I have below project requirement which is mentioned on very high level.

as an expert Technical Project Manager your job is to create detailed Project requirement out of it. I may have missed many critical points but you have to identify the gaps and think deeper to understand my project requirement and come up with practical project requirement.
You are free to add , modify points if you think that will be crucial for this project.

You may ask questions if you need some more clarification:

High level Project requirement:

We want to build an application for assessing Students Cognizance/IQ Level on different levels/capabilities/skills like , Numeracy Skill, Logical Reasoning , Verbal, Visual and etc
The Test may provide scores on each front.
The Idea is to analyze students current level and provide special training if required, base on the results of this test.
Application Requirement.
Frontend: (ReactJs, css)
A Web based application with different level of user access like Admin, Teacher, Student
Different Dashboards - Cumulative , Grade wise, age wise , individual student.
The question on screen could be in many forms (multi Modality)
a. Text based
b. Audio
c. Video
d. Pictorial
e. Drag and drop where student can interact with the elements
f. A input box where student can write using mouse or e-pen
The idea is to assess student on each front
Backend (Python, Fast API , Django, any database)
User management services to add , modify , delete users
Question bank services to add , modify , delete questions
backend should support questions of different type
a. Text based
b. Audio
c. Video
d. Pictorial
e. Drag and drop where student can interact with the elements
f. A input box where student can write using mouse or e-pen

services to create dashboard.
database to store all relevant data.

Here is a comprehensive, gap-free project requirements specification that translates your high-level idea into a detailed, build-ready plan, including assumptions, options, and clarifying questions where needed.

Project overview

The application will assess students' cognitive abilities across domains (Numeracy, Logical Reasoning, Verbal, Visual, and others), deliver multi-modal tests, compute domain-level scores and proficiency levels, and recommend targeted training plans. It will support Admin, Teacher, and Student roles, with robust dashboards for cumulative, grade-wise, age-wise, and individual insights.

Objectives

- Deliver reliable, fair, and accessible cognitive assessments across domains.
- Provide precise, actionable analytics at student, class, grade, and organization levels.
- Enable teachers/admins to manage users, classes, tests, questions, and training plans.
- Support multi-modal questions and interactions, including freehand input with mouse/e-pen.
- Ensure security, compliance, and scalability for institutional deployments.

User roles and permissions

- Admin
  - Organization setup, school/class structures, role management, permissions, global settings.
  - Full user, content, test, and reporting control.
- Teacher
  - Manage class rosters, assign/schedule tests, proctor sessions, view analytics for their classes.
  - Create/curate questions and test forms (if permitted).
- Student
  - Take assigned tests, view personal results and recommended trainings.
- Optional roles
  - Content Author/Reviewer (content lifecycle and quality checks).
  - Parent/Guardian (view child's progress).
  - Proctor (monitoring without content edit permissions).

Functional requirements

Authentication and access

- Email/password, institution SSO (OIDC/SAML), optional OAuth.
- Multi-tenant support (separate orgs/schools).
- Password policies, MFA (optional), session timeout, device registration (optional).
- Role-based access control with fine-grained permissions.

Organizational structure

- Tenants → Schools → Grades → Classes → Students.
- Enrollment management: import via CSV, UI, or SIS integration (optional).
- Academic year terms and test windows.

Question bank and content management

- Question types:
  - Text (MCQ, single/multiple select, short answer).
  - Audio (playback prompt).
  - Video (prompt, silent or narrated).
  - Image-based (pictorial stimuli).
  - Drag-and-drop interactions (labeling, ordering, matching).
  - Digital ink/freehand input (mouse/e-pen) with stroke capture; image export; optional handwriting recognition.
- Metadata and tagging:
  - Domain (Numeracy, Verbal, Logical, Visual, etc.), sub-skill, grade/age band, language, curriculum mapping.
  - Difficulty level, discrimination/IRT parameters (optional).
  - Time estimate, stimulus complexity, accessibility attributes (alt text, captions).
- Authoring tools:
  - WYSIWYG editor, media upload, drag-and-drop builder, ink canvas, validation rules.
  - Versioning, status workflow (draft → review → published), review comments.
  - Item preview and test-player preview.
- Media handling:
  - Upload, transcode, and store audio/video/images; image compression; streaming support.
  - DRM/light asset protection; signed URLs; CDN delivery.

Assessment and test management

- Test blueprints:
  - Define domain coverage, sub-skill distribution, difficulty mix, total items, time limits.
  - Fixed-form and adaptive routing (optional phased rollout).
- Test assembly:
  - Randomization by section/pool, shuffling of options.
  - Accommodations by profile (extended time, larger fonts, color contrast).
- Scheduling and assignment:
  - Assign to classes/groups/individuals with open/close windows.
  - Retake policies, proctoring controls, make-up sessions.
- Delivery/session management:
  - Secure test player, full-screen mode, minimal navigation, autosave every N seconds.
  - Timer, pause/resume rules, offline resilience (local cache with background sync).
  - Integrity controls: copy/paste prevention, tab switch detection (soft), item exposure control.

Student test player interactions

- Multi-modal rendering: text, audio/video playback, images.
- Interactions:
  - MCQ single/multi select; short answer; numeric entry with keypad.
  - Drag-and-drop labels, reorder lists, matching pairs; snapping and validation.
  - Ink input: capture strokes as vectors and PNG; eraser, undo/redo; pressure support when available.
- Accessibility:
  - Keyboard-only navigation, screen reader labels, alt text, captions/transcripts for media.
  - Adjustable font size, color themes, high-contrast mode; toggle animations.
  - Audio speed control and replay limits (configurable).
- Test navigation:
  - One item per screen or paginated; review flagging; optional back-navigation rules.

Scoring and analytics

- Scoring models:
  - Objective items: key-based scoring, partial credit (e.g., multi-select, drag-and-drop).
  - Constructed responses:
    - Numeric tolerance ranges.

        - Rubrics for text or process-based tasks (teacher-graded or auto rules).
        - Ink input scoring: rubric-based; optional OCR/handwriting recognition for numeric/text detection.
    - Normalization:
        - Raw score → scaled score per domain.
        - Percentiles by grade/age bands.
        - Proficiency levels (e.g., Below Basic, Basic, Proficient, Advanced) with cut-scores.
    - Optional psychometrics:
        - Item calibration, classical stats (p-value, discrimination).
        - IRT scoring with $ \theta $ estimation after pilot data is available.
- Result aggregation:
    - Student-level domain and sub-skill scores, growth over time.
    - Class, grade, school, and tenant aggregates with filters and drill-down.
    - Cohort comparisons and trend analysis.

Recommendations and training plans

- Rules engine mapping score bands to interventions/training modules per domain/sub-skill.
- Auto-assignment of practice sets; teacher approval workflow.
- Link to internal training content or external LTI tools.
- Track completion and measure post-intervention growth.

Dashboards and reporting

- Cumulative dashboard:
    - Domain-level distributions, proficiency breakdowns, heatmaps of sub-skills, item statistics.
- Grade- and age-wise dashboards:
    - Filters for grade, age ranges, class, teacher, test windows.
- Teacher dashboard:
    - Upcoming tests, completion rates, outliers, suggested interventions.
- Student dashboard:
    - Personal progress, strengths/weaknesses, recommended practice, badges (optional).
- Exports:
    - CSV/PDF for class rosters, results, item analysis; scheduled email exports (optional).
- Data governance:
    - Field-level masking for PII; role-based access to identifiable reports.

Notifications and communications

- In-app notifications for assignments, due dates, results available.
- Optional email/SMS push with consent and rate limits.

Audit and compliance

- Audit logs of user actions, item edits, test status changes, score overrides.
- Data retention policies; right to access/delete (as applicable).

Data model (high level)

- Tenant, School, Grade, Class, User, Role, Permission.
- StudentProfile (DOB, grade, accommodations, guardian links).
- Assessment, TestForm, Section, Item, ItemVersion, Stimulus, MediaAsset.
- Assignment, TestSession, Response, Score, Rubric.
- Domain, SubSkill, Difficulty, ProficiencyLevel, CutScore.
- TrainingModule, Recommendation, InterventionAssignment, CompletionRecord.
- AnalyticsAggregate (materialized summaries).
- AuditLog, Notification, IntegrationConfig.

API design (representative)

- Auth: POST /auth/login, POST /auth/refresh, POST /auth/logout.
- Users: GET/POST/PATCH/DELETE /users; bulk import; role assignment; enrollments.
- Orgs: /tenants, /schools, /grades, /classes, roster endpoints.
- Items: CRUD /items; /items/{id}/versions; media upload; preview; search with filters.
- Assessments: CRUD /assessments; /assessments/{id}/forms; publish/unpublish.
- Assignments: POST /assignments; GET by class/student; status tracking.
- Sessions: POST /sessions (start); PATCH /sessions/{id} (pause/resume/submit); autosave responses.
- Responses: POST /sessions/{id}/responses; batch save; file uploads for ink artifacts.
- Scoring: POST /sessions/{id}/score; async scoring jobs; GET /scores.
- Analytics: /analytics/cumulative, /analytics/grade, /analytics/age, /analytics/student/{id}.
- Recommendations: GET /students/{id}/recommendations; POST override.
- Reports: /reports/export; signed download links.
- Admin: /settings, /proficiency-levels, /cut-scores, /accommodations, /feature-flags.
- Webhooks: result posted, session completed (optional).

Non-functional requirements

Security and privacy

- Encryption in transit (TLS 1.2+) and at rest.
- Secrets management; signed URLs for media; CSRF/XSS/Clickjacking protections.
- Principle of least privilege; scoped API tokens; rate limiting and IP allowlists (admin).
- Privacy compliance for minors: FERPA/COPPA equivalents, GDPR/DPDP as applicable.
- Data residency configuration (per tenant, if required).

Performance and scalability

- Target concurrency: define expected peak active test-takers.
- Response times: p95 under 300 ms for standard APIs; p95 under 1s for analytics queries (cached).
- Horizontal scalability for API, WebSocket autosave channel, and scoring workers.
- CDN for static assets and media.

Availability and reliability

- 99.9% uptime target; health checks; zero-downtime deploys.
- Autosave every 10–15 seconds; local cache for network blips; resume on reconnection.
- Backup/restore; disaster recovery RTO/RPO targets.

Accessibility

- WCAG 2.2 AA compliance.
- Screen reader support, keyboard navigation, captions/transcripts, adjustable UI.

Localization

- i18n for UI and content; right-to-left support; locale-aware number/date formats.

Observability

- Centralized logging, distributed tracing, metrics dashboards, alerting.

Browser/device support

- Latest 2 versions of Chrome/Edge/Firefox, Safari 15+.
- Desktop and tablet; optional limited mobile support for practice only.

Technology stack and architecture

Frontend

- React, TypeScript, CSS Modules/Tailwind.
- State management (Redux/RTK or React Query).
- Accessibility and media players; custom drag-and-drop and ink canvas.
- Test player as a separate shell with isolated routing and hardening.

Backend

- Python with FastAPI for APIs; Uvicorn/Gunicorn.
- PostgreSQL as primary relational DB; Redis for caching/queues.
- Celery/RQ for async scoring and analytics jobs.
- Object storage for media (S3-compatible) with CDN in front.

Services

- Auth service, Content service, Assessment service, Delivery service (sessions/responses), Scoring service, Analytics service, Recommendation service, Notification service, Admin/config service.

Integrations (optional)

- SSO via SAML/OIDC.
- LMS via LTI 1.3 for rostering and grade passback.
- Webhooks for completed sessions and result ingestion.

Scoring details and proficiency

- Raw scoring per item with partial credit rules.
- Scaling:
  - Compute domain scaled score with normalization by grade/age reference groups.
  - Percentile: $ percentile = \Phi\left(\frac{score - \mu_{group}}{\sigma_{group}}\right) $, updated periodically as norms evolve.
- Proficiency levels:
  - Cut scores per domain and grade band; configurable in admin.
- Item analytics:
  - Difficulty index, discrimination, distractor analysis; flagging of anomalous items for review.

Content quality and governance

- Content review workflow with checklists (alignment, clarity, bias, accessibility).
- Bias and fairness checks across demographics (where permissible).
- Item exposure tracking; retired items; A/B forms to minimize leakage.

Testing and QA

- Unit, integration, contract tests; frontend component tests.
- Load testing of autosave and concurrent session flows.
- Accessibility audits and keyboard-only test runs.
- UAT with teachers and pilot classrooms; telemetry-driven refinements.

DevOps and environments

- Environments: Dev, Staging, Prod; feature flags for gradual rollouts.
- CI/CD pipelines with automated testing and security scans.
- Infrastructure as code; blue/green or canary deployments.

Data migration and seeding

- Seed initial domains, sub-skills, sample questions, proficiency levels, and cut scores.
- Import tools for CSV/JSON item banks and rosters.

Project plan (indicative)

- Phase 0: Discovery and design (3–4 weeks)
  - Detailed requirements, UX research, wireframes, content schema, test blueprint.
- Phase 1: Foundations (6–8 weeks)
  - Auth/RBAC, org structure, question bank CRUD, media service, item editor v1.
- Phase 2: Delivery and scoring (6–8 weeks)
  - Test assembly/assignment, test player with core item types, autosave, scoring engine v1.
- Phase 3: Analytics and dashboards (4–6 weeks)
  - Dashboards (cumulative, grade/age/student), exports, recommendation engine v1.
- Phase 4: Accessibility, polish, and pilots (4–6 weeks)
  - WCAG hardening, proctoring basics, performance tuning, UAT, pilot launch.
- Phase 5: Scale and enhancements (ongoing)
  - Adaptive testing v2, IRT, integrations, mobile/tablet optimizations.

Risks and mitigations

- Content complexity and editor scope: prioritize item types by impact; staged rollout.
- Performance during peak testing: load testing with realistic media; CDN and autosave tuning.
- Accessibility regressions: build a11y checks into CI and manual audits.
- Data privacy for minors: parental consent flows, strict PII handling, least-privilege access.

Acceptance criteria (samples)

- A teacher can create a test aligned to a blueprint and assign it to a class with a time window.
- A student can complete a mixed-modality test with autosave and resume after a network drop.
- Scores per domain are computed with partial credit and displayed in teacher and student dashboards within 2 minutes of submission.
- Admin can define proficiency cut scores and see updated level distributions immediately.
- All media renders with captions/alt text and passes WCAG keyboard navigation checks.

Would you like this to target fixed-form in v1 and phase adaptive testing later, or include basic adaptive routing from the start?
Do you prefer FastAPI-only or FastAPI plus Django Admin for rapid internal content operations?
Should we include audio/voice responses and auto-scoring for verbal items, or limit audio/video to prompts only in v1?
What are your peak concurrent test-taker and media size expectations to set performance budgets?
Which compliance regimes apply in your deployment regions (e.g., DPDP, GDPR, FERPA/COPPA), and do you require data residency controls?