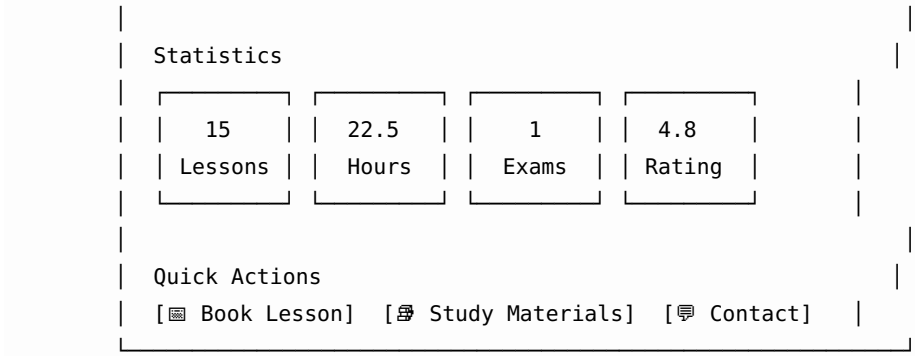
Similar layout but includes: - Instructor license number field - License expiry date - Upload certificate button - Multiple category selection (checkboxes) - Multiple transmission type selection - Employment type dropdown

---

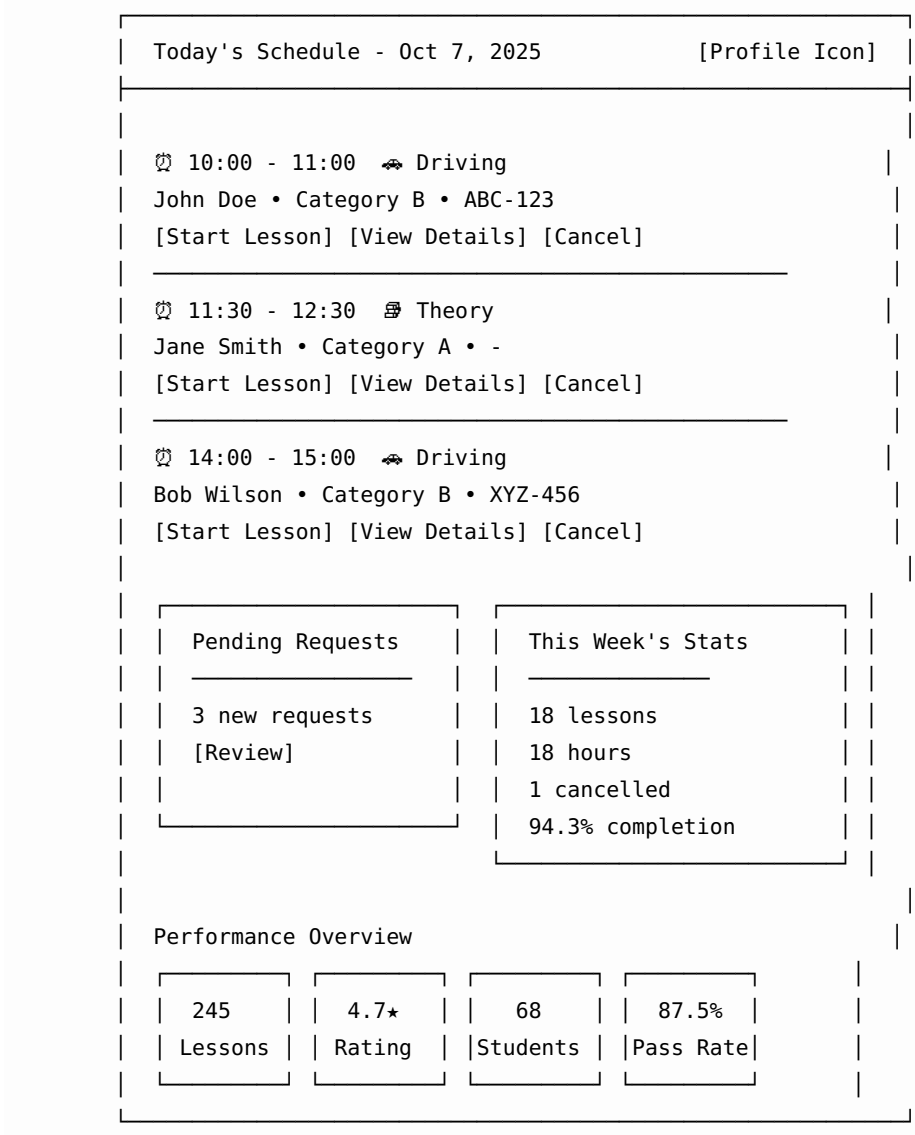
2. Dashboard Views

Student Dashboard

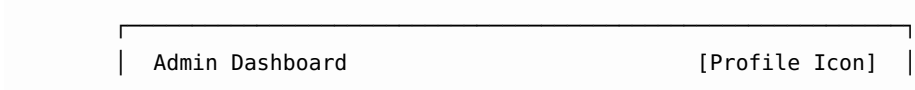


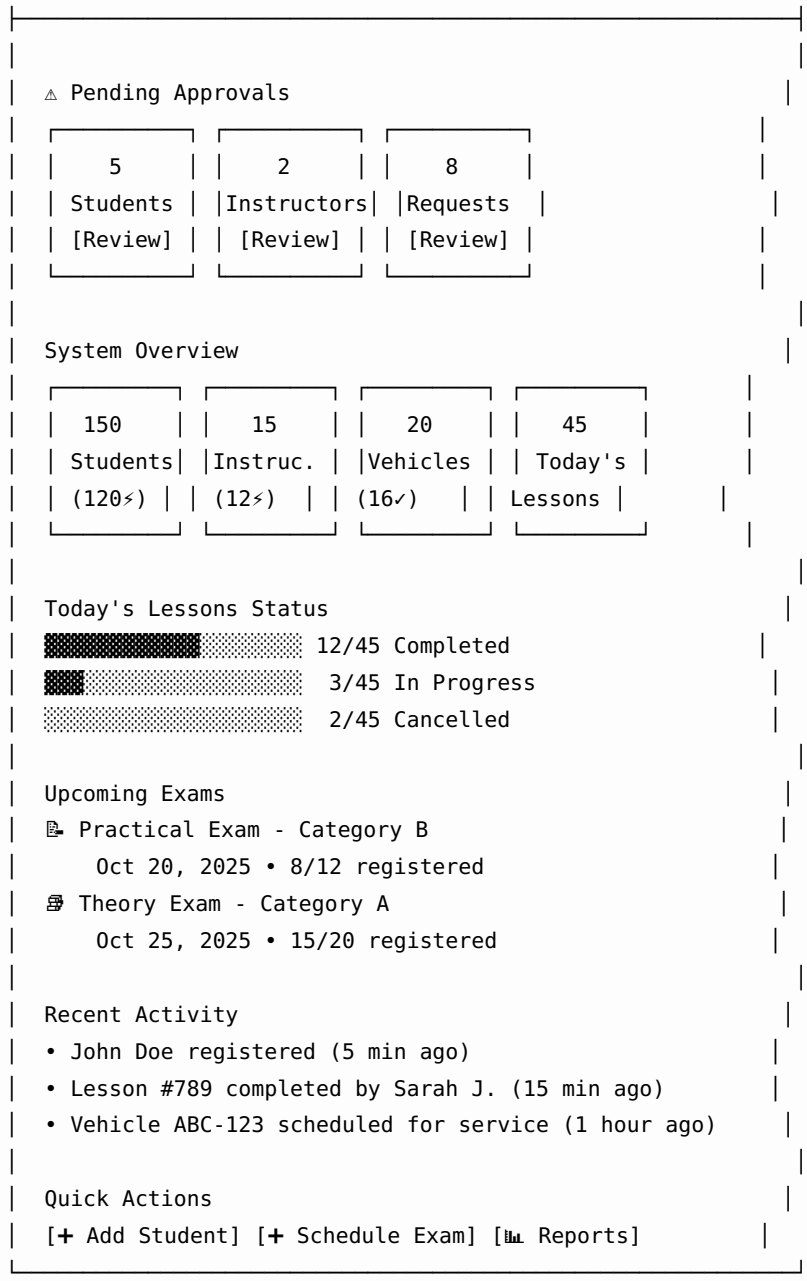


Instructor Dashboard



Admin Dashboard

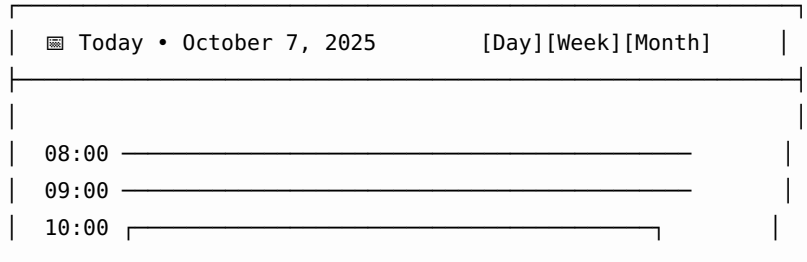


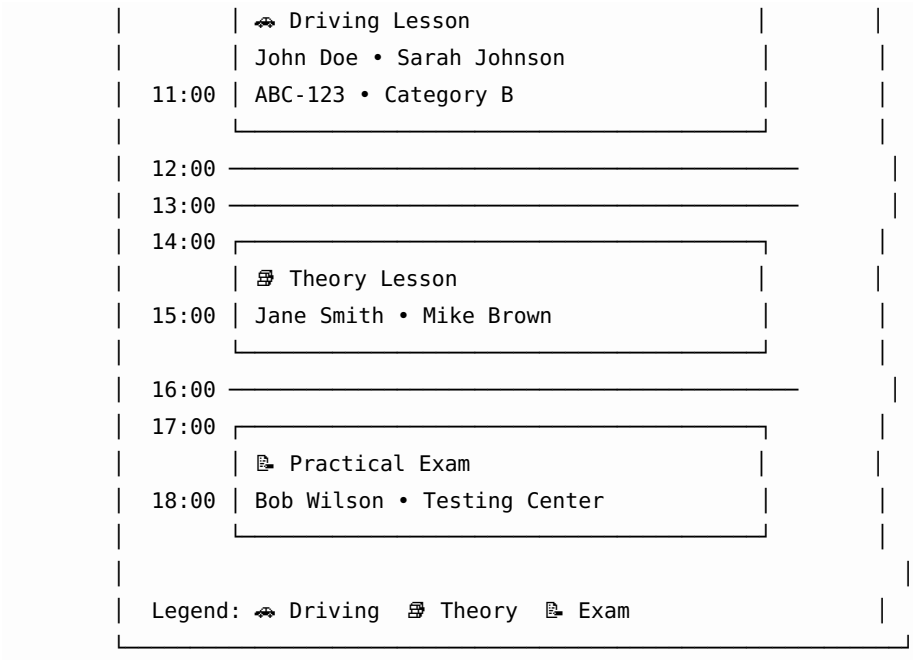


---

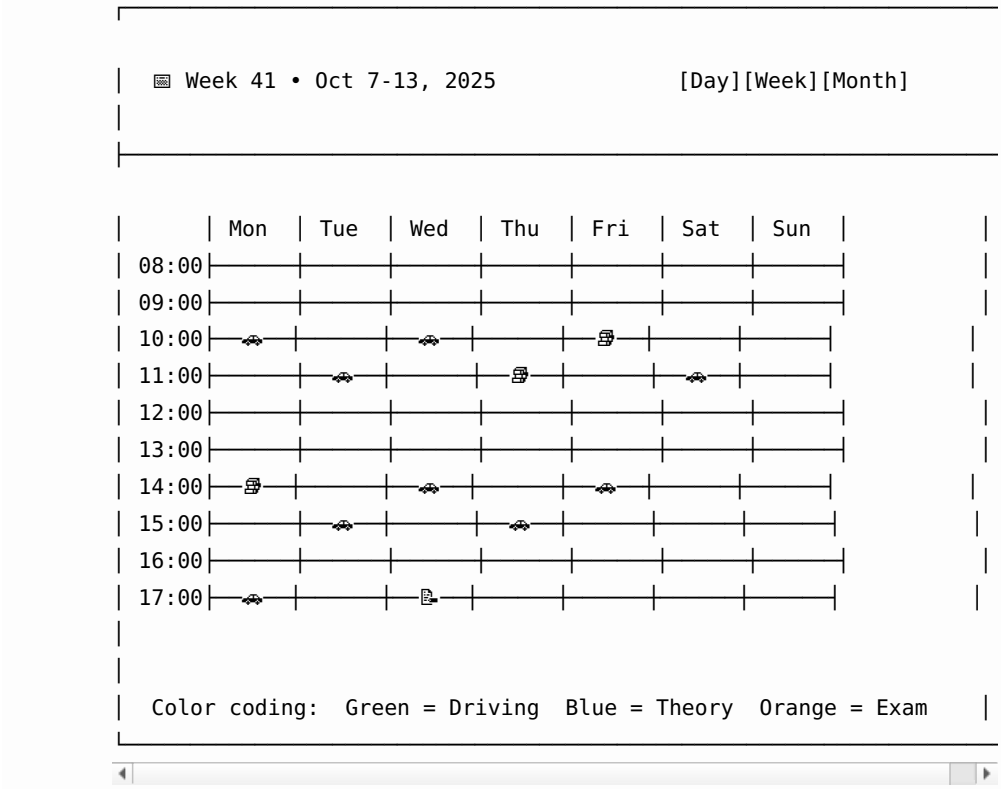
3. Calendar Views

Daily View

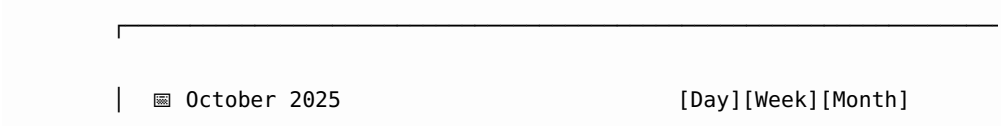


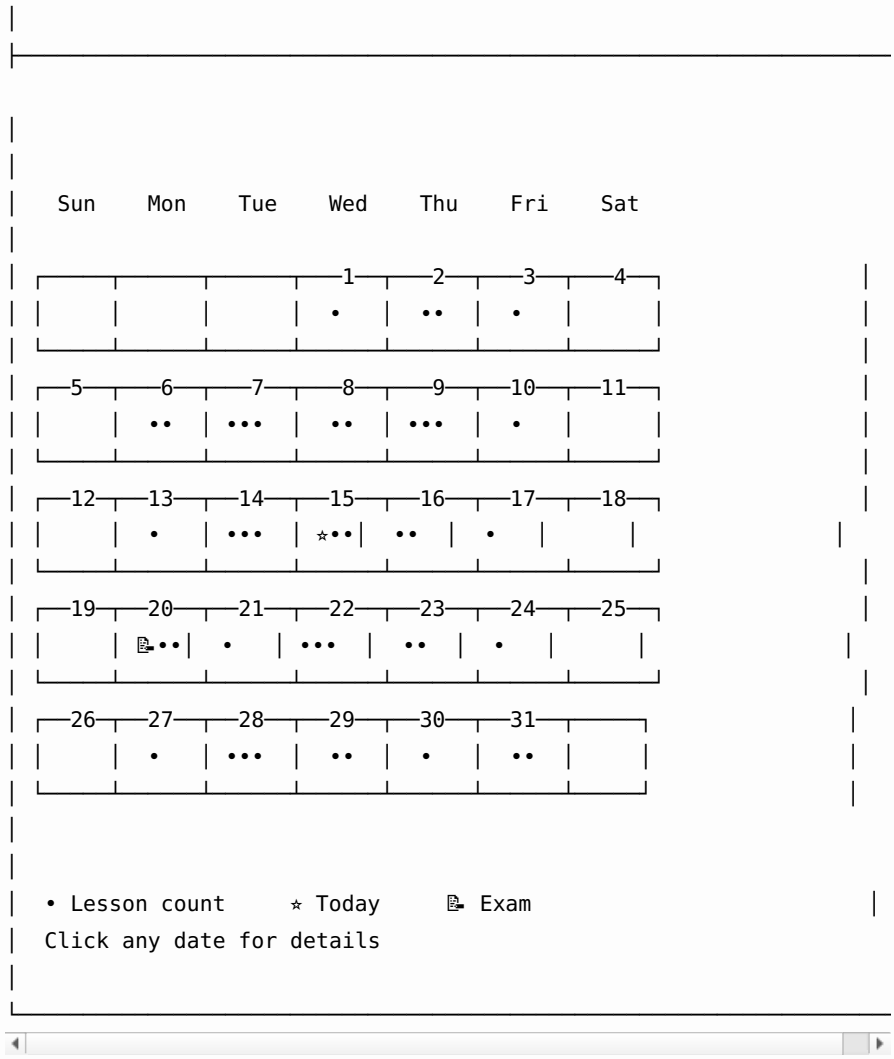


Weekly View



Monthly View

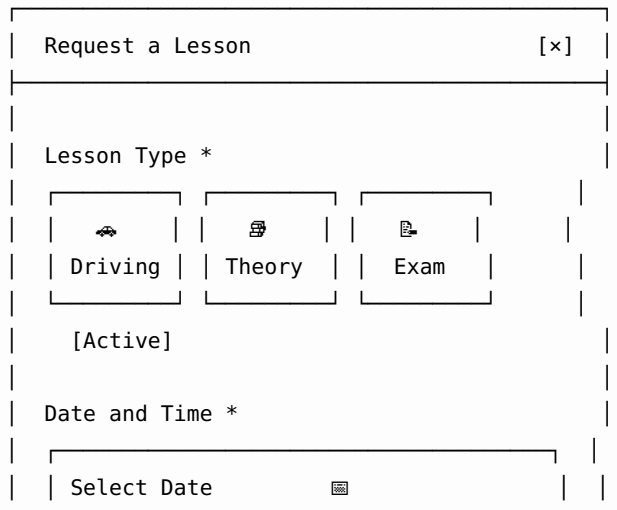


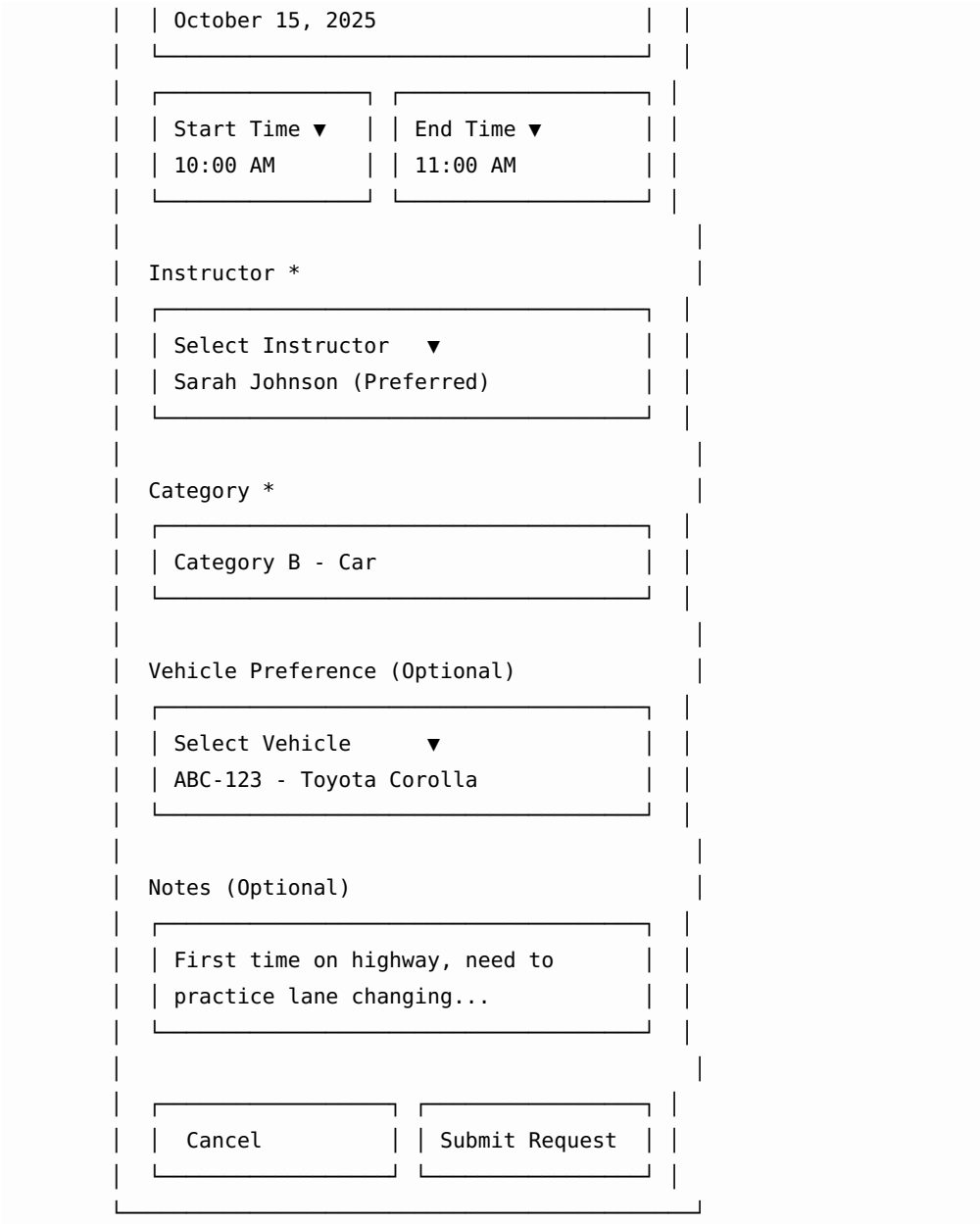


---

4. Lesson Creation Modal

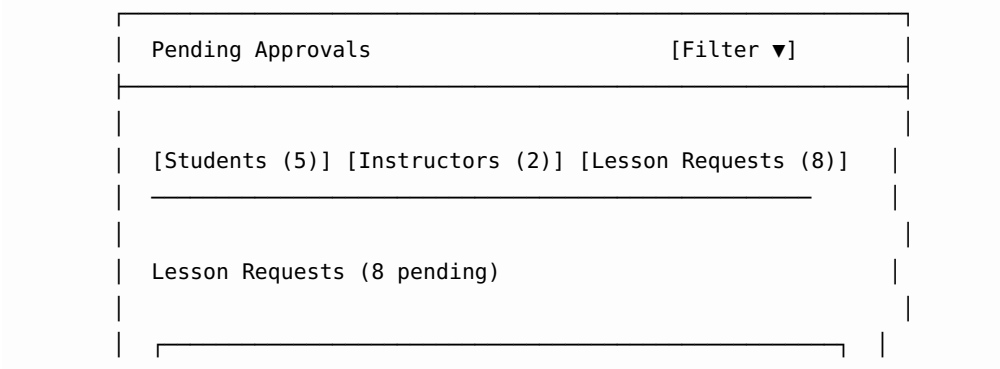
Create Lesson Request (Student)

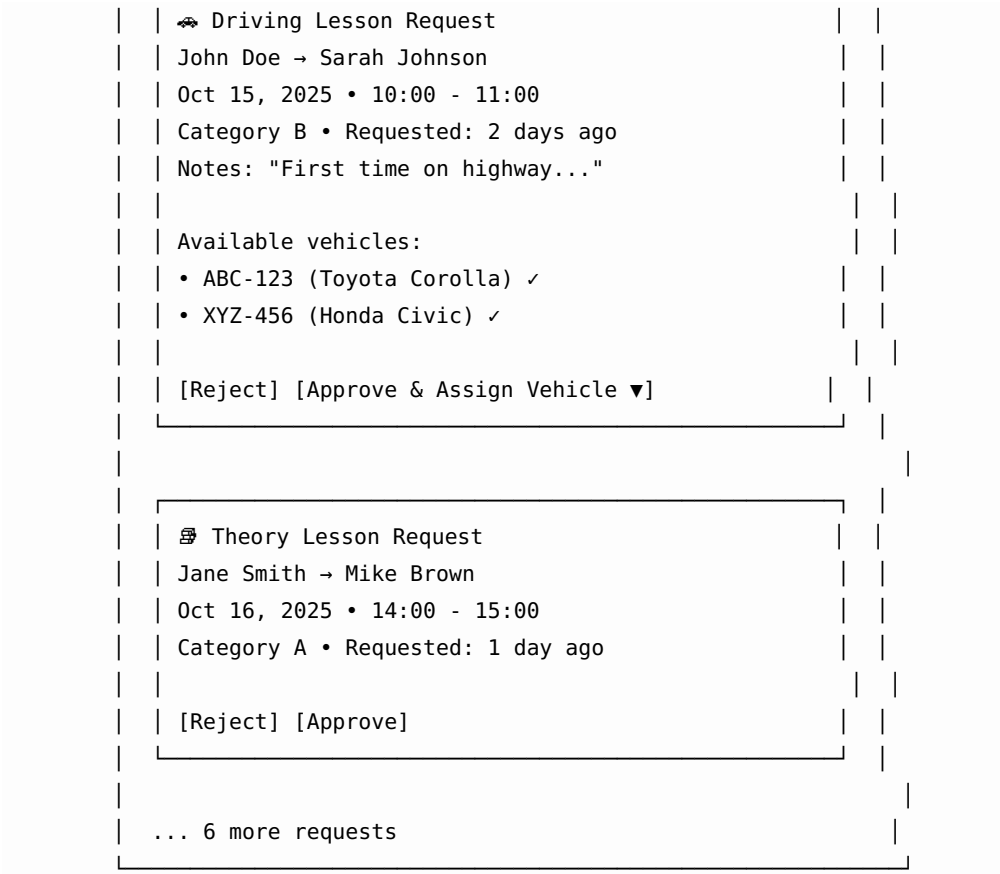




---

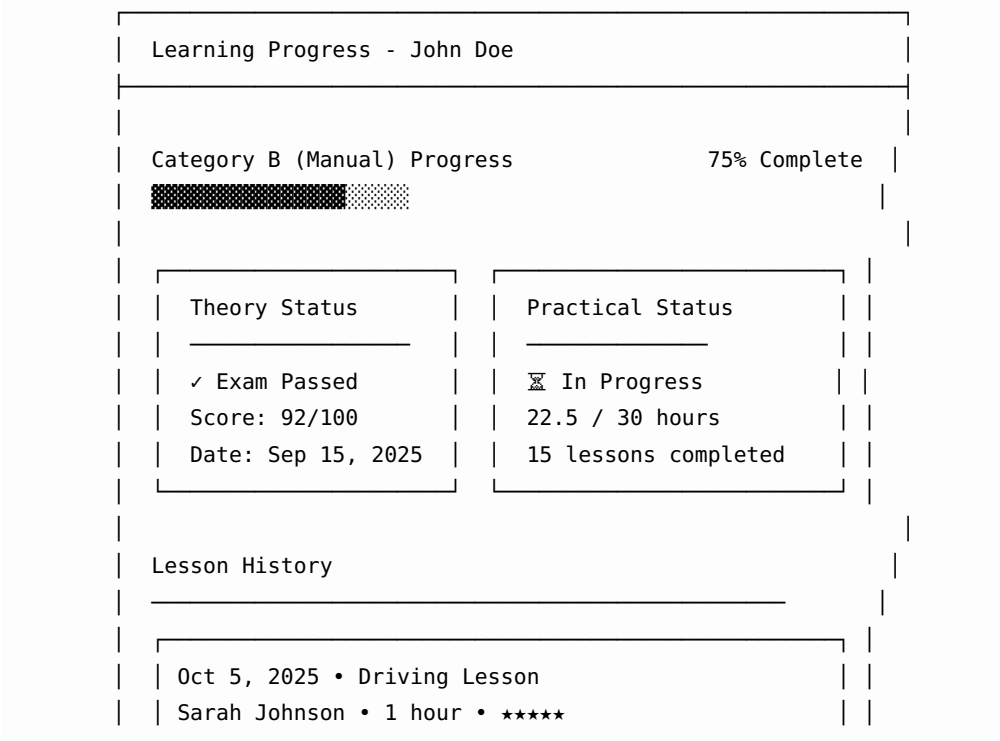
5. Pending Approvals View (Admin)

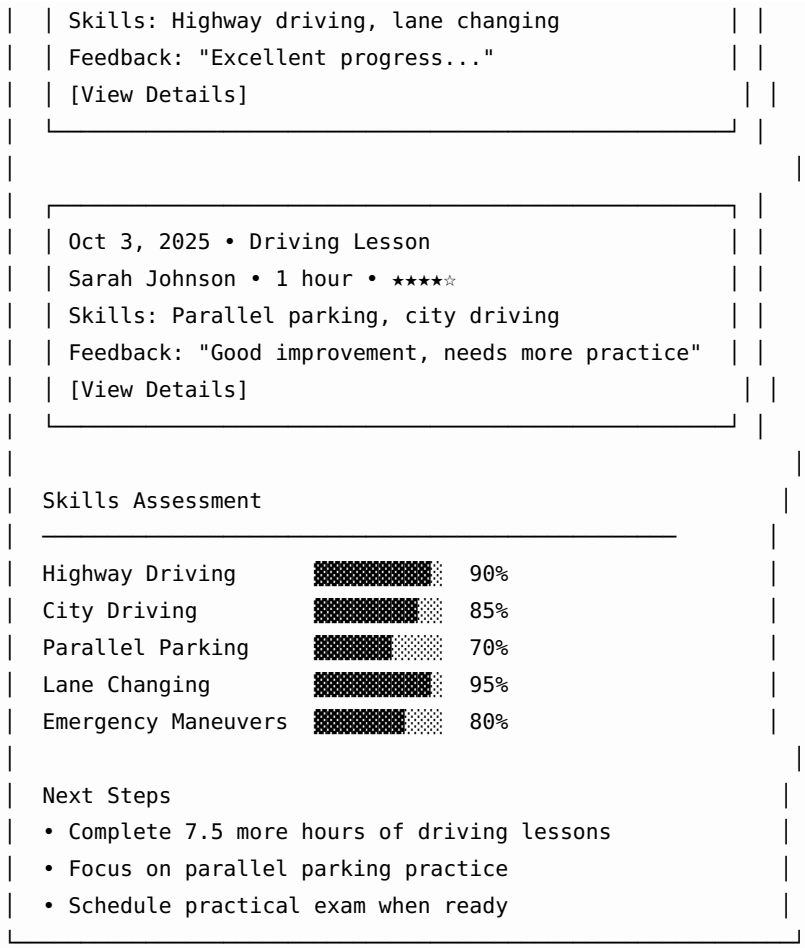




---

6. Student Progress View





Component Library

Buttons

```css
/* Primary Button */
.btn-primary {
  background: var(--color-primary);
  color: white;
  padding: var(--space-3) var(--space-6);
  border-radius: 8px;
  font-weight: var(--font-medium);
  transition: all 0.2s;
}

.btn-primary:hover {
  background: #1E40AF; /* Darker blue */
  transform: translateY(-1px);
  box-shadow: 0 4px 12px rgba(37, 99, 235, 0.3);
}
```

```css
/* Secondary Button */
.btn-secondary {
  background: var(--color-surface);
  color: var(--color-text-primary);
  border: 1px solid var(--color-border);
  padding: var(--space-3) var(--space-6);
  border-radius: 8px;
}

/* Danger Button */
.btn-danger {
  background: var(--color-error);
  color: white;
}
```

Cards

```css
.card {
  background: var(--color-surface);
  border-radius: 12px;
  padding: var(--space-6);
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
  border: 1px solid var(--color-border);
}

.card:hover {
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
  transition: box-shadow 0.3s;
}

.card-header {
  font-size: var(--text-xl);
  font-weight: var(--font-semibold);
  margin-bottom: var(--space-4);
}
```

Badges

```css
.badge {
  display: inline-block;
  padding: var(--space-1) var(--space-3);
  border-radius: 16px;
  font-size: var(--text-sm);
  font-weight: var(--font-medium);
}

.badge-driving {
  background: rgba(16, 185, 129, 0.1);
```

```
  color: var(--color-driving);
}

.badge-theory {
  background: rgba(59, 130, 246, 0.1);
  color: var(--color-theory);
}

.badge-exam {
  background: rgba(245, 158, 11, 0.1);
  color: var(--color-exam);
}
```

Form Inputs

```
.input {
  width: 100%;
  padding: var(--space-3) var(--space-4);
  border: 1px solid var(--color-border);
  border-radius: 8px;
  font-size: var(--text-base);
  transition: border-color 0.2s;
}

.input:focus {
  outline: none;
  border-color: var(--color-primary);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1);
}

.input-error {
  border-color: var(--color-error);
}

.input-error:focus {
  box-shadow: 0 0 0 3px rgba(239, 68, 68, 0.1);
}
```

Responsive Breakpoints

```
/* Mobile */
@media (max-width: 767px) {
  /* Stack layouts vertically */
  /* Full-width buttons */
  /* Simplified navigation */
}
```

```
/* Tablet */
@media (min-width: 768px) and (max-width: 1023px) {
  /* Two-column layouts */
  /* Collapsible sidebar */
}

/* Desktop */
@media (min-width: 1024px) {
  /* Multi-column layouts */
  /* Fixed sidebar */
  /* Expanded views */
}
```

Accessibility Guidelines

1. **Keyboard Navigation**
   - All interactive elements accessible via Tab key
   - Visible focus indicators
   - Skip links for main content
2. **Screen Readers**
   - Semantic HTML (nav, main, section, article)
   - ARIA labels for icons
   - Alt text for images
   - Form labels properly associated
3. **Color Contrast**
   - Minimum 4.5:1 for normal text
   - Minimum 3:1 for large text
   - Don't rely solely on color for information
4. **Touch Targets**
   - Minimum 44x44 pixels for mobile
   - Adequate spacing between interactive elements

Migration Strategy

Overview

Migration from an existing database (SQLite, MySQL, or proprietary) to PostgreSQL requires careful planning to ensure zero data loss and minimal downtime.

Pre-Migration Checklist

1. Assessment Phase

- [ ] Document current database schema
- [ ] Identify all data types and their PostgreSQL equivalents
- [ ] List all stored procedures, triggers, and functions
- [ ] Document all foreign key relationships
- [ ] Identify indexes and constraints
- [ ] Measure current database size
- [ ] List all database users and their permissions

2. Environment Setup

- [ ] Install PostgreSQL 14+ on target server
- [ ] Configure PostgreSQL settings (memory, connections, etc.)
- [ ] Set up database backups
- [ ] Create development/staging environments
- [ ] Install migration tools (pgloader, AWS DMS, or custom scripts)

3. Testing Plan

- [ ] Create test data set
- [ ] Define acceptance criteria
- [ ] Plan rollback procedures
- [ ] Identify critical queries to test
- [ ] Establish performance benchmarks

Migration Process

Phase 1: Schema Migration

Step 1: Export Current Schema

```
# For MySQL
mysqldump -u username -p --no-data dbname > schema.sql

# For SQLite
sqlite3 database.db .schema > schema.sql
```

Step 2: Convert to PostgreSQL

Create PostgreSQL schema from the database design document (Section 3).

```
# Run schema creation script
psql -U postgres -d driving_school_db -f schema.sql
```

Step 3: Create Triggers and Functions

```
# Run triggers and functions script
psql -U postgres -d driving_school_db -f triggers.sql
```

Phase 2: Data Migration

Option 1: Using pgloader (Recommended for MySQL/SQLite)

```
# Install pgloader
apt-get install pgloader

# Create migration command file
cat > migration.load << EOF
LOAD DATABASE
  FROM mysql://user:pass@localhost/old_db
  INTO postgresql://user:pass@localhost/driving_school_db

WITH include drop, create tables, create indexes, reset sequences

SET work_mem to '256MB',
    maintenance_work_mem to '512MB';

BEFORE LOAD DO
  $$ DROP SCHEMA IF EXISTS public CASCADE; $$,
  $$ CREATE SCHEMA public; $$;
EOF

# Run migration
pgloader migration.load
```

Option 2: Custom Scripts

```
# migration_script.py
import psycopg2
import mysql.connector  # or sqlite3

def migrate_users():
    # Connect to source database
    source_conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="password",
        database="old_db"
    )

    # Connect to PostgreSQL
    target_conn = psycopg2.connect(
        host="localhost",
        database="driving_school_db",
        user="postgres",
        password="password"
    )
```

```
    source_cursor = source_conn.cursor(dictionary=True)
    target_cursor = target_conn.cursor()

    # Fetch data from source
    source_cursor.execute("SELECT * FROM users")
    users = source_cursor.fetchall()

    # Insert into target
    for user in users:
        target_cursor.execute("""
            INSERT INTO users (
                email, password_hash, role, first_name, last_name,
                phone_number, date_of_birth, address, city,
        postal_code,
                is_email_verified, is_active, is_approved,
        created_at
            ) VALUES (
                %(email)s, %(password_hash)s, %(role)s, %
        (first_name)s,
                %(last_name)s, %(phone_number)s, %(date_of_birth)s,
                %(address)s, %(city)s, %(postal_code)s,
                %(is_email_verified)s, %(is_active)s, %
        (is_approved)s,
                %(created_at)s
            )
        """, user)

    target_conn.commit()
    source_cursor.close()
    target_cursor.close()
    source_conn.close()
    target_conn.close()

    print(f"Migrated {len(users)} users")

# Run migration for all tables
if __name__ == "__main__":
    migrate_users()
    migrate_students()
    migrate_instructors()
    migrate_lessons()
    migrate_exams()
    # ... etc
```

Phase 3: Data Validation

```
-- Compare record counts
SELECT 'users' as table_name, COUNT(*) FROM users
UNION ALL
```

```sql
SELECT 'students', COUNT(*) FROM students
UNION ALL
SELECT 'instructors', COUNT(*) FROM instructors
UNION ALL
SELECT 'lessons', COUNT(*) FROM lessons
UNION ALL
SELECT 'exams', COUNT(*) FROM exams;

-- Check for NULL values in required fields
SELECT 'users' as table_name,
       COUNT(*) as null_emails
FROM users
WHERE email IS NULL;

-- Verify foreign key integrity
SELECT COUNT(*) as orphaned_students
FROM students s
LEFT JOIN users u ON s.user_id = u.id
WHERE u.id IS NULL;

-- Check date ranges
SELECT
  MIN(lesson_date) as earliest_lesson,
  MAX(lesson_date) as latest_lesson
FROM lessons;
```

Phase 4: Performance Optimization

```sql
-- Analyze tables for query optimization
ANALYZE users;
ANALYZE students;
ANALYZE instructors;
ANALYZE lessons;
ANALYZE exams;

-- Create additional indexes if needed
CREATE INDEX CONCURRENTLY idx_lessons_composite
ON lessons(instructor_id, lesson_date, status)
WHERE status IN ('scheduled', 'in_progress');

-- Update statistics
VACUUM ANALYZE;
```

Phase 5: Application Migration

Step 1: Update Connection Strings

```
// Old (MySQL/SQLite)
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'old_db'
});

// New (PostgreSQL)
const { Pool } = require('pg');
const pool = new Pool({
  host: 'localhost',
  user: 'postgres',
  password: 'password',
  database: 'driving_school_db',
  port: 5432
});
```

Step 2: Update Queries

```
// MySQL uses ? placeholders
const query = "SELECT * FROM users WHERE id = ?";
connection.query(query, [userId]);

// PostgreSQL uses $1, $2, etc.
const query = "SELECT * FROM users WHERE id = $1";
pool.query(query, [userId]);
```

Step 3: Handle Date/Time Differences

```
// MySQL/SQLite date formatting
const date = new Date().toISOString().slice(0, 19).replace('T', '
        ');

// PostgreSQL (ISO 8601 format)
const date = new Date().toISOString();
```

Step 4: Update Boolean Handling

```
// MySQL uses 0/1 for boolean
const isActive = 1;

// PostgreSQL uses true/false
const isActive = true;
```

Migration Timeline

Week 1: Preparation

- Day 1-2: Schema mapping and conversion
- Day 3-4: Set up PostgreSQL environment
- Day 5-7: Create migration scripts and test

Week 2: Testing

- Day 1-3: Migrate test data
- Day 4-5: Validate data integrity
- Day 6-7: Performance testing

Week 3: Staging Migration

- Day 1-2: Migrate staging environment
- Day 3-5: Application testing in staging
- Day 6-7: Fix issues, final preparations

Week 4: Production Migration

- Day 1: Pre-migration backup
- Day 2: Execute migration (off-peak hours)
- Day 3: Validation and monitoring
- Day 4-7: Post-migration support

Rollback Plan

```
-- 1. Keep old database running in read-only mode
ALTER DATABASE old_db SET default_transaction_read_only = on;

-- 2. If rollback needed, switch application back
-- Update application config to use old database

-- 3. Document issues for retry
-- Create incident report with specific failures

-- 4. Fix issues in PostgreSQL while application uses old DB
-- Implement fixes

-- 5. Retry migration
-- Follow same process with improvements
```

Post-Migration Tasks

- [ ] Monitor query performance
- [ ] Set up automated backups
- [ ] Configure replication (if needed)
- [ ] Update documentation
- [ ] Train team on PostgreSQL

- [ ] Decommission old database (after 30 days)
- [ ] Celebrate successful migration! 🎉

Flutter Integration Guidelines

Overview

The Flutter frontend will communicate with the backend via REST API. This section provides guidelines for integrating the Flutter app with the backend.

Project Structure

```
lib/
├── main.dart
├── config/
│   ├── api_config.dart
│   ├── routes.dart
│   └── theme.dart
├── models/
│   ├── user.dart
│   ├── student.dart
│   ├── instructor.dart
│   ├── lesson.dart
│   ├── exam.dart
│   └── vehicle.dart
├── services/
│   ├── api_service.dart
│   ├── auth_service.dart
│   ├── student_service.dart
│   ├── lesson_service.dart
│   └── storage_service.dart
├── providers/
│   ├── auth_provider.dart
│   ├── student_provider.dart
│   └── lesson_provider.dart
├── screens/
│   ├── auth/
│   │   ├── login_screen.dart
│   │   ├── register_screen.dart
│   │   └── verify_email_screen.dart
│   ├── student/
│   │   ├── dashboard_screen.dart
│   │   ├── lessons_screen.dart
│   │   └── progress_screen.dart
```

```
│   ├── instructor/
│   │   ├── dashboard_screen.dart
│   │   └── schedule_screen.dart
│   └── admin/
│       └── dashboard_screen.dart
├── widgets/
│   ├── common/
│   │   ├── custom_button.dart
│   │   ├── custom_card.dart
│   │   └── loading_indicator.dart
│   ├── lesson/
│   │   ├── lesson_card.dart
│   │   └── calendar_view.dart
│   └── forms/
│       └── lesson_request_form.dart
└── utils/
    ├── constants.dart
    ├── validators.dart
    └── date_utils.dart
```

Dependencies

Add to `pubspec.yaml`:

```yaml
dependencies:
  flutter:
    sdk: flutter

  # State Management
  provider: ^6.0.5

  # HTTP Client
  http: ^1.1.0
  dio: ^5.3.3

  # Local Storage
  shared_preferences: ^2.2.2
  flutter_secure_storage: ^9.0.0

  # UI Components
  flutter_spinkit: ^5.2.0
  fluttertoast: ^8.2.4
  intl: ^0.18.1

  # Calendar
  table_calendar: ^3.0.9
```

```yaml
  # Forms
  flutter_form_builder: ^9.1.1

  # Notifications
  flutter_local_notifications: ^16.1.0

  # Deep Linking
  uni_links: ^0.5.1

  # Image Handling
  image_picker: ^1.0.4
  cached_network_image: ^3.3.0
```

API Configuration

lib/config/api_config.dart

```
class ApiConfig {
  // Base URLs
  static const String baseUrl = String.fromEnvironment(
    'API_BASE_URL',
    defaultValue: 'http://localhost:3000/api/v1',
  );

  static const String prodUrl =
'https://api.drivingschool.com/api/v1';
  static const String devUrl = 'http://localhost:3000/api/v1';

  // Get current base URL based on environment
  static String get apiUrl {
    const environment = String.fromEnvironment('ENV', defaultValue:
'dev');
    return environment == 'prod' ? prodUrl : devUrl;
  }

  // API Endpoints
  static const String login = '/auth/login';
  static const String register = '/auth/register';
  static const String refreshToken = '/auth/refresh';
  static const String verifyEmail = '/auth/verify-email';

  static const String students = '/students';
  static const String instructors = '/instructors';
  static const String lessons = '/lessons';
  static const String lessonRequests = '/lesson-requests';
  static const String exams = '/exams';
  static const String vehicles = '/vehicles';
```

```
  // Timeouts
  static const Duration connectTimeout = Duration(seconds: 30);
  static const Duration receiveTimeout = Duration(seconds: 30);
}
```

Models

lib/models/user.dart

```
class User {
  final int id;
  final String email;
  final String role;
  final String firstName;
  final String lastName;
  final String? phoneNumber;
  final bool isEmailVerified;
  final bool isApproved;
  final DateTime? createdAt;

  User({
    required this.id,
    required this.email,
    required this.role,
    required this.firstName,
    required this.lastName,
    this.phoneNumber,
    required this.isEmailVerified,
    required this.isApproved,
    this.createdAt,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'],
      email: json['email'],
      role: json['role'],
      firstName: json['firstName'],
      lastName: json['lastName'],
      phoneNumber: json['phoneNumber'],
      isEmailVerified: json['isEmailVerified'],
      isApproved: json['isApproved'],
      createdAt: json['createdAt'] != null
        ? DateTime.parse(json['createdAt'])
        : null,
    );
```

```
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'email': email,
      'role': role,
      'firstName': firstName,
      'lastName': lastName,
      'phoneNumber': phoneNumber,
      'isEmailVerified': isEmailVerified,
      'isApproved': isApproved,
      'createdAt': createdAt?.toIso8601String(),
    };
  }

  String get fullName => '$firstName $lastName';
}
```

lib/models/lesson.dart

```
class Lesson {
  final int id;
  final DateTime lessonDate;
  final String startTime;
  final String endTime;
  final String lessonType;
  final String status;
  final int studentId;
  final String studentName;
  final int instructorId;
  final String instructorName;
  final int? vehicleId;
  final String? vehicleRegistration;
  final String? category;

  Lesson({
    required this.id,
    required this.lessonDate,
    required this.startTime,
    required this.endTime,
    required this.lessonType,
    required this.status,
    required this.studentId,
    required this.studentName,
    required this.instructorId,
    required this.instructorName,
    this.vehicleId,
```

```
      this.vehicleRegistration,
      this.category,
    });

    factory Lesson.fromJson(Map<String, dynamic> json) {
      return Lesson(
        id: json['id'],
        lessonDate: DateTime.parse(json['lessonDate']),
        startTime: json['startTime'],
        endTime: json['endTime'],
        lessonType: json['lessonType'],
        status: json['status'],
        studentId: json['student']['id'],
        studentName: json['student']['name'],
        instructorId: json['instructor']['id'],
        instructorName: json['instructor']['name'],
        vehicleId: json['vehicle']?['id'],
        vehicleRegistration: json['vehicle']?['registration'],
        category: json['category'],
      );
    }

    Color get typeColor {
      switch (lessonType) {
        case 'driving':
          return Color(0xFF10B981); // Green
        case 'theory':
          return Color(0xFF3B82F6); // Blue
        case 'exam':
          return Color(0xFFF59E0B); // Orange
        default:
          return Colors.grey;
      }
    }

    IconData get typeIcon {
      switch (lessonType) {
        case 'driving':
          return Icons.directions_car;
        case 'theory':
          return Icons.menu_book;
        case 'exam':
          return Icons.assignment;
        default:
          return Icons.event;
      }
    }
  }
```

Services

lib/services/api_service.dart

```
import 'package:dio/dio.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import '../config/api_config.dart';

class ApiService {
  late Dio _dio;
  final FlutterSecureStorage _storage = FlutterSecureStorage();

  ApiService() {
    _dio = Dio(BaseOptions(
      baseUrl: ApiConfig.apiUrl,
      connectTimeout: ApiConfig.connectTimeout,
      receiveTimeout: ApiConfig.receiveTimeout,
      headers: {
        'Content-Type': 'application/json',
      },
    ));

    // Add interceptors
    _dio.interceptors.add(InterceptorsWrapper(
      onRequest: (options, handler) async {
        // Add auth token to headers
        final token = await _storage.read(key: 'access_token');
        if (token != null) {
          options.headers['Authorization'] = 'Bearer $token';
        }
        return handler.next(options);
      },
      onError: (error, handler) async {
        // Handle 401 errors (token expired)
        if (error.response?.statusCode == 401) {
          // Try to refresh token
          if (await _refreshToken()) {
            // Retry the request
            return handler.resolve(await
_retry(error.requestOptions));
          }
        }
        return handler.next(error);
      },
    ));
  }
```

```
  Future<Response> get(String path, {Map<String, dynamic>?
queryParams}) async {
    try {
      return await _dio.get(path, queryParameters: queryParams);
    } catch (e) {
      throw _handleError(e);
    }
  }

  Future<Response> post(String path, {dynamic data}) async {
    try {
      return await _dio.post(path, data: data);
    } catch (e) {
      throw _handleError(e);
    }
  }

  Future<Response> patch(String path, {dynamic data}) async {
    try {
      return await _dio.patch(path, data: data);
    } catch (e) {
      throw _handleError(e);
    }
  }

  Future<Response> delete(String path) async {
    try {
      return await _dio.delete(path);
    } catch (e) {
      throw _handleError(e);
    }
  }

  Future<bool> _refreshToken() async {
    try {
      final refreshToken = await _storage.read(key:
'refresh_token');
      if (refreshToken == null) return false;

      final response = await _dio.post(
        ApiConfig.refreshToken,
        data: {'refreshToken': refreshToken},
      );

      if (response.statusCode == 200) {
        final newAccessToken = response.data['data']['accessToken'];
        await _storage.write(key: 'access_token', value:
newAccessToken);
```

```
        return true;
      }
      return false;
    } catch (e) {
      return false;
    }
  }

  Future<Response> _retry(RequestOptions requestOptions) async {
    final options = Options(
      method: requestOptions.method,
      headers: requestOptions.headers,
    );
    return _dio.request(
      requestOptions.path,
      data: requestOptions.data,
      queryParameters: requestOptions.queryParameters,
      options: options,
    );
  }

  Exception _handleError(dynamic error) {
    if (error is DioException) {
      switch (error.type) {
        case DioExceptionType.connectionTimeout:
        case DioExceptionType.receiveTimeout:
          return Exception('Connection timeout. Please try again.');
        case DioExceptionType.badResponse:
          final message = error.response?.data['error']?['message']
            ?? 'An error occurred';
          return Exception(message);
        default:
          return Exception('Network error. Please check your
connection.');
      }
    }
    return Exception('An unexpected error occurred');
  }
}
```

lib/services/auth_service.dart

```
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import '../config/api_config.dart';
import '../models/user.dart';
import 'api_service.dart';

class AuthService {
```

```
  final ApiService _apiService = ApiService();
  final FlutterSecureStorage _storage = FlutterSecureStorage();

  Future<User> login(String email, String password) async {
    final response = await _apiService.post(
      ApiConfig.login,
      data: {
        'email': email,
        'password': password,
      },
    );

    final data = response.data['data'];

    // Store tokens
    await _storage.write(
      key: 'access_token',
      value: data['tokens']['accessToken'],
    );
    await _storage.write(
      key: 'refresh_token',
      value: data['tokens']['refreshToken'],
    );

    return User.fromJson(data['user']);
  }

  Future<void> register(Map<String, dynamic> registrationData) async
{
    await _apiService.post(
      ApiConfig.register,
      data: registrationData,
    );
  }

  Future<void> logout() async {
    await _apiService.post(ApiConfig.logout);
    await _storage.deleteAll();
  }

  Future<bool> isLoggedIn() async {
    final token = await _storage.read(key: 'access_token');
    return token != null;
  }

  Future<String?> getAccessToken() async {
    return await _storage.read(key: 'access_token');
  }
```

```
}
```

lib/services/lesson_service.dart

```
import '../config/api_config.dart';
import '../models/lesson.dart';
import 'api_service.dart';

class LessonService {
  final ApiService _apiService = ApiService();

  Future<List<Lesson>> getLessons({
    int? studentId,
    int? instructorId,
    String? startDate,
    String? endDate,
    String? status,
  }) async {
    final queryParams = <String, dynamic>{};

    if (studentId != null) queryParams['studentId'] = studentId;
    if (instructorId != null) queryParams['instructorId'] =
instructorId;
    if (startDate != null) queryParams['startDate'] = startDate;
    if (endDate != null) queryParams['endDate'] = endDate;
    if (status != null) queryParams['status'] = status;

    final response = await _apiService.get(
      ApiConfig.lessons,
      queryParams: queryParams,
    );

    final List lessonsJson = response.data['data'];
    return lessonsJson.map((json) =>
Lesson.fromJson(json)).toList();
  }

  Future<Lesson> getLessonById(int id) async {
    final response = await
_apiService.get('${ApiConfig.lessons}/$id');
    return Lesson.fromJson(response.data['data']);
  }

  Future<void> createLessonRequest(Map<String, dynamic> requestData)
async {
    await _apiService.post(
      ApiConfig.lessonRequests,
      data: requestData,
```

```
    );
  }

  Future<void> cancelLesson(int lessonId, String reason) async {
    await _apiService.post(
      '${ApiConfig.lessons}/$lessonId/cancel',
      data: {'cancellationReason': reason},
    );
  }

  Future<void> rateLesson(
    int lessonId,
    int rating,
    String feedback,
  ) async {
    await _apiService.post(
      '${ApiConfig.lessons}/$lessonId/rate',
      data: {
        'instructorRating': rating,
        'studentFeedback': feedback,
      },
    );
  }
}
```

State Management with Provider

lib/providers/auth_provider.dart

```
import 'package:flutter/foundation.dart';
import '../models/user.dart';
import '../services/auth_service.dart';

class AuthProvider with ChangeNotifier {
  final AuthService _authService = AuthService();

  User? _user;
  bool _isLoading = false;
  String? _error;

  User? get user => _user;
  bool get isLoading => _isLoading;
  String? get error => _error;
  bool get isLoggedIn => _user != null;

  Future<void> login(String email, String password) async {
    _isLoading = true;
```

```
    _error = null;
    notifyListeners();

    try {
      _user = await _authService.login(email, password);
      _isLoading = false;
      notifyListeners();
    } catch (e) {
      _error = e.toString();
      _isLoading = false;
      notifyListeners();
      rethrow;
    }
  }

  Future<void> register(Map<String, dynamic> data) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      await _authService.register(data);
      _isLoading = false;
      notifyListeners();
    } catch (e) {
      _error = e.toString();
      _isLoading = false;
      notifyListeners();
      rethrow;
    }
  }

  Future<void> logout() async {
    await _authService.logout();
    _user = null;
    notifyListeners();
  }

  Future<void> checkAuthStatus() async {
    final isLoggedIn = await _authService.isLoggedIn();
    if (!isLoggedIn) {
      _user = null;
      notifyListeners();
    }
  }
}
```

Example Screens

lib/screens/auth/login_screen.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../providers/auth_provider.dart';

class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _obscurePassword = true;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: SingleChildScrollView(
          padding: EdgeInsets.all(24.0),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                SizedBox(height: 40),

                // Logo
                Icon(
                  Icons.drive_eta,
                  size: 80,
                  color: Theme.of(context).primaryColor,
                ),

                SizedBox(height: 16),

                Text(
                  'Welcome Back',
                  style: Theme.of(context).textTheme.headlineMedium,
                  textAlign: TextAlign.center,
                ),

                SizedBox(height: 8),
```

```
Text(
  'Sign in to continue',
  style: Theme.of(context).textTheme.bodyMedium,
  textAlign: TextAlign.center,
),

SizedBox(height: 40),

// Email field
TextFormField(
  controller: _emailController,
  keyboardType: TextInputType.emailAddress,
  decoration: InputDecoration(
    labelText: 'Email',
    prefixIcon: Icon(Icons.email),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12),
    ),
  ),
  validator: (value) {
    if (value?.isEmpty ?? true) {
      return 'Please enter your email';
    }
    if (!RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$')
        .hasMatch(value!)) {
      return 'Please enter a valid email';
    }
    return null;
  },
),

SizedBox(height: 16),

// Password field
TextFormField(
  controller: _passwordController,
  obscureText: _obscurePassword,
  decoration: InputDecoration(
    labelText: 'Password',
    prefixIcon: Icon(Icons.lock),
    suffixIcon: IconButton(
      icon: Icon(
        _obscurePassword
          ? Icons.visibility
          : Icons.visibility_off,
      ),
      onPressed: () {
```

```
                        setState(() {
                          _obscurePassword = !_obscurePassword;
                        });
                      },
                    ),
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                  ),
                  validator: (value) {
                    if (value?.isEmpty ?? true) {
                      return 'Please enter your password';
                    }
                    return null;
                  },
                ),

                SizedBox(height: 24),

                // Login button
                Consumer<AuthProvider>(
                  builder: (context, authProvider, child) {
                    return ElevatedButton(
                      onPressed: authProvider.isLoading
                        ? null
                        : () => _handleLogin(context),
                      style: ElevatedButton.styleFrom(
                        padding: EdgeInsets.symmetric(vertical: 16),
                        shape: RoundedRectangleBorder(
                          borderRadius: BorderRadius.circular(12),
                        ),
                      ),
                      child: authProvider.isLoading
                        ? SizedBox(
                            height: 20,
                            width: 20,
                            child: CircularProgressIndicator(
                              strokeWidth: 2,
                              valueColor:
AlwaysStoppedAnimation<Color>(
                                Colors.white,
                              ),
                            ),
                          )
                        : Text(
                            'Sign In',
                            style: TextStyle(fontSize: 16),
                          ),
```

```
                    );
                  },
                ),

                SizedBox(height: 16),

                // Register link
                TextButton(
                  onPressed: () {
                    Navigator.pushNamed(context, '/register');
                  },
                  child: Text('Don\'t have an account? Register'),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Future<void> _handleLogin(BuildContext context) async {
    if (!_formKey.currentState!.validate()) return;

    final authProvider = Provider.of<AuthProvider>(context, listen:
false);

    try {
      await authProvider.login(
        _emailController.text,
        _passwordController.text,
      );

      // Navigate based on role
      final user = authProvider.user!;
      String route;

      switch (user.role) {
        case 'student':
          route = '/student/dashboard';
          break;
        case 'instructor':
          route = '/instructor/dashboard';
          break;
        case 'super_admin':
          route = '/admin/dashboard';
          break;
        default:
```

```
          route = '/';
      }

      Navigator.pushReplacementNamed(context, route);

    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(e.toString()),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
}
```

Best Practices

1. Error Handling

```
try {
  final lessons = await lessonService.getLessons();
  setState(() {
    _lessons = lessons;
  });
} on DioException catch (e) {
  if (e.response?.statusCode == 401) {
    // Handle unauthorized
    Navigator.pushReplacementNamed(context, '/login');
  } else {
    // Show error message
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Failed to load lessons')),
    );
  }
} catch (e) {
  // Handle general errors
  print('Unexpected error: $e');
}
```

2. Loading States

```
class LessonsScreen extends StatefulWidget {
  @override
  _LessonsScreenState createState() => _LessonsScreenState();
}

class _LessonsScreenState extends State<LessonsScreen> {
  bool _isLoading = true;
  List<Lesson> _lessons = [];

  @override
  void initState() {
    super.initState();
    _loadLessons();
  }

  Future<void> _loadLessons() async {
    setState(() => _isLoading = true);
    try {
      final lessons = await lessonService.getLessons();
      setState(() {
        _lessons = lessons;
        _isLoading = false;
      });
    } catch (e) {
      setState(() => _isLoading = false);
      // Show error
    }
  }

  @override
  Widget build(BuildContext context) {
    if (_isLoading) {
      return Center(child: CircularProgressIndicator());
    }

    if (_lessons.isEmpty) {
      return Center(child: Text('No lessons found'));
    }

    return ListView.builder(
      itemCount: _lessons.length,
      itemBuilder: (context, index) {
        return LessonCard(lesson: _lessons[index]);
      },
    );
  }
```

```
}
```

3. Pagination

```
class LessonsScreen extends StatefulWidget {
  @override
  _LessonsScreenState createState() => _LessonsScreenState();
}

class _LessonsScreenState extends State<LessonsScreen> {
  final ScrollController _scrollController = ScrollController();
  List<Lesson> _lessons = [];
  int _currentPage = 1;
  bool _isLoading = false;
  bool _hasMore = true;

  @override
  void initState() {
    super.initState();
    _loadLessons();
    _scrollController.addListener(_onScroll);
  }

  void _onScroll() {
    if (_scrollController.position.pixels ==
        _scrollController.position.maxScrollExtent) {
      if (!_isLoading && _hasMore) {
        _loadMore();
      }
    }
  }

  Future<void> _loadLessons() async {
    setState(() => _isLoading = true);
    try {
      final lessons = await lessonService.getLessons(
        page: _currentPage,
        limit: 20,
      );
      setState(() {
        _lessons = lessons;
        _isLoading = false;
      });
    } catch (e) {
      setState(() => _isLoading = false);
    }
  }
```

```
Future<void> _loadMore() async {
  setState(() => _isLoading = true);
  _currentPage++;

  try {
    final lessons = await lessonService.getLessons(
      page: _currentPage,
      limit: 20,
    );

    setState(() {
      if (lessons.isEmpty) {
        _hasMore = false;
      } else {
        _lessons.addAll(lessons);
      }
      _isLoading = false;
    });
  } catch (e) {
    _currentPage--;
    setState(() => _isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return ListView.builder(
    controller: _scrollController,
    itemCount: _lessons.length + (_hasMore ? 1 : 0),
    itemBuilder: (context, index) {
      if (index == _lessons.length) {
        return Center(
          child: Padding(
            padding: EdgeInsets.all(16),
            child: CircularProgressIndicator(),
          ),
        );
      }
      return LessonCard(lesson: _lessons[index]);
    },
  );
}

@override
void dispose() {
  _scrollController.dispose();
  super.dispose();
}
```

```
}
```

Performance & Scalability Considerations

Database Optimization

1. Indexing Strategy

```sql
-- Composite indexes for common queries
CREATE INDEX CONCURRENTLY idx_lessons_instructor_date_status
ON lessons(instructor_id, lesson_date, status)
WHERE status IN ('scheduled', 'in_progress');

CREATE INDEX CONCURRENTLY idx_lessons_student_date
ON lessons(student_id, lesson_date DESC);

-- Partial indexes for frequently filtered data
CREATE INDEX CONCURRENTLY idx_active_users
ON users(role, is_active)
WHERE is_active = TRUE;

-- GIN indexes for array columns
CREATE INDEX CONCURRENTLY idx_instructors_categories
ON instructors USING GIN(qualified_category_ids);
```

2. Query Optimization

```sql
-- Use EXPLAIN ANALYZE to optimize queries
EXPLAIN ANALYZE
SELECT l.*, s.first_name, i.first_name
FROM lessons l
JOIN students s ON l.student_id = s.id
JOIN instructors i ON l.instructor_id = i.id
WHERE l.lesson_date BETWEEN '2025-10-01' AND '2025-10-31'
  AND l.status = 'scheduled';

-- Optimize with proper indexes and query restructuring
```

3. Connection Pooling

```javascript
// Node.js with pg-pool
const { Pool } = require('pg');

const pool = new Pool({
  host: 'localhost',
  database: 'driving_school_db',
```

```
  user: 'postgres',
  password: 'password',
  port: 5432,
  max: 20, // Maximum connections
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});
```

Caching Strategy

1. Application-Level Caching

```
const NodeCache = require('node-cache');
const cache = new NodeCache({ stdTTL: 600 }); // 10 minutes

async function getCategories() {
  const cacheKey = 'categories';

  // Check cache first
  let categories = cache.get(cacheKey);

  if (categories) {
    return categories;
  }

  // Fetch from database
  categories = await db.query('SELECT * FROM categories WHERE
        is_active = TRUE');

  // Store in cache
  cache.set(cacheKey, categories);

  return categories;
}
```

2. Redis for Distributed Caching

```
const redis = require('redis');
const client = redis.createClient({
  host: 'localhost',
  port: 6379,
});

// Cache user sessions
async function cacheUserSession(userId, sessionData) {
  await client.setEx(
    `session:${userId}`,
    3600, // 1 hour expiry
```

```
    JSON.stringify(sessionData)
  );
}

// Retrieve from cache
async function getUserSession(userId) {
  const data = await client.get(`session:${userId}`);
  return data ? JSON.parse(data) : null;
}
```

Horizontal Scaling

1. Load Balancing

```
# nginx.conf
upstream api_servers {
    least_conn;
    server api1.example.com:3000;
    server api2.example.com:3000;
    server api3.example.com:3000;
}

server {
    listen 80;
    server_name api.drivingschool.com;

    location / {
        proxy_pass http://api_servers;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

2. Database Replication

```
-- Primary-Replica setup
-- Primary server (write operations)
-- Replicas (read operations)

-- Application routing
const primaryPool = new Pool({ /* primary config */ });
const replicaPool = new Pool({ /* replica config */ });

// Write operations use primary
async function createLesson(data) {
  return primaryPool.query('INSERT INTO lessons...', data);
}
```

```
// Read operations use replica
async function getLessons() {
  return replicaPool.query('SELECT * FROM lessons...');
}
```

Monitoring & Logging

1. Application Monitoring

```
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level:
        'error' }),
    new winston.transports.File({ filename: 'combined.log' }),
  ],
});

// Log all API requests
app.use((req, res, next) => {
  logger.info({
    method: req.method,
    url: req.url,
    userId: req.user?.id,
    timestamp: new Date().toISOString(),
  });
  next();
});
```

2. Performance Metrics

```
const prometheus = require('prom-client');

// Create metrics
const httpRequestDuration = new prometheus.Histogram({
  name: 'http_request_duration_ms',
  help: 'Duration of HTTP requests in ms',
  labelNames: ['method', 'route', 'status_code'],
});

// Track request duration
app.use((req, res, next) => {
  const start = Date.now();
```

```
  res.on('finish', () => {
    const duration = Date.now() - start;
    httpRequestDuration
      .labels(req.method, req.route?.path, res.statusCode)
      .observe(duration);
  });

  next();
});
```

Security Best Practices

1. Input Validation

```
const { body, validationResult } = require('express-validator');

app.post('/lessons',
  [
    body('studentId').isInt(),
    body('instructorId').isInt(),
    body('lessonDate').isISO8601(),
    body('startTime').matches(/^([01]\d|2[0-3]):([0-5]\d)$/),
    body('lessonType').isIn(['driving', 'theory', 'exam']),
  ],
  (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    // Process request
  }
);
```

2. SQL Injection Prevention

```
// BAD - Vulnerable to SQL injection
const query = `SELECT * FROM users WHERE email = '${email}'`;

// GOOD - Use parameterized queries
const query = 'SELECT * FROM users WHERE email = $1';
const result = await pool.query(query, [email]);
```

3. Rate Limiting

```
const rateLimit = require('express-rate-limit');
```

```
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // Limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP',
});

app.use('/api/', limiter);
```

4. CORS Configuration

```
const cors = require('cors');

app.use(cors({
  origin: [
    'https://app.drivingschool.com',
    'http://localhost:3000', // Development only
  ],
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization'],
}));
```

Appendices

A. Error Codes

Code	Message	Description
AUTH_001	Invalid credentials	Email or password is incorrect
AUTH_002	Email not verified	User must verify email first
AUTH_003	Account not approved	Account pending admin approval
AUTH_004	Token expired	Access token has expired
VAL_001	Validation error	Input data validation failed
CONFLICT_001	Instructor not available	Instructor has conflicting lesson
CONFLICT_002	Vehicle not available	Vehicle is already booked
CONFLICT_003	Student not available	Student has conflicting lesson
NOT_FOUND_001	Resource not found	Requested resource doesn’t exist

PERM_001	Insufficient permissions	User lacks required permissions

B. HTTP Status Codes

Status Code	Meaning	Usage
200	OK	Successful GET, PATCH, DELETE
201	Created	Successful POST (resource created)
204	No Content	Successful DELETE (no response body)
400	Bad Request	Invalid request data
401	Unauthorized	Missing or invalid authentication
403	Forbidden	Authenticated but insufficient permissions
404	Not Found	Resource doesn't exist
409	Conflict	Resource conflict (e.g., scheduling conflict)
422	Unprocessable Entity	Validation error
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server error
503	Service Unavailable	Server temporarily unavailable

C. Database Backup Script

```bash
#!/bin/bash
# backup_database.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups/postgresql"
DB_NAME="driving_school_db"
BACKUP_FILE="$BACKUP_DIR/${DB_NAME}_${DATE}.sql.gz"

# Create backup
pg_dump -U postgres $DB_NAME | gzip > $BACKUP_FILE

# Remove backups older than 30 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete
```

```
echo "Backup completed: $BACKUP_FILE"
```

D. Environment Variables

```
# .env.example

# Database
DATABASE_HOST=localhost
DATABASE_PORT=5432
DATABASE_NAME=driving_school_db
DATABASE_USER=postgres
DATABASE_PASSWORD=your_secure_password

# JWT
JWT_SECRET=your_jwt_secret_key
JWT_EXPIRES_IN=1h
REFRESH_TOKEN_EXPIRES_IN=30d

# Email
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your_email@gmail.com
SMTP_PASSWORD=your_app_password
EMAIL_FROM=noreply@drivingschool.com

# Application
NODE_ENV=production
PORT=3000
API_BASE_URL=https://api.drivingschool.com

# File Upload
MAX_FILE_SIZE=10485760 # 10MB in bytes
UPLOAD_DIR=/uploads

# Redis (optional)
REDIS_HOST=localhost
REDIS_PORT=6379
```

E. Testing Checklist

Unit Tests

- ☐ User authentication (login, register, logout)
- ☐ Password hashing and verification
- ☐ JWT token generation and validation
- ☐ Lesson conflict detection

- [ ] Student progress calculation
- [ ] Exam capacity management

Integration Tests

- [ ] User registration flow
- [ ] Lesson request approval workflow
- [ ] Exam registration process
- [ ] Payment processing
- [ ] Email notifications

API Tests

- [ ] All endpoints return correct status codes
- [ ] Authentication required for protected endpoints
- [ ] Role-based access control enforced
- [ ] Pagination works correctly
- [ ] Error responses formatted properly

Performance Tests

- [ ] Database queries under 100ms
- [ ] API response times under 500ms
- [ ] Handle 100 concurrent users
- [ ] Load testing with 1000+ users

F. Deployment Checklist

Pre-Deployment

- [ ] All tests passing
- [ ] Environment variables configured
- [ ] Database migrations ready
- [ ] Backup current production database
- [ ] Security audit completed
- [ ] Performance testing done

Deployment

- [ ] Deploy database migrations
- [ ] Deploy backend application
- [ ] Update environment variables
- [ ] Configure load balancer
- [ ] Enable monitoring
- [ ] Test critical user flows

Post-Deployment

- [ ] Monitor error logs
- [ ] Check performance metrics
- [ ] Verify email sending
- [ ] Test user registration
- [ ] Test lesson booking
- [ ] Monitor database performance

Document Change Log

Version	Date	Author	Changes
1.0	2025-10-07	DeepAgent	Initial technical specification

End of Document

For questions or clarifications, contact the development team at dev@drivingschool.com