Technical Assignment

Your task is to develop an exam system with Admin and Student views. Admins can create/update exams (with cooldown time between multiple exam attempts), and students can take (mock) exams, view attempt history, and are restricted by cooldowns. Technology to use DotNet 8+ and React 17+. Here is the user story for this feature:

# User Story

As an Admin, I want to create (or update) an exam with multiple attempts, so that a student can take the exam with restrictions applied (cooldown time between attempts) and view their history of attempts.

# Functional Requirements

**Admin Features/Page (Refer to Admin View mockup in next section)**

1. Create/update a single exam with:
   1. Exam Id (Guid)
   2. Title (can be in any language and can contain special characters)
   3. Maximum number of attempts (Between 1 to 1000)
   4. Cooldown time between attempts (in minutes – 0 to 525600)
2. When an (existing) exam is updated by admin → all student attempt history is reset
3. View table of student’s past and current attempts (if any exist):
   1. Attempt Id (Guid)
   2. Attempt number
   3. Status (In Progress / Completed)
   4. Start time (display in UTC time)
   5. End time (if completed, display in UTC time)

**Student Features/Page (Refer to Student View mockup in next section)**

1. View exam details (title, max attempts, attempts remaining, cooldown time)
2. View table of past attempts and current attempt with (if any exist):
   1. Attempt number
   2. Status (In Progress / Completed)
   3. Start time (display in current browser timeZone)
   4. End time (if completed, display in current browser timeZone)
3. If attempts remaining and cooldown is satisfied
   1. Show “Start Attempt” button for current attempt
      1. When Student clicks Start, a “Submit Attempt” button should appear, attempt will be set to InProgress and Start Time updated
      2. When Student click Submit, mark it as completed, InProgress changed to completed and End Time updated
4. If no attempts left, show “No attempts left.”
5. If blocked by cooldown, show “Your next attempt will be available at [date/time].”

# UI Expectations

1. Keep UI functional and show validation messages. Use React + Typescript.
2. Provide a way to switch between:
   1. Admin view → form to create/update exam + table of student attempt
   2. Student view → exam details + 'Start Attempt' button + 'Submit Attempt' button + table of attempts

# Security Expectations

1. Keep a User’s table with columns like: **Id | Username | PasswordHash | Role**
2. Seed the In-Memory DB with one Admin and one Student User. You can decide how to use the user log-In and how UI and API secure the pages/endpoints (secure cookies, JWT tokens, etc.).
3. Endpoints should be Restful and use attributes like [HttpGet] [Authorize(Roles = "Admin")], etc.

# Data Persistence

1. Use Entity Framework In-Memory provider.
2. Suggested tables/entities:
   1. **Exam**: ExamId, Title, MaxAttempts, CooldownMinutes, LastModified
   2. **Attempt**: AttemptId, ExamId, StudentId, AttemptNumber, AttemptStatus, StartTime, EndTime

# Rough UI mockup

**Admin View**  
**Exam Form**

Title: [ ]

Max Attempts: [ ]

Cooldown Minutes: [ ]

[**Submit Button**]

**Student Attempts History**  
|------------------------------------------------------------------------------------------------|  
| Attempt Id |Attempt Number | Status | Start Time | End Time |  
|----------------|------------------------|-----------------|-------------------|----------------|  
| 123-456-7 |1 | Completed | 10:00 UTC | 10:15 UTC |  
|123-456-8 | 2 | InProgress | 11:00 UTC | - |

|--------------------------------------------------------------------------------------------|

**Student View**

Exam: Title  
Max Attempts: 3  
Remaining Attempts: 2

Cooldown: 30 minutes  
[**Start Attempt Button**] | or | Next Attempt Available at 3:00 PM | or | No Attempts left

**My Attempts History**  
|------------------------------------------------------------------------|  
| Attempt # | Status | Start Time | End Time |  
|----------------|-----------------|------------------|------------------|  
| 1 | Completed | 15:30 IST | 15:45 IST |  
| 2 | InProgress | 16:00 IST | - |

|------------------------------------------------------------------------|

# Acceptance Criteria

1. Give a mechanism to switch between Admin view and Student View
2. Only valid exam details should be saved.
3. On exam update, all student attempts are cleared.
4. Student can see the current exam and attempt status in student view.
5. A student cannot start a new attempt unless cooldown has passed and attempts remain.
6. In-progress attempts can only be submitted.
7. Both Admin and Student can view attempt history in table.
8. Unit tests/Integration Tests are expected.
9. Please submit a solution that you think is **production quality code** (validations errors shown to user, relevant code comments, consistent naming, following good design principles so that it is extensible in future, etc.)

# Important Submission Instructions

* Application (both API and UI) should be able to be executed from VS Code or Visual Studio without any extra setup. If you have any other preferences than these and some other setup is needed to execute the application, then attach a **readme.md** file for the steps.
* Please **do not host** the code on publicly accessible source control systems.
* Submit your solution as a ZIP file containing source code (**without obj, bin, node\_modules folders**).
* Please name the solution and zip **Firstname.Lastname.ExamAttemptsWithCooldown**. (ex: Deepak.Mishra.ExamAttemptsWithCooldown)
* **Bonus**: Expose API with Swagger, add UI tests or E2E test coverage.

# Demo Video Submission Requirements

Along with your code submission, please provide a **demo video** that includes the following:

1. **Screen Recording + Face Cam**
   * Record your screen while demonstrating the application.
   * Your face must be clearly visible with **good lighting**.
2. **Application Walkthrough**
   * Explain the functionality of both the **UI (React)** and **API (.NET)**.
   * Show how the two parts interact.
   * Run **all test cases** for both UI and API and demonstrate the results.
3. **Technical Considerations**
   * During your explanation, discuss aspects such as:
     + **Extensibility/Future Improvements** (e.g., features you did not add, but would add if this were a production-ready application)
   * You do not need to implement all of these, but you should describe your thought process.