Prompt 1 — repo bootstrap & python backend deps

Paste to Claude Code:

Create a new repo named qbank with this structure and files. Use exact paths and contents.

Files:

README.md

QBank (v9)
- FastAPI backend + Redis/RQ worker
- Full Sympson–Hetter (S–H) iterative calibration
- Admin UI (Next.js) with progress & run history
- Postgres-backed calibration runs + CSV diff
- Filters & pagination on run list

Quick start:
- `psql $DATABASE_URL -f sql/content_ddl.sql`
- `psql $DATABASE_URL -f sql/item_exposure_control.sql`
- `psql $DATABASE_URL -f sql/feature_flags.sql`
- `psql $DATABASE_URL -f sql/calibration_runs.sql`
- `pip install -r qbank-backend/requirements.txt`
- API: `uvicorn app.main:app --reload` (in `qbank-backend/`)
- Worker: `python -m app.jobs.worker`
- Admin UI: `cd admin-ui && npm install && npm run dev`

qbank-backend/requirements.txt

fastapi==0.115.0
uvicorn==0.30.6
pydantic==2.8.2
python-dotenv==1.0.1
redis==5.0.8
rq==1.16.2
kafka-python==2.0.2
psycopg2-binary==2.9.9
SQLAlchemy==2.0.32
PyJWT==2.9.0
black==24.8.0
flake8==7.1.1
pytest==8.3.2

httpx==0.27.2

qbank-backend/app/core/config.py

```
import os
from dotenv import load_dotenv
load_dotenv()

DATABASE_URL = os.getenv("DATABASE_URL", "postgresql+psycopg2://
qbank:qbank@localhost:5432/qbank")
REDIS_URL = os×getenv("REDIS_URL", "redis://localhost:6379/0")
RQ_QUEUE = os×getenv("RQ_QUEUE", "calibration")
RQ_WORKER_CONCURRENCY = int(os.getenv("RQ_WORKER_CONCURRENCY",
"1"))
KAFKA_BOOTSTRAP = os.getenv("KAFKA_BOOTSTRAP", "localhost:9092")
KAFKA_TOPIC_EVENTS = os.getenv("KAFKA_TOPIC_EVENTS", "events.qbank")
TENANT_ID = os.getenv("APP_TENANT_ID",
"00000000-0000-0000-0000-000000000001")
APP_SECRET = os.getenv("APP_SECRET", "dev-secret-change-me")
MAX_DAILY_EXPOSURES = int(os.getenv("MAX_DAILY_EXPOSURES", "500"))
```

qbank-backend/app/core/database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.core.config import DATABASE_URL

engine = create_engine(DATABASE_URL, future=True, pool_pre_ping=True)
SessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False,
future=True)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

qbank-backend/app/core/cache.py

```
import redis
```

```
from datetime import datetime
from app.core.config import REDIS_URL, MAX_DAILY_EXPOSURES

redis_client = redis.Redis.from_url(REDIS_URL, decode_responses=True)

def exposure_key(question_id: int, version: int) -> str:
    day = datetime×utcnow()×strftime("%Y%m%d")
    return f"exp:{day}:{question_id}:{version}"

def can_serve(question_id: int, version: int) -> bool:
    key = exposure_key(question_id, version)
    count = int(redis_client.get(key) or 0)
    return count < MAX_DAILY_EXPOSURES

def bump_exposure(question_id: int, version: int) -> None:
    key = exposure_key(question_id, version)
    pipe = redis_client.pipeline()
    pipe.incr(key, 1)
    pipe.expire(key, 86400)
    pipe.execute()
```

Prompt 2 — models & SQL DDL (content + exposure + runs + flags)

Paste to Claude Code:

Add these ORM models and SQL schema files.

qbank-backend/app/models/orm.py

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import BigInteger, Integer, String, Text, Boolean, ForeignKey,
JSON, Float

class Base(DeclarativeBase): pass

class Topic(Base):
    __tablename__ = "topics"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    tenant_id: Mapped[str] = mapped_column(String)
    parent_id: Mapped[int | None] = mapped_column(BigInteger,
ForeignKey("topics.id"), nullable=True)
    name: Mapped[str] = mapped_column(String)
    blueprint_code: Mapped[str | None] = mapped_column(String, nullable=True)
```

```
class Question(Base):
    __tablename__ = "questions"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    tenant_id: Mapped[str] = mapped_column(String)
    external_ref: Mapped[str | None] = mapped_column(String, nullable=True)
    created_by: Mapped[str] = mapped_column(String)
    is_deleted: Mapped[bool] = mapped_column(Boolean, default=False)

class QuestionVersion(Base):
    __tablename__ = "question_versions"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    question_id: Mapped[int] = mapped_column(BigInteger,
ForeignKey("questions.id"))
    version: Mapped[int] = mapped_column(Integer)
    state: Mapped[str] = mapped_column(String)
    stem_md: Mapped[str] = mapped_column(Text)
    lead_in: Mapped[str] = mapped_column(Text)
    rationale_md: Mapped[str] = mapped_column(Text)
    difficulty_label: Mapped[str | None] = mapped_column(String, nullable=True)
    bloom_level: Mapped[int | None] = mapped_column(Integer, nullable=True)
    topic_id: Mapped[int | None] = mapped_column(BigInteger,
ForeignKey("topics.id"), nullable=True)
    tags: Mapped[dict] = mapped_column(JSON)
    assets: Mapped[list] = mapped_column(JSON)
    references: Mapped[list] = mapped_column(JSON)

class QuestionOption(Base):
    __tablename__ = "question_options"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    question_version_id: Mapped[int] = mapped_column(BigInteger,
ForeignKey("question_versions.id"))
    option_label: Mapped[str] = mapped_column(String(1))
    option_text_md: Mapped[str] = mapped_column(Text)
    is_correct: Mapped[bool] = mapped_column(Boolean)

class QuestionPublication(Base):
    __tablename__ = "question_publications"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    question_id: Mapped[int] = mapped_column(BigInteger,
ForeignKey("questions.id"))
    live_version: Mapped[int] = mapped_column(Integer)
    exam_code: Mapped[str] = mapped_column(String)
    tenant_id: Mapped[str] = mapped_column(String)
```

```
class QuizSession(Base):
    __tablename__ = "quiz_sessions"
    id: Mapped[str] = mapped_column(String, primary_key=True)
    user_id: Mapped[str] = mapped_column(String)
    tenant_id: Mapped[str] = mapped_column(String)
    mode: Mapped[str] = mapped_column(String)
    adaptive: Mapped[bool] = mapped_column(Boolean, default=True)
    exam_code: Mapped[str | None] = mapped_column(String, nullable=True)

class QuizItem(Base):
    __tablename__ = "quiz_items"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    quiz_id: Mapped[str] = mapped_column(String)
    question_id: Mapped[int] = mapped_column(BigInteger)
    version: Mapped[int] = mapped_column(Integer)
    position: Mapped[int] = mapped_column(Integer)

class UserResponse(Base):
    __tablename__ = "user_responses"
    id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    quiz_id: Mapped[str] = mapped_column(String)
    user_id: Mapped[str] = mapped_column(String)
    question_id: Mapped[int] = mapped_column(BigInteger)
    version: Mapped[int] = mapped_column(Integer)
    option_label: Mapped[str] = mapped_column(String(1))
    is_correct: Mapped[bool] = mapped_column(Boolean)
    time_taken_ms: Mapped[int | None] = mapped_column(Integer, nullable=True)

class ItemCalibration(Base):
    __tablename__ = "item_calibration"
    question_id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    version: Mapped[int] = mapped_column(Integer, primary_key=True)
    model: Mapped[str] = mapped_column(String, primary_key=True)
    a: Mapped[float | None] = mapped_column(Float)
    b: Mapped[float | None] = mapped_column(Float)
    c: Mapped[float | None] = mapped_column(Float)
    n_respondents: Mapped[int | None] = mapped_column(Integer)

class ItemExposureControl(Base):
    __tablename__ = "item_exposure_control"
    question_id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
    version: Mapped[int] = mapped_column(Integer, primary_key=True)
    sh_p: Mapped[float] = mapped_column(Float)
```

```
class FeatureFlag(Base):
    __tablename__ = "feature_flags"
    key: Mapped[str] = mapped_column(String, primary_key=True)
    enabled: Mapped[bool] = mapped_column(Boolean, default=True)
    value_json: Mapped[dict] = mapped_column(JSON)

class CohortAssignment(Base):
    __tablename__ = "cohort_assignments"
    user_id: Mapped[str] = mapped_column(String, primary_key=True)
    cohort_key: Mapped[str] = mapped_column(String, primary_key=True)
    cohort_value: Mapped[str] = mapped_column(String)


sql/content_ddl.sql

CREATE TABLE IF NOT EXISTS topics(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  tenant_id TEXT NOT NULL,
  parent_id BIGINT NULL,
  name TEXT NOT NULL,
  blueprint_code TEXT NULL
);
CREATE TABLE IF NOT EXISTS questions(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  tenant_id TEXT NOT NULL,
  external_ref TEXT,
  created_by TEXT NOT NULL,
  is_deleted BOOLEAN NOT NULL DEFAULT FALSE
);
CREATE TABLE IF NOT EXISTS question_versions(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  question_id BIGINT NOT NULL REFERENCES questions(id),
  version INT NOT NULL,
  state TEXT NOT NULL,
  stem_md TEXT NOT NULL,
  lead_in TEXT NOT NULL,
  rationale_md TEXT NOT NULL,
  difficulty_label TEXT,
  bloom_level INT,
  topic_id BIGINT REFERENCES topics(id),
  tags JSONB NOT NULL DEFAULT '{}'::jsonb,
  assets JSONB NOT NULL DEFAULT '[]'::jsonb,
  references JSONB NOT NULL DEFAULT '[]'::jsonb
```

```
);
CREATE TABLE IF NOT EXISTS question_options(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  question_version_id BIGINT NOT NULL REFERENCES question_versions(id),
  option_label TEXT NOT NULL,
  option_text_md TEXT NOT NULL,
  is_correct BOOLEAN NOT NULL
);
CREATE TABLE IF NOT EXISTS question_publications(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  question_id BIGINT NOT NULL REFERENCES questions(id),
  live_version INT NOT NULL,
  exam_code TEXT NOT NULL,
  tenant_id TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS quiz_sessions(
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL,
  tenant_id TEXT NOT NULL,
  mode TEXT NOT NULL,
  adaptive BOOLEAN NOT NULL DEFAULT TRUE,
  exam_code TEXT
);
CREATE TABLE IF NOT EXISTS quiz_items(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  quiz_id TEXT NOT NULL,
  question_id BIGINT NOT NULL,
  version INT NOT NULL,
  position INT NOT NULL
);
CREATE TABLE IF NOT EXISTS user_responses(
  id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  quiz_id TEXT NOT NULL,
  user_id TEXT NOT NULL,
  question_id BIGINT NOT NULL,
  version INT NOT NULL,
  option_label TEXT NOT NULL,
  is_correct BOOLEAN NOT NULL,
  time_taken_ms INT
);
CREATE TABLE IF NOT EXISTS item_calibration(
  question_id BIGINT NOT NULL,
  version INT NOT NULL,
  model TEXT NOT NULL,
```

```
  a DOUBLE PRECISION, b DOUBLE PRECISION, c DOUBLE PRECISION,
  n_respondents INT,
  PRIMARY KEY(question_id, version, model)
);
```

sql/item_exposure_control.sql

```
CREATE TABLE IF NOT EXISTS item_exposure_control (
  question_id BIGINT NOT NULL,
  version INT NOT NULL,
  sh_p DOUBLE PRECISION NOT NULL DEFAULT 1.0,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  PRIMARY KEY (question_id, version)
);
```

sql/feature_flags.sql

```
CREATE TABLE IF NOT EXISTS feature_flags (
  key TEXT PRIMARY KEY,
  enabled BOOLEAN NOT NULL DEFAULT TRUE,
  value_json JSONB NOT NULL DEFAULT '{}'::jsonb,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE TABLE IF NOT EXISTS cohort_assignments (
  user_id TEXT NOT NULL,
  cohort_key TEXT NOT NULL,
  cohort_value TEXT NOT NULL,
  assigned_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  PRIMARY KEY(user_id, cohort_key)
);
```

sql/calibration_runs.sql

```
CREATE TABLE IF NOT EXISTS calibration_runs (
  id UUID PRIMARY KEY,
  exam_code TEXT NOT NULL,
  status TEXT NOT NULL,
  params JSONB NOT NULL,
  history JSONB NOT NULL DEFAULT '[]'::jsonb,
  result JSONB,
  error TEXT,
```

```
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  started_at TIMESTAMPTZ,
  finished_at TIMESTAMPTZ
);
CREATE INDEX IF NOT EXISTS idx_cal_runs_exam ON
calibration_runs(exam_code);
CREATE INDEX IF NOT EXISTS idx_cal_runs_created ON
calibration_runs(created_at DESC);
```

Prompt 3 — auth & authoring & quizzes APIs

Paste to Claude Code:

Add these API modules.

qbank-backend/app/core/auth.py

```
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from pydantic import BaseModel
from typing import List
import jwt
from datetime import datetime, timedelta, timezone
from app.core.config import APP_SECRET

class TokenData(BaseModel):
    sub: str
    roles: List[str]

bearer = HTTPBearer()

def create_token(user_id: str, roles: List[str], ttl_minutes: int = 120) -> str:
    now = datetime×now(timezone×utc)
    payload = {"sub": user_id, "roles": roles, "iat": int(now.timestamp()), "exp":
int((now + timedelta(minutes=ttl_minutes)).timestamp())}
    return jwt.encode(payload, APP_SECRET, algorithm="HS256")

def get_current_user(creds: HTTPAuthorizationCredentials = Depends(bearer)) ->
TokenData:
    try:
        payload = jwt×decode(creds×credentials, APP_SECRET,
algorithms=["HS256"])
        return TokenData(sub=payload["sub"], roles=payload×get("roles", []))
    except Exception:
```

```
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid or expired token")

def require_roles(*required: str):
    def checker(user: TokenData = Depends(get_current_user)):
        roles = set(user.roles)
        if not roles.intersection(set(required)):
            raise HTTPException(status_code=403, detail="Insufficient role")
        return user
    return checker
```

qbank-backend/app/api/auth.py

```
from fastapi import APIRouter
from pydantic import BaseModel
from typing import List
from app.core.auth import create_token

router = APIRouter()

class MockLogin(BaseModel):
    user_id: str
    roles: List[str]

@router.post("/mock-login")
def mock_login(payload: MockLogin):
    token = create_token(payload.user_id, payload.roles)
    return {"access_token": token, "token_type": "bearer", "roles": payload.roles}
```

qbank-backend/app/services/adaptive.py

```
import math, random
from typing import List, Dict, Optional
D = 1.7
def logistic(x: float) -> float: return 1.0 / (1.0 + math.exp(-x))
def prob_3pl(theta: float, a: float, b: float, c: float) -> float:
    return c + (1.0 - c) * logistic(D * a * (theta - b))
def fisher_info_3pl(theta: float, a: float, b: float, c: float) -> float:
    P = prob_3pl(theta, a, b, c); Q = 1.0 - P
    if P<=0 or Q<=0 or (1.0-c)<=0: return 0.0
    return (D**2)*(a**2)*(Q/P)*((P-c)/(1.0-c))**2
def select_vanilla(candidates: List[Dict], theta: float) -> Optional[Dict]:
```

```
    best, best_I = None, -1
    for it in candidates:
        I = fisher_info_3pl(theta, it.get("a",1.0), it.get("b",0.0), it.get("c",0.2))
        if I > best_I: best_I, best = I, it
    return best
def select_sympson_hetter(candidates: List[Dict], theta: float) -> Optional[Dict]:
    scored = []
    for it in candidates:
        I = fisher_info_3pl(theta, it.get("a",1.0), it.get("b",0.0), it.get("c",0.2))
        scored.append((I, it))
    scored×sort(key=lambda x: x[0], reverse=True)
    for _, it in scored:
        sh_p = it.get("sh_p", 1.0)
        if random.random() <= max(0.0, min(1.0, sh_p)):
            return it
    return scored[0][1] if scored else None
```

qbank-backend/app/services/selector_choice.py

```
from sqlalchemy.orm import Session
from sqlalchemy import select
from app.models.orm import FeatureFlag, CohortAssignment

def get_selector_for_user(db: Session, user_id: str) -> str:
    ff = db×scalar(select(FeatureFlag)×where(FeatureFlag×key ==
"selector_strategy"))
    default = "sympson_hetter" if (ff and ff.enabled and (ff.value_json or
{}).get("value") == "sympson_hetter") else "vanilla"
    cohort =
db.scalar(select(CohortAssignment).where(CohortAssignment.user_id==user_id,
CohortAssignment.cohort_key=="selector_strategy"))
    return cohort.cohort_value if cohort else default
```

qbank-backend/app/api/author.py

```
from fastapi import APIRouter, Depends, HTTPException
from pydantic import BaseModel, constr
from typing import List, Optional
from sqlalchemy.orm import Session
from sqlalchemy import select, func
from app.core.database import get_db
from app.core.config import TENANT_ID
```

```
from app.core.auth import require_roles, TokenData
from app.models.orm import Topic, Question, QuestionVersion, QuestionOption,
QuestionPublication

router = APIRouter()

class OptionIn(BaseModel):
    label: constr(min_length=1, max_length=1)
    text_md: str
    is_correct: bool

class QuestionCreate(BaseModel):
    external_ref: Optional[str] = None
    topic_name: str
    exam_code: str = "DEMO-EXAM"
    stem_md: str
    lead_in: str
    rationale_md: str
    difficulty_label: Optional[str] = "medium"
    options: List[OptionIn]

@router.post("/questions",
dependencies=[Depends(require_roles("author","admin"))])
def create_question(payload: QuestionCreate, user: TokenData =
Depends(require_roles("author","admin")), db: Session = Depends(get_db)):
    t = db×scalar(select(Topic)×where(Topic×name == payload.topic_name))
    if not t:
        t = Topic(tenant_id=TENANT_ID, parent_id=None,
name=payload.topic_name, blueprint_code=None)
        db.add(t); db.flush()
    q = Question(tenant_id=TENANT_ID, external_ref=payload.external_ref,
created_by=user.sub, is_deleted=False)
    db.add(q); db.flush()
    next_v = (db×scalar(select(func×coalesce(func×max(QuestionVersion×version),
0))×where(QuestionVersion×question_id == q.id)) or 0) + 1
    qv = QuestionVersion(
        question_id=q×id, version=next_v, state="published",
        stem_md=payload.stem_md, lead_in=payload.lead_in,
rationale_md=payload.rationale_md,
        difficulty_label=payload×difficulty_label, topic_id=t×id, tags={}, assets=[],
references=[]
    )
    db.add(qv); db.flush()
    for o in payload.options:
```

```
        db.add(QuestionOption(question_version_id=qv×id,
option_label=o.label.upper(), option_text_md=o.text_md,
is_correct=o.is_correct))
    db.add(QuestionPublication(question_id=q×id, live_version=next_v,
exam_code=payload.exam_code, tenant_id=TENANT_ID))
    db.commit()
    return {"question_id": q.id, "version": next_v, "topic_id": t.id}
```

qbank-backend/app/api/quizzes.py

```
from fastapi import APIRouter, HTTPException, Depends
from pydantic import BaseModel, Field, constr
from typing import List, Optional, Literal
from uuid import uuid4
from datetime import datetime, timedelta
import json
from sqlalchemy.orm import Session
from sqlalchemy import select
from app.core.cache import redis_client, bump_exposure
from app.core.database import get_db
from app.core.auth import require_roles, TokenData
from app.models.orm import QuestionVersion, QuestionOption,
QuestionPublication, ItemCalibration, ItemExposureControl
from app.services.selector_choice import get_selector_for_user
from app.services.adaptive import select_vanilla, select_sympson_hetter

router = APIRouter()

class QuizFilters(BaseModel):
  topics: Optional[List[str]] = None
  difficulty: Optional[List[Literal["easy","medium","hard"]]] = None
  num_questions: int = Field(ge=1, le=120, default=40)
  mode: Literal["tutor","exam"] = "tutor"
  exam_code: Optional[str] = "DEMO-EXAM"

class QuizCreate(BaseModel):
  tenant_id: constr(min_length=8)
  filters: QuizFilters
  adaptive: bool = True

class QuizCreated(BaseModel):
  quiz_id: str
  question_ids: List[int]
```

```
  expires_at: datetime
  mode: Literal["tutor","exam"]

class NextQuestion(BaseModel):
  question_id: int
  version: int
  payload: dict

class AnswerSubmit(BaseModel):
  question_id: int
  selected: constr(min_length=1, max_length=1)
  time_taken_ms: Optional[int] = 0
  client_latency_ms: Optional[int] = 0

class AnswerResult(BaseModel):
  correct: bool
  correct_option: constr(min_length=1, max_length=1)
  explanation: dict
  difficulty: float

def _rk(qid: str, suf: str) -> str: return f"quiz:{qid}:{suf}"

@router.post("", response_model=QuizCreated, status_code=201,
dependencies=[Depends(require_roles("student","admin"))])
def create_quiz(payload: QuizCreate, user: TokenData =
Depends(require_roles("student","admin")), db: Session = Depends(get_db)):
  quiz_id = str(uuid4()); mode = payload×filters×mode
  expires_at = datetime.utcnow() + timedelta(hours=2)
  stmt = select(QuestionPublication, QuestionVersion).join(
    QuestionVersion,
    (QuestionVersion.question_id == QuestionPublication.question_id) &
(QuestionVersion.version == QuestionPublication.live_version)
  ).where(QuestionPublication.exam_code == (payload.filters.exam_code or
"DEMO-EXAM"), QuestionVersion.state == "published")
  rows = db.execute(stmt)×all()
  versions = [r[1] for r in rows]
  vcache = [{"q": v.question_id, "v": v.version, "t": v.topic_id, "d": v.difficulty_label}
for v in versions]
  redis_client.set(_rk(quiz_id, "versions"), json.dumps(vcache), ex=7200)
  redis_client.set(_rk(quiz_id, "cursor"), 0, ex=7200)
  redis_client.set(_rk(quiz_id, "mode"), mode, ex=7200)
  redis_client.set(_rk(quiz_id, "user"), user.sub, ex=7200)
  selector = get_selector_for_user(db, user.sub)
  redis_client.set(_rk(quiz_id, "selector"), selector, ex=7200)
```

```
  qids = list({v["q"] for v in vcache})[:payload.filters.num_questions]
  return QuizCreated(quiz_id=quiz_id, question_ids=qids, expires_at=expires_at,
mode=mode)

@router.get("/{quiz_id}/next", response_model=NextQuestion,
dependencies=[Depends(require_roles("student","admin"))])
def next_question(quiz_id: str, db: Session = Depends(get_db)):
  import json
  raw = redis_client.get(_rk(quiz_id,"versions"))
  if not raw: raise HTTPException(404, "Quiz not found or expired")
  versions = json×loads(raw)
  selector = redis_client.get(_rk(quiz_id,"selector")) or "vanilla"
  curk = _rk(quiz_id, "cursor"); cur = int(redis_client.get(curk) or 0)
  if cur >= len(versions): raise HTTPException(404, "No more questions")
  window = versions[cur : min(cur+20, len(versions))]
  candidates = []
  for w in window:
    ic =
db×scalar(select(ItemCalibration)×where(ItemCalibration×question_id==w["q"],
ItemCalibration.version==w["v"]).limit(1))
    exp =
db×scalar(select(ItemExposureControl)×where(ItemExposureControl×question_id
==w["q"], ItemExposureControl.version==w["v"]).limit(1))
    a = (ic×a if ic and ic.a is not None else 1.0) if ic else 1.0
    b = (ic.b if ic and ic.b is not None else 0.0) if ic else 0.0
    c = (ic.c if ic and ic.c is not None else 0.2) if ic else 0.2
    sh_p = exp.sh_p if exp else 1.0
    candidates.append({"question_id": w["q"], "version": w["v"], "topic_id": w["t"],
"a": a, "b": b, "c": c, "sh_p": sh_p})
  best = (select_sympson_hetter if selector=="sympson_hetter" else
select_vanilla)(candidates, theta=0.0) or candidates[0]
  redis_client.set(curk, cur+1)
  qv =
db.scalar(select(QuestionVersion).where(QuestionVersion.question_id==best["que
stion_id"], QuestionVersion.version==best["version"]))
  if not qv: raise HTTPException(500, "Item not found")
  opts =
db.execute(select(QuestionOption).where(QuestionOption.question_version_id==
qv.id)).scalars().all()
  bump_exposure(best["question_id"], best["version"])
  payload = {"stem_md": qv.stem_md, "lead_in": qv.lead_in, "options": [{"label":
o.option_label, "text": o.option_text_md} for o in opts]}
  return NextQuestion(question_id=best["question_id"], version=best["version"],
payload=payload)
```

```
@router.post("/{quiz_id}/answers", response_model=AnswerResult,
dependencies=[Depends(require_roles("student","admin"))])
def submit_answer(quiz_id: str, payload: AnswerSubmit, db: Session =
Depends(get_db)):
  qv =
db×scalar(select(QuestionVersion)×where(QuestionVersion×question_id==payloa
d.question_id).order_by(QuestionVersion.version.desc()))
  if not qv: raise HTTPException(404, "Question not found")
  opts =
db.execute(select(QuestionOption).where(QuestionOption.question_version_id==
qv.id)).scalars().all()
  correct = next((o×option_label for o in opts if o.is_correct), None)
  if not correct: raise HTTPException(500, "No correct option set")
  ok = (payload×selected×upper() == correct)
  return AnswerResult(correct=ok, correct_option=correct,
explanation={"rationale_md": qv.rationale_md}, difficulty=0.5)
```

qbank-backend/app/main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api.quizzes import router as quizzes_router
from app.api.author import router as author_router
from app.api.auth import router as auth_router
from app.api.admin import router as admin_router
from app.api.admin_runs import router as runs_router

app = FastAPI(title="QBank API v9", version="9.0.0")
app.add_middleware(CORSMiddleware, allow_origins=["*"],
allow_credentials=True, allow_methods=["*"], allow_headers=["*"])
app.include_router(auth_router, prefix="/v1/auth", tags=["auth"])
app.include_router(quizzes_router, prefix="/v1/quizzes", tags=["quizzes"])
app.include_router(author_router, prefix="/v1/author", tags=["authoring"])
app.include_router(admin_router, prefix="/v1/admin", tags=["admin"])
app.include_router(runs_router, prefix="/v1/admin", tags=["calibration-runs"])

@app.get("/health")
def health(): return {"status": "ok"}
```

Prompt 4 — SH core + CLI + jobs/worker + admin endpoints + runs endpoints

Paste to Claude Code:

Add calibration core, CLI, RQ queue + worker, admin calibration endpoints, and run-history endpoints.

analytics/calibration/sh_core.py

```
import math, random, statistics
import psycopg2, psycopg2.extras
D = 1.7
def logistic(x): return 1/(1+math.exp(-x))
def prob_3pl(theta,a,b,c): return c + (1-c)*logistic(D*a*(theta-b))
def fisher_info_3pl(theta,a,b,c):
    P=prob_3pl(theta,a,b,c); Q=1-P
    if P<=0 or Q<=0 or (1-c)<=0: return 0.0
    return (D**2)*(a**2)*(Q/P)*((P-c)/(1-c))**2
def sample_theta(dist="normal0,1"):
    return random.gauss(0,1) if dist.startswith("normal") else (random.uniform(-1,1)
if dist.startswith("uniform") else 0.0)
def load_pool(conn, exam_code):
    sql = '''
    SELECT qv.question_id, qv.version, qv.topic_id,
           COALESCE(ic.a,1.0) a, COALESCE(ic.b,0.0) b, COALESCE(ic.c,0.2) c,
           COALESCE(iec.sh_p,1.0) sh_p
    FROM question_publications qp
    JOIN question_versions qv ON qv.question_id=qp.question_id AND
qv×version=qp.live_version
    LEFT JOIN item_calibration ic ON ic.question_id=qv.question_id AND
ic×version=qv.version AND ic×model='3pl'
    LEFT JOIN item_exposure_control iec ON iec.question_id=qv.question_id AND
iec×version=qv.version
    WHERE qp.exam_code=%s AND qv×state='published'
    '''
    with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cur:
        cur.execute(sql,(exam_code,)); rows=cur×fetchall()
    return
[{"qid":int(r["question_id"]),"ver":int(r["version"]),"topic":r["topic_id"],"a":float(r["
a"]),"b":float(r["b"]),"c":float(r["c"]),"k":float(r["sh_p"]) if r["sh_p"] is not None
else 1.0} for r in rows]
def simulate_once(pool,n,test_len,theta_dist,use_k=True,seed=None):
    if seed is not None: random.seed(seed)
    seen={ (it["qid"],it["ver"]):0 for it in pool }
    for _ in range(n):
        theta=sample_theta(theta_dist); administered=set()
        for _pos in range(test_len):
```

```
            cand=[it for it in pool if (it["qid"],it["ver"]) not in administered]
            if not cand: break
            scored=sorted(((fisher_info_3pl(theta,it["a"],it["b"],it["c"]),it) for it in
cand), key=lambda x:x[0], reverse=True)
            chosen=None
            for _,it in scored:
                if (not use_k) or (random.random()<=max(0.0,min(1.0,it["k"]))):
                    chosen=it; break
            if chosen is None: chosen=scored[0][1]
            administered.add((chosen["qid"],chosen["ver"]))
        for key in administered: seen[key]+=1
    return seen, {}
def _compute_topic_tau(tau, topic_tau, topic_weights):
    if topic_tau: return {str(k): float(v) for k,v in topic_tau.items()}
    if topic_weights:
        s=sum(float(v) for v in topic_weights.values()) or 1.0
        return {str(k): tau*(float(v)/s) for k,v in topic_weights.items()}
    return None
def
iterative_sh(pool,tau,n,test_len,iters,alpha,theta_dist,floor,ceil,seed=None,topic_ta
u=None,topic_weights=None):
    tmap=_compute_topic_tau(tau,topic_tau,topic_weights)
    k={ (it["qid"],it["ver"]):float(it["k"]) for it in pool }
    history=[]
    for t in range(iters):
        for it in pool: it["k"]=k[(it["qid"],it["ver"])]
        seen,_=simulate_once(pool,n,test_len,theta_dist,use_k=True,seed=None if
seed is None else seed+t)
        r={ key: seen[key]/max(1,n) for key in seen }
        newk={}
        for it in pool:
            key=(it["qid"],it["ver"]); ri=r×get(key,0.0); ki=k[key]
            cap = float(tmap×get(str(it["topic"]),tau)) if tmap is not None else tau
            if ri<=0.0: val=min(1.0,max(floor,ki*1.1))
            else:
                ratio=cap/ri; val=ki*(ratio**alpha); val=min(1.0,max(floor,min(ceil,val)))
            newk[key]=val
        k=newk
        avg_exp=(sum(r×values())/len(r)) if r else 0.0
        if tmap is None:
            max_over=max((ri-tau for ri in r.values()), default=0.0)
        else:
            ovs=[]
            for it in pool:
```

```
            key=(it["qid"],it["ver"]); cap=float(tmap.get(str(it["topic"]),tau));
ovs.append(r.get(key,0.0)-cap)
        max_over=max(ovs) if ovs else 0.0
      history.append({"iter":t+1,"avg_exp":avg_exp,"max_over":max_over})
  return k, seen, history
def upsert_k(conn,kmap):
  with conn.cursor() as cur:
    cur.execute("SET search_path TO public")
    for (qid,ver),kval in kmap.items():
      cur.execute(
       '''INSERT INTO item_exposure_control(question_id,version,sh_p)
         VALUES (%s,%s,%s)
         ON CONFLICT (question_id,version) DO UPDATE SET
sh_p=EXCLUDED.sh_p, updated_at=now()''',
        (qid,ver,float(kval))
      )
  conn.commit()


analytics/calibration/sh_iterative.py

import argparse, json, psycopg2
from sh_core import load_pool, iterative_sh, upsert_k

def main():
  ap = argparse×ArgumentParser()
  ap.add_argument("--dsn", required=True)
  ap.add_argument("--exam", "--exam_code", dest="exam_code",
required=True)
  ap.add_argument("--tau", type=float, default=0.2)
  ap.add_argument("--n", type=int, default=500)
  ap.add_argument("--len", "--test_len", dest="test_len", type=int, default=30)
  ap.add_argument("--iters", type=int, default=6)
  ap.add_argument("--alpha", type=float, default=0.6)
  ap.add_argument("--theta", "--theta_dist", dest="theta_dist",
default="normal0,1")
  ap.add_argument("--floor", type=float, default=0.02)
  ap.add_argument("--ceil", type=float, default=1.0)
  ap.add_argument("--seed", type=int, default=None)
  ap.add_argument("--topic_tau", type=str, default=None)
  ap.add_argument("--topic_weights", type=str, default=None)
  ap.add_argument("--dry_run", action="store_true")
  args = ap×parse_args()
  t_tau=json.loads(args.topic_tau) if args.topic_tau else None
```

```
    t_w=json.loads(args.topic_weights) if args.topic_weights else None
    conn=psycopg2.connect(args×dsn)
    pool=load_pool(conn,args.exam_code)
    if not pool: print("No items found"); return

kmap,seen,hist=iterative_sh(pool,args.tau,args.n,args.test_len,args.iters,args.alph
a,args.theta_dist,args.floor,args.ceil,args.seed,t_tau,t_w)
    print("history=",hist)
    if not args.dry_run:
        upsert_k(conn,kmap); print("upserted=",len(kmap))
    conn.close()
if __name__=="__main__": main()
```

qbank-backend/app/jobs/queue.py

```
from rq import Queue
from redis import Redis
from app.core.config import REDIS_URL, RQ_QUEUE
redis = Redis.from_url(REDIS_URL)
queue = Queue(RQ_QUEUE, connection=redis)
```

qbank-backend/app/jobs/worker.py

```
from rq import Worker
from app.jobs.queue import queue, redis
from app.core.config import RQ_QUEUE
if __name__ == "__main__":
    w = Worker([RQ_QUEUE], connection=redis)
    w.work(with_scheduler=True)
```

qbank-backend/app/jobs/calibration_job.py

```
import os, json, psycopg2, uuid
from rq import get_current_job
from analytics.calibration.sh_core import load_pool, iterative_sh, upsert_k

def calibrate_job(exam_code, dsn, tau, n, test_len, iters, alpha, theta_dist, floor,
ceil, topic_tau, topic_weights, dry_run, run_id=None):
    job = get_current_job()
    job.meta.update({"state":"running","current_iter":0,"total_iters":iters});
job.save_meta()
```

```
    conn=psycopg2.connect(dsn); cur=conn×cursor()
    if run_id is None:
        run_id=str(uuid×uuid4())
        cur.execute("INSERT INTO
calibration_runs(id,exam_code,status,params,created_at,started_at) VALUES (%s,
%s,%s,%s,now(),now())",
          (run_id, exam_code, "running",
json.dumps({"tau":tau,"n":n,"test_len":test_len,"iters":iters,"alpha":alpha,"theta_di
st":theta_dist,"floor":floor,"ceil":ceil,"topic_tau":topic_tau,"topic_weights":topic_w
eights,"dry_run":dry_run})))
        conn.commit()
    else:
        cur.execute("UPDATE calibration_runs SET status='running',
started_at=now() WHERE id=%s",(run_id,)); conn.commit()
    try:
        pool=load_pool(conn, exam_code)
        if not pool:
            job.meta.update({"state":"empty"}); job.save_meta()
            cur.execute("UPDATE calibration_runs SET status='empty',
finished_at=now(), result=%s WHERE id=%s", (json.dumps({"updated":0,"history":
[],"diff":[]}), run_id)); conn.commit()
            cur.close(); conn.close(); return {"updated":0,"history":[],"diff":[]}
        before=[{"qid":it["qid"],"ver":it["ver"],"sh_p":float(it["k"])} for it in pool]
        kmap={ (it["qid"],it["ver"]):float(it["k"]) for it in pool }
        history=[]
        for t in range(iters):

km,seen,hist=iterative_sh(pool,tau,n,test_len,1,alpha,theta_dist,floor,ceil,None,topi
c_tau,topic_weights)
            for it in pool: it["k"]=km[(it["qid"],it["ver"])]
            kmap=km; history.extend(hist)
            job.meta.update({"current_iter":t+1,"avg_exp":hist[-1]
["avg_exp"],"max_over":hist[-1]["max_over"]}); job.save_meta()
            cur.execute("UPDATE calibration_runs SET history =
COALESCE(history,'[]'::jsonb) || %s::jsonb WHERE id=%s", (json.dumps([hist[-1]]),
run_id)); conn.commit()
        after=[{"qid":it["qid"],"ver":it["ver"],"sh_p":float(kmap[(it["qid"],it["ver"])])}
for it in pool]
        amap={(a["qid"],a["ver"]):a["sh_p"] for a in after}
        diff=[{"qid":b["qid"],"ver":b["ver"],"before":float(b["sh_p"]), "after":
float(amap.get((b["qid"],b["ver"]), b["sh_p"])), "delta":
float(amap.get((b["qid"],b["ver"]), b["sh_p"]))-float(b["sh_p"])} for b in before]
        if not dry_run: upsert_k(conn,kmap)
        result={"updated":len(kmap),"history":history,"diff":diff}
```

```
        cur.execute("UPDATE calibration_runs SET status='done', finished_at=now(),
result=%s WHERE id=%s",(json.dumps(result),run_id)); conn.commit()
        job.meta.update({"state":"done"}); job.save_meta()
        cur.close(); conn.close(); return result
    except Exception as e:
        job.meta.update({"state":"failed"}); job.save_meta()
        cur.execute("UPDATE calibration_runs SET status='failed', finished_at=now(),
error=%s WHERE id=%s",(str(e),run_id)); conn.commit()
        cur.close(); conn.close(); raise


qbank-backend/app/api/admin.py

from fastapi import APIRouter, Depends, HTTPException
from pydantic import BaseModel
from typing import List, Optional
from sqlalchemy.orm import Session
from sqlalchemy import text
import os, json, uuid, psycopg2
from app.core.database import get_db
from app.core.auth import require_roles
from app.jobs.queue import queue
from app.jobs.calibration_job import calibrate_job
from app.models.orm import ItemExposureControl

router = APIRouter()

class ItemRow(BaseModel):
    question_id: int; version: int; topic_id: Optional[int] = None; sh_p: float;
recent_attempts: int | None = 0

@router.get("/exposure/items", response_model=List[ItemRow],
dependencies=[Depends(require_roles("admin"))])
def list_items(limit: int = 100, db: Session = Depends(get_db)):
    rows = db.execute(text(\"\"\"
      SELECT qv.question_id, qv.version, qv.topic_id, COALESCE(iec.sh_p,1.0) sh_p,
        (SELECT count(*) FROM user_responses ur WHERE
ur.question_id=qv.question_id AND ur×version=qv.version AND
ur.created_at>now()-interval '7 days') recent_attempts
      FROM question_versions qv
      LEFT JOIN item_exposure_control iec ON iec.question_id=qv.question_id AND
iec×version=qv.version
      WHERE qv×state='published' ORDER BY recent_attempts DESC NULLS LAST
LIMIT :lim
```

```
    \"\"\"), {"lim": limit}).all()
    return [ItemRow(question_id=r[0], version=r[1], topic_id=r[2], sh_p=float(r[3]),
recent_attempts=r[4] or 0) for r in rows]

class SetSh(BaseModel): question_id:int; version:int; sh_p:float

@router.post("/exposure/set", dependencies=[Depends(require_roles("admin"))])
def set_sh(payload: SetSh, db: Session = Depends(get_db)):
    if payload.sh_p < 0 or payload.sh_p > 1: raise HTTPException(400, "sh_p must
be in [0,1]")
    row = db.get(ItemExposureControl, {"question_id": payload.question_id,
"version": payload.version})
    if row: row.sh_p = payload.sh_p
    else: db.add(ItemExposureControl(question_id=payload.question_id,
version=payload×version, sh_p=payload×sh_p))
    db.commit(); return {"ok": True}

class StartCalib(BaseModel):
    exam_code: str; tau: float = 0.2; n:int=400; test_len:int=25; iters:int=5;
alpha:float=0.6
    theta_dist:str="normal0,1"; floor:float=0.02; ceil:float=1.0
    topic_tau: Optional[dict] = None; topic_weights: Optional[dict] = None; dry_run:
bool = False

@router.post("/exposure/calibrate_sh/start",
dependencies=[Depends(require_roles("admin"))])
def start_calibration(payload: StartCalib):
    dsn = os×getenv("DATABASE_URL", "postgresql+psycopg2://
qbank:qbank@localhost:5432/qbank")
    run_id = str(uuid×uuid4())
    conn = psycopg2.connect(dsn); cur=conn×cursor()
    cur.execute("INSERT INTO
calibration_runs(id,exam_code,status,params,created_at) VALUES (%s,%s,%s,
%s,now())",
      (run_id, payload.exam_code, "queued", json.dumps(payload.model_dump())) )
    conn.commit(); cur.close(); conn.close()
    job = queue.enqueue(calibrate_job, payload.exam_code, dsn, payload.tau,
payload.n, payload.test_len, payload.iters,
                        payload.alpha, payload.theta_dist, payload.floor, payload.ceil,
                        payload.topic_tau, payload.topic_weights, payload.dry_run, run_id,
job_timeout=3600)
    return {"job_id": job.get_id(), "run_id": run_id}
```

qbank-backend/app/api/admin_runs.py

```
from fastapi import APIRouter, Depends, HTTPException, Query
from pydantic import BaseModel
from typing import List, Optional
from sqlalchemy.orm import Session
from sqlalchemy import text
from app.core.database import get_db
from app.core.auth import require_roles

router = APIRouter()

class RunRow(BaseModel):
    id: str; exam_code: str; status: str; created_at: str; started_at:
Optional[str]=None; finished_at: Optional[str]=None

@router.get("/exposure/calibrate_sh/runs", response_model=List[RunRow],
dependencies=[Depends(require_roles("admin"))])
def list_runs(exam_code: Optional[str] = None, start: Optional[str] = Query(None),
end: Optional[str] = Query(None),
              page: int = Query(1, ge=1), page_size: int = Query(25, ge=1, le=200), db:
Session = Depends(get_db)):
    where=[]; params={}
    if exam_code: where.append("exam_code = :exam");
params["exam"]=exam_code
    if start: where.append("created_at >= :start"); params["start"]=start
    if end: where.append("created_at < :end"); params["end"]=end
    where_sql=("WHERE "+ " AND ".join(where)) if where else ""
    offset=(page-1)*page_size
    q=f"""
      SELECT id::text, exam_code, status, created_at::text, started_at::text,
finished_at::text
      FROM calibration_runs {where_sql}
      ORDER BY created_at DESC LIMIT :lim OFFSET :off
    """
    params.update({"lim":page_size,"off":offset})
    rows=db.execute(text(q), params).all()
    return [RunRow(id=r[0], exam_code=r[1], status=r[2], created_at=r[3],
started_at=r[4], finished_at=r[5]) for r in rows]

class RunDetail(BaseModel):
    id: str; exam_code: str; status: str; params: dict; history: list
    result: Optional[dict]=None; error: Optional[str]=None; created_at: str;
```

started_at: Optional[str]=None; finished_at: Optional[str]=None

@router.get("/exposure/calibrate_sh/runs/{run_id}", response_model=RunDetail, dependencies=[Depends(require_roles("admin"))])
def run_detail(run_id: str, db: Session = Depends(get_db)):
    row=db.execute(text("""
      SELECT id::text, exam_code, status, params, history, result, error, created_at::text, started_at::text, finished_at::text
      FROM calibration_runs WHERE id=:id
    """), {"id": run_id}).first()
    if not row: raise HTTPException(404, "Run not found")
    return RunDetail(id=row[0], exam_code=row[1], status=row[2], params=row[3], history=row[4] or [], result=row[5], error=row[6], created_at=row[7], started_at=row[8], finished_at=row[9])

Prompt 5 — Admin UI (Next.js) with list, calibration start/progress, history w/ chart & CSV

Paste to Claude Code:

Create the admin UI (Next.js 14). Use minimal inline styles, no extra deps.

admin-ui/package.json

```
{
  "name": "admin-ui",
  "private": true,
  "version": "0.1.0",
  "scripts": { "dev": "next dev -p 4000", "build": "next build", "start": "next start -p 4000" },
  "dependencies": { "next": "14.2.5", "react": "18.2.0", "react-dom": "18.2.0" }
}
```

admin-ui/next.config.js

```
module.exports = { reactStrictMode: true };
```

admin-ui/pages/index.tsx

```
import { useState } from 'react';
const API = process.env.NEXT_PUBLIC_API || 'http://localhost:8000';
type ItemRow = { question_id:number; version:number; topic_id?:number;
```

```
sh_p:number; recent_attempts:number };
export default function Admin() {
  const [token, setToken] = useState(''); const [items, setItems] =
useState<ItemRow[]>([]);
  const headers = { 'Content-Type':'application/json', 'Authorization': `Bearer $
{token}` };
  const fetchItems = async () => { const r = await fetch(`${API}/v1/admin/exposure/
items?limit=200`, { headers }); setItems(await r.json()); };
  const setSh = async (qid:number, ver:number, sh_p:number) => { await fetch(`$
{API}/v1/admin/exposure/set`, { method:'POST', headers, body:
JSON.stringify({ question_id: qid, version: ver, sh_p }) }); await fetchItems(); };
  return (<main style={{padding:24}}>
    <h1>Admin: Sympson–Hetter</h1>
    <p>Paste an <b>admin</b> JWT</p>
    <textarea value={token} onChange={(e)=>setToken(e.target.value)} rows={4}
style={{width:'100%'}} />
    <div style={{marginTop:12}}><button onClick={fetchItems}>Load</button></
div>
    <table style={{marginTop:16, width:'100%', borderCollapse:'collapse'}}>
      <thead><tr><th>QID</th><th>Ver</th><th>Topic</th><th>sh_p</
th><th>Attempts(7d)</th><th>Save</th></tr></thead>
      <tbody>{items.map(it => (
        <tr key={`${it.question_id}-${it.version}`} style={{borderTop:'1px solid #eee'}}
>
          <td>{it.question_id}</td><td>{it.version}</td><td>{it.topic_id ?? ''}</td>
          <td><input type="number" min="0" max="1" step="0.05"
defaultValue={it.sh_p} onBlur={(e)=>setSh(it.question_id, it.version,
parseFloat(e.target.value))} /></td>
          <td>{it.recent_attempts}</td><td><button
onClick={()=>setSh(it.question_id, it.version, it.sh_p)}>Save</button></td>
        </tr>))}</tbody>
    </table></main>);
}
```

admin-ui/pages/calibration.tsx (start & live status)

```
import { useEffect, useState } from 'react';
const API = process.env.NEXT_PUBLIC_API || 'http://localhost:8000';
type Status = { state:string; current_iter:number; total_iters:number;
avg_exp?:number; max_over?:number; result?:any };
export default function Calibration() {
  const [token, setToken] = useState(''); const [jobId, setJobId] =
useState<string>(''); const [status, setStatus] = useState<Status | null>(null);
```

```
  const [form, setForm] = useState({ exam_code:'DEMO-EXAM', tau:0.2, n:400,
test_len:25, iters:5, alpha:0.6 });
  const headers = { 'Content-Type':'application/json', 'Authorization': `Bearer $
{token}` };
  const start = async () => { const r = await fetch(`${API}/v1/admin/exposure/
calibrate_sh/start`, { method:'POST', headers, body: JSON.stringify(form) }); const
data = await r.json(); setJobId(data.job_id); };
  useEffect(()=>{ const t = setInterval(async () => { if (!jobId) return; const r =
await fetch(`${API}/v1/admin/exposure/calibrate_sh/status?job_id=${jobId}`,
{ headers }); if (r.ok) setStatus(await r.json()); }, 1500); return () =>
clearInterval(t); }, [jobId, token]);
  const pct = status?.total_iters ? Math.round(100 * (status!.current_iter /
status!.total_iters)) : 0;
  return (<main style={{padding:24, maxWidth:800}}>
    <h1>Calibration</h1><p>Paste an <b>admin</b> JWT</p>
    <textarea value={token} onChange={(e)=>setToken(e.target.value)} rows={3}
style={{width:'100%'}} />
    <section style={{marginTop:16}}><h3>Parameters</h3>
      <div style={{display:'grid', gridTemplateColumns:'repeat(3, 1fr)', gap:12}}>
        <label>Exam <input value={form.exam_code}
onChange={(e)=>setForm({...form, exam_code:e.target.value})} /></label>
        <label>τ <input type="number" step="0.01" value={form.tau}
onChange={(e)=>setForm({...form, tau:parseFloat(e.target.value)})} /></label>
        <label>n <input type="number" value={form.n}
onChange={(e)=>setForm({...form, n:parseInt(e.target.value)})} /></label>
        <label>length <input type="number" value={form.test_len}
onChange={(e)=>setForm({...form, test_len:parseInt(e.target.value)})} /></label>
        <label>iters <input type="number" value={form.iters}
onChange={(e)=>setForm({...form, iters:parseInt(e.target.value)})} /></label>
        <label>α <input type="number" step="0.1" value={form.alpha}
onChange={(e)=>setForm({...form, alpha:parseFloat(e.target.value)})} /></label>
      </div><button style={{marginTop:12}} onClick={start}>Start</button>
    </section>
    {status && (<section style={{marginTop:24}}>
      <h3>Status: {status.state}</h3>
      <div style={{height:16, background:'#eee', borderRadius:8, overflow:'hidden'}}
><div style={{width:`${pct}%`, height:'100%', background:'#4a90e2'}} /></div>
      <p style={{marginTop:8}}>iter {status.current_iter} / {status.total_iters} ·
avg_exp {status.avg_exp?.toFixed(3)} · max_over {status.max_over?.toFixed(3)}</
p>
      {status.result && <pre style={{maxHeight:200, overflow:'auto',
background:'#fafafa', padding:12}}>{JSON.stringify(status.result, null, 2)}</pre>}
    </section>)}
  </main>);
```

```
}
```

admin-ui/pages/calibration-history.tsx (filters, pagination, chart, CSV)

```
import { useEffect, useMemo, useState } from 'react';
const API = process.env.NEXT_PUBLIC_API || 'http://localhost:8000';
type RunRow = { id:string; exam_code:string; status:string; created_at:string;
started_at?:string; finished_at?:string };
type DiffRow = { qid:number; ver:number; before:number; after:number;
delta:number };
type RunDetail = { id:string; exam_code:string; status:string; params:any;
history:any[]; result?:{updated:number; history:any[]; diff:DiffRow[]}; error?:string;
created_at:string; started_at?:string; finished_at?:string };
function toCSV(rows: DiffRow[]) { const h="qid,ver,sh_p_before,sh_p_after,delta\
\n"; const
b=rows.map(r=>[r.qid,r.ver,r.before.toFixed(6),r.after.toFixed(6),r.delta.toFixed(6)].
join(",")).join("\\n"); return h+b+"\\n"; }
function downloadCSV(filename:string, content:string){ const blob=new
Blob([content],{type:"text/csv;charset=utf-8;"}); const
url=URL.createObjectURL(blob); const a=document×createElement("a");
a×href=url; a×download=filename; a.click(); URL.revokeObjectURL(url); }
function LineChart({ points, width=520, height=180 }:{ points:
{x:number;y:number}[]; width?:number;height?:number }) {
  if (!points.length) return <svg width={width} height={height} />;
  const xs=points.map(p=>p.x), ys=points.map(p=>p.y);
  const minX=Math.min(...xs), maxX=Math.max(...xs); const
minY=Math.min(...ys,0), maxY=Math.max(...ys,0.001); const pad=24;
  const sx=(x:number)=> pad + ((x-minX)/Math.max(1,(maxX-
minX)))*(width-2*pad);
  const sy=(y:number)=> height - pad - ((y-minY)/Math.max(1e-9,(maxY-
minY)))*(height-2*pad);
  const path=points×map((p,i)=> (i===0?`M ${sx(p.x)} ${sy(p.y)}`:`L ${sx(p.x)} $
{sy(p.y)}`)).join(" ");
  const xTicks=Array.from(new Set(points.map(p=>p.x))); const yTicks=[minY,
(minY+maxY)/2,maxY];
  return (<svg width={width} height={height}>
    <rect x={0} y={0} width={width} height={height} fill="#fff" stroke="#e5e7eb" />
    <line ×1={pad} y1={height-pad} ×2={width-pad} y2={height-pad}
stroke="#9ca3af" />
    <line ×1={pad} y1={pad} ×2={pad} y2={height-pad} stroke="#9ca3af" />
    {xTicks.map((t,i)=>(<text key={i} x={sx(t)} y={height-pad+12} fontSize={10}
textAnchor="middle">{t}</text>))}
    {yTicks.map((t,i)=>(<g key={i}><line ×1={pad-4} y1={sy(t)} ×2={pad} y2={sy(t)}
```

```
stroke="#9ca3af" /><text x={4} y={sy(t)} fontSize={10}
dominantBaseline="middle">{t.toFixed(3)}</text></g>))}
    <path d={path} fill="none" stroke="#4a90e2" strokeWidth={2} />
    {points.map((p,i)=>(<circle key={i} cx={sx(p.x)} cy={sy(p.y)} r={2.5}
fill="#1f77b4" />))}
    <text x={pad} y={16} fontSize={12} fontWeight={600}>max_over vs iteration</
text>
  </svg>);
}
export default function CalibHistory() {
  const [token, setToken] = useState(''); const [runs, setRuns] =
useState<RunRow[]>([]); const [selected, setSelected] = useState<RunDetail |
null>(null);
  const [filters, setFilters] = useState({ exam_code:'', start:'', end:'', page:1,
page_size:25 });
  const headers = { 'Content-Type':'application/json', 'Authorization': `Bearer $
{token}` };
  const loadRuns = async () => { const qs = new URLSearchParams(); if
(filters.exam_code) qs.set('exam_code', filters.exam_code); if (filters.start)
qs.set('start', filters.start); if (filters.end) qs.set('end', filters.end); qs.set('page',
String(filters.page)); qs.set('page_size', String(filters.page_size)); const r = await
fetch(`${API}/v1/admin/exposure/calibrate_sh/runs?`+qs.toString(), { headers }); if
(r.ok) setRuns(await r.json()); };
  const loadRun = async (id:string) => { const r = await fetch(`${API}/v1/admin/
exposure/calibrate_sh/runs/${id}`, { headers }); if (r.ok) setSelected(await
r.json()); };
  useEffect(()=>{ if (token) loadRuns(); }, [token]);
  useEffect(()=>{ if (token) loadRuns(); }, [filters.exam_code, filters.start,
filters.end, filters.page, filters.page_size]);
  const points = useMemo(()=> selected?.history?.map((h:any)=> ({ x: h.iter, y:
Number(h.max_over || 0) })) || [], [selected?.history]);
  const exportCSV = () => { if (!selected?.result?.diff?.length) return; const csv =
toCSV(selected×result×diff); downloadCSV(`calibration_diff_${selected.id}.csv`,
csv); };
  return (<main style={{padding:24, display:'grid', gridTemplateColumns:'1fr 1fr',
gap:24}}>
    <section>
      <h1>Calibration Runs</h1><p>Paste an <b>admin</b> JWT</p>
      <textarea value={token} onChange={(e)=>setToken(e.target.value)} rows={3}
style={{width:'100%'}} />
      <div style={{display:'grid', gridTemplateColumns:'1fr 1fr', gap:12,
marginTop:12}}>
        <label>Exam <input value={filters.exam_code}
onChange={(e)=>setFilters({...filters, exam_code:e.target.value, page:1})} /></
```

```
label>
      <label>Page size <input type="number" value={filters.page_size}
onChange={(e)=>setFilters({...filters, page_size:parseInt(e.target.value)||25,
page:1})}/></label>
      <label>Start (ISO) <input placeholder="2025-08-01" value={filters.start}
onChange={(e)=>setFilters({...filters, start:e.target.value, page:1})} /></label>
      <label>End (ISO) <input placeholder="2025-08-31" value={filters.end}
onChange={(e)=>setFilters({...filters, end:e.target.value, page:1})} /></label>
    </div>
    <div style={{marginTop:8}}>
      <button onClick={()=>setFilters({...filters, page: Math.max(1, filters.page-1)})}
>Prev</button>
      <span style={{margin:'0 8px'}}>Page {filters.page}</span>
      <button onClick={()=>setFilters({...filters, page: filters.page+1})}>Next</
button>
      <button style={{marginLeft:12}} onClick={loadRuns}>Refresh</button>
    </div>
    <table style={{marginTop:12, width:'100%', borderCollapse:'collapse'}}>
      <thead><tr><th>Started</th><th>Exam</th><th>Status</th><th>Run</th></
tr></thead>
      <tbody>{runs.map(r => (<tr key={r.id} style={{borderTop:'1px solid #eee'}}>
        <td>{r.started_at || r.created_at}</td><td>{r.exam_code}</td><td>{r.status}
</td><td><button onClick={()=>loadRun(r.id)}>View</button></td>
      </tr>))}</tbody>
    </table>
  </section>
  <section>
    <h1>Details</h1>
    {!selected && <p>Select a run</p>}
    {selected && (<div>
      <p><b>ID:</b> {selected.id}</p><p><b>Exam:</b> {selected.exam_code} —
<b>Status:</b> {selected.status}</p>
      <p><b>Window:</b> {selected.started_at} → {selected.finished_at}</p>
      <h3>max_over chart</h3><LineChart points={points} />
      <h3 style={{marginTop:12}}>Params</h3><pre style={{background:'#fafafa',
padding:12, maxHeight:180, overflow:'auto'}}>{JSON.stringify(selected.params,
null, 2)}</pre>
      <h3>History</h3><pre style={{background:'#fafafa', padding:12,
maxHeight:200, overflow:'auto'}}>{JSON.stringify(selected.history, null, 2)}</pre>
      <h3>Diff (before/after sh_p)</h3>
      <div style={{display:'flex', gap:8, alignItems:'center'}}><button
onClick={exportCSV}>Export CSV</button><span style={{color:'#666'}}>rows:
{selected.result?.diff?.length || 0}</span></div>
      <pre style={{background:'#fafafa', padding:12, maxHeight:200,
```

```
overflow:'auto'}}>
{`qid,ver,sh_p_before,sh_p_after,delta
`}{selected.result?.diff?.slice(0,10)?.map((d:any)=>`${d.qid},${d.ver},$
{d.before.toFixed(4)},${d.after.toFixed(4)},${d.delta.toFixed(4)}`).join("\\n")}
{selected.result?.diff?.length>10 ? "\\n… (see CSV for full list)" : ""}
        </pre>
        {selected.error && (<><h3>Error</h3><pre style={{background:'#fff0f0',
padding:12}}>{selected.error}</pre></>)}
      </div>)}
    </section>
  </main>);
}
```

Prompt 6 — calibration status endpoint for UI polling

Paste to Claude Code:

Add status endpoint the UI polls.

qbank-backend/app/api/admin_status.py (optional; if not present in your copy)

```
from fastapi import APIRouter, Depends
from pydantic import BaseModel
from rq.job import Job
from app.core.auth import require_roles
from app.jobs.queue import redis

router = APIRouter()

class CalibStatus(BaseModel):
    state: str
    current_iter: int
    total_iters: int
    avg_exp: float | None = None
    max_over: float | None = None
    result: dict | None = None

@router.get("/exposure/calibrate_sh/status", response_model=CalibStatus,
dependencies=[Depends(require_roles("admin"))])
def calib_status(job_id: str):
    job = Job×fetch(job_id, connection=redis)
    meta = job×meta or {}
    state = meta×get("state") or job.get_status()
    return CalibStatus(
```

```
        state=state,
        current_iter=int(meta×get("current_iter") or 0),
        total_iters=int(meta×get("total_iters") or 0),
        avg_exp=float(meta×get("avg_exp") or 0.0) if meta.get("avg_exp") is not
None else None,
        max_over=float(meta.get("max_over") or 0.0) if meta.get("max_over") is not
None else None,
        result=job×result if state == "done" else None
    )
```

Update main to include it:

qbank-backend/app/main.py — add:

```
from app.api.admin_status import router as status_router
app.include_router(status_router, prefix="/v1/admin", tags=["calibration-status"])
```

Prompt 7 — tests & CI (optional but recommended)

Paste to Claude Code:

Add a minimal backend e2e test and CI.

qbank-backend/tests/test_api_smoke.py

```
import pytest
import httpx
BASE = "http://localhost:8000"
def test_health():
    r = httpx.get(f"{BASE}/health"); assert r.status_code==200
def test_login_and_seed():
    r = httpx×post(f"{BASE}/v1/auth/mock-login", json={"user_id":"tester","roles":
["author","publisher","student","admin"]})
    assert r.status_code==200; token=r.json()["access_token"];
hdr={"Authorization":f"Bearer {token}"}

payload={"external_ref":"E2E-1","topic_name":"Cardiology","exam_code":"DEMO-
EXAM","stem_md":"Stem","lead_in":"Pick
one","rationale_md":"Because","difficulty_label":"medium","options":
[{"label":"A","text_md":"Alpha","is_correct":True},
{"label":"B","text_md":"Bravo","is_correct":False}]}
    r = httpx.post(f"{BASE}/v1/author/questions", headers=hdr|{"Content-
Type":"application/json"}, json=payload); assert r.status_code==200
```

.github/workflows/backend.yml

```
name: backend-ci
on: { push: { paths: ["qbank-backend/**","sql/**",".github/workflows/
backend.yml"] }, pull_request: { paths: ["qbank-backend/**","sql/**",".github/
workflows/backend.yml"] } }
jobs:
  api:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:16-alpine
        env: { POSTGRES_USER: qbank, POSTGRES_PASSWORD: qbank,
POSTGRES_DB: qbank }
        ports: ["5432:5432"]
      redis:
        image: redis:7-alpine
        ports: ["6379:6379"]
    env:
      DATABASE_URL: postgresql+psycopg2://qbank:qbank@localhost:5432/qbank
      REDIS_URL: redis://localhost:6379/0
      APP_SECRET: ci-secret
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: "3.11" }
      - name: Install deps
        run: pip install -r qbank-backend/requirements.txt
      - name: Init DB schema
        run: |
          psql postgresql://qbank:qbank@localhost:5432/qbank -f sql/content_ddl.sql
          psql postgresql://qbank:qbank@localhost:5432/qbank -f sql/
item_exposure_control.sql
          psql postgresql://qbank:qbank@localhost:5432/qbank -f sql/
feature_flags.sql
          psql postgresql://qbank:qbank@localhost:5432/qbank -f sql/
calibration_runs.sql
      - name: Launch API
        run: |
          nohup python -m uvicorn qbank-backend.app.main:app --host 0.0.0.0 --port
8000 &
          sleep 3
```

```
      - name: Pytest
        run: pytest -q qbank-backend/tests/test_api_smoke.py
```

Prompt 8 — run instructions (dev)

Paste to Claude Code:

Add a dev helper doc.

DEV.md

Dev setup
1) Postgres + Redis running; export `DATABASE_URL` and `REDIS_URL` if non-default.
2) Apply schema:

```
psql "$DATABASE_URL" -f sql/content_ddl.sql
psql "$DATABASE_URL" -f sql/item_exposure_control.sql
psql "$DATABASE_URL" -f sql/feature_flags.sql
psql "$DATABASE_URL" -f sql/calibration_runs.sql
```

3) Backend:

```
cd qbank-backend
pip install -r requirements.txt
uvicorn app.main:app --reload
```

4) Worker:

```
python -m app.jobs.worker
```

5) Admin UI:

```
cd admin-ui
npm install
npm run dev # http://localhost:4000
```

6) Get admin JWT:

POST /v1/auth/mock-login {"user_id":"admin","roles":
["admin","author","publisher","student"]}

7) Start calibration in `/calibration`; review history & export CSV in `/calibration-history`.