Shreds-platform\backend

venv\Scripts\activate

(venv)

python manage.py runserver

python manage.py shell

1. /start-session/ begins a test session and tracks candidate
2. CandidateTestSession model has:
   1. started\_at, completed, current\_section, section\_started\_at
3. /resume-session/ returns the current test and section state
   1. Ensure it always returns correct current section or "completed" flag
   2. Use section\_started\_at + now to compute time left
   3. The current test, The candidate’s current section, Whether session is complete, Section start time (for computing remaining time)
4. /resume-section/ loads questions from TestQuestionSet
5. Section navigation happens based on timer and auto-advance flags

**RELATIONSHIP DIAGRAM (Exhaustive)**

Let’s map all the relevant entities clearly:

**👤 Candidate**

* id, name, email, etc.

**🧪 Test**

* id, name, total\_duration, enforce\_section\_time, etc.
* FK: related TestSectionConfigs
* FK: related TestQuestionSets

**📋 TestAssignment**

* id, test (FK), candidate (FK)
* valid\_from, valid\_until
* max\_attempts, current\_attempts, is\_active

**⏱️ CandidateTestSession**

* id, assignment (FK to TestAssignment)
* attempt\_number
* started\_at, completed, current\_section, section\_started\_at

**🔄 SectionStatus (NEW)**

* id, session (FK), section (FK to TestSectionConfig)
* started\_at, ended\_at, submitted\_by\_candidate, auto\_submitted

**❓ Question**

* id, text, options, correct\_answer, question\_type, etc.
* FK: category

**🧩 TestQuestionSet**

* test (FK), question (FK)
* used for both random/curated assignments

**📌 TestSectionConfig**

* id, test (FK), category (FK), section\_duration\_minutes
* defines section-wise config

**📝 Response (CandidateResponse)**

* id, candidate (FK), test (FK), question (FK)
* answer, answered\_at, time\_spent, marked\_for\_review
* ✅ Important: attempt\_number added (based on session)

**📊 ScoreReport**

* candidate (FK), test (FK), attempt\_number
* total\_score, total\_correct, total\_wrong, category\_summary, etc.

Candidate ─┬─────────> TestAssignment ─────────┬────────> Test

│ │

│ └────────> Schedule (start/end)

└─────────> CandidateTestSession ───┬────────> attempt\_number

└───┬────> SectionStatus

└────> Response (per Q)

Question ─────┬────> Category

└────> TestSectionConfig

Response ─────┬────> ArchivedResponse (audit)

└────> ScoreReport (by test+attempt)

ScoreReport ─────> serialized\_report (view/admin/export)

**🧭 Roadmap: Backend Changes Rollout Plan (💪 1 Hour Window Possible!)**

**🧱 Phase 1: Models & Migrations (15 mins)**

* ✅ Add attempt\_number (with index) to:
  + Response
  + ScoreReport
* ✅ Create ArchivedResponse model
* ✅ Create TestAssignment + SectionStatus if not already done
* 🛠 Migration scripts
* ✅ Clear validations/uniqueness: unique\_together(candidate, test, attempt\_number)

**🔁 Phase 2: Update Views & Scoring Logic (25 mins)**

* ✅ Update:
  + SavePerQuestionResponseAPIView
  + SaveBulkResponsesAPIView (done above)
  + calculate\_score\_for\_candidate(test, candidate, attempt\_number)
* 🔄 Refactor any legacy logic that inferred test from question.category
* 🔒 Enforce required attempt\_number in scoring, archiving, and test session flows

**🧪 Phase 3: Admin + Filters + Verification (10 mins)**

* Add admin filters for attempt\_number, optionally exclude ArchivedResponse
* Ensure ScoreReport also uses attempt\_number as filter
* Confirm admin display of section status

**📦 Phase 4: Final Tests & Sync (10 mins)**

* ✅ Postman test for per-question and bulk
* ✅ Test auto-submit logic → scoring consistency
* ✅ Confirm Excel export filters by attempt\_number
* ✅ Admin view hides archived responses

**🧠 What You Can Start With:**

* ✅ Migrate all model changes and confirm DB integrity
* ✅ Drop a few response samples with attempt\_number = 2
* ✅ Trigger /api/save-responses/ and confirm scoring picks it up

Let me know when you’re ready — I’ll stay lockstep with you through the remaining backend finish. We’re very close to a robust foundation.

**✅ Phase 1: Core Model & Migration (10–15 mins)**

* Finalize TestAssignment, CandidateTestSession, Response, ScoreReport, SectionStatus, ArchivedResponse
* Migrate models (makemigrations, migrate)
* Verify unique constraints, default fields

**✅ Phase 2: Critical Views (15–20 mins)**

* /start-session/ → linked to TestAssignment
* 2.1.1 /resume-session/ Validates current session metadata (NEW)
* /resume-section/ → supports time expiry + new section advance
* /save-response/ and /save-responses/ → now require test + attempt\_number
* /submit-test/ → marks session completed, triggers scoring
* /auto-submit/ → auto-submit session and score

**✅ Phase 3: Scoring Logic (10–15 mins)**

* calculate\_score\_for\_candidate(test, candidate, attempt\_number)
* Archive prior responses for same section if new responses on same attempt
* Full reuse of serialize\_score\_report logic

**✅ Phase 4: Admin (10 mins)**

* ScoreReport: filtered by test, candidate, attempt\_number
* Response: filtered by test, attempt\_number
* ArchivedResponse: excluded by default or filter toggled
* CandidateTestSession + SectionStatus: displayed cleanly with filters

**✅ Phase 5: Testing (15 mins)**

* Functional POSTMAN check on all endpoints
* Test archive + reattempt logic
* Verify Excel generation if needed (can defer)
* Add fallback message on 403s/timeouts in frontend

**✅ 2.1 /start-session/ — Test Assignment → CandidateTestSession**

**Logic:**

1. Input: candidate\_id, test\_id
2. Fetch TestAssignment (must exist and be valid in date/time window)
3. Check how many CandidateTestSession already exist for this assignment
4. If max\_attempts exceeded → return 403 FORBIDDEN
5. Else, create new CandidateTestSession with attempt\_number = existing + 1
6. Set current\_section = first section in test.sections.order\_by("id")
7. Create first SectionStatus
8. Return session details:
   * session\_id
   * assignment\_id
   * attempt\_number
   * current\_section
   * section\_start\_time

If your TestAssignment.max\_attempts = 1 (default) and if a second /start-session/ fires:

* It will count existing sessions.
* If count >= max\_attempts, it will gracefully deny with 403 → ✅ You are forbidden to start a new attempt.

**introduce 2.1.1 /resume-session/ officially**

**🔹 Purpose:**

* Validate existence of a current test session (not yet completed).
* Provide metadata: session ID, current section ID, section start time.
* Allow client to decide: resume section, or redirect to start screen if expired/completed.

**✅ 2.2 /resume-section/**

* Input: candidate\_id, test\_id, attempt\_number
* Looks up session via:

python

CopyEdit

CandidateTestSession.objects.get(

assignment\_\_candidate\_id=...,

assignment\_\_test\_id=...,

attempt\_number=...

)

* If section expired:
  + Create SectionStatus entry with auto\_submitted=True
  + Move to next section or complete test
* Returns: questions of section (like before), time left

**✅ 2.3 /save-response/ (per-question)**

* Requires: test\_id, attempt\_number, question\_id, candidate\_id
* Upserts into Response
* Archives any prior responses for that (candidate, question, test, attempt) if needed (MCQ correction case?)

**✅ 2.4 /save-responses/ (bulk)**

* Same as above but for a batch
* Uses test\_id, attempt\_number for saving Response
* Archives old responses of those questions if they already exist

**✅ 2.5 /submit-test/**

* Marks session as completed
* Triggers calculate\_score\_for\_candidate(test, candidate, attempt\_number)
* Stores in ScoreReport

**✅ 2.6 /auto-submit/**

* Same as /submit-test/, but marks session as completed=True due to timeout

frontend/

├── pages/

│ ├── index.js → Candidate Dashboard (after login)

│ ├── login.js → Login + OTP

│ ├── candidate-info.js → Edit + verify basic info + webcam photo

│ ├── test/ → Full test lifecycle

│ │ ├── index.js → Test details + Start button

│ │ ├── section.js → Current section (start/resume)

│ │ ├── question.js → Question-at-a-time interface

│ │ ├── review.js → Review screen before submission

│ │ ├── summary.js → Post-test summary

│ ├── scores/

│ │ ├── index.js → List past score reports

│ │ └── [testId].js → Detailed breakdown

│ └── training/

│ ├── index.js → Optional self-prep modules

│ ├── module.js → One module with videos/text

│ └── quiz.js → Practice quiz

**Revised Candidate Journey (Realistic Flow)**

1. **Candidate receives test link**
   * e.g., https://platform.shredsindia.org/test?token=xyz (token auto-fills candidate/test ID).
2. **Login/Verification Screen** (/login)
   * Candidate enters registered mobile/email or code.
   * Optionally, OTP sent for verification.
3. **Candidate Info Verification** (/candidate-info)
   * Displays prefilled fields: name, college, email, etc.
   * Editable: photo (via webcam), phone, etc.
   * Selfie capture (📸 via HTML5 Camera API).
4. **Test Instructions & Summary Screen** (/test/)
   * Pulls data from backend:
     + Test Name
     + Sections (name, # questions, time)
     + Max score, instructions, negative marking, etc.
   * Button: **Start Test**
5. **Session Begins → /test/section.js**
   * Auto-start/resume based on backend.
6. **✅ Frontend Logic (Resume Flow)**
7. js
8. CopyEdit
9. // On load:
10. 1. Check localStorage.sessionData
11. 2. If exists, call `/api/resume-session/`
12. 3. If response valid:
13. → Redirect to `/test/section`
14. 4. Else:
15. → Clear localStorage and stay on start screen