# Thomas GIA Exam

Awesome idea — let’s spec a clean, production-ready MVP for **“Thomas”** (Angular front + .NET Core REST API + EF Core + SQL Server) that mirrors the GIA flow: consent → practice (8 Q, untimed with instant feedback) → full timed exam (5 sections) → report.

Below is a tight blueprint you can build from right away.

## 1) High-level architecture

* **Frontend**: Angular 17+, standalone components, Router, Reactive Forms, RxJS. Optional NgRx later.
* **Backend**: .NET 8 Web API, EF Core, SQL Server, JWT auth, FluentValidation.
* **DB**: SQL Server with normalized schema (Questions, Sections, Attempts, Answers, etc.).
* **Auth**: Email+password (JWT), role: Admin (content authoring) / Candidate.
* **Content**: All questions, timings, instructions are stored in DB so you can match the “Aptitude Example Booklet” once you upload it.

## 2) GIA-like domain (configurable)

* Sections (example names, editable in DB):
  1. Reasoning
  2. Perceptual Speed
  3. Number Speed & Accuracy
  4. Spatial Visualization
  5. Word Meaning
* Each section has: name, description, **time limit** (e.g., minutes), question order, scoring policy.

## 3) Database schema (core tables)

**Users**

- Id (PK, GUID)

- Email (unique), PasswordHash, FullName

- Role (Admin|Candidate)

- CreatedAt, LastLoginAt

**Exams**

- Id (PK)

- Code (e.g., "GIA-EN"), Title, Description

- IsActive, CreatedAt

**ExamSections**

- Id (PK)

- ExamId (FK)

- Name, Description

- TimeLimitSeconds

- OrderIndex

**Questions**

- Id (PK)

- ExamSectionId (FK)

- Stem (rich text)

- Type (SingleChoice|MultipleChoice|Numeric|Ordering|Matrix etc.)

- Difficulty (optional), OrderIndex, IsPractice (bool)

**QuestionOptions**

- Id (PK)

- QuestionId (FK)

- Text

- IsCorrect (bool)

- OrderIndex

- Value (optional for numeric)

**Attempts**

- Id (PK)

- ExamId (FK), UserId (FK)

- Mode (Practice|Real)

- Status (NotStarted|InProgress|Completed|Cancelled|Expired)

- StartedAt, CompletedAt

- TotalTimeSeconds

- ClientMeta (UA, IP, etc.)

**AttemptSections**

- Id (PK)

- AttemptId (FK), ExamSectionId (FK)

- Status (NotStarted|InProgress|Completed|Expired)

- StartedAt, CompletedAt

- TimeSpentSeconds

- RawScore, MaxScore

**AttemptAnswers**

- Id (PK)

- AttemptId (FK)

- ExamSectionId (FK)

- QuestionId (FK)

- SelectedOptionIds (JSON)

- NumericValue (nullable)

- IsCorrect (computed)

- AnsweredAt

- TimeToAnswerMs

**Reports**

- Id (PK)

- AttemptId (FK)

- SummaryJson (per-section stats, overall, percentiles if you add norms)

- GeneratedAt

4) EF Core entities (sample)

public class ExamSection {

public int Id { get; set; }

public int ExamId { get; set; }

public string Name { get; set; } = "";

public string? Description { get; set; }

public int TimeLimitSeconds { get; set; }

public int OrderIndex { get; set; }

public Exam Exam { get; set; } = null!;

public ICollection<Question> Questions { get; set; } = new List<Question>();

}

public class Question {

public int Id { get; set; }

public int ExamSectionId { get; set; }

public string Stem { get; set; } = "";

public string Type { get; set; } = "SingleChoice";

public bool IsPractice { get; set; }

public int OrderIndex { get; set; }

public ExamSection ExamSection { get; set; } = null!;

public ICollection<QuestionOption> Options { get; set; } = new List<QuestionOption>();

}

## 5) REST API (versioned, e.g., /api/v1)

**Auth**

* POST /auth/register → {email, password, fullName}
* POST /auth/login → JWT + user
* Use Authorization: Bearer <token>.

**Config/metadata**

* GET /exams/active → active exam with sections (for candidate start page)
* GET /exams/{examId}/sections → list with time limits
* GET /sections/{sectionId}/practice-questions?limit=8
* Admin content endpoints (CRUD for sections/questions/options) under /admin/\*.

**Attempt lifecycle**

* POST /attempts → body: { examId, mode: Practice|Real } → creates attempt, returns attemptId and first section id.
* POST /attempts/{attemptId}/sections/{sectionId}/start → server stamps StartedAt.
* GET /attempts/{attemptId}/sections/{sectionId}/next-question → server enforces sequence.
* POST /attempts/{attemptId}/answers → submit one answer.
* POST /attempts/{attemptId}/sections/{sectionId}/complete → locks section, computes section score.
* POST /attempts/{attemptId}/complete → finalizes exam, triggers report generation.
* GET /attempts/{attemptId}/report → report JSON (and optionally PDF).

**Practice feedback**

* For practice mode only, POST /attempts/{id}/answers responds with {isCorrect, explanation?} immediately.

**Timing & integrity**

* Server-side guards enforce:
  + Per-section timer based on StartedAt + TimeLimitSeconds.
  + Auto-expire when time is up; late answers rejected.
  + One active section per attempt; once completed, cannot reopen.
  + Optional anti-refresh token: a short-lived sectionToken returned by /start required on answer posts.

## 6) Scoring & report

* **Scoring**: Each question = 1 point by default; IsCorrect derived from SelectedOptionIds vs correct set (or numeric tolerance).
* **Per-section**: RawScore, MaxScore, Accuracy %, Avg time per question.
* **Overall**: Sum across sections, radar chart data.
* **Percentiles**: If you later add norms, store in a Norms table and map raw→percentile per section.
* **PDF** (optional MVP+1): Generate with QuestPDF or DinkToPdf.

Example report JSON:

{

"candidate": {"name":"Rami Issa"},

"exam":"GIA-like",

"mode":"Real",

"overall":{"score":37,"max":50,"accuracy":74},

"sections":[

{"name":"Reasoning","score":8,"max":10,"accuracy":80,"avgMs":22000},

...

]

}

## 7) Angular app structure

**Routing**

/ (Home/Landing)

/consent (4 checkboxes + Start Exam)

/practice (intro → 8 Q flow → small summary)

/exam (guarded)

/lobby (shows sections, time limits, Start Section)

/section/:id (question runner + timer)

/result/:attemptId (report view)

/admin (protected)

/content (sections, questions)

/uploads (booklet parsing later)

/auth/login

**Key components (standalone)**

* ConsentPageComponent – 4 conditions (FormGroup with checkboxes) → “Start Exam” enabled only if all checked.
* PracticeRunnerComponent – pulls 8 practice questions, shows **immediate toast/card** after each answer (Success/Fail).
* ExamLobbyComponent – lists 5 sections with time limits and statuses.
* SectionRunnerComponent – renders stem/options, **server-driven next question**, shows countdown timer, disables inputs when time ends, auto posts pending answer on timeout.
* ReportComponent – charts (radar for sections, bars), share/download.
* AdminQuestionEditorComponent – CRUD questions/options (rich text stem).
* Shared: TimerDisplay, OptionList, ProgressBar, GuardUnsaved (prevents leaving mid-answer in practice).

**Services (RxJS)**

* AuthService (login/register, token in localStorage, HttpInterceptor adds Authorization header).
* ExamService (exam/section metadata).
* AttemptService (create attempt, start section, get next question, submit answers, complete).
* ReportService (fetch/generate PDF).
* AdminService (content management).

**State**

* Lightweight with BehaviorSubject in services (attemptId, section state, remaining time). Add NgRx if you want auditability or time-travel later.

**Timer strategy**

* Timer driven from server timestamps:
  + On /start, backend returns serverNow, expiresAt.
  + UI uses expiresAt - clientNow + (clientNow-serverNow) to compensate drift.
  + When timer hits 0 → call /sections/{id}/complete.

**Practice feedback UX**

* After posting answer, show result banner (“Correct!”/“Incorrect. Correct answer: …”) + **Next** button.
* Keep running accuracy widget (e.g., “4 / 8 correct”).

## 8) Security & exam integrity (MVP scope)

* JWT auth, HTTPS, CORS restricted.
* Rate-limit answer endpoints.
* Store ClientMeta (user agent, IP) in Attempts.
* Anti-refresh sectionToken as noted.
* Optional: blur/focus tracking events (front only, store as telemetry).

## 9) Migrations & seeding

* EF Core migrations.
* Seed:
  + Exams (1 row), ExamSections (5 rows with time limits),
  + 8 practice questions (IsPractice=true),
  + A few real questions per section to start,
  + Admin user.

## 10) Minimal endpoint contracts (snippets)

**Create attempt**

POST /api/v1/attempts

{

"examId": 1,

"mode": "Real" // or "Practice"

}

→ 201

{

"attemptId": 123,

"examId": 1,

"mode": "Real",

"sections": [{ "sectionId": 10, "name":"Reasoning", "timeLimitSeconds": 600 }, ...]

}

**Start section**

POST /api/v1/attempts/123/sections/10/start

→ 200

{

"sectionToken":"eyJhbGciOi...",

"serverNow":"2025-09-13T09:10:00Z",

"expiresAt":"2025-09-13T09:20:00Z"

}

Next question

GET /api/v1/attempts/123/sections/10/next-question

Authorization: Bearer ...

X-Section-Token: ...

→ 200

{

"questionId": 501,

"stem":"<p>…</p>",

"type":"SingleChoice",

"options":[{"id":9001,"text":"A"}, {"id":9002,"text":"B"}, ...],

"index": 3, "total": 20

}

**Submit answer (practice returns feedback immediately)**

POST /api/v1/attempts/123/answers

{

"examSectionId": 10,

"questionId": 501,

"selectedOptionIds": [9002]

}

→ Practice: { "isCorrect": true, "correctOptionIds":[9002] }

→ Real: { "accepted": true }

**Complete section / exam**

POST /api/v1/attempts/123/sections/10/complete

→ { "status":"Completed", "rawScore": 14, "maxScore": 20 }

POST /api/v1/attempts/123/complete

→ { "status":"Completed", "reportId": 777 }

## 11) Consent page (4 conditions)

* Store a signed consent record or include consent flags on the Attempt:
  + consentDataPrivacy, consentNoCheating, consentTimeLimits, consentResultsUsage (bools).
* “Start Exam” disabled until all checked.

## 12) Admin content authoring flow

* CRUD for Sections (name, time limit).
* CRUD for Questions (rich stem, type, order, practice flag).
* CRUD for Options (and mark correct ones).
* Import helper: once you upload the **Aptitude Example Booklet**, we’ll parse/extract and populate Questions (without violating any licensing—use your own content or allowed excerpts).

## 13) Reporting UI

* Candidate view: section table + radar chart + per-section accuracy + total score.
* Admin view: attempt list, filters by date/user, CSV export.

## 14) Dev notes / recommended libs

* **Backend**:
  + Auth: Microsoft.AspNetCore.Authentication.JwtBearer
  + Mapping: AutoMapper
  + Validation: FluentValidation
  + PDF (optional): QuestPDF
* **Frontend**:
  + Angular Material or Tailwind for quick UI,
  + ngx-charts (or Chart.js) for report charts,
  + Route guards: AuthGuard, ExamGuard (block access if no active attempt/section).

## 15) Build order checklist (you can follow this exactly)

1. **Backend skeleton**: projects, EF Core, migrations, JWT auth.
2. **Seed**: exam + 5 sections + 8 practice Qs.
3. **Attempts API**: create/start/next/answer/complete (+ practice feedback).
4. **Angular auth** + interceptor.
5. **Consent page** → **Practice runner** (instant feedback).
6. **Exam lobby** + **Section runner** with server-synced timer.
7. **Scoring** + **Report page**.
8. Admin CRUD for content.
9. Polish: telemetry, CSV export, optional PDF.

If you want, upload the **Aptitude Example Booklet** and I’ll align the section names, instructions, and sample question styles precisely. Ready to start with step 1 (backend skeleton) or would you rather begin with the Angular consent + practice flow?

## 16 DB Creation

-- === 0) פרמטרים: שנה שמות/סיסמה לפי הצורך ===

DECLARE @DbName sysname = N'ThomasDb';

DECLARE @LoginName sysname = N'thomas\_app';

DECLARE @UserName sysname = N'thomas\_app'; -- בד"כ זהה ל-Login

DECLARE @Password nvarchar(128) = N'ThomasJan2025!@'; -- החלף בסיסמה חזקה

-- === 1) יצירת הדאטאבייס ===

IF DB\_ID(@DbName) IS NULL

BEGIN

DECLARE @sql nvarchar(max) = N'CREATE DATABASE [' + @DbName + N']';

EXEC (@sql);

PRINT N'Database created: ' + @DbName;

END

ELSE

PRINT N'Database already exists: ' + @DbName;

-- === 2) יצירת Login ברמת השרת (SQL Authentication) ===

IF NOT EXISTS (SELECT 1 FROM sys.sql\_logins WHERE name = @LoginName)

BEGIN

DECLARE @sqlLogin nvarchar(max) = N'CREATE LOGIN [' + @LoginName + N'] WITH PASSWORD = N''' + @Password + N''', CHECK\_POLICY = ON, CHECK\_EXPIRATION = ON';

EXEC (@sqlLogin);

PRINT N'Login created: ' + @LoginName;

END

ELSE

PRINT N'Login already exists: ' + @LoginName;

-- === 3) יצירת User בתוך הדאטאבייס ומיפוי ל-Login ===

DECLARE @sqlUser nvarchar(max) = N'

USE [' + @DbName + N'];

IF NOT EXISTS (SELECT 1 FROM sys.database\_principals WHERE name = N''' + @UserName + N''')

BEGIN

CREATE USER [' + @UserName + N'] FOR LOGIN [' + @LoginName + N'] WITH DEFAULT\_SCHEMA = [dbo];

PRINT N''User created in DB: ' + @UserName + N''';

END

ELSE

PRINT N''User already exists in DB: ' + @UserName + N''';

';

EXEC (@sqlUser);

-- === 4) הענקת הרשאות מלאות רק על הדאטאבייס הזה (db\_owner) ===

DECLARE @sqlRole nvarchar(max) = N'

USE [' + @DbName + N'];

IF NOT EXISTS (

SELECT 1 FROM sys.database\_role\_members drm

JOIN sys.database\_principals r ON r.principal\_id = drm.role\_principal\_id AND r.name = N''db\_owner''

JOIN sys.database\_principals u ON u.principal\_id = drm.member\_principal\_id AND u.name = N''' + @UserName + N'''

)

BEGIN

ALTER ROLE [db\_owner] ADD MEMBER [' + @UserName + N'];

PRINT N''Added to db\_owner: ' + @UserName + N''';

END

ELSE

PRINT N''Already member of db\_owner: ' + @UserName + N''';

';

EXEC (@sqlRole);

-- === 5) אימות מהיר ===

PRINT '--- VERIFY ---';

EXEC (N'USE [' + @DbName + N'];

SELECT DB\_NAME() AS DbName, USER\_NAME() AS CurrentUser;

SELECT r.name AS RoleName, u.name AS UserName

FROM sys.database\_role\_members drm

JOIN sys.database\_principals r ON r.principal\_id = drm.role\_principal\_id

JOIN sys.database\_principals u ON u.principal\_id = drm.member\_principal\_id

WHERE u.name = N''' + @UserName + N'''');

{

"ConnectionStrings": {

"Default": "Server=YOUR\_SERVER,1433;Database=ThomasDb;User Id=thomas\_app;Password= ThomasJan2025!@;Encrypt=True;TrustServerCertificate=True;MultipleActiveResultSets=True"

}

}

Users:

Rami, Admin, [ram\_fad@yahoo.com](mailto:ram_fad@yahoo.com), Rami#2025!Temp

Fadwa, Candidate, [fadwa.issa@gmail.com](mailto:fadwa.issa@gmail.com), Fadwa#2025!Temp

Create DB User Code:

SET @Email = N'fadwa.issa@gmail.com';

SET @FullName = N'Fadwa';

SET @Role = 'Candidate';

SET @TempPassword = N'Fadwa#2025!Temp'; -- החלף מיד אחרי התחברות ראשונה

SET @Salt = CRYPT\_GEN\_RANDOM(16);

SET @SaltHex = CONVERT(VARCHAR(200), @Salt, 2);

SET @ToHash = @Salt + CONVERT(VARBINARY(2048), @TempPassword);

SET @Hash = HASHBYTES('SHA2\_256', @ToHash);

SET @HashHex = CONVERT(VARCHAR(64), @Hash, 2);

SET @PasswordHash = CONCAT(@SaltHex, ':', @HashHex);

IF NOT EXISTS (SELECT 1 FROM dbo.Users WHERE Email = @Email)

BEGIN

INSERT INTO dbo.Users (Email, PasswordHash, FullName, Role, IsActive, CreatedAt)

VALUES (@Email, @PasswordHash, @FullName, @Role, 1, @Now);

END

ELSE

BEGIN

UPDATE dbo.Users

SET FullName=@FullName, Role=@Role, IsActive=1, PasswordHash=@PasswordHash

WHERE Email=@Email;

END