Student Management System — Deep Audit & Plan

Repository: `harikiranadangi/student-management-system`

---

1) Quick repo snapshot

- **Stack**: Next.js + React + TypeScript + Tailwind; API routes with Prisma; PostgreSQL; optional Clerk/Auth.js; deployable to Vercel/Railway/DO.
- **Feature areas**: Students, Fees (ledger + per-student), Attendance, Exams/Marks, Reports, RBAC, Admin dashboard.
- **Structure (high-level)**: `prisma/` (schema + migrations), `src/api` (routes), `src/app` (routes like `/fees`, `/fees/fee_ledger`, `/fees/[id]`, `/students`, `/teachers`, `/reports`), shared components, config, lib.

  Notes: The README lists the above tech and pages; exact table/field names are inferred from your earlier messages and common SIS patterns.

---

2) Proposed data model (normalized, analytics-friendly)

Below is a practical schema that covers the features you've built and adds analytics readiness. If some names differ from your current Prisma schema, treat these as refactor targets.

Core entities

- **User** (`id`, `email`*, `phone?`, `passwordHash`/external auth id, `role` enum: `ADMIN|TEACHER|STAFF|STUDENT|PARENT`, `status`, `createdAt`, `updatedAt`)
- **Student** (`id`, `admissionNo`*, `firstName`, `lastName`, `name` (denormalized), `dob`, `gender`, `bloodType?`, `parentName`, `email?`, `phone?`, `address?`, `img?`, `joinDate`, `status` enum `ACTIVE|TRANSFERRED|ALUMNI`, timestamps)
- **Guardian** (`id`, `studentId` FK, `name`, `relation`, `email?`, `phone?`, `address?`)
- **Teacher** (`id`, `empCode`*, `name`, `email?`, `phone?`, `doj?`, `status`, timestamps)
- **Grade** (`id`, `name`, `code`, `academicYear`*, `classTeacherId?` FK Teacher, `capacity?`)
- **Section** (`id`, `gradeId` FK, `name`*, `room?`, unique `(gradeId, name)`)
- **Enrollment** (`id`, `studentId` FK, `gradeId` FK, `sectionId?` FK, `academicYear`*, `rollNo?`, `isActive`, unique `(studentId, academicYear)`)
- **Subject** (`id`, `code`, `name`, `gradeId?` FK for grade-specific syllabi)
- **TeacherSubject** (`id`, `teacherId` FK, `subjectId` FK, `gradeId?` FK, unique `(teacherId, subjectId, gradeId)`)

Attendance

- **AttendanceSession** (`id`, `gradeId`, `sectionId?`, `date`, `period?`, `takenById` Teacher, unique `(gradeId, sectionId, date, period)`)
- **AttendanceEntry** (`id`, `sessionId` FK, `studentId` FK, `status` enum `PRESENT|ABSENT|LATE|EXCUSED`, `notes?`)

Exams & Marks

- **Exam** (`id`, `name`, `type` enum `TERM|UNIT|PRACTICAL|FINAL`, `term` enum, `academicYear`, `startDate?`, `endDate?`)
- **ExamPaper** (`id`, `examId` FK, `subjectId` FK, `maxMarks`, unique `(examId, subjectId)`)
- **Mark** (`id`, `examPaperId` FK, `studentId` FK, `marksObtained`, `grade?`, `remarks?`)

Fees & Accounting

- **FeeHead** (`id`, `name`, `code`, `isRecurring`, `isDiscountable`)
- **FeeStructure** (`id`, `gradeId` FK, `term` enum, `academicYear`, `startDate`, `dueDate`, `lineTotal`, unique `(gradeId, term, academicYear)`)
- **FeeStructureItem** (`id`, `feeStructureId` FK, `feeHeadId` FK, `amount`)
- **StudentFeeSchedule** (`id`, `studentId` FK, `feeStructureId` FK, `amount`, `discountAmount default 0`, `fineAmount default 0`, `status enum PENDING|PARTIAL|PAID`, unique `(studentId, feeStructureId)`)
- **Receipt** (`id`, `studentId` FK, `receiptNo`*, `receiptDate`, `mode enum CASH|UPI|CHEQUE|NEFT`, `remarks?`, `createdById` User)
- **ReceiptLine** (`id`, `receiptId` FK, `studentFeeScheduleId` FK, `amount`)
- **LedgerEntry** (`id`, `studentId` FK, `txnDate`, `type enum DEBIT|CREDIT`, `amount`, `narration`, `linkType enum RECEIPT|FEE|ADJUSTMENT`, `linkId`, index on `(studentId, txnDate)`)

Reporting & Audit

- **ReportJob** (`id`, `type`, `params jsonb`, `status`, `requestedById`, `createdAt`, `completedAt?`, `filePath?`)
- **AuditLog** (`id`, `actorId` User, `action`, `entity`, `entityId`, `before jsonb?`, `after jsonb?`, `ip?`, `at`)

Keys marked with `*` *should be unique**.

---

3) Suggested Prisma schema refinements

- Use **UUID** primary keys; keep human-friendly codes (admissionNo, empCode, receiptNo) as unique fields with indexes.
- Add **composite uniques** noted above to prevent duplicates (e.g., `(gradeId, term, academicYear)` on `FeeStructure`).
- Prefer **enum** for `term`, `role`, attendance status, payment mode.

- Add **soft delete** flags only where needed (e.g., `isActive`) rather than global deletes.
- **Timestamps** (`createdAt`, `updatedAt`) defaulted via Prisma.
- **Foreign keys** with `onDelete: Restrict` for finance tables; `onDelete: Cascade` for child collections like `AttendanceEntry`.
- Introduce **computed/denormalized** helpers: `Student.name`, `LedgerEntry.balanceAfter` (materialized view or nightly job) for faster ledger screens.

---

4) Indexing & performance plan

- Indexes:
- `Enrollment(studentId, academicYear)`, `AttendanceSession(gradeId, date)`, `AttendanceEntry(sessionId, studentId)`
- `StudentFeeSchedule(studentId, status)`, `Receipt(receiptNo)`, `LedgerEntry(studentId, txnDate)`
- API pagination: **cursor-based** for large lists (students, receipts, ledger).
- Fee ledger screen: pre-aggregate **running balance** per student per academic year; cache in Redis or materialized views.
- Reports: long-running report jobs via `ReportJob` + background worker; serve files from object storage.

---

5) Validation & data quality

- Strong Zod/Yup schemas on all forms + server-side validation.
- Guard rails:
- Prevent editing `FeeStructure` after receipts exist; require adjustment entries.
- Validate `receiptNo` uniqueness per year/branch.
- Enforce sum(ReceiptLine) == Receipt.amount and sum(FeeStructureItem) == FeeStructure.lineTotal.

---

6) RBAC & security

- Roles: `ADMIN`, `ACCOUNTANT`, `TEACHER`, `STAFF`, `STUDENT`, `PARENT`.
- Policy highlights:
- Finance endpoints require `ACCOUNTANT|ADMIN`.
- Teachers restricted to sections/subjects they teach.
- Students/Parents scoped to own `studentId`.
- Add **row-level filtering** in services (don’t rely on UI).
- **AuditLog** for critical actions: receipts, marks edit, attendance overrides.

---

7) Analytics KPIs & dashboards

- **Finance**: Collection %, Outstanding by grade/term, DSO (days-sales-outstanding), Receipts by mode, Top defaulters.
- **Attendance**: Daily % by grade/section, Student streaks, Chronic absence (<=75%), Late/Excused trends.
- **Academics**: Subject averages, Top/bottom 10, Term-wise progress, Correlation (attendance vs marks).
- **Ops**: Admissions trend, Transfers, Capacity utilization by section.

Data marts / summary tables: - `fact_receipt_daily (date, amount, count, mode)` - `fact_attendance_daily (date, gradeId, sectionId, present, absent, late)` - `fact_marks (examId, subjectId, studentId, marks, grade)` - Dimension tables for `dim_grade`, `dim_subject`, `dim_student` (SCD-2 optional).

---

8) Exports & external analytics API

Endpoints (read-only) with pagination and filters: - `GET /api/export/students?updatedSince=...` - `GET /api/export/fees/ledger?studentId=...&from=...&to=...` - `GET /api/export/receipts?from=...&to=...&mode=...` - `GET /api/export/attendance?date=...&gradeId=...` - `GET /api/export/marks?examId=...&subjectId=...`

CSV samples - Students: `admissionNo,name,grade,section,phone,email,joinDate,status` - Receipts: `receiptNo,date,studentId,amount,mode,createdBy`

Rate-limits + tokens (JWT or PAT) for secure external access.

---

9) Reporting catalogue (starter)

- **Fees**: Term wise outstanding, Student ledger, Receipt register, Concession report, Aging (0–30/31–60/61–90/90+).
- **Attendance**: Daily class register, Monthly summary card per student, Absence letters.
- **Academics**: Exam result sheet, Subject-wise performance, Report card PDF.

---

10) Sample Prisma snippets

```
// Find student with active enrollment
await prisma.student.findUnique({
  where: { admissionNo },
  include: {
    enrollments: { where: { academicYear, isActive: true }, include: { grade: true, section: true } },
  },
});

// Create receipt with lines in a transaction
await prisma.$transaction(async (tx) => {
  const receipt = await tx.receipt.create({ data: { studentId, receiptNo, receiptDate, mode, remarks, createdById } });
  for (const line of lines) {
    await tx.receiptLine.create({ data: { receiptId: receipt.id, studentFeeScheduleId: line.scheduleId, amount: line.amount } });
    await tx.ledgerEntry.create({ data: { studentId, txnDate: receiptDate, type: 'CREDIT', amount: line.amount, linkType: 'RECEIPT', linkId: receipt.id } });
  }
});
```

---

11) SQL indexes (PostgreSQL)

```
CREATE INDEX ON "Enrollment" ("studentId", "academicYear");
CREATE UNIQUE INDEX ON "FeeStructure" ("gradeId", "term", "academicYear");
CREATE INDEX ON "StudentFeeSchedule" ("studentId", "status");
CREATE INDEX ON "LedgerEntry" ("studentId", "txnDate");
CREATE UNIQUE INDEX ON "Receipt" ("receiptNo");
```

---

12) DevOps/production checklist

- CI: lint, typecheck, build, `prisma migrate deploy`, smoke tests.
- **Backups**: daily PG dump + PITR; test restores.
- **Observability**: error tracking (Sentry), logs, slow query log, uptime monitor.
- **Secrets**: separate envs, no secrets in repo; rotate DB creds.
- **Data protection**: access logs, PII encryption at rest (if needed), annual data retention policy.

---

13) Next actions (concrete)

1) Compare this model vs current `schema.prisma`; create a diff plan. 2) Add composite uniques + indexes; generate migrations. 3) Implement read-only `/api/export/...` endpoints. 4) Build finance & attendance summary tables and dashboards. 5) Add `AuditLog` on receipts/marks and tighten RBAC guards.

Tell me if you want me to tailor the schema around your exact `schema.prisma` *(I can map one-to-one fields if you paste it or grant me read access to the raw file), and I'll refactor the API routes accordingly.*