API End Points

1. Auth & User APIs

```
Base: /auth/

    POST   /auth/register/

    POST   /auth/login/

    POST   /auth/logout/

    GET    /auth/profile/

    PUT    /auth/profile/
```

Purpose:

- Authentication
- User role handling (test-maker / student)
- Token-based auth (JWT recommended)

2. Test Maker (Admin/Test Creator) APIs

```
Base: /tests/
```

Test CRUD

```
POST   /tests/                     → create test

GET    /tests/my/                  → list tests created by me

GET    /tests/{test_id}/           → test details

PUT    /tests/{test_id}/           → update test

DELETE /tests/{test_id}/           → delete test
```

Passcode Control (Important)

```
PUT    /tests/{test_id}/passcode/
```

```
Payload:

    {

      "passcode": "NEW1234"

    }
```

Purpose:

- Test maker can **change passcode anytime**
- Immediately invalidates old access

3. Question Bank APIs

```
Base: /questions/

    POST   /questions/                    → add question

    GET    /questions/?test_id=           → list test
    questions

    PUT    /questions/{question_id}/      → update question

    DELETE /questions/{question_id}/      → delete question
```

Supports:

- MCQ
- MSQ
- NAT
- Explanation field

4. Test Access (Student Side – Secure)

```
Base: /access/

    Access via Code or Link

    POST   /access/validate/


Payload:

    {

      "test_code": "ABC123",

      "passcode": "9999"

    }
```

```
Response:

    {
      "access_granted": true,
      "test_id": 42,
      "session_token": "uuid-session-token"
    }
```

Rules:

- No passcode → no access
- Passcode change → old sessions invalidated

5. Test Session APIs

```
Base: /session/

    POST   /session/start/
    POST   /session/submit/
    GET    /session/{session_id}/result/
```

Start Test

```
Payload:

    {
      "test_id": 42,
      "session_token": "uuid-session-token"
    }
```

Submit Test

```
Payload:

    {
      "session_id": 77,
      "answers": {
        "q1": "A",
        "q2": ["A","C"],
        "q3": 42
      }
    }
```

6. Result & Analytics APIs

```
Base: /results/

    GET  /results/my/                         → student results

    GET  /results/test/{test_id}/             → test-wise
    analytics (creator)
```

```
Provides:
```

- `Score`
- `Accuracy`
- `Time taken`
- `Avg time per question`

7. Utility / Security APIs

```
POST /tests/{test_id}/lock/

POST /tests/{test_id}/unlock/
```

Purpose:

- Temporarily disable test access
- Useful during misuse or maintenance

8. API Ownership Rules (Important)

Role	Allowed APIs
Test Maker	create/update/delete tests & questions
Student	access, attempt, submit, view own results
Admin	full access

9. Recommended App-wise Routing

```
auth/

tests/

questions/

access/

session/

results/
```

```
Clean. Scalable. Maintainable.
```

CSS Rules

1. Core Philosophy (THIS IS NON-NEGOTIABLE)

Your project follows **Mobile-First + Layout Primitives**.

That means:

- Mobile styles = **default**
- Desktop styles = **only inside media queries**
- Layout responsibility is **strictly separated**

If these rules are followed, your UI will stay clean even at 100+ pages.

---

2. Mental Model (Most Important)

Think in 3 layers only

```
.page      → controls page-level structure (height, stacking)
.section   → controls vertical spacing only
.container → controls width only
```

Nothing else.
No color, no background mixing, no layout hacks.

---

3. What Controls What (Very Clear)

3.1 .page — PAGE CONTROLLER

Purpose

- Represents one full screen / route
- Controls height and page-level layout behavior

Allowed

- min-height
- flex layout (if needed)
- page-level stacking

Not allowed

- padding
- background colors
- width control

Correct usage

```
<div class="page">
  ...
</div>
```

Rule

- Every route MUST have exactly **one .page root**

3.2 .section — VERTICAL SPACING ONLY

Purpose

- Controls **vertical rhythm**
- Separates logical sections (hero, content, footer blocks)

Allowed

- padding-block only

Not allowed

- background colors
- width
- flex/grid
- margins

Correct usage

```
<section class="section">
  <div class="container">
    ...
  </div>
</section>
```

Variants

- `.section → normal spacing`
- `.section--tight → compact spacing`

This keeps spacing consistent across the entire app.

3.3 .container — WIDTH CONTROLLER

Purpose

- Controls readable width
- Centers content
- Adds horizontal padding for mobile

Allowed

- max-width
- margin-inline
- padding-inline

Not allowed

- vertical padding
- background colors

Golden rule

> Backgrounds always go **outside** the container.

Correct:

```
<section class="section bg-primary-soft">
  <div class="container">
    ...
  </div>
</section>


Wrong:

<div class="container bg-primary-soft"> ❌
```

---

4. Mobile-First Rules (Lock This In)

4.1 Default = Mobile

Your CSS already does this correctly.

- grid-template-columns: 1fr
- Single-column layouts
- Touch-friendly spacing

4.2 Desktop Enhancements Only Here

```
@media (min-width: 1024px) {
  .grid-2 { grid-template-columns: repeat(2, 1fr); }
  .grid-3 { grid-template-columns: repeat(3, 1fr); }
}
```

Rule:

- **Never** reduce spacing on desktop
- Desktop only **adds layout**, not removes comfort

---

5. How Components Should Be Built

Components NEVER control:

- Page spacing
- Width
- Global layout

Components MAY control:

- Internal spacing
- Background
- Borders
- Shadows

Example (correct):

```
<div class="card">
  <h3>Title</h3>
  <p>Description</p>
</div>


Example (wrong):

<div class="card section container"> ❌
```

---

6. Recommended Additions to main.css (Small but Powerful)

6.1 Page Content Wrapper (Optional but Clean)

Add this:

```
.page-content {
  display: flex;
  flex-direction: column;
  gap: var(--space-6);
}


Usage:

<div class="page">
  <div class="page-content">
    <section class="section">...</section>
    <section class="section">...</section>
  </div>
</div>
```

This avoids random margins everywhere.

6.2 Safe Full-Height Centering (For Login / Access Pages)

Add:

```
.page-center {
  min-height: 100vh;
  display: flex;
  align-items: center;
}


Usage:

<div class="page page-center">
  <div class="container">
    <div class="card">Login</div>
  </div>
</div>
```

6.3 Mobile Tap Safety (Optional but Professional)

button,

```
a {
  min-height: 44px;
}
```

This improves mobile UX without changing layout.

---

7. Naming & Discipline Rules (Very Important)

- `.page-*` → page-level only
- `.section-*` → spacing variants only
- `.card-*` → components
- `.btn-*` → buttons
- `.text-*` / `.bg-*` → utilities

Never mix responsibilities in one class.

---

8. How Your React Pages Should Look (Canonical Pattern)

```
export default function TestAccessPage() {
  return (
    <div className="page">
      <section className="section">
        <div className="container">
          <div className="card">
            {/* content */}
          </div>
        </div>
      </section>
    </div>
  );
}
```

If every page follows this, your UI will scale cleanly.

---

9. Final Verdict on Your `main.css`

Honest assessment:

- Design tokens: excellent
- Spacing scale: correct
- Layout primitives: senior-level
- Mobile-first: correctly implemented

- Separation of concerns: very clean

This is **production-grade CSS**, not tutorial CSS.

NO MAGIC NUMBERS ARE ALLOWED IN CSS.

That means:

- No raw `px`, `rem`, `%`, `vh`, `vw` values inside components
- No ad-hoc margins, paddings, font sizes, widths, heights
- **All values must come from variables (design tokens)**

If a value is needed and no variable exists → **create a variable first.**

---

1. Single Source of Truth

All layout, spacing, sizing, typography, and colors must come from:

```
:root {

  /* spacing */

  --space-1 … --space-7


  /* typography */

  --font-size-base

  --line-height-base


  /* layout */

  --container-max-width


  /* color system */

  --color-*


  /* elevation */

  --shadow-*

}
```

Nothing bypasses this layer.

---

2. Page / Section / Container Control (Reconfirmed)

`.page`

- Controls: height, stacking
- Must not define spacing or width

`.section`

- Controls: vertical spacing ONLY
- Uses:

```
padding-block: var(--space-5 | --space-6 | --space-7)
```

`.container`

- Controls: width ONLY
- Uses:

```
max-width: var(--container-max-width);

padding-inline: var(--space-3 | --space-4);
```

No numeric values allowed.

---

3. Spacing Rules (VERY STRICT)

❌ Not allowed

```
margin-top: 20px;

padding: 12px;

gap: 10px;
```

✅ Correct

```
margin-top: var(--space-4);
```

```
padding: var(--space-3);

gap: var(--space-2);
```

---

4. Font Sizes & Line Heights

❌ **Not allowed**

```
font-size: 18px;

line-height: 1.7;
```

✅ **Correct**

```
font-size: var(--font-size-base);

line-height: var(--line-height-base);
```

If a new scale is required:

```
:root {

  --font-size-sm: 0.875rem;

  --font-size-lg: 1.125rem;

}
```

Then use the variable.

---

5. Border Radius, Shadows, Widths

Border Radius

❌

```
border-radius: 10px;
```

✅

```
border-radius: 0.5rem; /* allowed ONLY if defined as token */
```

Recommended addition:

```
:root {
  --radius-sm: 0.375rem;
  --radius-md: 0.5rem;
  --radius-lg: 0.75rem;
}
```

Then:

```
border-radius: var(--radius-md);
```

Shadows

```
Already correct:
box-shadow: var(--shadow-sm);
```

No custom shadows allowed.

---

6. Heights, Widths & Positioning

❌ **Not allowed**

```
height: 300px;
width: 80%;
top: 12px;
```

✅ **Allowed patterns**

- Content-driven sizing
- Flex/grid alignment
- Token-based sizes

If fixed sizing is absolutely required:

```
:root {
  --size-avatar: 250px;
}
```

Then:

```
width: var(--size-avatar);
height: var(--size-avatar);
```

---

7. Responsive Rules (No Breakpoint Magic Numbers)

❌ **Not allowed**

```
@media (min-width: 768px)
```

✅ **Correct (tokenized breakpoints)**

Add once:

```
:root {
  --bp-desktop: 1024px;
}

Then:
@media (min-width: var(--bp-desktop)) {
  ...
}
```

---

8. Component Authoring Rule

Components must:

- Use only variables
- Never invent spacing
- Never affect page layout
- Be composable inside `.section > .container`

If a component needs new spacing:

1. Add token
2. Document token
3. Use token

---

9. Utility Classes Are Also Bound by This Rule

❌

```
.mt-10 { margin-top: 10px; }
```

✅

```
.mt-4 { margin-top: var(--space-4); }
```

---

10. CSS Review Checklist (MANDATORY)

Before any CSS is accepted:

  - `No raw numbers (px, rem, %, vh, vw)`
  - `All spacing via --space-*`
  - `All colors via --color-*`
  - `All layout via .page → .section → .container`
- Desktop styles only inside media queries

- Mobile-first defaults respected

---

11. Why This Rule Exists (Rationale)

- Prevents inconsistency
- Makes global redesign trivial
- Enables theming
- Keeps mobile UX predictable
- Scales to large teams and long-lived projects

This is **how design systems stay alive for years**.

---

12. Status

CSS Standard: LOCKED

Violations are refactors, not exceptions.

Database models

FINAL DATABASE DESIGN

1. User & Role (RBAC – future safe)

```
users
id (PK)
name
email (UNIQUE)
password_hash
is_active
created_at
updated_at


roles
id (PK)
name (UNIQUE)   // SUPER_ADMIN, ADMIN, TEST_MAKER, STUDENT


user_roles
id (PK)
user_id (FK → users.id)
role_id (FK → roles.id)

UNIQUE(user_id, role_id)
```

Why:

- A user can be **both test maker + student**
- Avoids role explosion later

2. Test (Immutable intent, mutable config)

tests

```
id (PK)
title
description
created_by (FK → users.id)

duration_minutes
```

```
    total_marks

    start_time (nullable)
    end_time (nullable)

    status              // DRAFT, PUBLISHED, ARCHIVED
    is_active           // hard kill switch

    created_at
    updated_at
```

Rules:

- ARCHIVED = visible but not attemptable
- is_active = false = instant shutdown (even mid-test)

3. Test Access & Security (VERY IMPORTANT)

test_access

```
    id (PK)
    test_id (FK → tests.id)

    access_code (UNIQUE)      // short code or UUID
    passcode_hash

    passcode_version         // integer (starts from 1)

    is_passcode_required
    max_attempts_per_user

    created_at
    updated_at
```

test_passcode_history

```
    id (PK)
    test_id (FK → tests.id)

    passcode_hash
    passcode_version
    changed_by (FK → users.id)
```
changed_at

Why this matters:

- If passcode changes **during an attempt**, you still know:
  - Which version user used
- Enables:
  - Audit
  - Dispute resolution
  - Security compliance

4. Question Bank (Reusable forever)

questions

```
id (PK)
question_text
question_type         // MCQ, MSQ, NAT
difficulty            // EASY, MEDIUM, HARD
explanation

created_by (FK → users.id)
created_at
updated_at
```

options

```
id (PK)
question_id (FK → questions.id)

option_text
is_correct


Rules:

  ● MCQ → exactly 1 correct
  ● MSQ → 1+ correct
  ● NAT → no options

Enforced at service layer (not DB)
```

5. Test–Question Mapping (Critical)

test_questions

```
id (PK)
test_id (FK → tests.id)
question_id (FK → questions.id)

marks
question_order

UNIQUE(test_id, question_id)
```

Why:

- Same question in multiple tests
- Same question with **different marks**
- Stable ordering

6. Attempts (This is the backbone)

```
test_attempts
id (PK)
test_id (FK → tests.id)
user_id (FK → users.id)

attempt_number
passcode_version_used

started_at
last_activity_at
submitted_at

status     // IN_PROGRESS, SUBMITTED, EXPIRED, FORCE_SUBMITTED
score

created_at
updated_at

UNIQUE(test_id, user_id, attempt_number)


Corner cases handled:

  ● App crash → resume using IN_PROGRESS
  ● Time over → EXPIRED
```

- Admin stops test → FORCE_SUBMITTED
- Passcode change mid-test → version tracked

7. Answers (Precise & Safe)

responses

```
id (PK)
attempt_id (FK → test_attempts.id)
question_id (FK → questions.id)

numerical_answer   // for NAT
is_correct
marks_obtained

created_at
updated_at

UNIQUE(attempt_id, question_id)
```

response_options

```
id (PK)
response_id (FK → responses.id)
option_id (FK → options.id)

UNIQUE(response_id, option_id)
```

Why split:

- MCQ → 1 row
- MSQ → multiple rows
- NAT → no option rows

8. Final Result Snapshot (Analytics-ready)

results

```
id (PK)
attempt_id (FK → test_attempts.id)

total_questions
```

```
correct
wrong
unattempted
percentage
rank

generated_at


Why snapshot:
```

- `Rankings should NOT change if logic updates later`
- `Stable certificates & reports`

CORNER CASES — EXPLICITLY SOLVED

Case	How it's handled
User refreshes page	Resume via IN_PROGRESS
Internet lost	last_activity_at
Passcode changed mid-test	passcode_version_used
Same question reused	test_questions
Student opens test twice	Attempt lock
Admin stops test	is_active=false
Cheating dispute	Passcode history + responses
Future paid tests	Add payments table

Without breaking anything:

- Paid tests
- Adaptive testing
- Question analytics
- Certificates
- Offline PWA sync
- AI-generated questions
- Instructor dashboards

SDD

SOFTWARE DESIGN DOCUMENT (SDD)

Online Test & Assessment Platform

1. Introduction

1.1 Purpose

This Software Design Document (SDD) provides a complete technical design of the **Online Test & Assessment Platform**, translating the approved **Software Requirements & Design (SWRD)**, **API specifications**, and **final database schema** into a concrete system blueprint.

The document defines:

- System architecture
- Component-level design
- Data design
- API interaction flow
- Security and access control mechanisms

This SDD is **scope-locked** and intended for direct implementation.

1.2 Design Goals

- Secure test access using **test code + passcode**
- Support both **guest and authenticated students**
- Scalable question reuse and analytics readiness
- Mobile-first, low-latency test experience
- Strong auditability and dispute handling

2. System Architecture

2.1 Architectural Overview

The system follows a **Client–Server architecture** with strict separation of concerns.

```
[ React Frontend (SPA) ]
        |
    REST APIs (JSON)
        |
[ Django REST Backend ]
        |
[ Relational Database ]
```

2.2 Architectural Style

- **Frontend**: Component-based SPA (React)
- **Backend**: Layered architecture
    - API Layer (Views / Controllers)
    - Service Layer (Business rules)
    - Data Layer (ORM Models)
- **Communication**: Stateless REST APIs
- **Authentication**: JWT for logged-in users

3. Major System Components

3.1 Frontend Component Design (React)

3.1.1 Component Structure

```
App
 ├── Auth
 │    ├── Login
 │    ├── Register
 │    └── Profile
 ├── TestCreator
 │    ├── TestList
 │    ├── CreateTest
 │    ├── EditTest
 │    ├── PasscodeManager
 │    └── AnalyticsDashboard
 ├── QuestionBank
 │    ├── QuestionList
 │    ├── CreateQuestion
 │    └── PublicQuestionSearch
 ├── TestAccess
 │    ├── TestCodeInput
 │    └── PasscodeInput
 ├── TestSession
 │    ├── QuestionRenderer
 │    ├── Timer
 │    ├── NavigationPanel
 │    └── SubmitHandler
 ├── Results
 └── Shared
      ├── Navbar
      ├── Loader
      └── ProtectedRoute
```

3.1.2 Frontend Responsibilities

Module	Responsibility
Auth	Login, JWT handling
TestCreator	Test & passcode management
QuestionBank	Question CRUD & reuse
TestAccess	Secure test entry
TestSession	Attempt lifecycle
Results	Score & analytics display

3.2 Backend Component Design (Django REST)

3.2.1 App-Level Structure

```
backend/
 ├── auth/
 ├── tests/
 ├── questions/
 ├── access/
 ├── session/
 ├── results/
 └── core/
```

3.2.2 Backend Responsibilities

App	Responsibility
auth	Authentication, RBAC
tests	Test lifecycle & passcodes
questions	Question bank
access	Secure test validation
session	Attempt management
results	Evaluation & snapshots

4. Data Design

4.1 Database Architecture

The database design emphasizes:

- **Immutability of intent**
- **Auditability**
- **Reuse**
- **Future extensibility**

4.2 Core Entities

Users & Roles

- Supports **multiple roles per user**
- RBAC ready for future admin features

Tables

- users
- roles
- user_roles

Tests

Represents the test entity and its configuration.

Key Design Principles

- Ownership enforced
- Soft archival supported
- Hard kill switch (is_active)

Table

- tests

Test Access & Security

Handles secure entry and passcode rotation.

Key Features

- Passcode hashing
- Versioned passcodes
- Historical audit trail

Tables

- test_access

- test_passcode_history

Question Bank

Reusable, visibility-controlled question repository.

Design Highlights

- Public/private questions
- Creator attribution
- Auto-gradable types only

Tables

- questions
- options

Test–Question Mapping

Decouples questions from tests.

Why

- Same question in multiple tests
- Different marks/order per test

Table

- test_questions

Attempts

Represents a single test attempt lifecycle.

States Supported

- IN_PROGRESS
- SUBMITTED
- EXPIRED
- FORCE_SUBMITTED

Table

- test_attempts

Responses

Fine-grained answer storage per question.

Design Choice

- Separate tables for options and responses
- Supports MCQ, MSQ, NAT cleanly

Tables

- responses
- response_options

Results Snapshot

Immutable evaluation snapshot for analytics and reporting.

Why Snapshot

- Logic changes should not affect past results
- Stable certificates and rankings

Table

- results

5. API Interaction Design

5.1 Authentication Flow

1. User logs in
2. JWT issued
3. Token attached to protected API calls

5.2 Secure Test Access Flow

Student → Enter Test Code
→ Passcode Validation
→ Session Token Issued
→ Test Session Start

Passcode change:

- Invalidates old access
- Does NOT affect active attempts

5.3 Test Attempt Flow

Start Session
→ Fetch Questions
→ Periodic Activity Updates
→ Manual / Auto Submission

→ Evaluation
→ Result Snapshot

6. Security Design

- Passcodes stored as **hashed values**
- Version-based passcode enforcement
- Ownership checks at service layer
- Attempt locking to prevent multi-tab misuse
- Audit logs via passcode history & attempts

7. Non-Functional Design

Aspect	Design Decision
Scalability	Stateless APIs
Performance	Indexed FKs, snapshots
Reliability	Resume IN_PROGRESS
Usability	Mobile-first UI
Security	RBAC + passcode versioning

8. Extensibility & Future Readiness

The design supports:

- Paid tests
- Adaptive testing
- AI-generated questions
- Advanced analytics
- Certificates & dashboards
- PWA offline sync

Without schema refactor

9. Design Status

SDD Status:

- Complete
- Consistent
- Scope-locked
- Implementation-ready

SWRD

SOFTWARE REQUIREMENTS DOCUMENT (SRD)

Online Test & Assessment Platform

1. Introduction

1.1 Purpose

This document defines the functional and non-functional requirements for an online test and assessment platform. The SRD serves as the single source of truth for system design, API planning, database modeling, and implementation.

1.2 Product Vision

The platform is designed for modern competitive exam preparation with a **mobile-first approach**. It enables test creators to securely create and distribute tests, while students can attempt tests with minimal friction, either as guests or logged-in users. The system is extensible to support analytics and AI-assisted features in future phases.

2. User Roles & Access Model

2.1 Test Creator

- Authentication: Mandatory login
- Responsibilities:
  - Create, edit, publish, and manage tests
  - Create and manage questions
  - Control test visibility and access
  - Choose test templates
  - View results and analytics for their own tests only
  - Search and reuse public questions
  - Clone public tests created by others

2.2 Student

Students may interact with the system in two modes:

2.2.1 Guest Student

- No login required
- Can attempt tests using:
  - Test link or test code
  - Test access passcode
- Limitations:
  - Results may be shown only immediately after submission
  - Attempt history is not permanently stored

2.2.2 Logged-in Student

- Login required
- Can:
  - Attempt tests
  - View previous test results
  - Maintain attempt history
  - Receive analytics and insights in future versions

3. Authentication & Authorization

3.1 Authentication Rules

- Test creators must be authenticated
- Student authentication is optional
- JWT-based authentication for all logged-in users

3.2 Test Access Control

- Each test is protected by:
  - A unique test code and/or shareable test link
  - A mandatory access passcode
- A test can be accessed only when **both**:
  - The test identifier is valid
  - The passcode is correct

3.3 Passcode Management

- Passcodes are set by the test creator
- Test creators can:
  - Update passcodes at any time
  - Immediately invalidate previous passcodes
- Passcode updates:
  - Do not affect already submitted attempts
  - Apply only to new test entries

4. Question Management

4.1 Question Creation

Each question must contain:

- Question text
- Question type
- Options (if applicable)
- Correct answer(s)
- Topic tag(s)
- Difficulty level (easy / medium / hard)

4.2 Supported Question Types

The platform must support all relevant modern competitive exam question types:

- Single-correct MCQ
- Multiple-correct (MSQ)
- Numeric Answer Type (NAT / TITA)
- Fill-in-the-blank
- True / False
- Matching (one-to-one)
- Matrix match
- Assertion–Reason
- Passage / paragraph-based questions

All question types must be auto-gradable.

4.3 Question Visibility

Each question has a visibility setting:

- **Private**: usable only by the creator
- **Public**: discoverable and reusable by other test creators

Rules:

- Public questions cannot be edited or deleted by others
- Original creator attribution is preserved
- Modification requires cloning into a private copy

4.4 Public Question Dataset

- Test creators can search public questions using:
  - Topic
  - Difficulty
  - Keywords
- Public questions can be added directly to tests without ownership transfer

5. Test Management

5.1 Test Configuration

Each test includes:

- Title
- Description
- Duration
- Number of questions
- Selected question set
- Test template
- Visibility (public / private)

5.2 Test Visibility

- **Private Test**
  - Accessible only via test code/link + passcode
- **Public Test**
  - Discoverable by other test creators
  - Can be cloned by others
  - Results are visible only to the original creator

5.3 Test Ownership

- Each test has exactly one owner
- Only the owner can:
  - Modify the test
  - Change passcodes
  - View test results

6. Test Templates

6.1 Template Selection

- A test template must be selected during test creation
- A default template is provided
- Templates cannot be changed after test publication

6.2 Template Capabilities

Templates define:

- Question layout (single or multiple per page)
- Navigation rules (free or sequential)
- Question palette behavior
- Mark-for-review option
- Scoring logic
- Negative marking configuration
- Global timer behavior

Templates are configuration-based, not hard-coded.

6.3 MVP Templates

- Practice Quiz (no negative marking)
- Competitive Exam Template
- Section-ready Template (structure only)

7. Test Attempt Flow

7.1 Attempt Rules

- Test starts only after passcode validation
- Test runs for a fixed duration
- Submission options:
    - Manual submission
    - Automatic submission on time expiry

Reattempt policy is configurable per test (single attempt recommended for MVP).

7.2 Guest vs Logged-in Attempts

- Guest attempts:
    - Results visible once
    - Not stored permanently
- Logged-in attempts:
    - Stored permanently
    - Accessible at any time

8. Results & Analytics

8.1 Result Generation

- Results are generated automatically on submission
- Result visibility:
    - Immediate or delayed (configurable by test creator)

8.2 Analytics (MVP)

- Test creator:
    - Student-wise scores
    - Overall test performance
- Logged-in student:
    - Test-wise results

8.3 Future Analytics

- Topic-wise weakness detection
- Performance trends across tests
- Difficulty-wise analysis
- AI-driven insights

Topic-level response data must be stored from the beginning.

9. AI-Assisted Question Generation (Future Scope)

- AI can generate questions based on:
  - Topic
  - Difficulty
  - Question type
- AI-generated questions:
  - Require manual review
  - Can be edited
  - Can be saved as public or private
- AI-generated questions follow the same lifecycle as manual questions

10. Non-Functional Requirements

10.1 Performance

- Support concurrent test attempts
- Stable under live test conditions

10.2 Security

- Secure passcode validation
- Role-based access control
- Strict test ownership enforcement

10.3 Usability

- Simple, distraction-free test UI
- Clear navigation and timer visibility

10.4 Device Compatibility

- Fully accessible on:
  - Mobile phones

    - Tablets
    - Laptops and desktops

10.5 Mobile-First Design

- UI must be designed primarily for mobile devices
- Touch-friendly controls are mandatory
- All features must work equally on mobile and desktop

10.6 Constraints

- Responsive web application only
- No native mobile apps in MVP
- Continuous internet connection required during tests

11. Out of Scope (MVP)

- Leaderboards and rankings
- Payments and monetization
- Native Android/iOS apps
- Automatic AI publishing

12. Document Status

This SRD is:

- Complete
- Consistent
- Scope-locked
- Ready for System Design (SDD)

.