E-System Backend — Project Workflow

Updated: 2025-10-20

Overview

- **Stack**: Node.js + TypeScript, Express, MongoDB (Mongoose), JWT auth, Joi validation.
- **Structure**: `src/` with `config/`, `middleware/`, `models/`, `controllers/`, `routes/`, `services/`, `utils/`; scripts in `scripts/`.
- **Key modules**: Auth, Users, Courses, Attendance, Grades, Announcements, Admin/Teacher/Student scoped features.

Startup and Configuration

- **Entrypoint**: `src/server.ts`
  - Loads env via `src/config/env.ts` (dotenv), connects to Mongo with `connectToDatabase`, starts HTTP server on `env.port`.
- **App setup**: `src/app.ts`
  - Global middleware: CORS, JSON/urlencoded parsers, logging (morgan), security headers (helmet), static `uploads/`, favicon handler.
  - Health: `GET /health` returns status/ts/env.
  - API docs: Swagger UI at `/api-docs`, spec at `/openapi.json`.
  - Rate limiting: `apiLimiter` globally; `authLimiter` for `/api/v1/auth`.
  - Routes mounted under `/api/v1` via `src/routes/index.ts`.

Environment and Settings (`src/config/env.ts`)

- Server: `PORT`, `NODE_ENV`.
- Database: `MONGO_URI`.
- Auth: `JWT_SECRET`, `JWT_EXPIRES_IN`, optional `ADMIN_SIGNUP_CODE`.
- SMTP: host/port/secure/user/password for email utilities.
- CORS: `CORS_ORIGIN`, `CORS_CREDENTIALS`.
- Rate limits: window and max per window.
- Uploads: max size, allowed mime types, `UPLOAD_PATH`.

Request Lifecycle

1. Request enters Express → CORS → parsing → logging → helmet → global `apiLimiter`.
2. Health/docs/static routes handled early.
3. Versioned API routes `/api/v1` → feature routers.
4. Per-route: `authenticate` (JWT to `req.user`) and `authorize` (role checks) as needed.
5. `validate()` (Joi) checks `body/query/params` and returns 400 on failure.
6. Controllers execute domain logic; async errors go to global `errorHandler`.
7. `notFoundHandler` for unmounted routes; `errorHandler` normalizes Mongoose/JWT/dup-key errors.

Authentication and Authorization

- **JWT issuance**: On register/login; token signed via `services/auth.service.ts` with configurable expiry.
- **Middlewares** (`src/middleware/auth.ts`):
  - `authenticate`: verifies Bearer token, loads user, attaches `req.user`.
  - `authorize(['admin'|'teacher'|'student'])`: enforces role access.
  - `requireAuth` / `requireRoles` are lighter variants using decoded payload only.

Data Models (Mongoose)

- **User**: `fullName`, `email` (unique), `passwordHash`, `role` in `{'admin','teacher','student'}`.
- **Course**: `title`, `description`, `code` (unique, uppercase), `credits`, `teacher`, `students[]`, `semester`, `academicYear`, `isActive`.
- **Attendance**: `student`, `course`, `date`, `status` in `{'present','absent','late','excused'}`, `recordedBy`, `notes`; unique `(student,course,date)`.
- **Grade**: `student`, `course`, `gradeType`, `title`, `score/maxScore`, computed `percentage` and `letterGrade` (pre-save), `gradedBy`, optional `submittedAt`.
- **Announcement**: `title`, `content`, `type`, `author`, `targetAudience[]`, optional `course`, `isActive`, `publishedAt`, optional `expiresAt`, `attachments[]`.

Core Middleware

- **Validation**: `middleware/validation.ts` with `validate()` and `commonSchemas` (pagination, `objectId`, `email`, `password`).
- **Errors**: `middleware/errorHandler.ts` handles Mongoose cast/dup/joi-ish and JWT errors; includes `asyncHandler` wrapper and `notFoundHandler`.
- **Rate limiting**: `middleware/rateLimit.ts` with `apiLimiter`, `authLimiter`, and `uploadLimiter` presets.

Feature Workflows

- **Auth** (`/api/v1/auth`)
  - `POST /register`: optional `role`, `adminCode` enforcement for admin role.
  - `POST /login`: returns JWT and user info.
  - `GET /profile`: via `authenticate`.
- **Courses** (`/api/v1/courses`)
  - List/filter/paginate; get by id.
  - Create/update by `teacher|admin` (Joi schemas enforced).
  - Enroll/remove student; delete course (admin only).
- **Attendance** (`/api/v1/attendance`)
  - List with filters (student, course, date range, status) and role scoping.
  - Record single/bulk by `teacher|admin` (enrollment and duplicates validated).
  - Update/delete by `teacher|admin`.
  - Per-course stats grouped by student/status (teacher/admin only).
- **Grades** (`/api/v1/grades`)
  - List with role scoping (student → own; teacher → own courses; admin → all) and filters.
  - Create/update/delete by course teacher or admin; validates enrollment and `score ≤ maxScore`.
  - Student-focused endpoints: list grades; compute GPA (weighted by maxScore) and average percentage.
- **Announcements** (`/api/v1/announcements`)
  - Create/list/update announcements with type, audience targeting, optional course linkage.

Seeding Workflow (`scripts/seed.ts`)

- Connects with Mongoose using `env.mongoUri`.

- Upserts deterministic sample data:
  - Users: admins, teachers, students (passwords hashed via bcrypt).
  - Courses: assign teacher, enroll students by email.
  - Grades: multiple types per course; computes `percentage` and `letterGrade`.
  - Attendance: ~30 days historical per student-course with weighted statuses.
  - Announcements: general, course, academic, emergency.
- Idempotent (`findOneAndUpdate` + `$setOnInsert`) for repeatable runs; logs a summary then disconnects.

Tooling and Scripts (`package.json`)

- Dev/start: `dev` (nodemon TS), `build` (tsc), `start` (node dist).
- Quality: `typecheck`, `lint` / `lint:fix`, `format`.
- Testing: `test`, coverage/watch variants.
- Ops: `db:seed` (ts-node), `db:test` (connectivity), `status` (app/db check), `test:api`.

Deployment and Ops

- **Docker Compose**: App (port 4000), MongoDB (port 27017), Mongo Express (port 8081). Volumes for Mongo and `uploads/`. Env vars provided in compose.
- **Static uploads**: Served from `/uploads`.

API Map (selected)

- Root: `GET /api/v1` returns API metadata and endpoint map.
- Health: `GET /health`.
- Docs: `GET /api-docs`, `GET /openapi.json`.
- Feature roots: `/api/v1/auth`, `/admin`, `/teacher`, `/student`, `/courses`, `/attendance`, `/users`, `/grades`, `/announcements`, `/data`.

Notes

- Role-based scoping is enforced in controllers in addition to route guards for defense in depth.
- Global error handler avoids leaking internals; development mode includes stack traces.
- Rate limiting is stricter for auth to mitigate brute force.