

ReCode Developer Manual

Technical Overview, Architecture, and Pseudocode

Version 1.0 • Author: Brandon van Vuuren • Scope: MVP → Phase 2

Table of Contents

Overview:	4
1. Executive Summary	4
Figure 1 — System Context (C4 L1)	4
2. System Overview	5
2.1 Actors & Roles	5
2.2 Seasonality & Content.....	5
3. Authentication & Authorization (MVP).....	5
Key guarantees:.....	5
4. Core Data Model (MVP scope).....	5
5. End-to-End Flow (Sequence)	6
Figure 2 — End-to-End Sequence (Login→Publish→Attempt→Grade).....	7
Narrative:	8
6. Ingestion & Generation Pipeline	8
Figure 3 — Slide→Topic→Challenge Pipeline	8
Stages:.....	9
7. Challenge Evaluation & Submissions	9
Figure 4 — Challenge Evaluation & Submission Flow.....	9
Key steps:	10
8. API Layer (Selected MVP Endpoints).....	10
Auth.....	10
Admin	10
Lecturer	10
Student.....	10
9. High-Level Pseudocode (End-to-End).....	11
9.1 Auth & Session.....	11
9.2 Admin Provisioning	11
9.3 Lecturer Upload → Extraction → Topics → AI Proposals → Curation → Publish	12
9.4 Student Attempt → Execute (Sample) → Submit (All Tests) → Award/ELO	13
9.5 Awarding & Progress.....	14
10. Deployment & Infrastructure	14
Figure 5 — Deployment Architecture (AWS/Supabase)	15
11. EC2 Setup & Docker (Judge0/Redis)	16
Figure 6 — EC2 + Docker Setup (Judge0/Redis/Bedrock)	16
Summary:	16
12. RPCs (Supabase SQL).....	16

13. Error Handling & Resilience.....	17
14. Frontend → Backend Contract (MVP).....	17
15. Non-Functional Requirements (MVP Targets)	17
16. Future Enhancements (Phase 2+).....	17
17. Appendix A — Pseudocode (Consolidated)	18
18. Appendix B — Diagrams (All)	18
19. Appendix C — Minimal API Examples (FastAPI)	18
20. Appendix D — RLS Policy Notes	19
21. Appendix E — Ops & Health	19
22. Sign-off	19

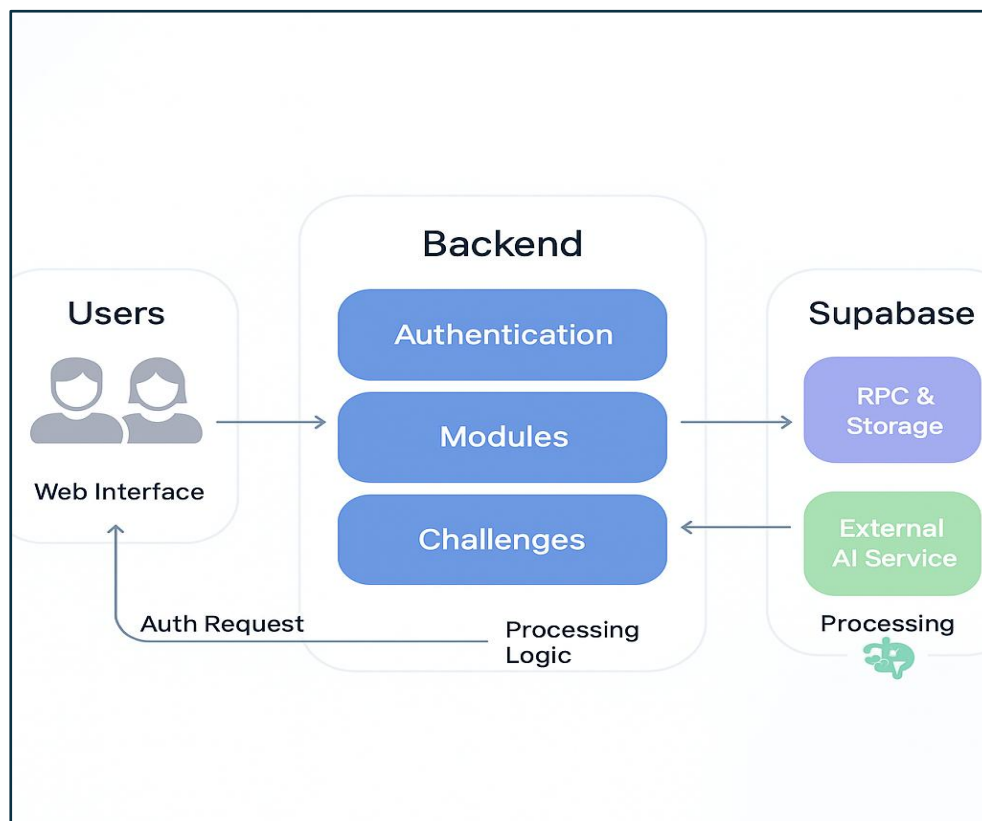
Overview:

- 🚩 **Project:** RECODE (AI-assisted gamified coding education)
- 🚩 **Stack:** FastAPI • Supabase (Auth/DB/Storage/RPC) • spaCy • AWS Bedrock (Claude) • Judge0 (EC2 Docker) • Celery • Redis • Gunicorn/Uvicorn
- 🚩 **Security:** Cookies (HttpOnly) • JWT • Supabase RLS • Role-based access (student/lecturer/admin)

1. Executive Summary

RECODE transforms lecturer slides into interactive programming challenges with automated grading. Flow: upload slides → extract text → detect topics (spaCy) → generate challenge proposals/tests (Claude via Bedrock) → curate/publish → students attempt → code runs in Judge0 (EC2 Docker) → verdicts → badges/ELO/GPA → analytics. Supabase provides Auth, RLS-secured Postgres, Storage, and RPCs. Backend uses cookies for session auth, Celery/Redis for background tasks, and Gunicorn/Uvicorn for concurrency.

Figure 1 — System Context (C4 L1)



2. System Overview

2.1 Actors & Roles

- **Admin:** creates semesters/modules, course instances; assigns lecturers; governance.
- **Lecturer:** uploads slides; reviews/curates AI proposals; publishes sets; views analytics.
- **Student:** attempts challenges; executes sample/public tests; submits for full grading; earns badges/ELO.

2.2 Seasonality & Content

- **Semester = Season** (6 months).
 - **Weekly drops:** basic set (2×bronze, 2×silver, 1×gold); special single-item sets (ruby/emerald/diamond).
 - **Weights (GPA-style):** bronze 10%, silver 20%, gold 40% (specials configured per table).
-

3. Authentication & Authorization (MVP)

- **Auth:** Supabase email+password with email verification (Outlook).
- **Session:** Backend sets **HttpOnly** cookie with access JWT (SameSite=Lax, Secure in prod).
- **RLS:** Per-user data isolation; role checks at API layer.
- **Roles:** student, lecturer, admin (in profiles.role).
- **Frontend:** always sends credentials (fetch(..., { credentials: 'include' })).

Key guarantees:

- The frontend can **hide** routes for UX, but **backend always enforces** authorization (never trust the client).
 - RLS protects DB rows even if a token is stolen; API still checks roles and course membership.
-

4. Core Data Model (MVP scope)

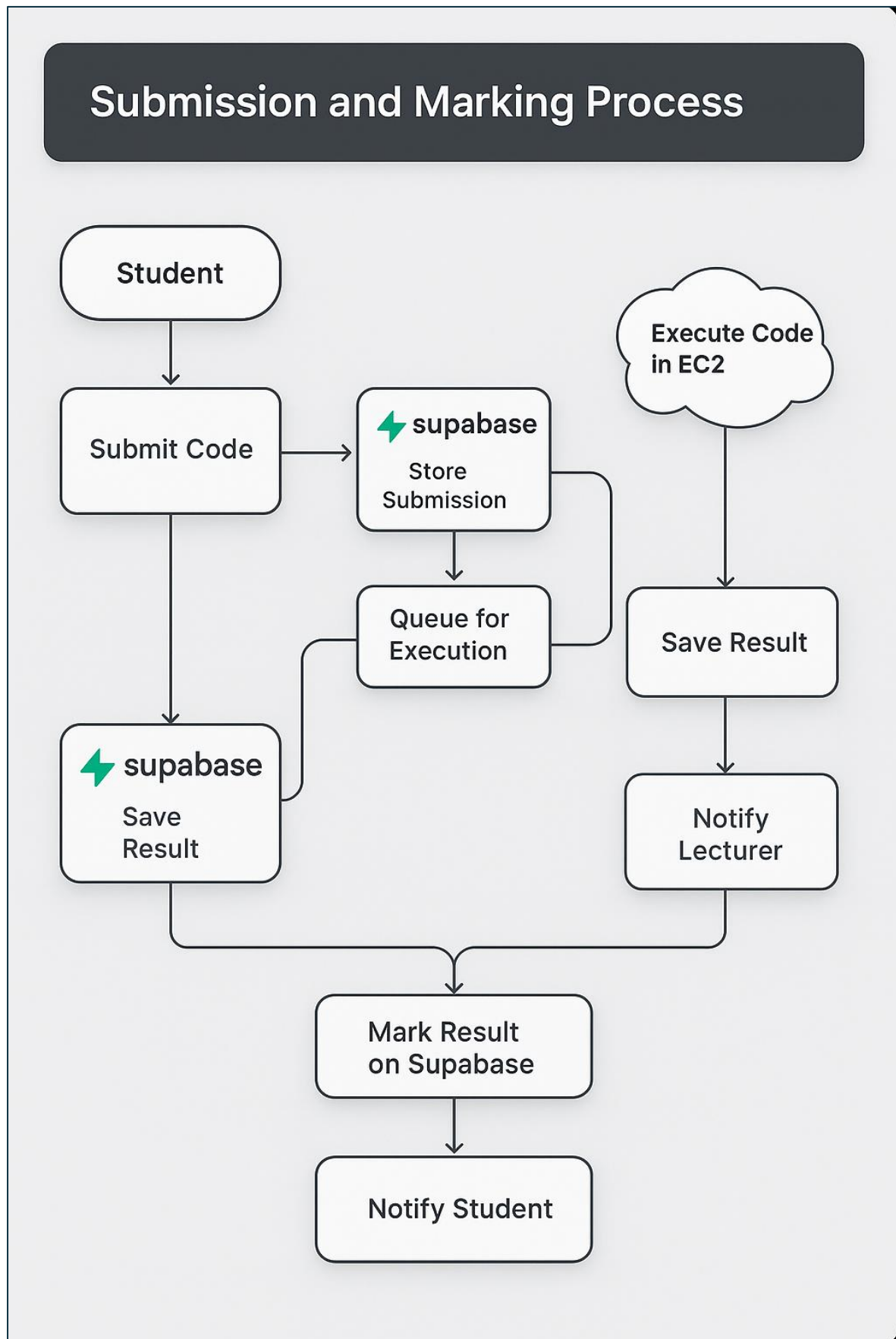
Minimal relational entities, normalized for clarity and speed:

- **profiles**(id PK, email, role, display_name, created_at)
- **semesters**(id, year, term, start_date, end_date)
- **modules**(id, code, name)
- **course_instances**(id, module_id FK, semester_id FK, title, start_date, end_date)
- **course_memberships**(id, course_instance_id FK, user_id FK, role_in_course)

- **slides**(id, course_instance_id FK, storage_path, filename, uploader_id, uploaded_at)
 - **slide_extractions**(id, slide_id FK, text, metadata JSONB, version, created_at)
 - **topics**(id, course_instance_id FK, name, week, source='spacy')
 - **challenge_sets**(id, course_instance_id FK, topic, set_kind='basic'|'special', special_type NULL|'ruby'|'emerald'|'diamond', available_from, due_date, published BOOL, published_at)
 - **challenge_set_requirements**(set_id FK, bronze_required, silver_required, gold_required, ruby_required, emerald_required, diamond_required)
 - **challenge_items**(id, set_id FK, difficulty='bronze'|'silver'|'gold'|specials, title, statement, io_format, examples JSONB, approved BOOL)
 - **challenge_item_tests**(id, item_id FK, input, expected_output, visibility='public'|'hidden', weight)
 - **attempts**(id, item_id FK, set_id FK, user_id FK, language, last_code, status='in_progress'|'submitted'|'graded', updated_at)
 - **submissions**(id, attempt_id FK, item_id FK, set_id FK, user_id FK, source_code, verdict, stdout, stderr, time_ms, created_at)
 - **user_badges**(user_id, item_id, set_id, badge difficulty, awarded_at)
 - **user_set_completion**(user_id, set_id, completed BOOL, completed_at)
 - **user_elo**(user_id, elo, updated_at)
 - **badge_config**(difficulty, weight_fraction)
 - **audit_log**(id, actor_id, action, target_id, meta, created_at)
 - **dead_letter**(id, kind, payload, error, created_at)
-

5. End-to-End Flow (Sequence)

Figure 2 — End-to-End Sequence (Login→Publish→Attempt→Grade)

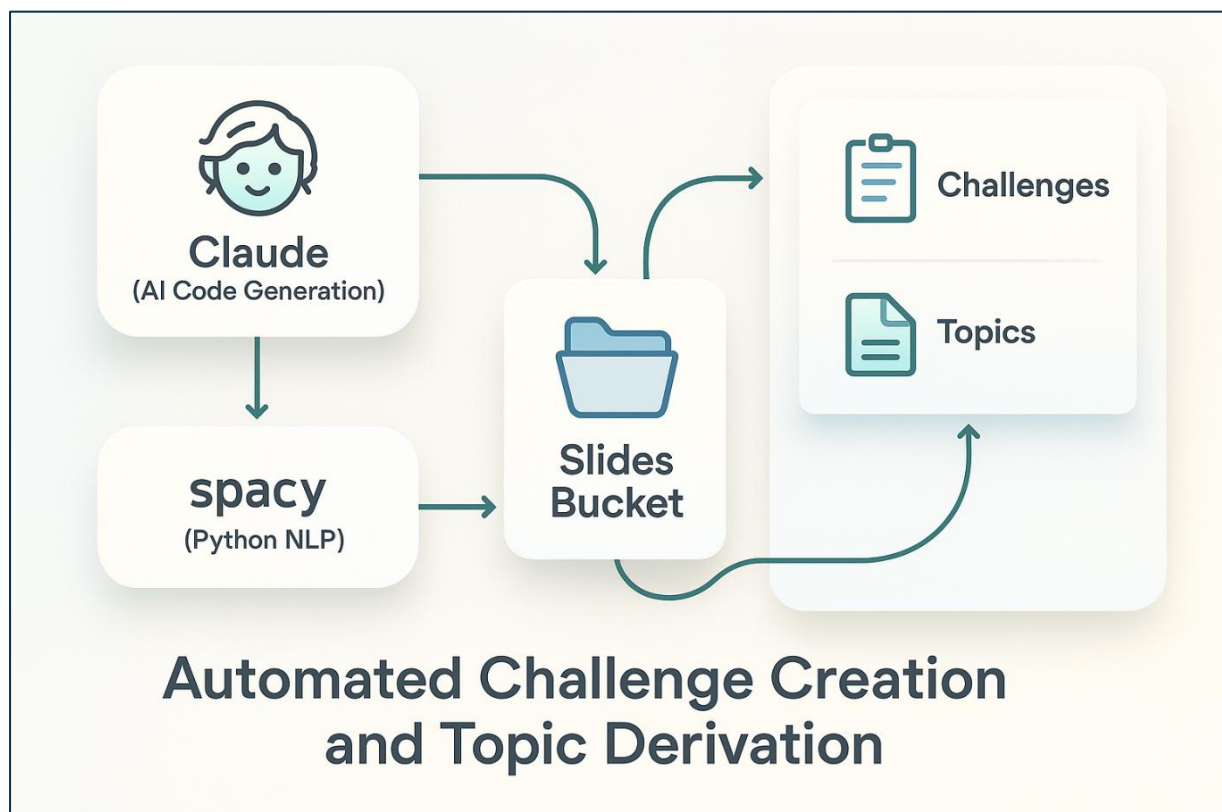


Narrative:

1. **Register/Login:** Supabase issues session; backend sets HttpOnly cookie.
 2. **Admin** creates semester/module/course instance; assigns lecturer (RPCs).
 3. **Lecturer** uploads slides → backend stores in Supabase Storage.
 4. **Extraction/NLP:** text extracted; spaCy detects topics/subtopics.
 5. **AI Proposals:** Claude generates challenge items & tests.
 6. **Curation/Publish:** lecturer approves; set published for date window.
 7. **Student** attempts item: “Run” (public test) → “Submit” (all tests).
 8. **Judge0** executes code; backend records submission; awards badge; recalculates set completion; updates ELO; logs audit.
-

6. Ingestion & Generation Pipeline

Figure 3 — Slide→Topic→Challenge Pipeline

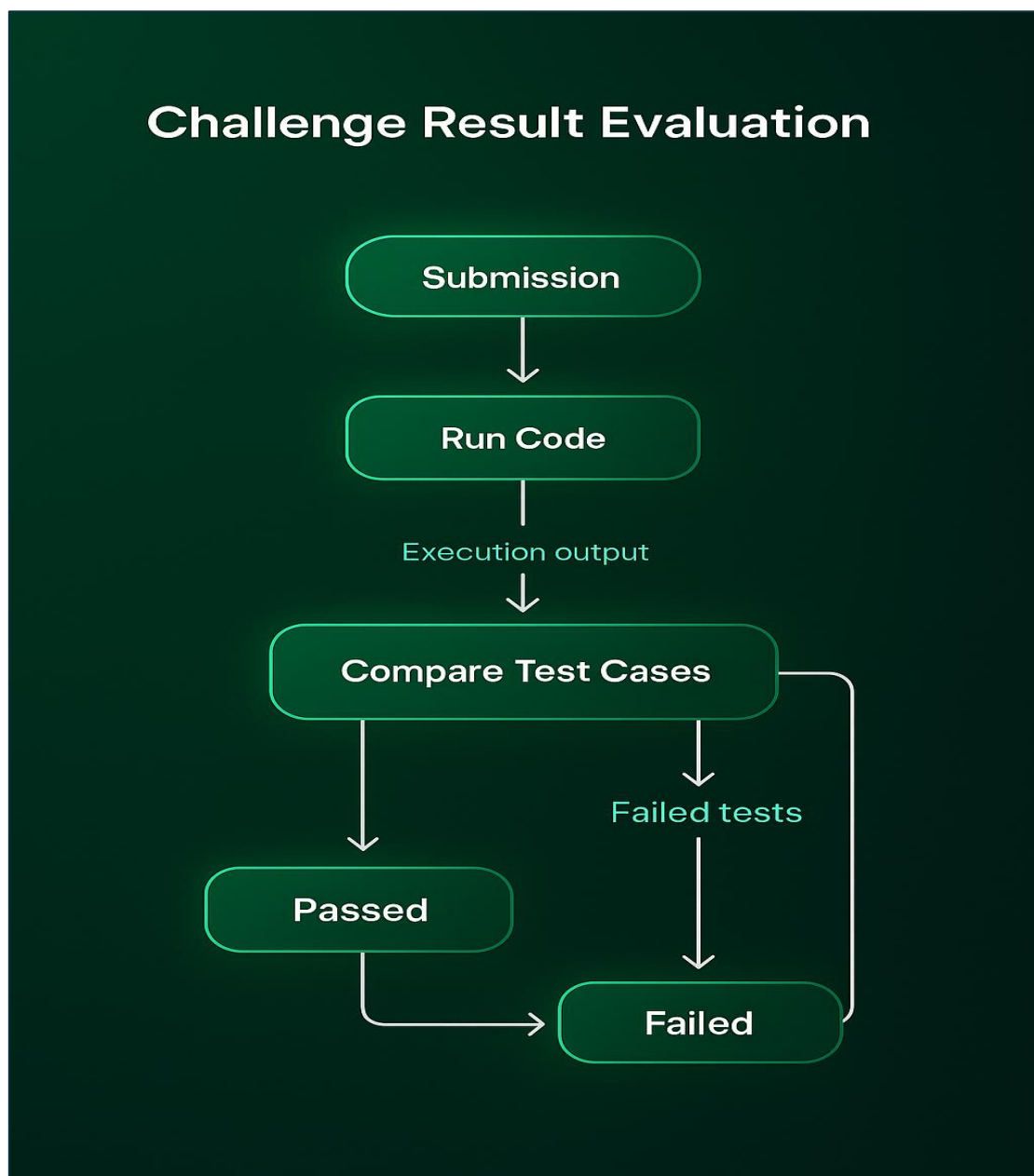


Stages:

- **Upload** (Lecturer) → **Slides** (Storage) → **Extraction** (python-pptx/PDF) → **NLP** (spaCy topics/subtopics) → **AI Proposal** (Bedrock Claude: 2B/2S/1G + specials) → **Curation** (approve) → **Publish** (weekly availability).
 - **Queues:** extract, nlp, gen (Celery/Redis); backoff & dead-letter for robustness.
 - **Artifacts:** slide_extractions, topics, challenge_sets, challenge_items, challenge_item_tests.
-

7. Challenge Evaluation & Submissions

Figure 4 — Challenge Evaluation & Submission Flow



Key steps:

- **Run** (sample): executes **only public** test; returns stdout/time; not graded.
 - **Submit**: executes **public + hidden** tests; compares outputs; verdict.
 - **On Accepted**: award badge (once), recompute set completion, update ELO, log audit.
 - **On Failure**: record submission; student can reattempt.
 - **Resilience**: retries/backoff to Judge0; persistent failures → dead_letter.
-

8. API Layer (Selected MVP Endpoints)

Auth

- POST /auth/register → Supabase sign-up, profile bootstrap.
- POST /auth/login → sets HttpOnly access-token cookie.
- POST /auth/refresh → rotate cookie.
- POST /auth/logout → clear cookie.

Admin

- POST /admin/semesters → RPC create_semester.
- POST /admin/modules → RPC create_module.
- POST /admin/course-instances → RPC create_course_instance.
- POST /admin/course-instances/{ci}/assign-lecturer → RPC assign_membership.

Lecturer

- POST /lecturer/slides/upload → storage + enqueue extraction.
- GET /lecturer/challenge-sets?published=false → curate queue.
- POST /lecturer/challenge-sets/{id}/approve-items → approve subset.
- POST /lecturer/challenge-sets/{id}/publish → RPC publish_set.

Student

- GET /me/weekly?course_instance_id=... → open sets.
- POST /attempts → create attempt.
- POST /execute → run public test (Judge0).
- POST /submissions → grade (all tests), award/update.

All protected (except register/login); FE always sends cookies; backend validates token/role.

9. High-Level Pseudocode (End-to-End)

9.1 Auth & Session

```
POST /auth/register(email, password):  
  
  supa_user := SUPABASE.auth.sign_up(email, password)  
  
  INSERT profiles(user_id:=supa_user.id, email, role="student")  
  
  return 204
```

```
POST /auth/login(email, password):  
  
  tokens := SUPABASE.auth.sign_in_with_password(email, password)  
  
  assert email_confirmed(tokens.user)  
  
  set_cookie("access_token", tokens.session.access_token, httpOnly, sameSite=Lax,  
secure=prod)  
  
  return { user: profile(tokens.user.id) }
```

```
POST /auth/refresh():  
  
  tokens := SUPABASE.auth.refresh_session(refresh_cookie)  
  
  set_cookie("access_token", tokens.session.access_token, ...)  
  
  return 204
```

9.2 Admin Provisioning

```
POST /admin/semesters(year, term, start_date, end_date):  
  
  RPC_create_semester(...)
```

```
POST /admin/modules(code, name):  
  
  RPC_create_module(...)
```

```
POST /admin/course-instances(module_id, semester_id, title, start, end):  
  
  RPC_create_course_instance(...)
```

```
POST /admin/course-instances/{ci}/assign-lecturer(lecturer_user_id):  
  
  RPC_assign_course_membership(ci, lecturer_user_id, role="lecturer")
```

9.3 Lecturer Upload → Extraction → Topics → AI Proposals → Curation → Publish

```
POST /lecturer/slides/upload(course_instance_id, file):

storage_key := SUPABASE.storage.put("slides", file)

slide_id := INSERT slides(course_instance_id, storage_path:=storage_key, uploader_id:=me)

enqueue task_extract_text(slide_id)

return {slide_id, status:"queued"}
```

```
task_extract_text(slide_id):

bin := SUPABASE.storage.get(slides.storage_path)

text, meta := EXTRACT_TEXT_AND_META(bin)

extraction_id := INSERT slide_extractions(slide_id, text, metadata:=meta, version:=auto)

enqueue task_nlp_topics(extraction_id)
```

```
task_nlp_topics(extraction_id):

topics := SPACY.detect_topics(extractions.text)

FOR t IN topics:

    topic_id := UPSERT topics(course_instance_id, name:=t, week:=infer_week(),
source:="spacy")

enqueue task_propose_challenges(course_instance_id, topic_ids)
```

```
task_propose_challenges(ci, topic_ids):

FOR topic_id IN topic_ids:

    set_id := INSERT challenge_sets(ci, topic:=TOPIC(topic_id), set_kind:"basic",
available_from:=next_week_start(), due_date:=next_week_end(),
published:false)

    INSERT challenge_set_requirements(set_id, bronze_required:2, silver_required:2,
gold_required:1)

    proposal := BEDROCK.CLAUDE.propose_items(topic_id)

    FOR item IN proposal.items:

        item_id := INSERT challenge_items(set_id, difficulty:=item.diff, title, statement, io_format,
examples, approved:false)

    INSERT tests: 1 public + 2 hidden
```

```
FOR st IN ["ruby","emerald","diamond"]:
    s_id := INSERT challenge_sets(ci, topic:=..., set_kind:"special", special_type:=st,
available_from:=schedule(st), published:false)

    sp := BEDROCK.CLAUDE.propose_single_item(topic_id, st)

    sp_item := INSERT challenge_items(s_id, difficulty:=st, ...)

    INSERT tests (public + hidden)
```

```
POST /lecturer/challenge-sets/{set_id}/approve-items(item_ids[]):

    UPDATE challenge_items SET approved=true WHERE id IN item_ids
```

```
POST /lecturer/challenge-sets/{set_id}/publish:

    assert requirements met

    RPC_publish_set(set_id)

    UPDATE challenge_sets SET published=true, published_at=now()
```

9.4 Student Attempt → Execute (Sample) → Submit (All Tests) → Award/ELO

```
GET /me/weekly(ci):

    return published sets for course_instance ci
```

```
POST /attempts(item_id, language, initial_code?):

    INSERT attempts(item_id, set_id:=ITEM.set_id, user_id:=me, status:"in_progress", language,
last_code:=initial_code)
```

```
POST /execute(attempt_id, code):

    visible := FIRST_PUBLIC_TEST(item_id)

    res := JUDGE0.run(language, code, visible.input) # wait=true

    return {stdout:res.stdout, passed:(trim(res.stdout)==trim(visible.expected))}
```

```
POST /submissions(attempt_id, code):

    tests := challenge_item_tests(item_id)

    all_passed := true; agg_stdout := []

    FOR t IN tests:

        r := JUDGE0.run(language, code, t.input) # wait=true
```

```
agg_stdout.push(r.stdout)

if trim(r.stdout) != trim(t.expected): all_passed=false
```

```
INSERT submissions(..., verdict := all_passed ? "Accepted":"Wrong Answer",
stdout:=join(agg_stdout))

if all_passed:

    award_badge_if_first(me, item_id, set_id)

    recompute_set_completion(me, set_id)  # compare counts vs requirements

    update_user_elo(me, item.difficulty)

UPDATE attempts SET status := all_passed ? "graded":"submitted"

return {verdict: all_passed ? "Accepted":"Wrong Answer"}
```

9.5 Awarding & Progress

```
award_badge_if_first(user, item):

    if not exists user_badges(user,item):

        INSERT user_badges(user, item, set_id, badge:=ITEM.difficulty, awarded_at:=now())

recompute_set_completion(user, set_id):

    req := challenge_set_requirements(set_id)

    counts := count_badges(user,set_id)

    if counts meet req: UPSERT user_set_completion(user,set_id, completed:true,
completed_at:now())

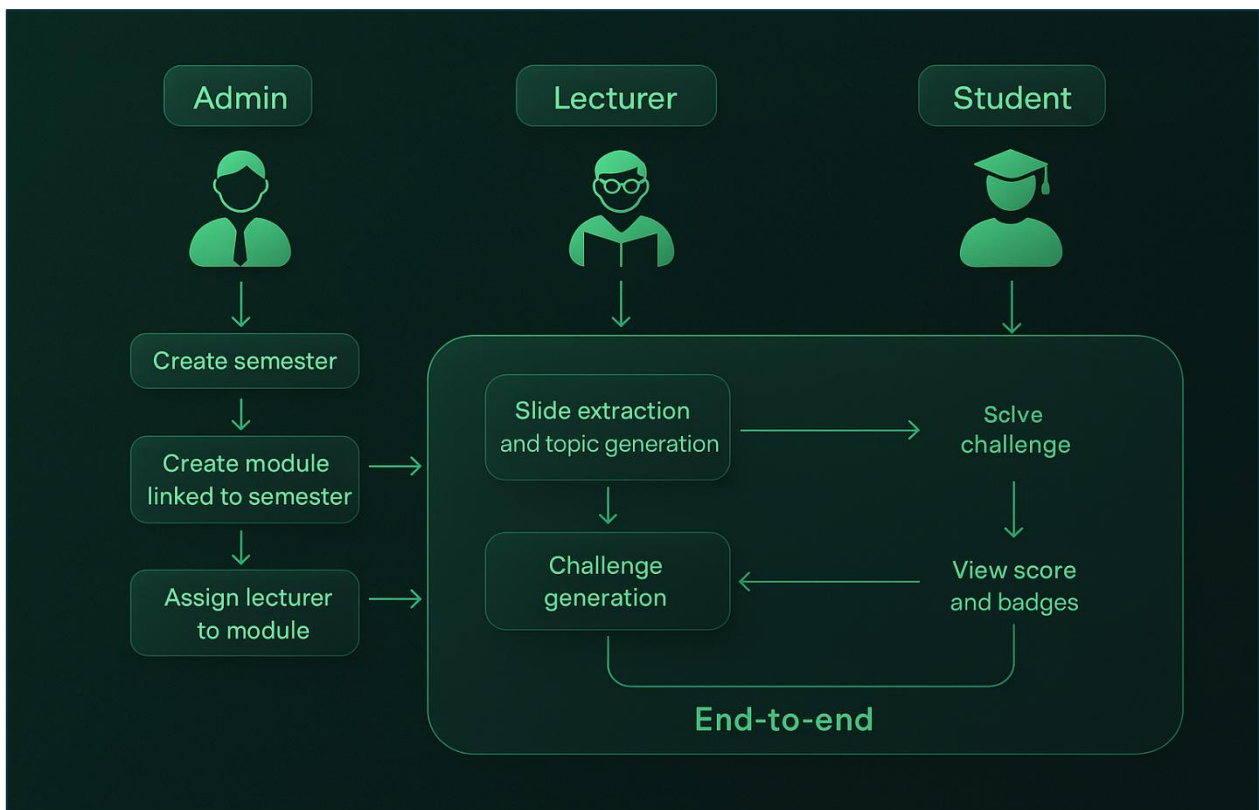
update_user_elo(user, difficulty):

    delta := elo_delta_map[difficulty]

    UPSERT user_elo(user, elo := greatest(0, current_elo + delta), updated_at:=now())
```

10. Deployment & Infrastructure

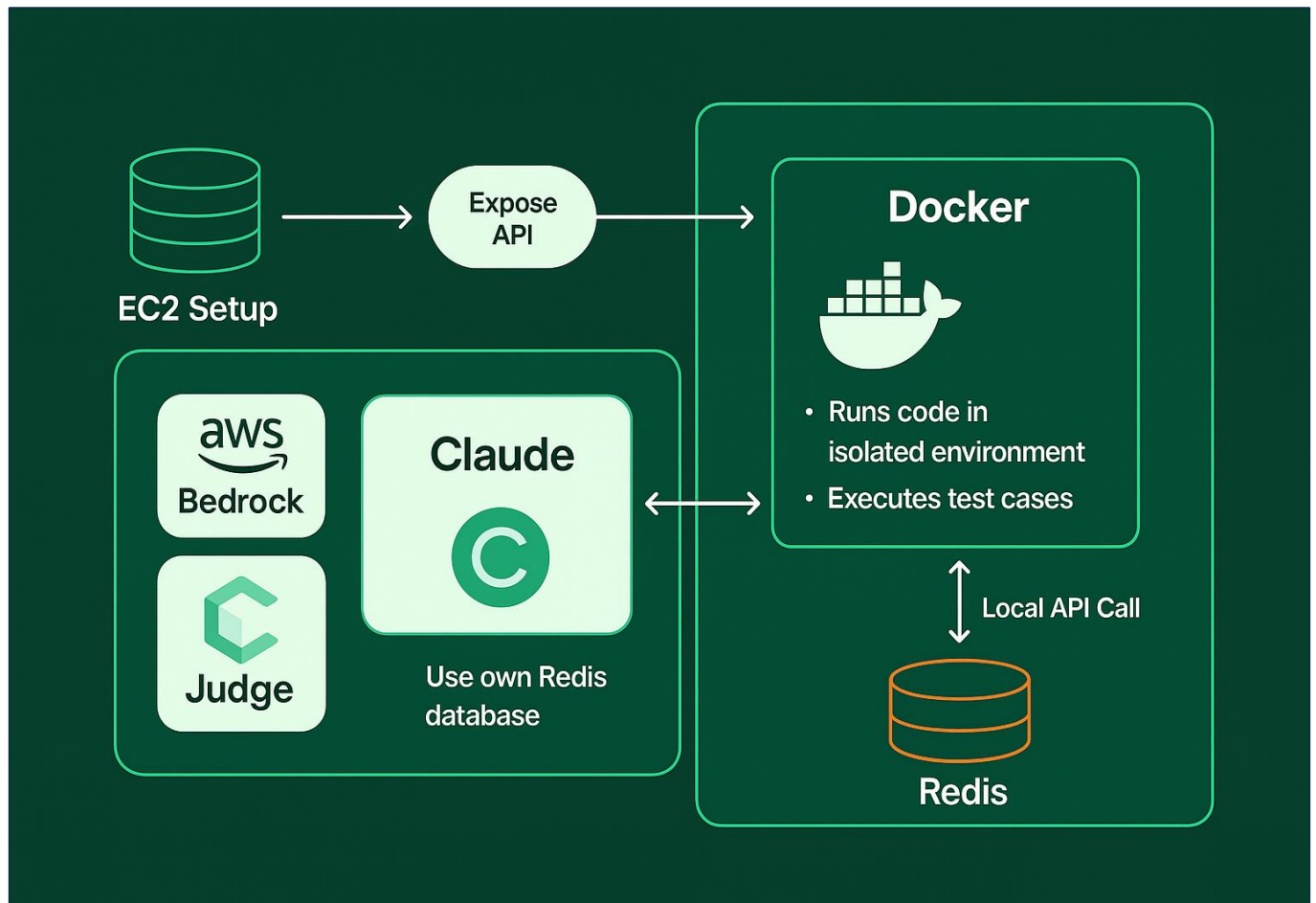
Figure 5 — Deployment Architecture (AWS/Supabase)



- **Backend:** FastAPI + Gunicorn/Uvicorn (workers=4), stateless.
 - **Queue:** Celery + Redis (extract/nlp/gen/mail).
 - **Exec:** Judge0 (Docker on EC2).
 - **AI:** Bedrock (Claude Sonnet).
 - **Data:** Supabase (Postgres with RLS + Storage).
 - **Security:** SG rules exposing required ports; HTTPS to backend; HTTP (private/VPC) to Judge0.
 - **Observability:** /healthz, audit logs, dead-letter table.
-

11. EC2 Setup & Docker (Judge0/Redis)

Figure 6 — EC2 + Docker Setup (Judge0/Redis/Bedrock)



Summary:

- **EC2** (Ubuntu LTS, t3.small+): install Docker & docker-compose.
- **Judge0**: run compose; expose API port (e.g., 2358) to backend SG.
- **Redis**: internal container; Celery connects via private network.
- **Bedrock**: enable model access; IAM role with bedrock:InvokeModel.
- **ENV**: JUDGE0_HOST, BEDROCK_MODEL, DB keys, CORS origins.

12. RPCs (Supabase SQL)

- `create_semester(year, term, start, end)`
- `create_module(code, name)`
- `create_course_instance(module_id, semester_id, title, start, end)`

- `assign_membership(course_instance_id, user_id, role)`
 - `publish_set(set_id)`
 - `award_badge_if_accepted(user_id, item_id)` (*optional enforcement*)
 - `update_elo(user_id, difficulty)` (*optional DB version*)
-

13. Error Handling & Resilience

- **Retries/backoff** on Bedrock/Judge0/network calls (2s, 4s, 8s, 16s).
 - **Dead-letter** table on persistent failure.
 - **Idempotency** for awarding (check history).
 - **RLS + Role enforcement** at API.
 - **Validation** via Pydantic schemas on all request bodies.
-

14. Frontend → Backend Contract (MVP)

- All protected requests with credentials: 'include'.
 - **Content-types:** JSON for metadata endpoints; multipart for upload; JSON for run/submit.
 - **Responses:** always typed (ids, names, arrays) and include timestamps where relevant.
 - **Pagination:** standard limit/offset for listings.
-

15. Non-Functional Requirements (MVP Targets)

- **Latency:** < 150 ms for small metadata calls; Judge0 runs variable by code.
 - **Availability:** single AZ acceptable; stateless API for easy restarts.
 - **Security:** HTTPS, HttpOnly cookies, RLS, least-privilege IAM.
 - **Docs:** OpenAPI via FastAPI; Swagger UI for local testing.
-

16. Future Enhancements (Phase 2+)

- Adaptive difficulty per student; leaderboard & cohort analytics.
- WebSocket push for execution status.
- Batch curation tools; import/export challenge banks.

- CI pipeline: lint/type/test on PRs; preview environments.
-

17. Appendix A — Pseudocode (Consolidated)

(Full end-to-end pseudocode block provided in §9, use as the canonical reference.)

18. Appendix B — Diagrams (All)

- Figure 1 — System Context (C4 L1)
 - Figure 2 — End-to-End Sequence
 - Figure 3 — Slide→Topic→Challenge Pipeline
 - Figure 4 — Evaluation & Submission Flow
 - Figure 5 — Deployment Architecture
 - Figure 6 — EC2 & Docker Setup
-

19. Appendix C — Minimal API Examples (FastAPI)

```
# auth_endpoints.py

@router.post("/auth/login")
async def login(credentials: LoginDTO, response: Response):
    tokens = supa.auth.sign_in_with_password(credentials)

    response.set_cookie("access_token", tokens.session.access_token, httponly=True,
samesite="lax", secure=IS_PROD)

    return {"user": await get_profile(tokens.user.id)}

@router.post("/auth/logout")
async def logout(response: Response):
    response.delete_cookie("access_token")

    return {"ok": True}

# attempts_endpoints.py

@router.post("/attempts")
async def create_attempt(dto: AttemptCreate, user=Depends(require_role(["student"])))
    # verify enrollment...
```

```
return repo.create_attempt(user.id, dto.item_id, dto.language, dto.initial_code)

# judge0_service.py

async def run_test(language: str, code: str, stdin: str):

    payload = {...}

    r = await http.post(f"{JUDGE0_HOST}/submissions?wait=true", json=payload)

    return r.json()
```

20. Appendix D — RLS Policy Notes

- Restrict profiles by `auth.uid() = profiles.id`.
 - Restrict slides/course_instances by membership.
 - Students: read own attempts/submissions; write own attempts/submissions.
 - Lecturers: read/write within assigned course_instances.
 - Admin: bypass via elevated role.
-

21. Appendix E — Ops & Health

- `/healthz` checks DB, Storage, Redis, Judge0, Bedrock.
 - Audit log on publish, award, role changes.
 - Backups: Supabase automated + export scripts.
-

22. Sign-off

This document is the authoritative MVP blueprint for RECODE's backend/infra and core student/lecturer workflows.