Nice — I’ll turn that raw feature list into a clear, no-fluff, step-by-step spec you can code to or hand off to teammates. I’ll separate **user flows** (how people use it) and **implementation steps** (how to build it). I’ll also give **API + DB sketches**, **security rules**, and a short **MVP acceptance checklist** so you know when to demo.

**Quick summary**

Build an MVP-first Student/Teacher/Parent/Admin dashboard with QR attendance, real-time updates, downloadable reports, a timetable, resource recommendations, and a gamified progress tracker. Priority: attendance → reports → timetable → notifications → recommendations → gamification.

**User flows — step by step (what happens, in order)**

**Student (daily flow)**

1. Student signs up / logs in (email/college ID + password / SSO).
2. Student lands on Dashboard: summary (today’s classes, attendance %, next tasks, badges).
3. Student views timetable — free periods are highlighted.
4. Student opens “Attendance” for a class: scans session QR (camera).
5. Scanner app reads QR payload → sends token + student credentials to server.
6. Server validates token and marks attendance; response returns updated percentage.
7. Dashboard shows real-time confirmation (toast) + updated attendance % and badge progress.
8. Student opens “Personal Goals / Free Period Tasks”: picks a micro-task or uses AI suggestion.
9. Student can download performance PDF or view semester graphs.

**Teacher (daily flow)**

1. Teacher logs in → selects a class/section.
2. Teacher opens “Generate Attendance” → server creates a session (id, start\_time, expiry) and signs a token.
3. Teacher gets QR (display fullscreen or print).
4. Students scan QR; teacher sees a live list of scanned students (and can manually add/remove).
5. Teacher uploads materials / posts assignment and sets deadline.
6. Teacher views analytics: student attendance list, low-attendance alerts, performance snapshots.
7. Teacher can send announcements to parents/students.

**Parent (daily flow)**

1. Parent logs in -> links to child (using invite code or student ID verified by admin).
2. Parent views child’s attendance overview, grades, timetable, and alerts for low attendance.
3. Parent receives notifications (absence, PTM invites, announcements).

**Admin (daily flow)**

1. Admin logs in → manages users, classes, and syllabus.
2. Admin sets global thresholds (attendance threshold 75%), manages calendar, and sees analytics dashboards.
3. Admin exports reports (CSV/PDF) and schedules institute-wide announcements.

**Implementation plan — step by step (MVP-first, iterative)**

**Phase: Requirements + design**

1. Convert features into a prioritized backlog. MVP = QR attendance + student/teacher auth + attendance reporting + basic timetable + notifications (in-app).
2. Create wireframes for four dashboards: Student, Teacher, Parent, Admin. Keep UI minimal and mobile-first.

**Phase: Data model & auth**

1. Design DB schema (sketch below). Decide roles: Admin, Teacher, Student, Parent. Implement RBAC (role-based access control).
2. Implement authentication: email/collegeID + password; optional SSO later. Use JWT or session cookies with secure flags.

**Phase: Core APIs & QR attendance flow**

1. Create REST endpoints for classes, sessions, attendance, timetables, resources, reports. (API examples below.)
2. QR attendance session creation: Teacher calls POST /attendance/sessions → server creates session\_id, starts\_at, ends\_at, signs payload (HMAC or short-lived JWT) → returns QR image that encodes {session\_id, signature}.
3. Student scanning flow: client extracts payload → calls POST /attendance/mark with {session\_id, student\_id, signature} → server verifies signature and time window → writes attendance record.
4. Manual override: /attendance/sessions/{id}/override (teacher/admin only).

**Phase: Real-time & UI feedback**

1. Implement WebSockets (Django Channels or Socket.IO): teacher’s UI receives live scan updates; students receive immediate confirmation.
2. Use optimistic UI for fast feedback: show “pending” while server confirms, then update.

**Phase: Reports, PDFs & Graphs**

1. Build performance endpoints and front-end graphs (Chart.js / Recharts).
2. PDF export: generate downloadable report cards via background job (Celery) using WeasyPrint/ReportLab.

**Phase: Timetable & Calendar sync**

1. Timetable CRUD endpoints and UI. Add Google Calendar sync as optional: OAuth2 flow to create events on permission.

**Phase: Notifications & Alerts**

1. Implement notification system: in-app push + email. Use background workers for scheduled alerts (e.g., low attendance alerts).
2. Admin teacher/parent notification roles and templates.

**Phase: Resource recommendation (basic AI)**

1. Start with rule-based recommendations (if grade < X → recommend Y course/video). Later replace with lightweight model or a RAG-based AI suggestion engine.

**Phase: Gamification & Leaderboards**

1. Add badges, points for attendance/task completion, and leaderboards. Ensure privacy (anonymize if required).

**Phase: QA, privacy, deployment**

1. Full QA: unit tests, integration tests, manual UAT. Test edge cases (QR replay, clock skew, offline scanning).
2. Deploy: containerize (Docker), CI/CD, HTTPS, monitoring, backups.

**Minimal viable database schema (sketch)**

* users (id, email, hashed\_password, role, name, created\_at)
* students (id, user\_id, college\_roll, class\_id, parent\_id)
* teachers (id, user\_id, department)
* classes (id, name, subject\_code, teacher\_id)
* timetables (id, class\_id, day, start\_time, end\_time, location)
* attendance\_sessions (id, class\_id, starts\_at, ends\_at, created\_by, signature, status)
* attendance\_records (id, session\_id, student\_id, status[present/absent/late], timestamp, marked\_by)
* assignments (id, class\_id, title, description, due\_date, attachments)
* resources (id, subject, url, type, recommended\_for)
* notifications (id, user\_id, title, body, read, created\_at)
* badges / points tables for gamification
* parents table linking to student(s) + permissions

**Suggested API endpoints (concise examples)**

* POST /auth/login → { token }
* GET /dashboard/student → student summary
* GET /classes/{id}/timetable
* POST /attendance/sessions (teacher creates session) → returns {session\_id, qr\_data}
* POST /attendance/mark (student marks) → {status: 'ok', attendance\_percent}
* POST /attendance/sessions/{id}/override (teacher) → manual add/remove
* GET /reports/student/{id} → JSON + GET /reports/student/{id}/pdf
* POST /resources/recommend → {recommended: [...]}

**QR security & anti-fraud (step-by-step)**

1. QR payload should not be a plain session id. Use a signed token (HMAC or short-lived JWT) that includes session\_id, issued\_at, expires\_at, and maybe class\_id.
2. Server verifies signature + time window before accepting. Reject repeated marks for same session\_id + student\_id (idempotency).
3. Prevent replay by including nonce or tracking attendance\_records and rejecting duplicates.
4. Use TLS (HTTPS) for all communications. Rate-limit marking endpoint to prevent abuse.

**Real-time & offline considerations**

* Use WebSockets to show live scan updates to teachers.
* Offline scanner: allow client to cache scans and batch-submit when online — server must validate timestamps and reject expired session tokens.

**Security & privacy (must do)**

1. HTTPS + HSTS.
2. Use Django’s built-in hashed passwords (pbkdf2) or bcrypt. Don’t store plaintext.
3. Role-based access control: enforce server-side checks on every endpoint.
4. Encrypt sensitive fields at rest if required (student personal info). Use DB-level encryption if mandated by the institute.
5. Audit logs for overrides and manual edits.
6. Data retention policy and GDPR-like export/delete for parents/students.

**Acceptance criteria — what “done” looks like (MVP)**

* Teacher can create a QR session and see live students scanning.
* Students can scan and mark attendance; attendance % updates correctly.
* Admin can export attendance CSV/PDF.
* Timetable displays correctly and highlights free periods.
* Parent can view linked child’s attendance and receive alerts.
* Notifications are delivered in-app.
* Basic resource recommendation (rule-based) appears under weak subjects.
* Manual override works and is auditable.

**Edge cases & tests to write**

* Duplicate scans / replay attack.
* Student changes device mid-session.
* Timezone and clock skew (server authoritative).
* Students with multiple classes at same time.
* Offline scanning and delayed submission.
* Permission and role escalation tests.

**Tech stack recommendation (strong opinion)**

* Backend: **Django + Django REST Framework** (you already lean Django). Use **Django Channels** for real-time.
* DB: **PostgreSQL**.
* Background jobs: **Celery + Redis**.
* Frontend: **React** + Tailwind (mobile-first). Use a camera + QR scanner package.
* Charts: **Chart.js** or Recharts.
* PDF: **WeasyPrint** or ReportLab.
* Deployment: Docker + GitHub Actions → cloud provider (your college infra, Render, or AWS).

**Hackathon / demo checklist (what to show)**

1. Live demo of teacher generating QR and students scanning (real-time list).
2. Show attendance % change live on student dashboard.
3. Export a PDF report and show a timetable with highlighted free period.
4. Show a teacher uploading an assignment and sending an announcement.
5. Show low-attendance alert email / in-app notification.
6. Quick slides: problem, demo, architecture diagram, next steps.

Alright ✅ so your **finalized tech stack** is:

* **Frontend** → React
* **Backend** → Django (Python) + MySQL
* **AI/ML** → TensorFlow / PyTorch (for Free Period Recommendation engine)
* **Hosting** → Locally hosted (judges’ laptops/server demo, not cloud)

That’s perfect for SIH since you don’t have to deal with cloud latency or Wi-Fi issues during demo.

Here’s how we’ll **map this into a flowchart layout like your mining example** 👇

**🔄 Flowchart Layout Plan**

**Top: Role-Based Login**

* Oval (Start) → Rectangle (“Role-Based Login”) → Diamond (Authorized? YES/NO)
* If NO → End
* If YES → Redirect to 4 lanes: Student / Teacher / Parent / Admin

**Lane 1: 🎓 Student Dashboard**

1. 📌 **Attendance**
   * Scan QR (Input parallelogram)
   * Diamond: “Low Attendance <75%?”
     + YES → Alert notification
     + NO → Store in DB
2. 📊 **Performance Report** (rectangle)
   * Show marks, generate PDF, Graphs
3. 🗓️ **Timetable**
   * Highlight free periods
   * Connect to **AI Module (Free Period Recommendation)**
4. 🎯 **Personal Goals & Progress Tracker**
   * Gamification → Badges / Leaderboard
5. 📚 **Resources Recommendation**
   * Based on weak subjects
6. 📥 **Class Tasks & Materials**
   * Download notes, assignments
7. 🔔 **Notifications**

**Lane 2: 👩‍🏫 Teacher Dashboard**

1. Select Group/Class
2. Generate QR for Attendance
   * Students scan → Validate → Store DB
3. Assign Tasks / Upload Materials
4. View Student Reports (Attendance %, Marks)
5. Announcements (broadcast)
6. Teacher’s Project Space (optional)

**Lane 3: 👨‍👩‍👦 Parent Dashboard**

1. View Child’s Report (Marks, Graphs)
2. View Timetable
3. Attendance Overview (Alerts if < threshold)
4. Notifications (PTM, Absentee alerts)

**Lane 4: 🛠️ Admin Dashboard**

1. Manage Users
2. Curriculum Planning
3. Announcements
4. Analytics (Attendance trends, Productivity insights)
5. Reports (Export PDFs, Dept-level)

**System AI Module (Side Box)**

* Connects only to **Student Timetable**
* Diamond: “Free Period Detected?”
  + YES → Suggest micro-tasks/resources (AI logic)
  + NO → Continue normal schedule

**Side Tech Stack Box**

* Frontend → React
* Backend → Django + MySQL
* AI/ML → TensorFlow / PyTorch
* Hosting → Local