Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б10-мм

Петров Владимир Сергеевич

Интеграционное тестирование сервиса курсов в HwProj 2

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доц. кафедры СП, к. т. н. Ю. В. Литвинов

Консультант:
инженер-программист JetBrains, А. В. Бережных

Санкт-Петербург
2022

Оглавление

1. Введение

В настоящее время тестирование является одним из главных этапов разработки программного обеспечения. Программа тестируется с целью проверки на соотвествие поставленным требованиям, нахождения критических и мелких ошибок в исходном коде и для того, чтобы убедиться, что система ведет себя адекватно даже при нестандартном её использовании.

HwProj 2 — веб сервис для проведения онлайн-курсов по программированию и информатике. В перспективе этот проект должен заменить Blackboard, MS Teams в качестве платформы для публикации материалов курса на кафедре системного программирования. На данный момент написана минимальная рабочая версия этого приложения, и она находится на этапе тестирования и исправления ошибок, мешающих комфортному использованию этой версии. Планируется тествое использование данного сервиса к началу следующего учебного года.

Проект написан на ASP.NET[1] и имеет микросервисную архитектуру. В связи с этим каждый микросервис должен быть прокрыт тестами. Интеграционное тестирование является наиболее подходящим для этих целей на данном этапе. Оно позволяет просимулировать реальное использование микросервиса и его взаимодествие с различными компонентами приложения.

Сервис курсов — один из микросервисов HwProj 2. В нем реализуется работа с курсами, домашними работами и домашними заданиями. Предполагается, что большее количество времени пользователи будут работать с различными компонентами, логика которых написана в этом микросервисе. Из этого следует, что задача проверки корректности работы сервсиса курсов первоочередная.

[1] ASP.NET — фреймворк для разработки веб-приложений на платформе .NET

2. Постановка задачи

Целью работы является тестирование сервиса курсов для проверки корректной работы приложения.
Для её выполнения поставлены следующие задачи:

1. Сформировать требования к тестам.
2. Рассмотреть подходы к написанию интеграционных тестов для ASP.NET приложений.
3. Выбрать подход к тестированию.
4. Написать тесты.
5. Оценить количество кода, покрытого тестами.

3. Обзор

3.1. Обзор используемых технологий

- **ASP.NET** [1]
  Фреймворк, работающий на платформе .NET, является разработкой компании Microsoft. Используется при создании веб приложений.

- **C#** [3]
  Объектно ориентированный язык программирования со строгой типизацией. Работает на платформе .NET.

- **NUnit** [5]
  Фреймворк для написания тестов для приложений, написанных на .NET.

- **React** [6]
  Java-script библиотека. Используется в разработке пользовательских интерфейсов в том числе и для одностраничных веб приложений.

- **AutoFixture** [2]
  Библиотека для автогенерации данных.

- **Moq** [4]
  Библиотека для создания mock объектов.

3.2. Архитектура HwProj 2.0.1

Клиентская часть является одностраничным веб-приложением, написанным на React. Серверная часть представляет микросервисное ASP.NET приложение. Клиент общается с сервером через API Gateway. API Gateway — общая API для всей серверной части, главной функцией которой является перенаправление запроса на нужный микросервис.

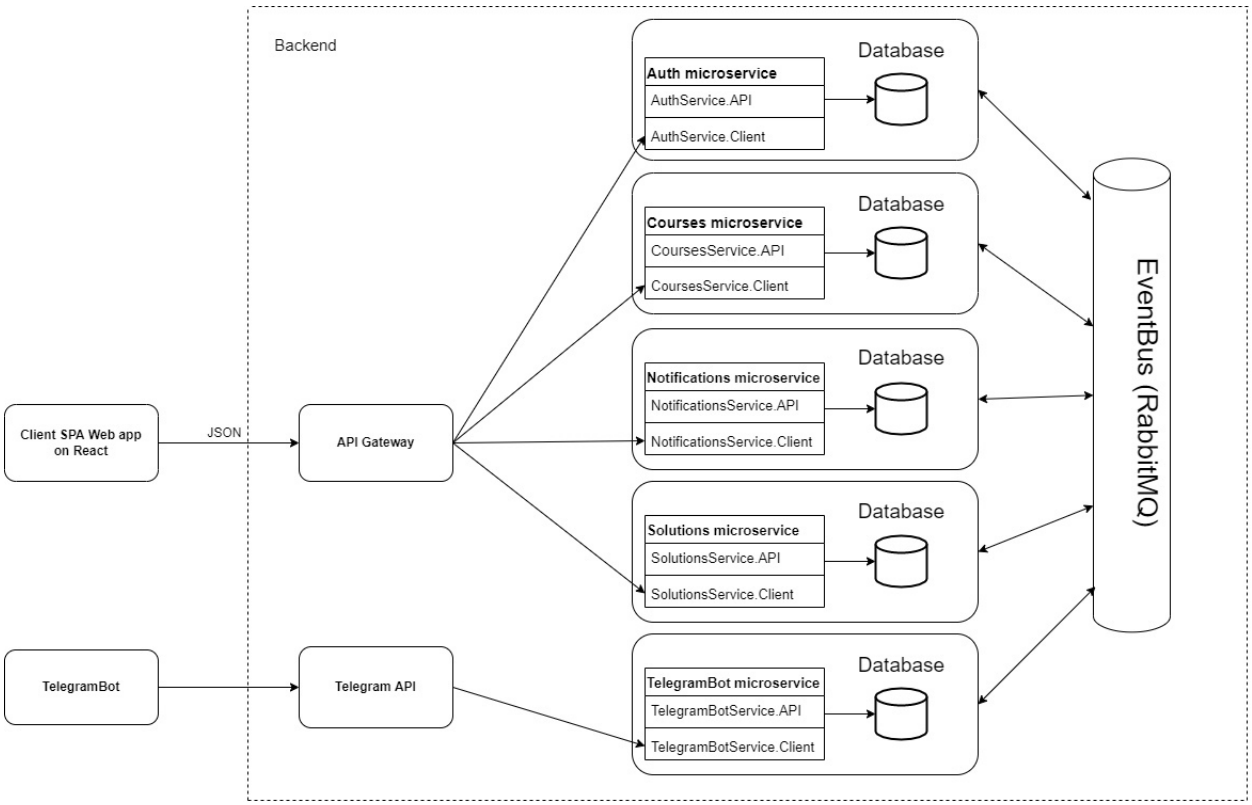

Рис. 1: Архитектура HwProj 2.0.1

Также используется шина сообщений RabbitMQ для реализации уведомлений. Каждый микросервис посылает на неё свои уведомления и их обработкой занимается сервис уведомлений.

4. Требования к тестам

Тесты должны симулировать работу реальной системы. Для этого нужна среда, которая максимально будет приближена к реальным условия. Требуется использовать мок объекты только там, где это необходимо. Сервис курсов должен взаимодействовать с локально запущенными базами данных и микросервисами. Для этих целей будет производится интеграционное тестирование.

Нужно протестировать базовые сценарии. К базовым сценариям относятся:

- Создание, редактирование и удаление курса преподавателем.
- Создание, редактирование и удаление домашней работы преподавателем.
- Создание, редактирование и домашнего задания преподавателем.
- Подача заявки на участие в курсе студентом.
- Принятие заявки студента на участие в курсе.
- Отклонение заявки студента на участние в курсе.
- Добавления преподавателя на курс.
- Проверка, что нельзя добавлять преподавателя на курс, если он не имеет роль "Преподаватель".
- Проверка, что задание, дата публикации которого еще не наступило, не отображается для студента.
- Проверка, что задание отображается для преподавателя курса вне зависимости от даты публикации.

Также в этом сервисе разделяются права доступа между преподавателями курса и остальными пользователями к некоторым функциям.

Нужно проверить, что пользователи отличные от преподавателей не могут выполнять следующие сценарии:

- Редактировать и удалять курс.
- Создавать, редактировать и удалять домашние работы внутри курса.
- Создавать, редактировать и удалять домашние задания.
- Принимать заявки пользователей на участие в курсе.
- Отклонять заявки пользователей на участие в курсе
- Добавлять преподавателя на курс.

5. Реализация

5.1. Выбор подхода к тестированию

Было рассмотрено 2 подхода к написанию интеграционных тестов.

Первый это подход, описанный в документации по интеграционному тестированию с MSDN[2] используя WebApplicationFactory.
WebApplicationFactory — класс, который создает тестовый сервер. Ему на вход подается ссылка на тестируемый проект, там же настраивается, какие дополнительные сервисы и базы данных мы используем.
Плюсы такого подхода:

- Оффициальная документация с примерами и кодом приложений.
- Можно использовать другую базу данных, другие параметры приложения.

Минусом такого подхода можно выделить проблемы присоединения к процессу приложения дебаггером и его отладки.

Второй подход заключается в локальном запуске сервисов и баз данных и обращения к ним.
Плюсы такого подхода:

- Используются реальные сервисы и базы данных, что делает тесты более приближенными к реальности.
- Проще в реализации в сравнении с предыдущим подходом.
- После тестов можно зайти в приложение и посмотреть, что произошло или что сломалось.

Минусом данного подхода является проблема с интеграцией CI, а именно, как CI будет создавать базу данных, в первом подходе можно

[2]MSDN — домашняя страница документации и учебных ресурсов Майкрософт для разработчиков и технических специалистов

сделать базу данных хранящуюся в памяти.

Так как сценарии, покрытые тестами, должны быть максимально приближены к реальному поведению веб-приложения, функции, предоставляющиемые первым подходом, не будут использоваться. Поэтому был выбран второй подход.

5.2. Выбор тестового фреймворка

Выбор фреймворка для тестирования был между xUnit и NUnit. Оба фреймворка отлично подходят для решения поставленной задачи, они поддерживают параллельный запуск тестов, используют атрибуты для информирования фреймворка, какой код и как интерпретировать, как тесты. Был выбран NUnit, так как он обладает более богатой документацией, большим количеством примеров проектов, где он используется, а также из-за того, что вся команда разработки имела опыт работы с ним.

5.3. Выбор библиотек для генерации данных и создания мок объектов

Для генерации данных был выбран AutoFixture, для создания мок объектов - Moq. Это одни из самых популярных библиотек для своих задач. У обеих библиотек имеется хорошая документация, после просмотра которой, отпала надобность искать что-то другое, так как они отлично подходили для выполнения поставленных задач. Дальнейшие поиски и сравнеия заняло бы лишнее время и не принесли бы проекту значительной пользы.

5.4. Тесты

Тесты пишутся по следующему принципу. Переиспользуется уже ранее написанный клиент для сервиса. Для каждого тесты генерируются данные с помощью AutoFixture. Такими данными могут быть модель

пользователя для регистрации, модель курса и другие подобные модели. Через клиент отправляется запрос к сервису и делаются проверки, чтобы убедиться, что выполнилось то, что ожидалось.

Листинг 1: Пример теста на создание курса

```
[Test]
public async Task TestCreateCourse()
{
    // Arrange
    var (userId, _) = await CreateAndRegisterLecture();
    var courseClient = CreateCourseServiceClient(userId);
    var newCourseViewModel = GenerateCreateCourseViewModel();
    // Act
    var courseId =
        await courseClient.CreateCourse(newCourseViewModel, userId);
    // Assert
    var courseFromServer =
        await courseClient.GetCourseById(courseId, userId);
    courseFromServer.Should().BeEquivalentTo(newCourseViewModel);
}
```

В этом примере создается модель преподавателя, а затем производится его регистрация с помощью метода CreateAndRegisterLecture(). Так как мы обращаемся к сервису через клиент, нужно его создать. Делается это через метод CreateCourseServiceClient(userId), которому на вход подается id пользователя, чтобы сервиса курсов мог знать от кого пришел запрос. Далее генерируется модель создаваемого курса с помощью GenerateCreateCourseViewModel(). Затем, используя ранее созданный клиент, вызывается метод CreateCourse, в которую передается модель курса. Проверка осуществляется с помощью того, что мы вызываем метод GetCourseById, который возвращает с сервера модель курса под заданным id, и сравниваем полученную модель с исходной.

5.5. Написание тестов на базовые случаи

Базовые тесты отражают ситуации стандартного использования сервиса.
В результате были написаны следующие тесты:

1. Создание курсов.
2. Редактирование курса.
3. Удаление курса.
4. Создание домашней работы для курса.
5. Редактирование домашней работы.
6. Удаление домашней работы.
7. Добавления задания для домашней работы.
8. Редактирование задания.
9. Удаление задания.
10. Подача заявки на участие в курсе.
11. Принятие заявки студента на участие в курсе.
12. Отклонение заявки студента на участние в курсе.
13. Добавления преподавателя на курс.
14. Проверка работоспособности системы публикаций по дате.

5.6. Тесты на права доступа

Некоторые функции могут использоваться только ментором данного курса. Чтобы это проверить создавались 5 пользователей: ментор курса, приглашенный ментор, преподаватель, но не ментор курса, студент вне курса, студент на курсе. Всеми этими пользователями вызывались методы доступные только менторам данного курса. Проверялось,

что всем кроме менторов приходит код возврата 403(Forbidden). Были написаны проверки на

1. Редактирование курса.
2. Удаление курса.
3. Создание домашней работы.
4. Редактирование домашней работы.
5. Удаление домашней работы.
6. Создание задания.
7. Редактирование задания.
8. Удаление задания.
9. Принятие заявки студента на участие в курсе.
10. Отклонение заявки студента на участие в курсе.
11. Добавления ментора в курс.
12. Добавляемый ментор должен иметь роль ”Преподаватель”.

6. Результаты

6.1. Оценка покрытия кода тестами

Для оценки покрытия кода тестами был использован Cover tests в Rider[3]. Он показывает количество строчек кода, которых покрывают тесты. На рис.2 представлены результаты.

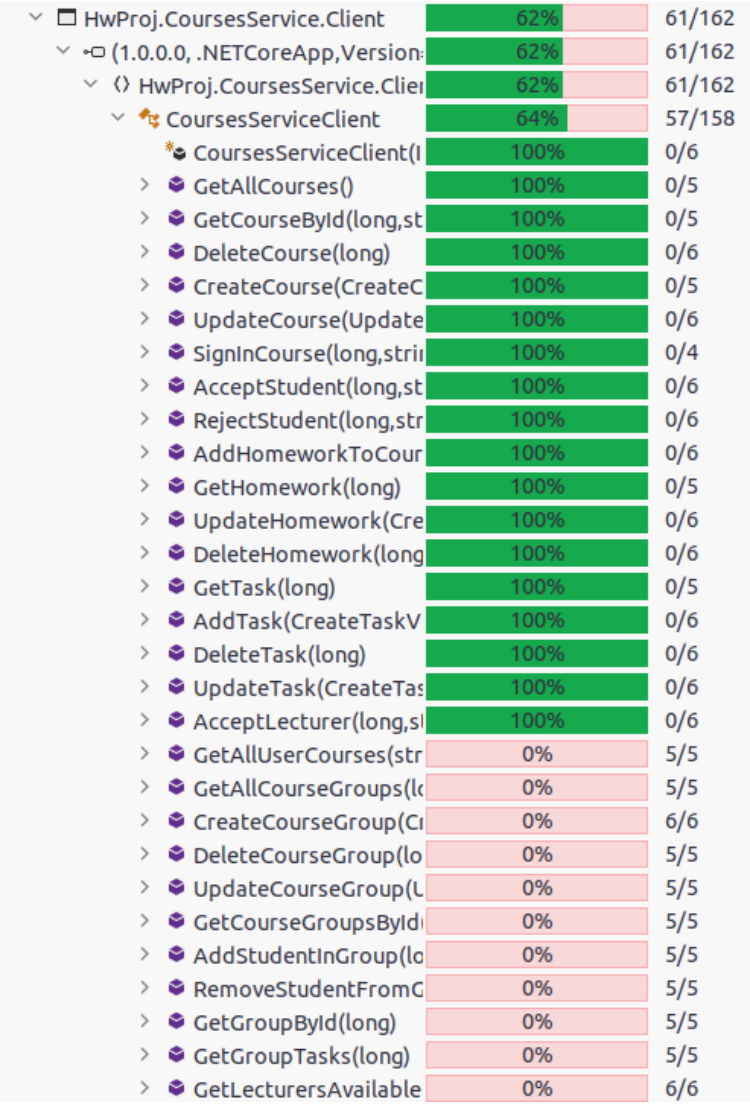

Рис. 2: Результаты после запуска cover tests

Статистика cover tests показывает, что после написания тестов, ко-

[3]Rider — это кросс-платформенная IDE для .NET-разработчиков, основанная на платформе IntelliJ и ReSharper.

личество кода покрытого тестами стала составлять 64%. Если смотреть на методы, которые отвечают за тестируемые сценарии, то там покрытия кода состовляет 100%.

Пуллреквест с реализацией тестов - https://github.com/InteIIigeNET/HwProj-2.0.1/pull/147

6.2. Баг с удалением курса и ошибки с именованием переменных

В результате тестирования был исправлен баг с удалением курса. Удаление курса один из методов, который ограничивает права доступа к нему. Перед тем, как выполнить этот метод, из запроса достается id юзера и проверяется, что он преподаватель данного курса. Баг заключался в том, что этот id не передавался через клиент. Из этого следует, что любой пользователь не мог удалить курс. Баг был исправлен путем того, что в http запрос был добавлен этот самый id.

Также был обнаружена ошибка с несогласованием именования переменных. UpdateCourseViewModel модель использующаяся для редактирования курса. Все её поля должны быть такого же имени, что и в CourseViewModel. Ошибка заключалась в том, что поле IsCompleted в данной модели было записано, как IsComplete. Ошибка была исправлена, как в бэкэнд, так и на фронтенд частях.

Пуллреквест с исправлением багов и ошибок - https://github.com/InteIIigeNET/HwProj-2.0.1/pull/149

6.3. Возвращение информации об ответе к методам с ограниченным правом доступа

Было добавлено возвращение информации о результатх запроса для методов с ограниченным доступом. Это позволило протестировать кор-

ректную работу этих функций.

Пуллреквест с реализацией - https://github.com/InteIligeNET/HwProj-2.0.1/pull/153

6.4. Баг CourseMentorOnlyAttribute

CourseMentorOnlyAttribute — атрибут отвечающий за ограничение прав доступа к методам доступных только преподавателям. Баг был выявлен для методов редактирования/удаления домашних работ, создания/удаления/редактирования домашних заданий. Ошибка заключается в том, что их может вызвать любой пользователь.

Также были выявлены причины этого бага, для этого рассмотрим листинг 2:

Листинг 2: CourseMentorOnlyAttribute

```
public void OnAuthorization(AuthorizationFilterContext context)
{
    var routeData = context.HttpContext.GetRouteData();
    var headers = context.HttpContext.Request.Headers;

    if (routeData.Values.TryGetValue("courseId", out var courseId))
    {
        headers.TryGetValue("UserId", out var userId);
        var mentorIds = _coursesService.GetCourseLecturers
            (long.Parse(courseId.ToString())).Result;

        if (!mentorIds.Contains(userId.ToString()))
        {
            context.Result =
            new StatusCodeResult(StatusCodes.Status403Forbidden);
        }
    }
```

}

Код реализует ту часть атрибута, которая отвечает за ограничения прав доступа.

Баг заключается в строчке:
'if (routeData.Values.TryGetValue("courseId", out var courseId))' - при вызове вышеописанных методов это условие не выполняется из-за того, что в запросах не передается никакая информация об id курса.

7. Заключение

В рамках учебной практики были выполнены следующие задачи:

- Были сформированы требования к тестам.
- Были рассмотрены подходы к написанию интеграционных тестов для ASP.NET приложений.
- Был выбран подход к тестированию.
- Были написаны тесты.
- Была проведена оценка количество кода, покрытого тестами.

Код работы представлен в [7].

Список литературы

[1] ASP.NET. — URL: https://ru.wikipedia.org/wiki/ASP.NET (online; accessed: 2022-05-20).

[2] AutoFixture. — URL: https://autofixture.github.io/.

[3] C#. — URL: https://ru.wikipedia.org/wiki/C_Sharp (online; accessed: 2022-05-20).

[4] Moq. — URL: https://github.com/moq/moq4/tree/ecd540409 (online; accessed: 2022-06-04).

[5] NUnit. — URL: https://nunit.org/ (online; accessed: 2022-05-20).

[6] React. — URL: https://reactjs.org/ (online; accessed: 2022-05-20).

[7] Реализация тестов. — URL: https://github.com/InteIIigeNET/HwProj-2.0.1/pull/147 (online; accessed: 2022-06-04).