

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б09-мм

Тагильцев Семен Вадимович

Автоматическое тестирование сервиса авторизации в HwProj 2.0.1

Отчёт по учебной практике
в форме «Производственное задание»

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

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

Санкт-Петербург
2022

Оглавление

1. Введение	3
2. Постановка задачи	4
3. Обзор	5
3.1. Обзор используемых технологий	5
3.2. Обзор архитектуры веб-сервиса	6
4. Реализация	7
4.1. Выбор подходов тестирования	7
4.2. Реализация тестовых сценариев	8
5. Апробация	10
6. Заключение	11
Список литературы	12

1. Введение

В настоящее время тестирование приложений является неотъемлемой частью разработки, которая позволяет облегчить поддержку и сопровождение продукта. Во время разработки небольших приложений при помощи ручного тестирования обнаруживается и устраняется большое количество ошибок. Но по мере увеличения объема исходного кода продукта, при добавлении новой функциональности становится невозможно ручное тестирование. В такие моменты на помощь разработчикам приходит автоматическое тестирование, которое призвано освободить программистов от рутинной, времязатратной работы.

Одним из таких небольших приложений является веб-сервис для помощи обучения программированию, который имеет микросервисную архитектуры и разрабатывается на математико-механическом факультете СПбГУ – HwProj-2.0.1 [3,4]. На момент развертывания первой рабочей версии приложения, остро встала проблема отсутствия тестов. Новой функциональности с каждым днем пишется только больше и ручное тестирование становится все более трудоемким. Вследствие чего было принято решение начать написание тестов.

Одна из основных функциональностей приложения, которая необходима для авторизации, аутентификации, редактирования данных пользователей и много других вещей, выполняется микросервисом авторизации. Функциональность данного микросервиса является первоочередной при взаимодействии с веб-сервисом и поэтому должна быть покрыта тестами одна из первых.

2. Постановка задачи

Целью работы является проектирование и написание тестов для клиентской и сервисной частей сервиса авторизации:

1. Рассмотреть различные подходы к тестированию, как клиентской, так и серверной частей приложения.
2. Сформировать набор тестовых сценариев.
3. Реализовать тестовые сценарии для серверной части.
4. Реализовать тестовые сценарии для клиентской части.

3. Обзор

3.1. Обзор используемых технологий

Хочется отметить, что технологии, которые напрямую не относятся к тестированию, выбирались без согласования со мной, поэтому лишь приведен их краткий обзор и нет обоснования выбора.

3.1.1. C#

C# [2] — объектно-ориентированный язык программирования. C# не только является одним из наиболее распространенных языков, который используется на математико-механическом факультете СПбГУ, но и имеет большое комьюнити веб-разработчиков. C# непрерывно поддерживается и развивается компанией Microsoft, благодаря чему становится с каждым днем все более и более удобным языком программирования для разработки [10].

3.1.2. ASP.NET Core

ASP.NET Core [1] — это высокопроизводительный, кроссплатформенный фреймворк, который предназначен для создания пользовательских веб-сервисов. Одними из плюсов данного фреймворка является то, что ASP.NET Core имеет подробную документационную базу и разработан для тестируемости.

3.1.3. React

React [6] — это open-source библиотека, предназначенная для создания пользовательских интерфейсов веб-приложений.

3.1.4. NUnit

NUnit [5] — это фреймворк с открытым исходным кодом, созданный для автоматического тестирования приложений под платформой .NET. Данный фреймворк является стандартом в мире тестирования

.NET приложений. NUnit имеет подробную документацию, регулярно выходят обновления, отсутствует привязанность к определенной среде разработки. Автоматические тесты написанные при помощи данной среды легко интегрируются в систему CI (continuous integration), в отличие от MSTest, которые привязаны к Visual Studio. Поэтому вместе с Fluent Assertions и AutoFixture, NUnit становится удобным и практичным инструментом для написания интеграционных тестов.

3.1.5. Selenium WebDriver

Selenium WebDriver [7] — это инструмент для тестирования пользовательского интерфейса веб-приложений. Selenium является стандартом в тестирование пользовательских интерфейсов веб-приложений и имеет библиотеку для платформы .NET, благодаря чему и был выбран данный инструмент.

3.2. Обзор архитектуры веб-сервиса

Веб-приложение HwProj 2.0.1 состоит из двух основных частей: клиентской и серверной. В основу клиентской части легла библиотека React, а микросервисная серверная часть написана на языке программирования C# при помощи фреймворка ASP.NET Core.

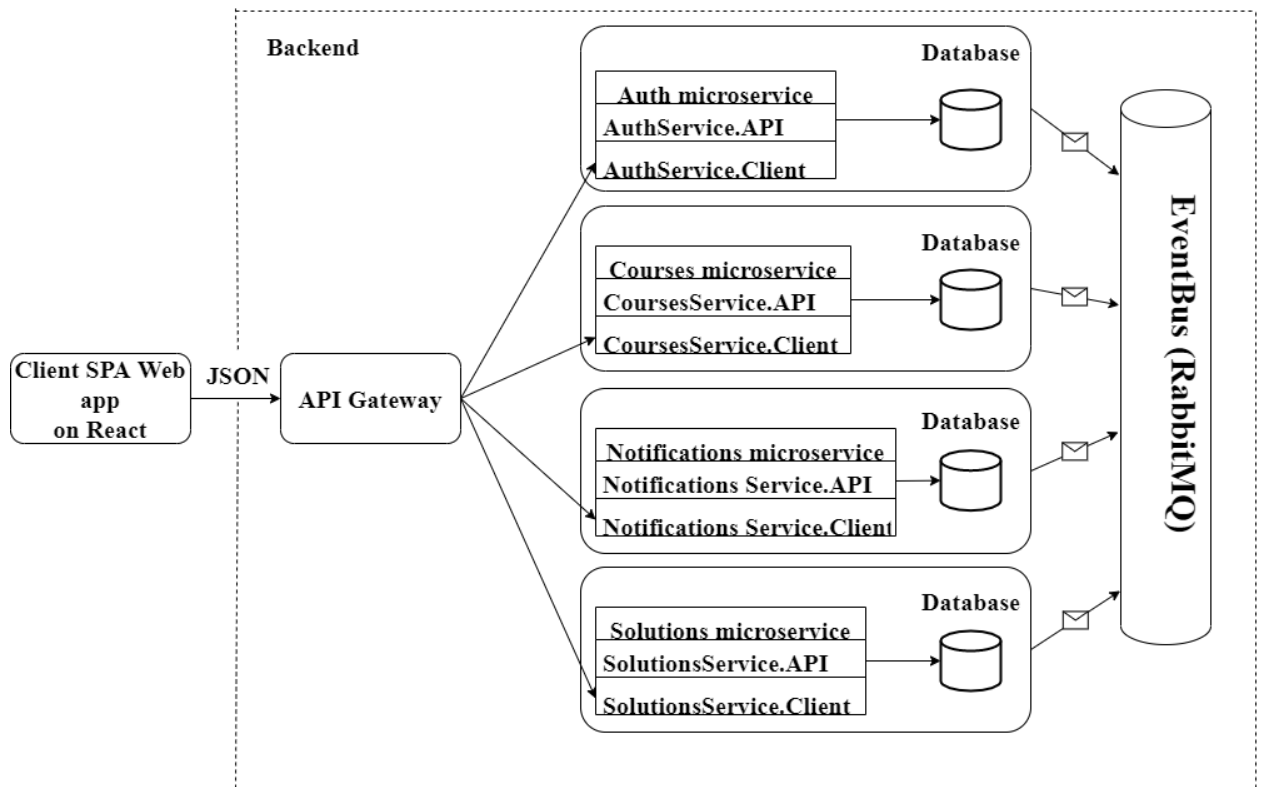


Рис. 1: Схема архитектуры веб-сервиса HwProj 2.0.1

4. Реализация

4.1. Выбор подходов тестирования

Для автоматического тестирования клиентской части сервиса авторизации приложения HwProj 2.0.1 были рассмотрены два подхода: UI-тесты и интеграционные тесты компонент. В интеграционных тестах осуществляется тестирование как целых компонентов, так и взаимодействие разных компонентов друг с другом. UI-тесты в отличие от интеграционных не проверяют мельчайшие детали компонентов, а тестируют их основные функции. При помощи этих тестов пытаются симитировать действия реального пользователя веб-сервиса, тесты переходят по ссылкам, нажимают на кнопки, заполняют поля, ждут выполнения загрузки.

Стоит заметить, что это два абсолютно разных подхода к тестированию, которые оправданно могут быть реализованы в одном проекте. Но в силу того, что для написания интеграционных тестов необходимы

глубокие знания React и TypeScript, а также малого количества пособий и трудоемкости в написании, от этой идеи пришлось отказаться в пользу UI-тестов, которые в свою очередь не требуют глубоких познаний в области тестирования и могут быть написаны на языке программирования C#.

Что касается серверной части, то здесь выбор стоял между интеграционными тестами с применением фреймворка Moq для изолирования микросервиса авторизации и баз данных, а также без данного фреймворка. Был выбран второй вариант исходя из задач: было необходимо протестировать, как работает весь веб-сервис с реально базой данных, а не только изолированный микросервис.

4.2. Реализация тестовых сценариев

После рассмотрения и выбора подходов были составлены тестовые сценарии. Некоторые из были общими для клиентской и серверной частей, но также были и тестовые сценарии, которые специфичны для серверной части. Их возникновение возможно только при прямом обращении к серверу веб-приложения.

Основной особенностью тестирования сервиса авторизации является большое количество появляющихся ошибок в результате невнимательности. Например, указывание некорректного или неправильного email-адреса, пароля при авторизации, регистрации или при попытках отредактировать профиль пользователя. При написании тестов особое внимание было обращено на такие случаи.¹

В тестах использовался паттерн AAA (Arrange-Act-Assert). В каждом тесте сначала производилась подготовка данных (arrange), затем на основе этих данных вызывался один или несколько методов представленных в клиенте сервиса авторизации (act), после чего методы возвращали данные, которые необходимо было проверить (assert).

Для подготовки данных для тестов, чтоб не возникало коллизий, использовалась библиотека AutoFixture. С ее помощью генерировались

¹Что такое авторизация?: <https://www.sravni.ru/enciklopediya/info/chto-takoe-avtorizacija/> (online; accessed: 2022-06-04).

имена, фамилии, пароли и другие данные пользователей. Для удобного сравнения актуальных и ожидаемых данных применялась библиотека Fluent Assertions.

Особенностью тестирования клиентской части приложения является то, что здесь применяется паттерн Page Object. Иными словами страницы веб-приложения представляются в виде классов на языке C#, после чего с ними можно удобно работать.

Во время написания тестов было найдено несколько ошибок и недостатков. Самые основные из них были исправлены, а остальные зафиксированы.

В конечном итоге было написано двенадцать UI-тестов и двадцать один интеграционный тест. Основные тестовые сценарии: авторизация, регистрация, изменения данных пользователей.

```
[Test]
public async Task UserAlreadyExistRegisterTest()
{
    var userData = GenerateRegisterViewModel();
    var registerResult = await _authServiceClient.Register(userData);

    registerResult.Succeeded.Should().BeTrue();
    registerResult.Errors.Should().BeNullOrEmpty();
    registerResult.Value.AccessToken.Should().NotBeNullOrEmpty();

    var result = await _authServiceClient.Register(userData);

    result.Succeeded.Should().BeFalse();
    result.Errors.Should()
        .BeEquivalentTo("Пользователь уже зарегистрирован");
    result.Value.Should().BeNull();
}
```

Рис. 2: Пример интеграционного теста для серверной части

```

[Test]
public void UserRegistrationTwiceTest()
{
    var mainMenu = new MainMenuPageObject(_webDriver);
    var name = GenerateString();
    var surname = GenerateString();
    var middleName = GenerateString();
    var email = GenerateEmail();
    var password = GenerateString();

    mainMenu
        .MoveToRegister()
        .Register(name, surname, email, password, password, middleName)
        .MoveToMenu()
        .SignOff()
        .MoveToRegister()
        .Register(name, surname, email, password, password, middleName);

    var result = new RegisterPageObject(_webDriver).GetResult();

    result.Should().Be("Пользователь уже зарегистрирован");
}

```

Рис. 3: Пример UI-теста для клиентской части

5. Апробация

Для проведения апробации результатов были предприняты следующие шаги:

- Была запущена конфигурация проекта, в которой находились все микросервисы приложения. Была поднята клиентская часть приложения для UI-тестов.
- Запущены интеграционные и UI-тесты.

В результате апробации все тестовые сценарии отработали корректно, приложение работало без критических ошибок, но были зафиксированы незначительные ошибки. Интеграционные тесты серверной части были приняты в основную ветку проекта, UI-тесты находятся на стадии проверки.

6. Заключение

В результате выполнения данной работы были достигнуты следующие результаты:

- Рассмотрены различные подходы к тестированию, как клиентской, так и серверной частей приложения.
- Сформирован набор тестовых сценариев.
- Реализованы тестовые сценарии для серверной и клиентской частей.

С реализацией можно ознакомиться в пулл реквестах на GitHub. [8, 9]

Список литературы

- [1] ASP.NET Core. — URL: <https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-6.0/> (online; accessed: 2022-05-20).
- [2] C#. — URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (online; accessed: 2022-05-20).
- [3] HwProj-2.0.1. — URL: <https://github.com/IntelliNET/HwProj-2.0.1> (online; accessed: 2022-06-04).
- [4] HwProj-2.0.1 Site. — URL: <http://195.70.201.5:3000/> (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] Selenium WebDriver. — URL: <https://www.selenium.dev/documentation/webdriver/> (online; accessed: 2022-05-20).
- [8] UI-тесты. — URL: <https://github.com/IntelliNET/HwProj-2.0.1/pull/154> (online; accessed: 2022-05-20).
- [9] Интеграционные тесты. — URL: <https://github.com/IntelliNET/HwProj-2.0.1/pull/148> (online; accessed: 2022-05-20).
- [10] История языка C#. — URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/whats-new/csharp-version-history/> (online; accessed: 2022-06-04).