ready blueprint:

1. Overall architecture (tech + flow)
2. Features by role (teacher / parent / owner / staff)
3. Database design (collections & fields)
4. API design (important endpoints)
5. App structure in Flutter (screens)
6. Admin panel structure
7. Week-by-week roadmap (what to do first → last)

I'll assume **Node.js + Express + MongoDB + Flutter + simple React/HTML admin panel**.

1️⃣ Overall Architecture (High Level)

Frontend:

- **Flutter mobile app**
    - One app, 3 roles:
        - Teacher
        - Parent
        - School Owner
- **Web Admin Panel**

- Built with React or simple HTML + Bootstrap
- Used by Management Staff

Backend:

- **Node.js + Express REST API**
- **MongoDB** as database
- **JWT Auth** (role-based)
- **Payment gateway**: Razorpay / Stripe (later step)
- Optional later: **Firebase Cloud Messaging (FCM)** for notifications

Flow Example (Fee Payment):

Parent opens app → sees fee due → presses “Pay” → payment gateway page → success → backend marks fee as PAID → admin panel + owner app show updated status.

2️⃣ Features by Role (MVP)

🔹 **Common (All Roles)**

- Login / Logout
- View profile
- Change password

👩‍🏫 **Teacher (Mobile app)**

- View classes assigned
- View students list
- Mark attendance (present/absent)
- Add homework / assignments
- View timetable (optional later)

👨‍👩‍👧 Parent (Mobile app)

- View child details (class, section, roll)
- View attendance summary
- View homework / announcements
- View fees:
    - Pending fees
    - Paid history
- Pay fees online

👑 School Owner (Mobile app)

- View total students, teachers
- View fee collection summary
- See list of defaulters (students with pending fees)
- Basic reports (monthly collection, etc.)

🧑‍💼 Management Staff (Admin Panel)

- CRUD:
    - Schools (if multiple)
    - Classes, Sections
    - Students, Parents, Teachers
- Assign students to classes
- Assign teachers to classes
- Create fee structures:
    - Fee type (tuition, transport)
    - Amount
    - Due dates
- Track payments
- Export basic reports (CSV maybe later)

3️⃣ Database Design (MongoDB – Collections)

You can refine later, but here's a solid starting point.

1. `users`

Stores login + role info.

```
{
```

```
  "_id": ObjectId,
  "name": "Rahul Sharma",
  "email": "rahul@example.com",
  "password_hash": "...",
  "role": "TEACHER" | "PARENT" | "OWNER" | "STAFF",
  "schoolId": ObjectId,  // which school they belong to
  "createdAt": ISODate,
  "updatedAt": ISODate
}
```

2. schools

```
{
  "_id": ObjectId,
  "name": "ABC Public School",
  "address": "Sikar, Rajasthan",
  "ownerUserId": ObjectId,  // reference to users
  "createdAt": ISODate
}
```

3. classes

```
{
  "_id": ObjectId,
  "schoolId": ObjectId,
  "name": "Class 8",
  "section": "A",
  "classTeacherId": ObjectId, // user (teacher)
  "createdAt": ISODate
}
```

4. students

```
{
  "_id": ObjectId,
  "schoolId": ObjectId,
  "classId": ObjectId,
  "rollNo": 12,
  "name": "Aman Kumar",
  "dob": "2010-05-11",
  "parentUserId": ObjectId,
  "admissionNo": "ADM2025-001",
  "status": "ACTIVE" | "INACTIVE",
  "createdAt": ISODate
}
```

5. `attendance`

One record per class per day (with student-level status list), or one per student per day.
For now, per student per day:

```
{
  "_id": ObjectId,
  "studentId": ObjectId,
  "classId": ObjectId,
  "date": "2025-12-01",
  "status": "PRESENT" | "ABSENT" | "LEAVE",
  "markedBy": ObjectId, // teacher
  "createdAt": ISODate
}
```

6. `homeworks`

```
{
  "_id": ObjectId,
```

```
  "schoolId": ObjectId,
  "classId": ObjectId,
  "title": "Maths – Fractions",
  "description": "Solve Q1–10",
  "dueDate": "2025-12-03",
  "createdBy": ObjectId, // teacher
  "createdAt": ISODate
}
```

7. `feeStructures`

Defines the fee plan.

```
{
  "_id": ObjectId,
  "schoolId": ObjectId,
  "classId": ObjectId,   // or null if common
  "title": "Monthly Tuition Fee",
  "amount": 1500,
  "frequency": "MONTHLY" | "ONE_TIME",
  "isActive": true
}
```

8. `feeInvoices` (Very important)

For each student, for each fee cycle:

```
{
  "_id": ObjectId,
  "schoolId": ObjectId,
  "studentId": ObjectId,
  "feeStructureId": ObjectId,
  "month": 11,
```

```
  "year": 2025,
  "amount": 1500,
  "status": "PENDING" | "PAID",
  "dueDate": "2025-11-10",
  "paidDate": "2025-11-07",
  "paymentRef": "RAZORPAY_ORDER_ID",
  "createdAt": ISODate
}
```

You can add more collections later (notifications, exams, etc.), but this is enough for a strong MVP.

4️⃣ API Design (Important Endpoints)

Base: `/api`

Auth

- `POST /api/auth/register` (for staff/owner – maybe only admin uses)
- `POST /api/auth/login`
- `GET /api/auth/me` – get current user (using JWT)

Schools

- `POST /api/schools` (create school – owner/staff only)
- `GET /api/schools/:id`
- `GET /api/schools` (for owner – list all his schools if multi)

Classes

- `POST /api/classes`
- `GET /api/classes?schoolId=...`
- `PUT /api/classes/:id`
- `DELETE /api/classes/:id`

Students

- `POST /api/students`
- `GET /api/students?classId=...`
- `GET /api/students/:id`
- `PUT /api/students/:id`
- `DELETE /api/students/:id`

Attendance

- `POST /api/attendance/mark`
    - body: `{ classId, date, records: [{ studentId, status }] }`
- `GET /api/attendance/class/:classId?date=2025-12-01`

- `GET /api/attendance/student/:studentId?month=11&year=2025`

Homework

- `POST /api/homeworks`
- `GET /api/homeworks?classId=...`
- `GET /api/homeworks/:id`

Fees

- `POST /api/fees/structures`
- `GET /api/fees/structures?classId=...`
- `POST /api/fees/invoices/generate`
  (for a month for a class, etc.)
- `GET /api/fees/invoices?studentId=...`
- `GET /api/fees/invoices/:id`

Payment Integration (rough idea)

- `POST /api/payments/create-order`
    - calls Razorpay/Stripe SDK, returns order ID
- Flutter app opens payment UI

- On success, backend:
    - `POST /api/payments/webhook` (from gateway)
    - marks `feeInvoice.status = "PAID"`

Owner / Dashboard

- `GET /api/dashboard/owner?schoolId=...`
    - returns:
        - total students
        - total collected fees this month
        - total pending
        - attendance stats

5️⃣ Flutter App Structure (Screens)

One app, login decides which dashboard to show.

Core Structure

- `login_screen.dart`
- `role_selection` is not needed; role comes from backend.

After login, based on `role`:

For Teacher

- `TeacherHomeScreen`
    - Today’s classes
- `ClassStudentsScreen` (list students)
- `MarkAttendanceScreen`
- `HomeworkListScreen`
- `CreateHomeworkScreen`

For Parent

- `ParentHomeScreen`
    - Child name + class + summary
- `AttendanceSummaryScreen`
- `HomeworkScreen`
- `FeesScreen`
    - List of invoices (Pending / Paid)
    - `PayNow` button → payment

For Owner

- `OwnerDashboardScreen`
    - Cards: total students, total teachers, total collection, pending

- `ClassListScreen`
- `DefaulterListScreen` (students with pending invoices)

Use:

- `BottomNavigationBar` (Home / Attendance / Fees / Profile etc.)
- `Dio` or `http` package for API calls
- Some basic state management (Provider is enough for MVP)

6️⃣ Admin Panel Structure (Web)

You don't need fancy design initially. Use:

- React + Vite + Tailwind/Bootstrap OR even simple HTML with a template.

Pages:

1. Login
2. Dashboard (cards: total students, teachers, fees)
3. Schools (if multiple)
4. Classes & Sections
5. Students
6. Teachers

7. Parents
8. Fees:
    - Fee Structures
    - Invoices & Payments
9. Reports (basic tables)

Each page will call your backend APIs.

7️⃣ Week-by-Week Roadmap (Realistic)

Assuming ~2–3 hrs/day and you already know JS basics.

🗓️ Week 1 – Backend Skeleton

- Setup Node.js + Express project
- Connect to MongoDB
- Implement:
    - `User` model
    - Auth (register + login + JWT)
- Test with Postman

🗓️ Week 2 – Core Models

- Add models: `School`, `Class`, `Student`

- APIs for:
    - Create school
    - Create class, assign teacher
    - Create & fetch students
- Start planning fee structure model

🗓️ Week 3 – Attendance & Homework

- Create `Attendance` + `Homework` models
- Build APIs:
    - Mark attendance
    - Get attendance
    - Create + get homework

🗓️ Week 4 – Fee System

- Implement `feeStructures` and `feeInvoices`
- API to:
    - Create fee structure
    - Generate invoices for a class/month
    - Fetch invoices for a student

🗓️ Week 5 – Payment Integration

- Choose gateway (Razorpay/Stripe test mode)
- Implement:
    - `create-order` endpoint
    - Handle webhook / success logic
    - Mark invoice as paid

🗓️ Week 6 – Flutter Basics + Auth

- Learn Flutter basics (layouts, navigation)
- Build:
    - Login screen
    - Network calls to login API
- After login, navigate to role-based home screen

🗓️ Week 7 – Teacher Flow

- Build teacher UI:
    - View classes
    - View students in class
    - Mark attendance (call API)
    - Create homework

🗓️ Week 8 – Parent Flow

- Parent home:
    - Child details
    - Attendance summary (API)
    - Homework view
    - Fees screen: list invoices

🗓️ **Week 9 – Parent Payment Flow**

- Integrate payment gateway in Flutter (webview / SDK)
- After success:
    - Confirm with backend
    - Update UI of invoices

🗓️ **Week 10 – Owner & Admin Panel**

- Simple owner dashboard in app
- Start basic admin panel:
    - Login
    - View schools, classes, students list

🗓️ **Week 11–12 – Polish & Extras**

- Validation, error handling
- Simple role-based protections

- Improve UI
- Write README + record demo video

You don’t have to follow this **100% strictly**, but this is a very solid path.