Course Sign-up System

Architecture

- One WEB API application
  - one API endpoint with which students can sign up for a course
  - one API endpoint that provides minimum/maximum/average age for a given course
- One BackgroundTask Worker application
  - one worker process that tries to sign up the student + sends success/failure email
- RabbitMQ for queueing/messaging
  - lightweight and easy to deploy on premises and in the cloud
  - supports multiple messaging protocols

Tools / Technologies

ASP.NET Core 2.0 for WEB API project

Entity Framework Core 2.0 for ORM

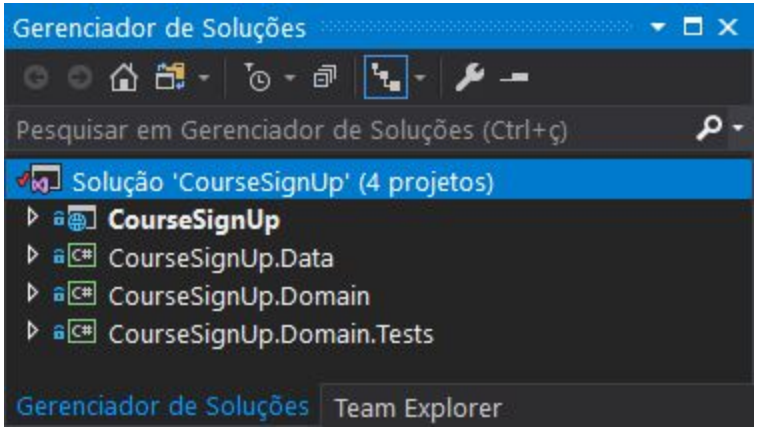
Sql Server for DBMS

RabbitMQ for asynchronous messaging

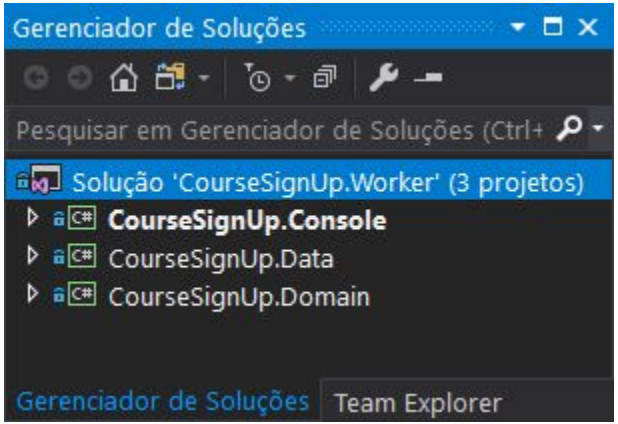
APIs

- POST api/Course
  - publishes a sign-up message to the open RabbitMQ connection
- GET api/Course/{courseCode}
  - retrieves minimum/maximum/average age for a given course

WEB API application & Worker application

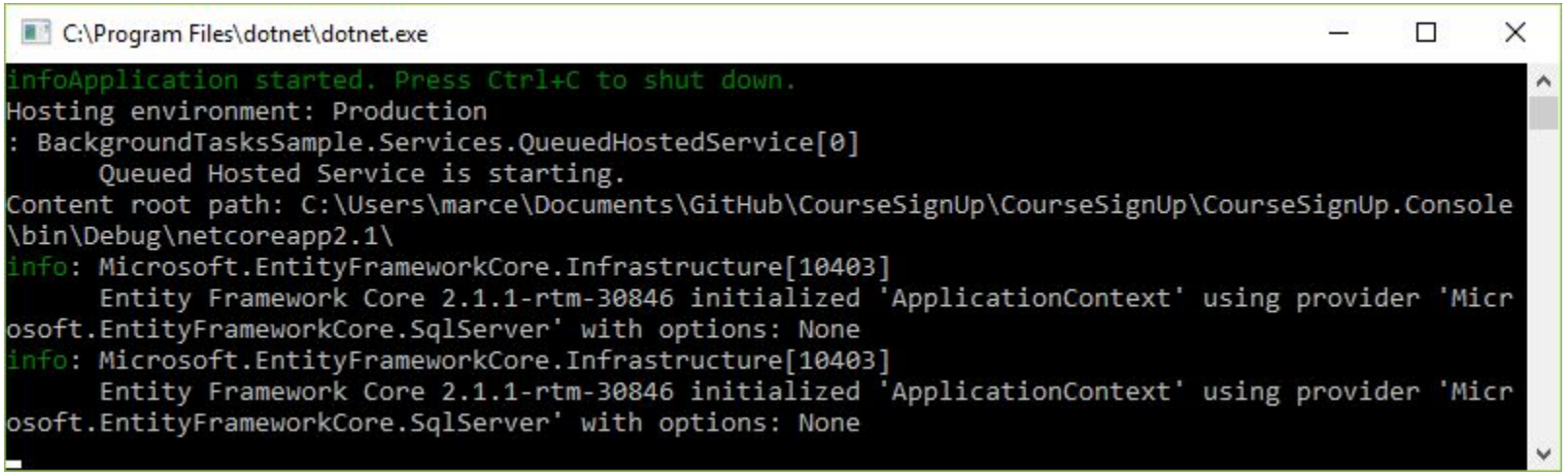


WEB API application

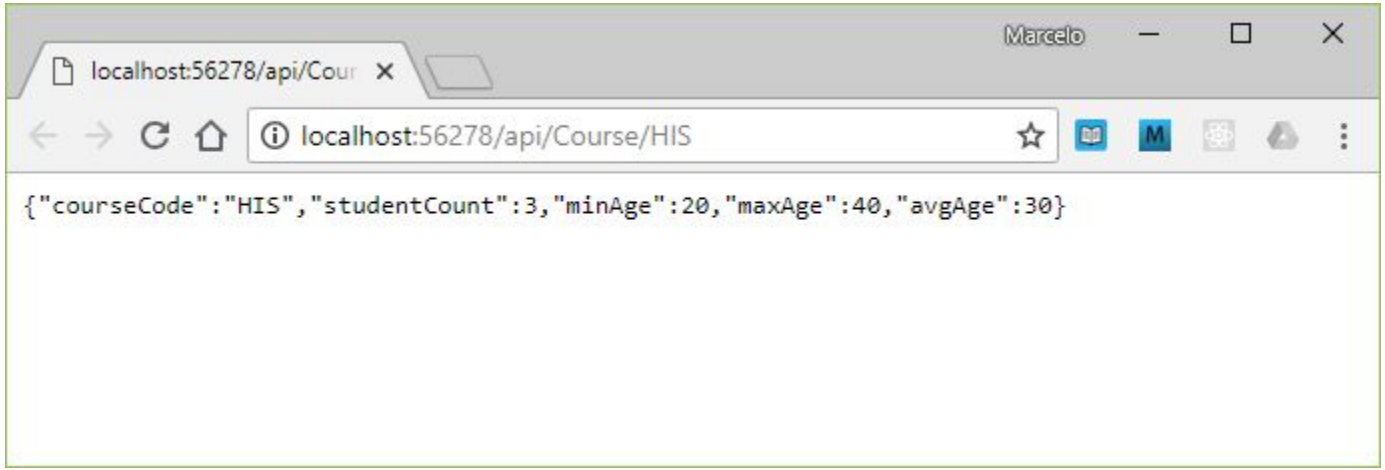


Worker application

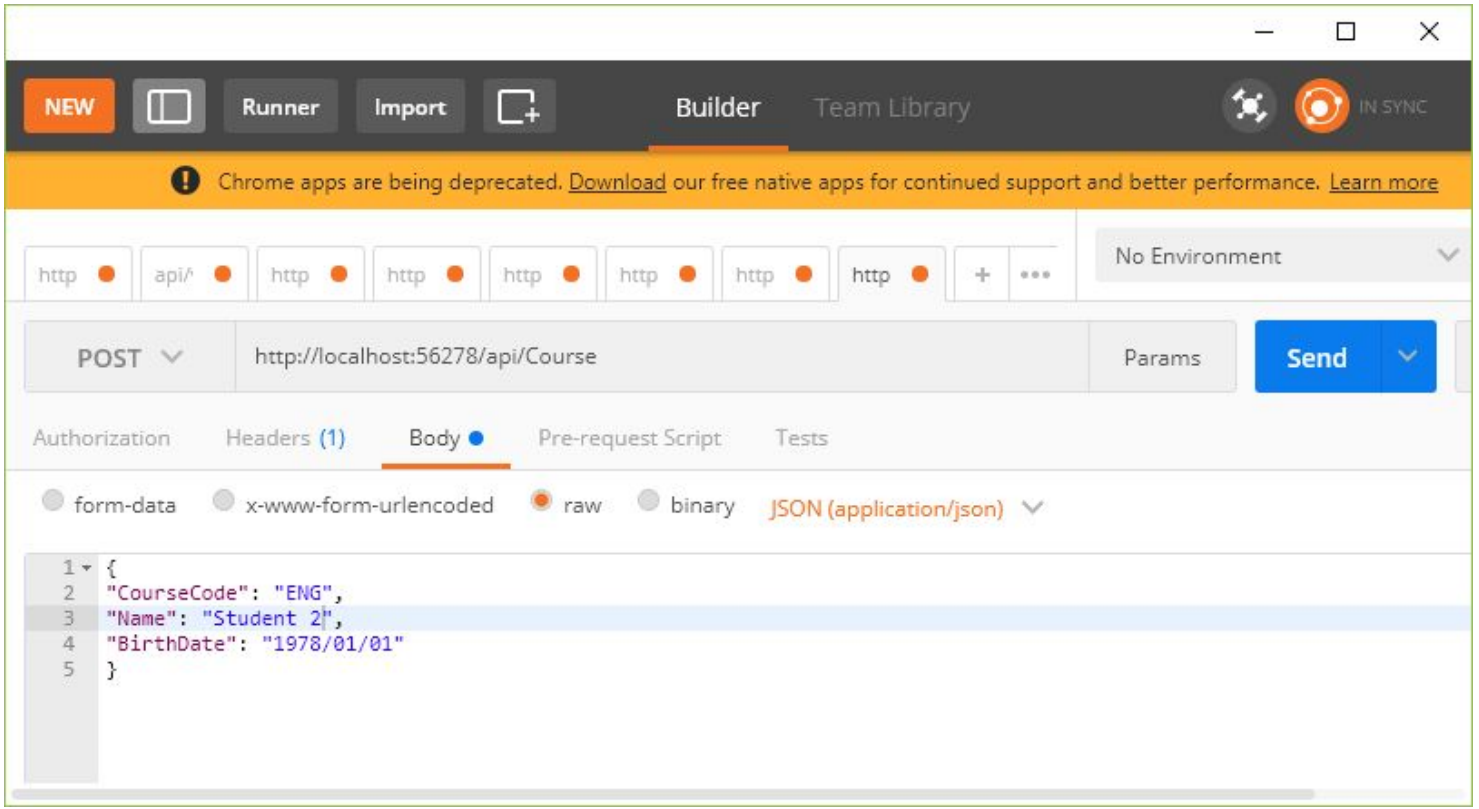
Worker application



Web API application - GET



Web API application - POST



Scaling out

Sign-up requests to the API does not wait for any synchronous sign up. Instead, each request produces a different message that is published to the RabbitMQ queue.

The worker process, which is another process, retrieves each individual message, and calls the CourseService.SignupStudent method. Every time a student is successfully enrolled in a course, this same process updates the minimum / maximum / average ages for that course.

Querying minimum / maximum / average ages for a given course does not impose an impact on the web API application, since these calculations have been made before.

These architecture allows a high-availability and high-scalability system.

Querying - Creating Course Stats

The Course table contains 4 fields for Querying:

- StudentCount
- MinBirthdate
- MaxBirthdate
- BirthdateTickSum

These 4 fields are always updated after a new student is enrolled (by the Worker process).

Querying - Retrieving Course Stats

The web API retrieves 4 fields from the given course:

- Student Count
- Minimum Age
- Maximum Age
- Average Age

These 4 fields are obtained from the Course Table (StudentCount, MinBirthdate, MaxBirthdate, BirthdateTickSum). MinBirthdate and MaxBirthdate are compared to the current date so the Min Age and Max Age can be obtained. Average Age is the quotient between BirthdateTickSum and StudentCount. The resulting ticks are translated into the average DateTime.

Problems & Challenges

It took me some time to figure out how to set up RabbitMQ. This is because that's my first time with this component.

I started unit-testing the repository, which relies heavily on Entity Framework components. Then I gave up and tested only CourseService, which does not depend on EF.

Unit testing the CourseService component was becoming increasingly complicated. Mocking dependencies can be hard sometimes.

What Can Be Improved

Student e-mail should be provided for the sign-up endpoint

Student uniqueness in course enrollment should be better enforced during sign-up

Not sure if SQL Server is the best approach for this system. Maybe alternative no-sql solutions could be tried.

I only unit-tested the CourseService component, because that’s where the business rules are. I would provide better unit testing code coverage, testing more components.