CourseSignUp Code flow and Steps

Part 1: API for signing up

Create an API endpoint with which students can sign up for a course.

If a course is full, it should not be possible to sign up any more.

The endpoint's response should indicate whether signing up was successful.

Step 1:

Created below two tables in SQL DB:

CREATE TABLE [dbo].[Course] (

[CourseId] INT NOT NULL,

[CourseName] VARCHAR (50) NULL,

[LecturerName] VARCHAR (50) NULL,

[MaxAllowedStudent] INT NULL,

PRIMARY KEY CLUSTERED ([CourseId] ASC)

);

CREATE TABLE [dbo].[Student] (

[StudentId] INT IDENTITY (1, 1) NOT NULL,

[Name] VARCHAR (50) NULL,

[DOB] DATETIME NULL,

[Age] INT NOT NULL,

[CourseId] INT NULL,

[Email] VARCHAR (50) NULL,

PRIMARY KEY CLUSTERED ([StudentId] ASC),

CONSTRAINT [FK\_Student\_ToTable] FOREIGN KEY ([CourseId]) REFERENCES [dbo].[Course] ([CourseId])

);

Step 2:

* Created CourseEnrollmentLib.csproj that has business logic for student enrollment.
* First checking total number of students already enrolled to the passed CourseID against MaxAllowedStudent for the course, if condition satisfied then save student in DB.

Step 3:

* Created CourseEnrollmentAPI WEB API in .net core that consumes CourseEnrollmentLib.dll to enroll student data.
* If success we get HTTP200 response code else HTTP500 error message.

Part 2: Scaling out

After few months, the company's courses grow wildly successfull, business is

booming. There are many courses and millions of sign ups, and your synchronous

in-process API which you have created in the Part 1 cannot handle the load any more.

Create a new endpoint for your API that defers the actual processing to a

worker process: signing up is processed asynchronously via a message bus.

This works as follows. The API puts a command message on a queue, and the

message is picked up by the worker process. The worker process tries to sign

up the student; it then sends an e-mail to inform the student whether signing

up succeeded.

You can implement "sending an email" with a mock implementation that logs

success or failure.

Step 1:

* To the existing solution added AzureQueueLibrary.csproj that is used to Queue and Enqueue Student enrolment data to Azure Storage that is configured.
* AzureQueueLibrary had business logic to serialise and deserialise Azure storage message as per data required.

Step 2:

* Made changes in CourseEnrollmentAPI to call AzureQueueLibrary instead directly storing data to DB for student enrolment.
* Instead pushing records to Azure storage.

Step 3:

* Added TriggerStudentEnrollment Azure function Queue trigger that is triggered on every message added on Azure storage that is configured.
* TriggerStudentEnrollment is referring CourseEnrollmentLib.dll to enroll student data.
* If enrolment is successful logging data as enrolment success full.

Part 3: Querying

For analysis purposes, the company needs to know per course the minimum age, the

maximum age and the average age of students that signed up for the courses.

Consider that this needs to keep working efficiently when there are millions

of sign-ups per day: calculating this information at every request is unfeasable.

Create two API endpoints:

- GET list: which returns a list with the above information for each course, plus

the course total capacity and current number of students

- GET details: which returns the above information for a single course, plus

the teacher and the list of registered students

Step 1:

Added business logic in CourseEnrollmentLib to use both Course and Student tables to get above details.

Step 2:

Added two new endpoints to existing API to get details.