Backend: Components, Flows, and API Design

Below is a single-source reference for what the backend will contain, how data/flows move, which endpoints exist, and which endpoints must be public or private. Use this as the contract for frontend and admin panel development.

1 — Core backend components

- Express API server (or Next.js API)
- MongoDB (collections listed below)
- Redis + Queue (BullMQ) for background jobs (certificate generation, emails)
- Puppeteer worker (generates PDF certificates)
- S3 / Cloudinary (store certificate PDFs, uploads)
- Mailer (nodemailer or transactional provider)
- Payment integration (Razorpay + webhook listener)
- Auth: JWT access + refresh (HttpOnly cookies), role-based checks
- Logging & Audit: `auditLogs` collection, request logs (morgan/sentry)

2 — Main data models (collections)

- **users** — user profile, email, passwordHash, role, createdAt
- **courses** — title, slug, price, description, modules, published
- **enrollments** — userId, courseId, paymentId, status, progress, enrolledAt
- **progress** — userId, courseId, moduleId, lessonId, completedAt (fine-grained)
- **submissions** — project submissions, grade, mentorComments, verified
- **certificates** — certId, userId, courseId, pdfUrl, issuedAt, status, hash
- **payments** — providerId / orderId, userId, courseId, amount, status, rawResponse
- **auditLogs** — action, actorId, targetId, timestamp, details

3 — High-level flows (short)

Signup / Auth

- User signs up → `POST /api/auth/signup` → creates user, returns tokens in HttpOnly cookies.
- Login → `POST /api/auth/login` → sets cookies. Refresh via `POST /api/auth/refresh`. Logout clears cookies.

Course browse → enroll → lesson progress

- Visitor fetches courses `GET /api/courses` (public).
- Course detail `GET /api/courses/:slug` (public).
- Enroll: `POST /api/enroll` (private; user must be logged in) → creates enrollment + Razorpay order or simulated success.
- Payment webhook updates `POST /api/payments/webhook` (public endpoint with HMAC verification) → marks enrollment active.
- Student marks lesson complete `POST /api/progress/complete-lesson` (private) → updates progress and `enrollments.progress`. If progress==100 → create certificate record and enqueue certificate job.

Certificate generation & verification

- Certificate job picks pending cert → Puppeteer renders HTML → PDF buffer.
- Upload to S3 → update certificate doc with `pdfUrl`, `issuedAt`, `status: active`, compute hash.
- Notify student via email + push.
- Public verification: `GET /api/certificates/:certId` (public) → returns student name, course, issuedAt, pdfUrl, status. Admin can revoke via private API.

Admin workflows

- Admin creates/edits courses, views students, approves/declines submissions, issues/revokes certificates, views analytics. All admin routes are private and role-guarded.

4 — API endpoints (organized, method, path, auth, purpose)

Auth

- `POST /api/auth/signup` — public — body: {name,email,password} → sets cookies, returns user.
- `POST /api/auth/login` — public — body: {email,password} → sets cookies.
- `POST /api/auth/refresh` — public (uses refresh cookie) — rotates access token.
- `POST /api/auth/logout` — private — clears cookies.

Public (no auth required)

- `GET /api/health` — health check
- `GET /api/courses` — list published courses (filter, pagination)
- `GET /api/courses/:slug` — course detail (including public preview lessons)
- `GET /api/certificates/:certId` — public certificate verification (read-only)
- `POST /api/payments/webhook` — payment provider webhook (verify signature, update payment + enrollment)

User (authenticated)

- `GET /api/user/me` — private — current user profile
- `GET /api/user/certificates` — private — list user certificates
- `POST /api/enroll` — private — enroll (creates payment order or simulated success)
  body: { courseId, paymentMode: 'razorpay'|'simulated' }
- `GET /api/enrollments` — private — list user's enrollments
- `POST /api/progress/complete-lesson` — private — body: { courseId, moduleId, lessonId }
- `POST /api/submissions` — private — project submission: { courseId, submissionUrl, files? }
- `GET /api/submissions` — private — user's submissions

Admin (authenticated + role=admin/instructor)

- `POST /api/admin/courses` — create course
- `PUT /api/admin/courses/:id` — update course
- `DELETE /api/admin/courses/:id` — remove course
- `GET /api/admin/students` — list students, search
- `GET /api/admin/enrollments` — list + filter
- `POST /api/admin/verify-submission` — approve/reject project
- `POST /api/admin/certificates/:certId/revoke` — revoke certificate (sets status, add audit log)
- `POST /api/admin/certificates/:certId/issue` — manually re-issue certificate (enqueue job)
- `GET /api/admin/analytics` — basic metrics (enrollments, completions, revenue)
- `GET /api/admin/logs` — audit logs (paginated)

5 — Which endpoints must be public vs private (quick list)

- **Public:** `/api/health`, `/api/courses`, `/api/courses/:slug`, `/api/certificates/:certId`, `/api/payments/webhook`
- **Private (authenticated user):** `/api/user/*`, `/api/enroll`, `/api/progress/*`, `/api/submissions/*`, `/api/user/certificates`
- **Admin-only (role protected):** `/api/admin/*`, `/api/admin/certificates/*`, `/api/admin/analytics`, course create/update/delete endpoints if you separate admin routes

6 — Data & background job responsibilities

- **Enrollment update:** on successful payment, create enrollment document and payment record.
- **Progress tracking:** store each lesson completion in `progress` collection (fine-grained) — derive percentage from lessons completed vs total lessons.
- **Certificate job:** worker reads certificate doc (status pending), fetches user+course, renders HTML, produces PDF, uploads, updates DB, sends email.
- **Webhook processing:** verify provider signature, update `payments` collection and set enrollment status to `active`. Log everything in `auditLogs`.

7 — Security rules & best-practices (must implement)

- JWT access token short life (e.g., 15m), refresh token rotation. Store both as HttpOnly, Secure cookies.
- Role checks middleware for admin routes.
- HMAC verification of payment webhook (verify Razorpay signature).
- Rate-limit login and webhook endpoints.
- Validate/escape all inputs; sanitize file uploads.
- CSP headers for pages that render certificate preview.
- Audit every admin action (who did what).

8 — Example request/response (important ones)

Enroll (private)

```
Request: POST /api/enroll
{ "courseId": "64f...", "paymentMode": "simulated" }

Response (simulated):
{ "ok": true, "enrollment": { "id": "...", "courseId": "...", "status": "active",
```

Mark lesson complete (private)

```
Request: POST /api/progress/complete-lesson
{ "courseId": "64f...", "moduleId":"m1", "lessonId":"l3" }

Response:
{ "ok": true, "progress": 65 }
(if progress becomes 100 -> server also returns { certificateCreated: true, certId
```

Certificate verify (public)

```
Request: GET /api/certificates/NEA-123456

Response:
{
  "certId":"NEA-123456",
  "student":{ "id":"...", "name":"Amit Kumar" },
  "course":{ "id":"...", "title":"MERN Bootcamp" },
  "issuedAt":"2025-11-20T12:00:00Z",
  "pdfUrl":"https://s3.amazonaws.com/.../NEA-123456.pdf",
  "status":"active"
}
```

9 — Webhooks & external integrations

- **Razorpay:** `POST /api/payments/webhook` — verify signature, update payments, set `enrollment.status='active'`, add audit log.
- **Email provider / SMTP:** used by worker to notify student.
- **S3/Cloudinary:** worker uploads certificate PDF. Use environment creds and keep private.
- Optional: add a small public /verify page on the frontend that calls the public certificate API for UX.

10 — Logging, monitoring & observability

- Store `auditLogs` for admin actions (create/issue/revoke).
- Use Sentry for error tracking; DB metrics via Atlas monitoring.
- Keep webhook retry logic and idempotency (store provider orderId to prevent duplicates).

11 — Minimal next-step checklist (to start backend build)

1. Implement models & DB connection.
2. Implement auth routes + cookies + middleware.
3. Implement `GET /api/courses` + `GET /api/courses/:slug`.
4. Implement enroll route (simulated mode) + seed course & users.
5. Implement `POST /api/progress/complete-lesson` with progress calc.
6. Implement certificate model + enqueue job on completion.
7. Build worker (Puppeteer) and S3 upload.
8. Add `GET /api/certificates/:certId` public verify.
9. Add payment webhook after above flow stable.