## horizontal line



ReCode Docs

Basic Processes and Data Flow

Each process slice has:

1. Description of the process
2. Core data involved
3. Data flow of process
4. DB entities and key attributes

### 1. Core User & Access Management

#### **1.1 User Authentication & Roles**

Process

* User logs in via JWT/Supabase Auth
* Roles assigned at account registration (student/lecturer/admin)
* Students cannot upload slides- lecturers can
* Role-based authorisation controls features

Data Needed

* email, password (login)
* email, password, full\_name, role, module\_code (registration)
* User profile information (participation mark, ELO points)
* hashed\_password, role permissions
* Supabase Auth JWT

DB Entities & Attributes

* USER(user\_id, name, email, hashed\_password, role, profile\_img\_url, elo\_points, participation\_mark, border\_tier)
* USER\_ROLE(role\_id, role\_name, permissions) (if seperate)

Data Flow

* User submits credentials → system validates → role info retrieved/confirmed → session established → UI shows allowed actions

#### **1.2 Enrolment & Access Control**

Process

* Lecturers manage course instances
* Manage module enrolment via codes or invites
* Enforce access to challenges; revoke when needed

Data Needed

* Invite status, enrolment role, validity windows

Dataflow

* User presents invite code/join request → system validates invite → enrolment granted or denied → access updated

DB Entities & Attributes

* ENROLMENT(enrolment\_id, user\_id, module\_id, role\_in\_module, enrolled\_at, revoked\_at)
* INVITE(invite\_id, module\_id, code, expires\_at, max\_uses, uses)

#### **1.3 Admin Console & Moderation**

Process

* Admins manage users, roles, modules
* Handle plagiarism/abuse flags, force recalculations, restore archived items

Data Needed

* Role assignments, moderation actions

Dataflow

* Admin performs action → system applies change → record written to audit logs → affected parties notified as needed

DB Entities & Attributes

* USER\_ROLE(user\_id, role)
* MOD\_ACTION(action\_id, moderator\_id, entity\_type, entity\_id, reason, performed\_at)

### 2. Content Creation & Management

#### **2.1 Slide Upload & Topic Generation**

Process

* Lecturer uploads .pptx, .pdf, .txt slides
* Parse via Python-pptx/OCR → extract text
* NLP suggests topics (spaCy + HuggingFace **TBD**)
* Lecturer can add/edit/remove topics
* Adding/removing topics updates question pools
* If parsing fails, notify lecturer with error and provide UI to manually select/type in relevant topics
* Record parse failure logs

Data Needed

* Slide file metadata
* Lecturer-modified topic list (topic\_name, importance\_weight, status)
* Auto-generated topic list parsed text content
* Topic selection (if fallback)

Dataflow

* Lecturer uploads file → parsing service extracts text → NLP suggests topics → lecturer reviews/edits → final topics saved and used to suggest questions

DB Entities & Attributes

* SLIDE\_STACK(stack\_id, lecturer\_id, module\_id, file\_url, upload\_date)
* SLIDE\_TEXT(text\_id, stack\_id, slide\_number, parsed\_text)
* TOPIC(topic\_id, stack\_id, topic\_name, is\_auto\_generated, importance\_weight)

#### **2.2 Topic Curation & Weighting**

Process

* Lecturer reviews auto-generated topics
* Edits, merges/splits topics, sets importance\_weight
* Weight influences question suggestion order and challenge composition

Data Needed

* Topic metadata
* Mapping from topics → questions
* Importance weights

Dataflow

* Lecturer adjusts topics → system updates weight/status → suggestion engine reorders question suggestions

DB Entities & Attributes

* TOPIC(topic\_id, stack\_id, topic\_name, is\_auto\_generated, importance\_weight, status)
* TOPIC\_MERGE\_LOG(op\_id, source\_topic\_ids[], target\_topic\_id, performed\_by, performed\_at)

#### **2.3 Question Bank Management**

Process

* CRUD operations on questions
* Tag by topic, difficulty, language/runtime
* Attach hints, canonical solutions, atomic tests

Data Needed

* Question text
* I/O tests
* difficulty tags
* hints
* Language

Dataflow

* Author/automated pipeline creates or edits question → system validates tests (optional) → question saved → available for challenge selection

DB Entities & Attributes

* QUESTION(question\_id, topic\_id, difficulty, language, question\_text, starter\_code, canonical\_solution, atomic\_tests\_json, is\_active)
* QUESTION\_TAG(question\_id, tag)
* HINT(hint\_id, question\_id, hint\_text, order\_index)

#### **2.4 Selecting challenges**

Process

* Lecturer selects exactly 5 bronze, 3 silver, 2 gold per challenge
* 1 challenge = 1 week’s work (e.g., “Recursion”)
* Challenges linked to topics and stored for students

Data Needed

* Topic ID mapping to questions
* Question difficulty classification (bronze/silver/gold)
* Challenge metadata (week number, topic name)

Dataflow

* Lecturer picks questions → system validates distribution (5/3/2) → challenge saved as draft/published

DB Entities & Attributes

* QUESTION: question\_id, topic\_id, difficulty (enum), question\_text, correct\_answer, hint\_list
* CHALLENGE: challenge\_id, topic\_id, week\_number, ruby\_unlock (bool), emerald\_unlock (bool)
* CHALLENGE\_QUESTION: challenge\_id, question\_id

#### **2.5 Challenge Scheduling & Publishing**

Process

* Lecturer can assemble a challenge per week (5 bronze, 3 silver,2 gold)
* Set open/close windows
* Save draft and then publishes

Data Needed

* Challenge metadata
* Windows
* Visibility

Dataflow

* Lecturer schedules/publishes → system sets state (draft → published) → notifications triggered on publish

DB Entities & Attributes

* CHALLENGE(challenge\_id, module\_id, topic\_id, title, week\_number, open\_at, close\_at, state)
* CHALLENGE\_QUESTION(challenge\_id, question\_id, order\_index)

#### **2.6 Question Suggestion**

Process

* NLP analyzes parsed slide text and computes topic weights
* System queries the question bank using topic tags + weights, returning ranked suggestions (tag-based lookup + ranking)
* Lecturer reviews ranked suggestions and selects questions for the challenge

Data Needed

* Topic text & weights from slides
* Question bank entries and tags (difficulty, language, topic tags)
* Lecturer selection / approval

Dataflow

* NLP extracts/weights topics → system queries question bank for matching tags → returns ranked suggestions to lecturer → lecturer selects questions → selections saved to challenge

DB Entities & Attributes

* GEN\_JOB: job\_id, stack\_id, topic\_id, status, started\_at, finished\_at, error
* GEN\_CANDIDATE: candidate\_id, job\_id, , scores, decision (approved/rejected), decided\_by, decided\_at

**2.7** **Question Validation Pipeline**

* Proposed questions are validated for:
  + Novelty: Avoids duplicates in the module.
  + Difficulty: Matches bronze/silver/gold criteria.
  + Atomicity: Ensures test cases are isolated/valid.
* Validation uses a service (e.g., n8n) to score each question.

Data Needed

Generation inputs (topic text), candidate items, validation scores

### 3. Assessment & Gameplay Loop

#### **3.1 Challenge Attempt & Judge0 Execution**

Process

* Students submit via embedded editor (CodeMirror)
* Code sent to Judge0 API for evaluation
* Atomicity check: correct/incorrect result only
* Max 3 resubmissions & max 3 hints per challenge
* Elo deductions apply for hints/time/resubmissions

Data Needed

* Submission code, time taken, number of resubmissions, hints used
* Judge0 execution result
* Elo calculation inputs

Dataflow

* Student writes code → submits for test/submit → code is sent to Judge0 -> Judge0 runs tests → result returned → system records submission, updates resubmission/hint counters, recalculates Elo/badges/progress

DB Entities & Attributes

* SUBMISSION(submission\_id, user\_id, challenge\_id, question\_id, submitted\_code, time\_taken, hints\_used, resubmissions\_count, result\_status (pass/fail), submitted\_at)

#### **3.2 Executing and marking code with Judge0**

##### **3.2.1 Scenario 1 – Using Judge0 API, No Docker (Goal is to minimize API calls to stay within rate limits)**

###### **3.2.1.1 Single Test Case Execution & Marking**

Process

* Test Code button: Send student’s code and single test input to Judge0 execute endpoint
* Judge0 runs code once and returns output
* Backend temporarily stores output linked to user/session
* Backend generates and stores hash of submitted code at test time
* Submit button: Backend hashes current student code again and compares with stored hash
  + If hash matches → use cached output for marking by comparing to expected output in DB
  + If hash differs → force student to test again before submitting

Data Needed

* Student source code and language
* Single test input
* Output from Judge0 execution
* Stored hash of tested code
* Expected output from question database

DB Entities & Attributes

###### 3.2.1.2 Multiple Test Case Execution & Marking with Wrapper

Process

* Test Code button: Backend fetches all test inputs and expected outputs for the question from DB
* Backend dynamically builds a wrapper program that:
  + Includes student’s code
  + Loops over all test inputs, runs student code for each input, collects outputs in a list
  + Prints outputs as a JSON string
* Backend sends wrapper program to Judge0 for single execution covering all test cases
* Judge0 returns combined JSON output with all test case results
* Backend parses output and temporarily stores it linked to user/session
* Backend generates and stores hash of submitted code at test time
* Submit button: Backend hashes current student code and compares to stored hash
  + If hash matches → compare stored outputs to expected outputs for marking
  + If hash differs → force retest before submitting

Data Needed

* Student source code and language
* List of all test inputs and expected outputs for the question
* JSON output from Judge0 wrapper execution
* Stored hash of tested code

DB Entities & Attributes

##### **3.2.2 Scenario 2 – Using local dockerized Judge0**

Process

* Pull and run Judge0 Docker image
* Backend will connect to local Judge0 API
* Code submissions sent to local Judge0 container for execution
* Wrappers will be used for cases that need to be tested against multiple test cases for efficiency
* Judge0 executes code securely in isolated containers
* Results returned instantly to backend without external API limits
* Backend handles parsing of execution results and passes feedback to frontend

Data Needed

* Code submission (source code, language, stdin)
* Judge0 Docker container endpoint URL and port
* Execution results (stdout, stderr, execution status)

#### **3.3 Hints & Resubmission Enforcement**

Process

* Max 3 hints per challenge
* Hints stored in question bank or generated (Hugging Face)
  + Each successive hint can become more specific
* Penalties applied to Elo; when limits reached, further hints/resubmissions are blocked

Data Needed

* Hint content
* Hint usage count per student per challenge

Dataflow

* Student requests a hint → check remaining hints → deliver hint (pre-authored or generated) → update usage count → apply penalty calculation

DB Entities & Attributes

* HINT(hint\_id, question\_id, hint\_text, order\_index)
* HINT\_USAGE(user\_id, challenge\_id, hint\_id, used\_at)

#### **3.4 Time Tracking & Deadlines**

Process

* Track time taken for Elo penalties and analytics
* Enforce open/close windows
* Late penalties optional (soft/hard deadlines)

Data Needed

* Timestamps, durations, deadline policies

Dataflow

* Student starts challenge → timer recorded → submission occurs → system computes duration and applies late penalties if applicable

DB Entities & Attributes

* USER\_CHALLENGE\_TIMER(user\_id, challenge\_id, started\_at, submitted\_at, duration\_seconds)
* DEADLINE\_POLICY(module\_id, late\_window\_minutes, late\_penalty\_percent)

#### **3.5 Paste/Plagiarism & Editor Telemetry**

Process

* Monitor editor for paste events (keyboard shortcuts, right-click paste, drag/drop).
* Log all paste actions with metadata (timestamp, source if available, code snippet length).
* If timestamp is >30s when submitted, mark as suspicious submission for review
* Surface warnings

Data Needed

* Paste flags, similarity scores, evidence

Dataflow

* Monitor editor for paste events → Log paste data (timestamp, source, snippet length) → Compare paste time to submission time → If >30s → Mark as suspicious → Surface warning

DB Entities & Attributes

* EDITOR\_EVENT(event\_id, user\_id, challenge\_id, question\_id, event\_type, occurred\_at, payload\_json)
* PLAGIARISM\_CHECK(check\_id, submission\_id, similarity\_score, matched\_user\_ids[], status)

### 4. Progression & Rewards

#### **4.1 Badge & Participation System**

Process

* Badge weights: Bronze = 6%, Silver = 10%, Gold = 20%
* Ruby/Emerald badges = 50% participation + 50% badge weighting
* Continuous recalculation
* Badges earned are stored and participation recalculated continuously

Data Needed

* Badge type and date earned
* Weight per badge type
* Calculated participation mark

Dataflow

* Challenge results → badges assigned → participation recalculated continuously → profile/leaderboards updated

DB Entities & Attributes

* USER\_BADGE(user\_id, challenge\_id, badge\_type, date\_awarded)
* BADGE\_TYPE(badge\_type, weight\_percent)
* USER\_PARTICIPATION(user\_id, module\_id, current\_percent)

**4.2 Eligibility Engine (Ruby/Emerald Unlocks)**

Process

* System computes eligibility of:
  + Ruby challenge unlock: after every 2 published challenges, participation ≥ 50%
  + Emerald challenge unlock: ≥ 2/4 Rubies passed, participation ≥ 50%

Data Needed

* User participation mark (%)
* Nr of Ruby challenges passed timeline

Dataflow

* When challenges are completed → eligibility engine evaluates rules → eligibility flags updated → students informed when unlocks occur

DB Entities & Attributes

* ELIGIBILITY(user\_id, module\_id, ruby\_available\_from\_week, emerald\_available\_from\_week, is\_ruby\_eligible, is\_emerald\_eligible, last\_evaluated\_at)

#### **4.3 Elo, Titles**

Process

* Award Elo after each challenge completed
* Elo gained/lost based on difficulty, speed, hints, resubmissions
* Titles unlocked at Elo thresholds

Data Needed

* Base gains, penalties, elo points gained in challenge, thresholds

Dataflow

* Submission graded → Elo for challenge computed → USER\_ELO updated → if threshold passed, TITLE awarded → leaderboards updated

DB Entities & Attributes

* USER\_ELO(user\_id, elo\_points, last\_updated)
* ELO\_EVENT(event\_id, user\_id, module\_id, challenge\_id, base\_points, penalties\_json, net\_points, created\_at)
* TITLE(title\_id, title\_name, elo\_threshold, order\_index)

#### **4.4 Leaderboards & Cohort Comparisons**

Process

* Computed leaderboards by Elo, badges, completion speed
* Per-module and global
* Weekly snapshots

Data Needed

* Aggregates per user/module/week

Dataflow

* Aggregate stats → compute rankings → snapshot weekly → frontend fetches snapshots for display

DB Entities & Attributes

* LEADERBOARD\_SNAPSHOT(snapshot\_id, module\_id, week\_number, rankings\_json, generated\_at)

#### **4.5 Semester Finalisation & Border Assignment**

Process

* At end of season freeze participation, award final titles, set profile border(bronze/silver/gold/ruby/emerald/diamond), archive challenges

Data Needed

* Final participation mark(%)
* final Elo/title
* border tier

Dataflow

* Admin triggers finalisation (or scheduled) → system computes final results → set border and titles → lock/freeze semester data and archive challenges

DB Entities & Attributes

* SEMESTER(semester\_id, season\_name, start\_date, end\_date, state)
* USER\_SEMESTER\_RESULT(user\_id, semester\_id, final\_participation, final\_elo, final\_title, final\_border\_tier)

#### **4.6 Profile & Borders**

Process

* Profile shows Elo, badges, participation mark, titles
* Profile border colour set by final badge tier at semester end
* Profile shows title obtained
* Student & lecturer dashboards pull from same source but display differently

Data Needed

* Profile border tier calculation
* Image URL and basic info

Dataflow

* Profile request → system gathers latest stats → render profile with border & title

DB Entities & Attributes

* USER (already holds border\_tier, profile\_img\_url)
* USER\_ELO()

### 5. Analytics, Reporting & Notifications

#### **5.0 Participation Mark Calculator**

Process

* Continuously recomputes participation mark from badge mix per challenge:
  + Bronze 6%, Silver 10%, Gold 20% (weighted by earned counts per challenge)
  + Ruby/Emerald each contribute a 50/50 split: 50% participation + 50% respective badge outcome

Data Needed

* Per-challenge badge outcomes, configured weights, user totals

Dataflow

* When badges change → recalclate participation → store & display updated percentage

DB Entities & Attributes

* BADGE\_TYPE: badge\_type (bronze/silver/gold/ruby/emerald), weight\_percent
* USER\_PARTICIPATION: user\_id, module\_id, current\_percent, calc\_breakdown(jsonb), last\_recalculated\_at

#### **5.1 Progress Tracking & Analytics**

Process

* Store weekly completion progress per student
* Cohort analytics for lecturers - graphs, participation averages, elo trends
* Students see personal dashboard

Data Needed

* Challenge completion status per user
* Badge counts per user
* Participation history
* ELO Points per user

Dataflow

* Student completes challenge → event logged → aggregates updated → dashboards refreshed

DB Entities & Attributes

* USER\_PROGRESS(user\_id, challenge\_id, completion\_status, score\_percent, date\_completed)
* LECTURER\_ANALYTICS\_EXPORT (on-demand)

#### **5.2 Data Export & Reporting**

Process

* Export participation, badges, Elo, completion to CSV/XLSX
* Async job with download URL

Dataflow

* Export request → background job builds file → store file and return download link → optional notification when ready

DB Entities & Attributes

* EXPORT\_JOB(job\_id, module\_id, requested\_by, params\_json, status, file\_url, requested\_at, completed\_at)

#### **5.3 Notifications & Reminders**

Process

* Notify students on new challenges, unlocks (Ruby/Emerald), deadlines, announcements

Data Needed

* User preferences, trigger events, delivery status

Dataflow

* Event occurs → notification composed → deliver via email

DB Entities & Attributes

* NOTIFICATION(notif\_id, user\_id, type, title, body, sent\_at, read\_at)
* USER\_PREF(user\_id, email\_opt\_in, push\_opt\_in, digest\_frequency)

#### **5.4 Audit Logging & Event Trail**

Process

* Record critical actions for accountability (topic edits, challenge publishes, grade recalcs, role changes)

Data Needed

* Actor, action, target entity, before/after snapshots

Dataflow

* Action performed → audit entry recorded → available for admin review

DB Entities & Attributes

* AUDIT\_LOG(log\_id, actor\_id, action, entity\_type, entity\_id, before\_json, after\_json, occurred\_at, ip)

### 6. Storage & Infrastructure

#### **6.1 Files & Storage Management**

Process

* Store slides and generated exports
* Signed URLs
* Cleanup old assets

Data Needed

* File metadata, retention windows

Dataflow

* User uploads file → storage stores file and metadata → signed access provided for retrieval → scheduled cleanup runs for expired assets

DB Entities & Attributes

* FILE\_ASSET(file\_id, owner\_id, module\_id, type, file\_url, size\_bytes, checksum, created\_at, expires\_at)