Санкт-Петербургский государственный университет

Системное программирование

Группа 21.Б07-мм

Перевод HwProj на .NET 6

Пожарский Роман Николаевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного, к. т. н., Ю. В. Литвинов

Консультант:
инженер-программист «Интеллиджей Лабс», А. В. Бережных

Санкт-Петербург
2023

Оглавление

1. Введение

HwProj — это сервис для ведения курсов (обычно по программированию), который предоставляет возможность преподавателям выкладывать материалы по курсу, домашние задания, а также оценивать присланные студентами решения. Сервис активно используется на направлении «Математическое обеспечение и администрирование информационных систем» на курсах по программированию для студентов 1-2 курсов, а также в некоторых других университетах (ИТМО, ВШЭ), а поддерживают его выпускники и студенты СПбГУ.

На данный момент бэкенд сервиса написан на языке $C\#$ на платформе .NET Core 2.2 [4], которая не поддерживается компанией Microsoft с 2019 года. Это значит, что проблемы и уязвимости, которые есть на этой платформе, уже давно никто не исправляет.

Также .NET Core не позволяет применять инструменты миграции базы данных, и это тормозит развитие HwProj, поскольку добавление новой функциональности сильно привязано к имеющейся базе данных.

Более того, .NET Core 2.2 не позволяет использовать новые версии языка $C\#$, поэтому такие нововведения 9, 10, и 11 версий языка, как top-level statements, records и многочисленные улучшения pattern-matching недоступны для тех, кто поддерживает HwProj.

Стоит отметить и большую разницу в производительности между .NET Core 2.2 и новыми версиями .NET.

2. Постановка задачи

Целью работы является перевод бэкенда HwProj на .NET 6 [2]. Для её выполнения были поставлены следующие задачи:

1. исправить появившиеся после смены SDK ошибки компиляции;
2. обновить используемые библиотеки и разрешить конфликты между ними;
3. исправить ошибки времени выполнения;
4. настроить инструменты кодогенерации;
5. исправить Docker-файлы.

3. Обзор

Бэкенд HwProj представлен 4 микросервисами: Auth Service, Solutions Service, Notifications Service, Courses Service. Брокером сообщений для них выступает RabbitMQ. Фронтенд использует порт ApiGateway — REST API сервиса, имеющего клиенты всех микросервисов, чтобы перенаправлять им запросы. Далее в этом разделе будут перечислены технологии, которые были использованы в ходе работы над учебной практикой.

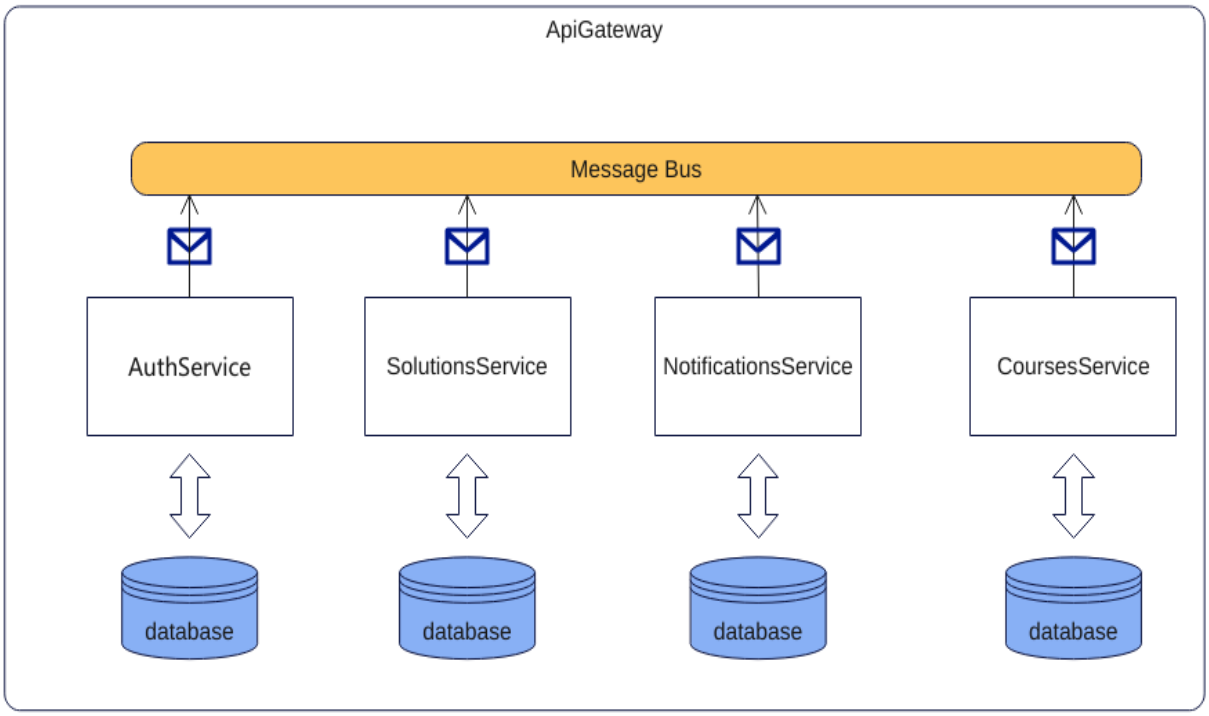

Рис. 1: Архитектура серверной части

3.1. Используемые технологии

.NET 6

.NET 6 — открытый, бесплатный, кроссплатформенный фреймворк для создания мобильных, настольных, облачных и веб приложений, который разрабатывается и поддерживается на GitHub. На момент начала работы над учебной практикой, уже вышел .NET 7 [3], было принято

решение переводить HwProj на .NET 6 , поскольку он является Long Term Supported (36 месяцев), в отличие от Short Term Supported (18 месяцев) .NET 7, и срок поддержки .NET 7 истекает раньше[1]. Также между .NET 6 и .NET 7 имеется незначительная разница в смысле производительности и предоставляемых возможностей.

Entity Framework Core

Entity Fremework Core — легковесная, расширяемая, открытая объектно-ориентированная технология доступа к данным, является Object-Relational Mapping решением для .NET от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Уже используется в production версии сервиса. Одной из основных целей перевода на .NET 6 и является возможность применения миграций при помощи Entity Framework Core Tools.

Swagger

Swagger [6] — наиболее популярный инструмент, предоставляющий возможность автоматически описывать API сервисов. Swagger Editor позволяет проектировать, описывать и документировать API, использующие OpenAPI (ранее известный как Swagger). Использовался для кодогенерации TypeScript клиента по *swagger.json* файлу, описывающему API бэкенда. Ранее использовался в сервисе из-за своей простоты и удобства. Для того чтобы не ломать обратную совместимость с фронтендом, было приято решение пользоваться этим же инструментом.

Docker

Docker [1] — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Для запуска микросервисов используется docker compose.

[1] https://dotnet.microsoft.com/en-us/platform/support/policy

4. Реализация

4.1. Ошибки компиляции

Поскольку работа по переводу на новый .NET уже была начата консультантом, осталось небольшое количество ошибок компиляции, связанных, в основном, с изменением сигнатур методов в новых версиях используемых библиотек, например, немного поменялась настройка JWT Bearer токена. Также без особых проблем были изменены версии используемых библиотек.

4.2. Ошибки времени выполнения

После исправления ошибок компиляции, нужно было проверить базовые сценарии работы с HwProj как со стороны студента, так и со стороны преподавателя.

4.2.1. Правки в Startup

Для того, чтобы запустить сервис, понадобилось сразу внести изменения в общий для всех микросервисов класс *Startup*. Этот класс отвечает за настройку API при запуске и содержит 2 метода: *ConfigureServices* для добавления в DI-контейнер сервисов, которые использует API, и *Configure*, отвечающий за порядок обработки сетевого запроса. Были изменены устаревшие методы для добавления контроллеров, а также были добавлены *Routing* и *Authorization* в обработку запроса.

4.2.2. Запросы, вызывавшие сбой

Были проверены различные сценарии работы с HwProj, начиная от регистрации пользователя, заканчивая оценкой преподавателем решения домашнего задания, и были обнаружены несколько ошибок, которые вызывали полный сбой в сервисе:

- последние непроверенные решения;

- открыть сданные на курсе решения;
- открыть пустой (только что созданный) курс.

Последние непроверенные решения вызывали один из самых сложных запросов к базе данных, и в .NET 6 этот запрос перестал работать, скорее всего, из-за неоптимальности, поэтому сам запрос к базе данных был упрощён, насколько это возможно, а остальная логика была сделана на стороне .NET.

Ошибку, связанную со сданными решениями и новым курсом пришлось исследовать через раздел «Сеть» в браузере и отладку в консоли. Проблема состояла в том, что клиенту приходили модели домашних работ, у которых некоторые поля были *null*, а к этим полям обращался TypeScript при отрисовке компонентов. Такая ситуация происходила из-узеза того, что когда переписывали код на бэкенде, забыли поставить публичные модификаторы доступа у домашних работ для соответстующих полей, что и мешало этим полям корректно сериализовываться в *json*.

4.2.3. Ошибки при многопоточной работе с базой данных

Entity Framework Core для работы с базой данных из кода на *C#* предоставляет класс *DatabaseContext*, который не является потоко-безопасным. Microsoft рекомендует всегда дожидаться окончания запроса к базе данных перед отправкой нового[2]. В HwProj в нескольких местах несколько потоков работали с *DatabaseContext* одновременно. Так, например, в запросе получения пользователей по массиву их идентификаторов для каждого пользователя создавался отдельный поток, а этот запрос ещё и вызывал получение роли каждого пользователя. Например, если нужно получить информацию о 20 пользователях, создавалось 40 задач, выполнявшихся в разных потоках. Было принято решение читать пользователей из базы данных за 1 запрос, аналогично было сделано и с ролями. При этом возникла ошибка, которая состояла в том, что при принятии или отклонении заявки на подключение к

[2]При использовании одного и того же экземпляра *DatabaseContext*

курсу, выбранная операция выполнялась либо не с тем студентом, либо не выполнялась вообще, также была возможна ситуация, когда среди отправителей заявок были уже ранее принятые студенты. Проблема была в том, что изменённый метод получения информации о студентах не гарантировал, что найденные студенты будут упорядочены по своим идентификаторам из тела запроса. А в другом микросерврисе, в методе получения статистики по курсу по курсу, выполнялась работа с индексами идентификаторов пользователей, и найденных по этим идентификаторам пользователей. Таким образом, было добавлено упорядочивание найденных пользователей по идентификаторам.

Другие проблемы с многопоточной работой решались либо синхронным выполнением запросов к базе данных, либо все необходимые данные получались за один запрос без каких-либо потерь,

4.2.4. Итог

Важно отметить, что ошибки, связанные с слишком сложными запросами к базе данных и многопоточностью, не воспроизводились на .NET Core. Это показывает, что переход на новый .NET был необходим, поскольку теперь Microsoft не позволяет писать ненадёжный небезопасный код, который работал раньше.

4.3. Кодогенерация

В HwProj для генерации клиента для фронтенда используется OpenAPI (ранее известный как Swagger), который по *swagger.json* файлу, описывающему какой-либо API, может сгенерировать клиент или сервер на многих популярных языках программирования.

Клиент, сгенерированный по новому *swagger.json*, заметно отличался от ранее использовавшегося: изменились названия большинства методов и определения enum-ов. Нужно было решить эту проблему, не изменяя код в генерирующемся клиенте и на фронтенде. За название сгенерированного метода отвечает поле *operationId* в *swagger.json* и тип запроса (get, post и т.д.). Проблема была решена через написание под-

сказки для Swagger, какой *operationId* нужно сгенерировать для метода, исходя из его роута и типа запроса. Проблема с генерацией enum-ов была решена использованием опции *UseInlineDefinitionsForEnums*.

Таким образом, исправления, внесённые в настройки кодогенератора, позволят легко генерировать рабочий клиент при изменениях на бэкенде.

4.4. Docker

API каждого микросервиса HwProj собирается в Docker-контейнере, все сервисы запускаются через **docker compose**. Без особых проблем были переписаны Docker-файлы под .NET 6. При этом было обнаружено, что ранее выполнялось несколько лишних действий: сначала выполнялось восстановление зависимостей через **dotnet restore**, затем сборка **Debug** версии, и только потом сборка **Publish** версии, которая, собственно, и запускалась. Первые две операции были убраны, потому что **Debug** версия никак не использовалась, а восстановление зависимостей и так вызывается при выполнении команды **dotnet build**. Так, если ранее контейнер с одним API собирался две минуты, теперь он собирается за одну.

При переходе на .NET 6 пришлось столкнуться с ошибкой **NETSDK1152** [5]. До .NET 6, если в директории, в которую собирается приложение, имелось несколько файлов, у которых был одинаковый относительный путь, то последний скопированный файл просто подменял другой. Подмена вызывала трудно отлавливаемые ошибки, поэтому .NET 6 теперь не позволяет так делать. В HwProj конфликтовали *appsettings.json* разных API. Поэтому были убраны зависимости типа API-API, которых и не должно было быть, поскольку для каждого сервиса имеется клиент, а логика, из-за которой нужны были эти зависимости, была вынесена в общие библиотеки.

5. Апробация

Апробация запланирована на лето, когда работоспособность сервиса не будет критически важна и можно будет тестировать сервис на .NET 6.

Заключение

В ходе работы были достигнуты следующие результаты:

1. исправлены появившиеся после смены SDK ошибки компиляции;
2. обновлены используемые библиотеки и разрешены конфликты между ними;
3. исправлены ошибки времени выполнения;
4. настроены инструменты кодогенерации;
5. исправлены Docker-файлы.

Код доступен по ссылке [3].

[3] https://github.com/InteIligeNET/HwProj-2.0.1/pull/212 Ник: oveeernight

Список литературы

[1] Docker. — URL: https://www.docker.com/ (дата обращения: 14 мая 2023 г.).

[2] .NET 6. — URL: https://dotnet.microsoft.com/en-us/download/dotnet/6.0 (дата обращения: 14 мая 2023 г.).

[3] .NET 7. — URL: https://dotnet.microsoft.com/en-us/download/dotnet/7.0 (дата обращения: 14 мая 2023 г.).

[4] .NET Core 2.2. — URL: https://dotnet.microsoft.com/en-us/download/dotnet/2.2 (дата обращения: 14 мая 2023 г.).

[5] NETSDK1152 Error. — URL: https://learn.microsoft.com/en-us/dotnet/core/compatibility/sdk/6.0/duplicate-files-in-output (дата обращения: 14 мая 2023 г.).

[6] Swagger. — URL: https://swagger.io/ (дата обращения: 14 мая 2023 г.).