МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики
Кафедра современных технологий программирования

МАКАРОВА ЕКАТЕРИНА АЛЕКСАНДРОВНА

Тестирование веб-сайта и API серверного ПО приложения дополненной реальности для визитных карт

Курсовая работа
по дисциплине «Алгоритмизация и программирование»
студентки 2 курса специальности
1-26 03 01 «Управление информационными ресурсами»
дневной формы получения образования

Научный руководитель
Гуща Юлия Вальдемаровна,
старший преподаватель
кафедры современных технологий
программирования

Гродно 2019

РЕЗЮМЕ

Курсовая работа: 44 с., 3 рис., 5 табл., 2 прил., 10 источников.

ТЕСТИРОВАНИЕ, АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ, ТЕСТ-КЕЙСЫ, ТЕСТ-РЕПОРТ, ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA.

Цель исследования: изучение методов и инструментов автоматизации тестирования веб-приложений на языке Java, автоматизация тестирования пользовательского интерфейса веб-сайта приложения и API серверного ПО приложения, улучшение покрытия модульными тестами приложения дополненной реальности для визитных карт.

Методы исследования: анализ, синтез, абстрагирование, моделирование.

В работе приводится обзор инструментов и методов для организации автоматизированного тестирования веб-приложений. В результате проделанной работы, создана тестовая документация для увеличения покрытия модульными тестами кода приложения и организовано автоматическое тестирование на языке Java веб-сайта и API серверного ПО приложения дополненной реальности для визитных карт.

SUMMARY

Course work: 44 pp, 3 fig., 5 tabl., 2 app., 10 ref.

TESTING OF WEB SITE AND SERVER API OF APPLICATION OF AUGMENTED REALITY FOR BUSINESS CARDS

The research objective: studying of the methods and tools for automating the testing of web applications in Java, automating the testing of the user interface of the website of the application and the API of the server software of the application, improving the coverage of augmented reality application tests for business cards with unit tests.

The methods of research: analysis, synthesis, abstraction, modeling.

The work provides an overview of tools and methods for organizing automated testing of web applications. As a result of the work done, test documentation was created to increase the coverage of application code with unit tests and automated testing was conducted in the Java language of the website and API of the server software of the augmented reality application for business cards.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Деловые визитные карточки – один из основных атрибутов современного делового общения. Однако, на данный момент, бизнес-сфера нуждается в новых подходах и современных технологиях, касающихся продвижения товаров и услуг. С целью усовершенствования делового взаимодействия и коммуникации в целом назрела необходимость разработать систему визуализации визитных карточек с применением технологии дополненной реальности.

В системе имеются три группы пользователей: гости – обычные пользователи, клиенты – те, кто приобретает визитки и системные администраторы.

Вся информация приложения хранится и обрабатывается на серверной части проекта, доступ к которому обеспечивается с помощью API. Для удобства пользователя, написано дополнительное ПО – веб-сайт. С помощью веб-сайта пользователь может управлять элементами приложения через привычный ему пользовательский интерфейс.

При столь объемном функционале важной частью создания системы является ее тестирование. Тестирование необходимо для того, чтобы понять, работает ли программа, как ожидается и соответствует ли она предъявляемым к ней требованиям. Актуальность тестирования обусловлена тем, что при его использовании обеспечивается детальное исследование и обнаружение дефектов системы, их анализ и предложение оптимальных способов решения проблемы. Своевременное выявление и исправление ошибок и недоработок имеет огромное значение в процессе разработки программного продукта, поскольку это уменьшает риски и при этом происходит снижение затрат и времени на разработку программного обеспечения. Часто процесс тестирования ПО может быть автоматизирован, что в некоторых случаях может положительно отразится на скорости и качестве тестирования, это позволяет ещё больше повысить качество продукта. В рамках курсовой работы применено как мануальное, так и автоматизированное тестирование, что в совокупности даёт полноценное покрытие функционала системы тестовыми сценариями и тестами отдельных модулей.

Цель данной курсовой работы – изучение методов и инструментов автоматизации тестирования веб-приложений на языке Java, автоматизация тестирования пользовательского интерфейса веб-сайта приложения и API

серверного ПО приложения, улучшение покрытия модульными тестами приложения дополненной реальности для визитных карт.

В процессе написания курсовой работы предусмотрено выполнение следующих функций:

— написание сценариев работы пользователя и, на их основе, создание последовательностей тестов, симулирующих работу пользователя;

— написание ПО для автоматического тестирования основных функций веб-сайта с проверкой всех вариаций действий пользователя;

— написание ПО для автоматического тестирования всех доступных API методов с проверкой всех возможных событий, указанных разработчиком в документации к методам;

— валидация запросов со стороны клиента системы;

— составление отчётов тестирования с подробной информацией о всех найденных ошибках.

РАЗДЕЛ 1. ТЕХНИЧЕСКАЯ ДОКУМЕНТАЦИЯ

1.1 Модульное тестирование веб-сайта системы

Модульное тестирование, или юнит-тестирование – процесс в программировании, позволяющий проверить на корректность отдельные части исходного кода программы. Данный подход заключается в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это дает возможность достаточно быстро проверить, не привело ли очередное изменение кода к ошибкам в уже протестированных местах программы, а также облегчает обнаружение и устранение таких ошибок [1].

1.1.1 Основные функции веб-сайта системы

Выявление и анализ функций и свойств, которыми должна обладать система являются ключевыми на начальном этапе тестирования, поскольку по итогу анализа выявляются модули исходной программы и в дальнейшем пишутся инструкции к тестированию и сценарии использования системы.

Методами анализа и синтеза в веб-сайте приложения дополненной реальности для визитных карт были выявлены следующие функции:

— регистрация нового пользователя;
— вход в систему уже зарегистрированного пользователя;
— переход между разделами навигации сайта;
— возможность перехода в социальные сети.

1.1.2 Тестовая документация

Для структуризации и прослеживаемости процесса тестирования обязательным атрибутом является документация, которая позволяет грамотно продумать покрытие отдельных модулей системы тестами.

Тестовый случай, или тест-кейс – это тестовая документация, описывающая совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Это спецификация входных данных, средств выполнения, процедуры тестирования и ожидаемых результатов, которые определяют один тест для

модуля, который должен быть выполнен для достижения конкретной цели тестирования программного обеспечения.

Тест-кейсы бывают позитивные и негативные. Позитивные тесты всегда идут первыми и служат для того, чтобы показать, что система работает корректно. Негативные тесты, в свою очередь, нужны для тестов с невалидными данными, чтобы проследить поведение системы в непредусмотренных случаях. В совокупности два вида тестов дают большую возможность найти неисправности системы [1].

В то же время существуют две другие классификации тестов, которые также указываются в тестовых случаях: по приоритету (представлены в таблице 1.1) и уровню тестирования (таблица 1.2).

Таблица 1.1 – Классификация тестов по приоритету

Приоритет	Описание
Высокий	Выполняются в первую очередь, отвечают за ключевой функционал.
Средний	Следуют за тестами с высоким приоритетам, отвечают за расширенные функции, которые не всегда являются обязательными.
Низкий	Могут быть опущены или пройдены в последнюю очередь.

Примечание – Источник: собственная разработка.

Таблица 1.2 – Классификация тестов по уровню тестирования

Уровень	Описание
Smoke	Минимальный набор тестов на явные ошибки, не проходящую этот тест программу не имеет смысла отдавать на более глубокое тестирование.
Critical-path	Значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании.
Extended	Проверяется нестандартное использование программного продукта, границы переполнения массивов данных.

Примечание – Источник: собственная разработка.

Любой тест-кейс обязательно включает в себя:

— Номер тест-кейса – необходим для удобной организации хранения тестов;

— Название – основная тема, или идея тест-кейса. Краткое описание его сути;

— Шаги – описание последовательности действий, которая должна привести нас к ожидаемому результату;

— Ожидаемый результат – результат: что мы ожидаем увидеть после выполнения шагов.

Пример полноценного тест-кейса представлен в таблице 1.3, в ней помимо обязательных атрибутов полей добавлена информация о предусловии, отметка о том, автоматизирован ли тест, отметка о прохождении, приоритет и уровень тестирования.

Таблица 1.3 – Тест-кейс

ID тест-кейса	**Модуль**	**Описание**	**Предусловие**	**Шаги**
27	Страница регистрации	Проверка поля Login на валидность	Открыть страницу регистрации	1. Ввести от 4 до 24 символов в поле Login 2. Нажать Sign in
Ожидаемый результат	**Приоритет**	**Уровень**	**Автоматизация**	**Отметка о прохождении**
Значение допустимо и ошибок не возникло	Высокий	Smoke	+	+

Примечание – Источник: собственная разработка.

Грамотно составленный тестовый набор (список тест-кейсов) позволяет протестировать ПО мануально и прослеживать выполнение и покрытие тестами. Полный список тест-кейсов веб-сайта приложения представлен в Приложении 1.

1.2 Средства автоматизация тестирования веб-сайта

Автоматизированное тестирование представляет собой процесс проверки таких функций, как запуск, инициализация, исполнения, анализа и вывода, производимый в автоматическом режиме. Автоматизированное тестирование осуществляется с помощью специального программного обеспечения и вспомогательных инструментов. Этот тип тестирования решает те же задачи что и ручное функциональное тестирования. У автоматизированного тестирования имеется множество достоинств. Так как тесты выполняются с большой скоростью, можно много раз выполнять одни и те же тесты.

1.2.1 Инструмент автоматизации управления браузерами Selenium

Selenium – это инструмент для автоматизированного управления браузерами. Сейчас наиболее популярной областью его применения является автоматизация тестирования веб приложений. Но, при помощи Selenium также можно автоматизировать любые другие рутинные действия в браузере.

При создании теста фреймворк для сборки тестов Maven автоматически загружает клиентскую библиотеку Selenium для языка Java вместе со всеми зависимостями и создает проект, используя файл pom.xml (конфигурационный файл проекта)[2]. Всё, что остаётся – добавить зависимость, отвечающую за использование Selenium и его версию:

```
<dependency>
     <groupId>org.seleniumhq.selenium</groupId>
     <artifactId>selenium-java</artifactId>
     <version>3.9.1</version>
</dependency>
```

Листинг 1.1 – Зависимость Selenium в файле pom.xml

Примечание – Источник: собственная разработка на основе [3].

1.2.2 Среда тестирования TestNG

TestNG - это среда тестирования, предназначенная для упрощения широкого диапазона потребностей в тестировании, от модульного тестирования (тестирование одного класса в отдельности) до интеграционного тестирования (тестирование целых систем, состоящих из нескольких классов, нескольких пакетов и даже нескольких внешних сред, таких как серверы приложений). TestNG позволяет выполнять сложные группировки методов тестов и выстраивать их в произвольном порядке, давать им описания и зависимости одних методов от других. Для использования данной среды в файл проекта pom.xml также нужно добавить зависимость.

```
<dependency>
     <groupId>org.testng</groupId>
     <artifactId>testng</artifactId>
     <version>6.8</version>
     <exclusions>
         <exclusion>
             <groupId>org.testing</groupId>
             <artifactId>org.hamcrest</artifactId>
         </exclusion>
     </exclusions>
</dependency>
```

Листинг 1.2 – Конфигурация TestNG в файле pom.xml

Примечание – Источник: [4].

1.3 Документация к тестированию API

API – это интерфейс прикладного программирования или интерфейс программирования приложений. API позволяет осуществлять связь и обмениваться данными между двумя отдельными модулями программы. Система программного обеспечения, реализующая API, содержит функции либо подпрограммы, которые могут быть выполнены с помощью другого программного обеспечения [1].

Для анализа документации к API использован ресурс для спецификации REST Swagger UI Данный ресурс предоставляет методы для запросов на сервер. Его особенность заключается в том, что он дает возможность не только интерактивно просматривать спецификацию, но и отправлять запросы.

Поскольку документация подразумевается динамической, то и генерируется она из кода с аннотациями, а не переписывается каждый раз вручную. Как итог, существенно облегчается анализ и тестирование запросов к API. Такая документация предоставляет всю необходимую информацию для начала тестирования сервиса. Описание методов даёт краткое определение тому, что должно произойти при удачном выполнении запроса.

Права доступа устанавливают ограничения на выполнения определённых команд разными группами пользователей, что, к примеру, позволяет создавать методы, которые будут доступны только администрации сервиса.

Параметры запроса – главная его часть. Swagger выводит подробное описание для каждого параметра запроса клиента и ответа сервера. Стоит отметить, что Swagger также следит за объектами, которые передаются от клиента к серверу (так называемые DTO), собирает их в список и выводит отдельным блоком. Для каждого DTO составляется дополнительная XML документация, раскрывающая функционал объекта.

1.4 Средства автоматизации тестирования API

1.4.1 Среда тестирования JUnit

JUnit – библиотека для модульного тестирования программ Java. Использование JUnit позволяет проверить код программы без значительных усилий и не занимает много времени. Юнит тесты классов и функций являются своего рода документацией к тому, что ожидается в результате их выполнения. И не просто документацией, а документацией, которая может автоматически проверять код на соответствие предъявленным функциям. Это удобно, и часто тесты разрабатывают как вместе, так и до реализации классов [5].

Поддерживает аннотации для идентификации методов и утверждения для тестирования получаемых результатов. Тесты могут быть организованы в связки тестов (test suites).

Является аналогом TestNG, но более предпочтителен в тестировании API. Для конфигурации среды нужно добавить зависимость в файл проекта pom.xml.

```
<dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.11</version>
     <scope>test</scope>
</dependency>
```

Листинг 1.3 – Конфигурация Junit в файле pom.xml

Примечание – Источник: собственная разработка на основе [6].

1.4.2 REST API

Многие современные веб-приложения используют REST API для взаимодействия с сервером и для интеграции друг с другом.

Неотделим от API понятие подхода REST (от англ. Representational State Transfer – «передача состояния представления»). Данный подход обеспечивает общение между клиентом (как правило, это браузер) и сервером с помощью обычных HTTP-запросов (GET, POST, PUT, DELETE и т. д). Информация от клиента передается в параметрах самих запросов, а информация от сервера – в теле ответа (который может быть, например, JSON-объектом или XML-документом) [7].

REST API подразумевает под собой простые правила:

— каждый URL является ресурсом;

— при обращении к ресурсу методом GET возвращается описание этого ресурса;

— метод POST добавляет новый ресурс;

— метод PUT изменяет ресурс;

— метод DELETE удаляет ресурс.

Эти правила предоставляют простой интерфейс для других приложений, взаимодействие с которым происходит через протокол HTTP.

РАЗДЕЛ 2. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ

2.1 Автоматизация тестирования веб-сайта

В реализации тестирования веб-сайта используется главный тестовый класс WebTestClass, а также пакет с тремя вспомогательными классами, созданными по принципу паттерна Page Factory. Один модуль тестирования – один класс. Класс FileReaderClass – для чтения из текстового файла данных, необходимых для регистрации и логина. Класс HelpMethods, в котором содержатся методы неявного ожидания драйвера браузера, а также метод с наиболее часто повторяющимися действиями в тестах.

2.1.1 Класс FileReaderClass

В представленном классе задействован класс BufferReader, который читает текст из потока ввода символов, буферизуя прочитанные символы, чтобы обеспечить эффективное считывание символов и строк. Основное его назначение в данном случае – чтение строк их текстового файла credentials, путь к которому объявлен в переменной credentialsDirectory.

```
public class FileReaderClass {
    private String credentialsDirectory = "./src/credentials";
    public String readFromFile(int k) throws IOException {
        BufferedReader reader =
 new BufferedReader(new FileReader(credentialsDirectory));
        String line;
        List<String> lines = new ArrayList<String>();
        while ((line = reader.readLine()) != null) {
            lines.add(line);
        }
        return (lines.get(k));
    }
}
```

Листинг 2.1 – Класс FileReaderClass

Примечание – Источник: собственная разработка.

BufferReader построчно считывает данные из файла и может быть вызван в других классах программы. Поскольку строки содержатся в листе,

можно добавлять новые данные строкового типа в текстовый файл без ограничений.

2.1.2 Класс RegisterPage

Класс RegisterPage служит для обработки страницы регистрации веб-сайта. Конструктор данного класса в качестве параметра принимает ссылку на веб-драйвер браузера вспомогательные классы, а также обращается к шаблону проектирования PageFactory (инициализирует объекты/элементы страницы и обращается к ним, только когда в коде есть обращение к ним).

```
public RegisterPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
}
```

Листинг 2.2 – Конструктор класса FileReaderClass

Примечание – Источник: собственная разработка.

Для взаимодействия с отдельными элементами веб-сайта, необходимо обратиться к ним. Чтобы сделать это нужно указать уникальный локатор и его тип. Например, у заголовка страницы регистрации локатор div[class='card-header'] h4, который вычисляется с использованием консоли браузера. Для того чтобы Selenium нашёл элемент используется аннотация @FindBy, которая позволяет считывать локатор элемента типа css или xpath. Локатор типа css обеспечивает быстрый поиск элемента, в то время как локатор типа xpath более надежен в поиске нужного элемента, так как он требует более детального указания пути. В любом случае создается переменная типа WebElement, к которой можно обращаться из любого метода класса. Подобные локаторы более разумно передавать в листы.

```
@FindBy(css = "div[class='card-header'] h4") public WebElement
header;
@FindBy(css = "input") public List<WebElement> fields;
```

Листинг 2.3 – Пример локаторов

Примечание – Источник: собственная разработка.

Поскольку в текущем классе реализуется тестирование регистрации со стороны пользовательского интерфейса необходимо, прежде всего, проверить

наличие полей для регистрации. Для этого нужно обратиться в лист fields к соответствующему полю по индексу, получить его содержимое по атрибуту и сравнить с ожидаемым результатом при помощи класса Assert, используемого TestNG: Assert.assertEquals(fields.get(2).getAttribute("placeholder"), "Login"). Если же нужно проверить наличие объекта на странице, Assert используется в немного иной форме, где указывается логическое значение в ожидаемом результате и опция isDisplayed, которая возвращает true или false, в актуальном: Assert.assertEquals(sign_in_button.isDisplayed(),true).

Одним из ключевых аспектов тестирования форм веб-приложений является валидация, поэтому в поля формы необходимлвводить не только верные, предусмотренные системой данные, но и значения с границ классов эквивалентности, которые служат для избежание избыточных тестов, а также абсурдные для системы значения, пустые запросы.

Например, чтобы отправить пустой запрос, драйвер должен сымитировать нажатие кнопки sign_in_button. Если вернется ответ с просьбой системы заполнить поля (в данном случае курсор возвращается на обязательное первое поле), то тест пройден.

Если требуется ввести в текстбокс какое-либо значение, используется метод sendKeys («значение, которое нужно ввести») сразу после переменной с локатором. Чтобы очистить поле, используется метод clear().

```
WebElement firstName = fields.get(0);
sign_in_button.click();
Assert.assertEquals(helpMethods.currentElements().getAttribute("
placeholder"), "Login");
firstName.sendKeys("a");
sign_in_button.click();
Assert.assertEquals(helpMethods.currentElements().getAttribute("
placeholder"), "First name");
firstName.clear();
```

Листинг 2.4 – Пример частичной валидации поля First Name

Примечание – Источник: собственная разработка.

Последовательность данных методов позволяет провести валидацию полей согласно тест-кейсам и существенно ускорить процесс тестирования. Так, для автоматизированного теста требуется около 2-3 минут, в то время как мануально эти же тесты выполнялись бы около получаса. Однако не всегда тесты могут не пройти из-за ошибок в ПО. Порой драйвер настолько быстро выполняет инструкции, что не успевает увидеть локатор. В таких случаях применяется команда Thread.sleep(), где в параметрах указывается количество

миллисекунд, на которые должен остановиться драйвер и позволить странице загрузиться до конца.

С целью проверки валидной регистрации, для каждого поля по индексу, в цикле посылаются значения из текстового файла с данными. Затем имитируется нажатие на кнопку подтверждения. Поскольку тест идёт со стороны клиента, а ответ ожидается со стороны сервера, то необходимо дождаться ответа об успешной регистрации и сравнить с оповещением (также имеет свой уникальный локатор, который объявлен в начале класса). Аналогичным образом построены классы InitialPage и LoginPage.

```
public void validRegistration() throws IOException,
InterruptedException {
        for (int i = 0; i <fields.size(); i++){
            fields.get(i).clear();
fields.get(i).sendKeys(fileReaderClass.readFromFile(i+1));
        }
sign_in_button.click();
Thread.sleep(200);
Assert.assertTrue(success_alert.isDisplayed());
System.out.println("REGISTRATION TEST PASSED");
}
```

Листинг 2.5 – Метод с регистрацией пользователя

Примечание – Источник: собственная разработка.

2.1.3 Класс HelpMethods

Текущий класс состоит из двух методов: implicitWait и currentElements. Метод implicitWait отвечает за неявные ожидания драйвера – те, которые нет необходимости каждый раз вызывать вручную. Достаточно вызвать его в классе и между каждым действием Selenium будет ожидать указанное количество секунд: driver.manage().timeouts().implicitlyWait(7,SECONDS).

Метод currentElements определяет элемент, на котором установлен курсор в данный момент. Он буквально переключается на него и возвращает ссылку на веб-элемент. Этот метод вынесен в отдельный класс с целью сделать код рациональнее, так как данная функция вызывается неоднократно.

2.1.4 Класс WebTestClass

Класс WebTestClass содержит все тесты и их последовательности. Он состоит из нескольких частей: часть с аннотацией @BeforeClass, в котором настраивается конфигурация для запуска драйвера, запускается всего один раз перед всем тестированием и тесты, разбитые по модулям.

Прежде всего, для запуска теста при помощи драйвера, необходимо указать его тип и путь к нему при помощи метода setProperty и объявить его. Метод manage() позволяет управлять драйвером, а метод window().maximize() разворачивает окно браузера в полноэкранный режим. Используя метод get() можно передать страницу, которую откроет браузер. В данном случае это будет первая строка из текстового файла, содержащая ссылку на локальный хост сайта.

```
@BeforeClass
    public void setUp() throws IOException {
        System.setProperty("webdriver.chrome.driver",
"/home/acciosky/Documents/Drivers/chromedriver");
        driver = new ChromeDriver();
        driver.manage().window().maximize();
            }
```

Листинг 2.6 – Тестовый метод с настройками драйвера

Примечание – Источник: собственная разработка.

Для указания на тест следует прописать аннотацию @Test перед телом метода, там же можно указать описание теста и зависимость от иного теста, указав его название [8]. Если этот шаг пропустить, тесты будут запускаться параллельно, что порой может вызывать ошибки запуска в случае конфликтов тестов. В теле метода указываются методы из классов, относящихся к отдельным страницам и передаются параметры, если это требуется.

```
@Test(description = "Test cases 44-50", dependsOnMethods =
"testRegisterPage")
    public void testLoginPage() throws IOException,
InterruptedException {
        initialPage.toLoginPage();
        loginPage.fieldsPresence();
    }
```

Листинг 2.7 – Тестовый метод для проверки страницы логина

Примечание – Источник: собственная разработка на основе [9].

2.1.5 Результаты теста

Для запуска теста следует запустить главный тестовый класс. Системой автоматически откроется браузер и будет выполняться тестовый сценарий. Результаты теста представлены на рисунке 2.1 и 2.2..

Рисунок 2.1 – Итоги теста в IntelliJ IDEA

Примечание – Источник: собственная разработка.

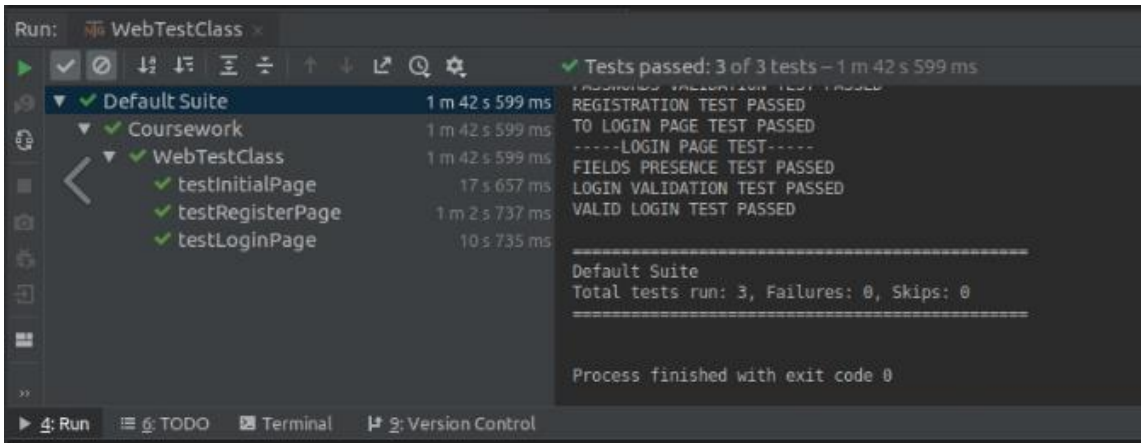

Рисунок 2.2 – Итоги теста в IntelliJ IDEA

Примечание – Источник: собственная разработка.

2.2 Итоги тестирования веб-сайта

В результате мануального и автоматизированного тестирования были выявлены следующие отличия полученного результата от ожидаемого:

Таблица 2.1 – Отчёт о тестировании

№	Описание	Тест-кейс	Шаги воспроизведения	Ожидаемый результат	Актуальный результат
1	При нажатии на иконку почты ничего не происходит	3	1. Нажать на иконку с изображением конверта	1. Открылся почтовый клиент	1. Иконка активна, но не вызывает действий
2	Можно отправить запрос на регистрацию с пробелами в поле First name	20	1. Ввести пробелы в поле First name и нажать Sign in	1. Сообщение об ошибке	1. Запрос успешен
3	Можно отправить запрос на регистрацию с пробелами в поле Second name	26	1. Ввести пробелы в поле Second name и нажать Sign in	1. Сообщение об ошибке	1. Запрос успешен
4	Логин можно зарегистрировать как набор спецсимволов	31	1. Ввести спецсимолы в поле Login и нажать Sign in	1. Сообщение об ошибке	1. Запрос успешен
5	Регистрация успешна с разными значениями в поле пароля и его повтора	43	1. Ввести от 6 до 24 символов в поле Password 2. Ввести от 6 до 24 других символов в поле Repeat password 3. Нажать Sign in	3. Сообщение об ошибке	1. Запрос успешен

Примечание – Источник: собственная разработка.

2.3 Автоматическое тестирование API

2.3.1 Пакет Models и DTO

Классы-модели, использующиеся при тестировании методов API, являются DTO объектами (Data Transfer Object) или объектами, которые содержат информацию для тела POST запроса к API [10].

На данном этапе тестирования используются следующие модели:

— AccountAuthenticationDTO содержит поля логина и пароля, необходимых для авторизации пользователя в системе;

— AccountBanDTO содержит информацию об идентификаторе или логине аккаунта, а также флага бана;

— AccountEmailDTO содержит в себе поле с новым почтовым адресом. Используется для смены email адреса аккаунта;

— AccountRegistrationDTO используется для передачи первоначальной информации об аккаунте, необходимой для его регистрации в системе. Этот класс включает в себя логин, пароль и email регистрирующегося пользователя.

— Social – модель, которая описывает ссылку на социальную сеть. Используется для отображения ссылок в веб-приложении, а также для создания ссылок клиента в мобильном приложении. Включает в себя данные о названии ресурса, ссылки на него, его иконка.

2.3.2 Класс Configuration

Configuration - класс с конфигурацией, который используется для настройки запросов к API. Файл содержит домен (адрес) API, который использует каждый тест для направления запроса. Также в Configuration содержится метод получения стандартной спецификации (параметров) запроса, которые определяют настройки HTTPS подключения, тип контента, отправляемого и получаемого в тестовых запросах (в системе используется JSON), а также хедер (header) bearer авторизации, куда автоматически вписывается токен авторизованного аккаунта.

```
      public static RequestSpecification getRequestSpec() {
        initializeRestAssuredSettings();

        RequestSpecBuilder specBuilder = new
RequestSpecBuilder();
        specBuilder.setRelaxedHTTPSValidation();

        RequestSpecification spec = specBuilder.build();
        spec.contentType(ContentType.JSON);
        spec.accept(ContentType.JSON);

        return spec;
    }
```

Листинг 2.6 – Метод получения стандартной спецификации

Примечание – Источник: собственная разработка.

2.3.4 Класс APITestClass

APITestClass - главный класс тестирования API, который содержит в себе последовательности тестов. Данный класс создаёт экземпляры тестовых классов, привязанных к контроллерам, и вызывает тесты из этих классов по последовательностям. Последовательности тестов строятся по следующему принципу: одна последовательность описывает все позитивные тесты определённого метода, другая – все негативные этого же метода. Таким образом можно просто отследить работоспособность API без использования логов тестирования.

AccountTests включает в себя все тесты, связанные с управлением аккаунтами в системе. Кроме положительных тестов он содержит и негативные тесты, проверяющих корректность возвращаемых ошибок. Класс содержит методы тестирования аутентификации аккаунта (позитивная аутентификация с верными данными пользователя), невалидного тестирования аккаунта со следующими ошибками: пропущен логин аккаунта, пропущен пароль, данные аккаунта не отправлены, данные несуществующего аккаунта.

Проверка регистрации пользователя с новыми действительными данными, регистрации с пропущенным логином, паролем, почтовым адресом, регистрации с данными, где логин или почтовый адрес уже заняты. Тесты получения кратких сведений об аккаунте и профиле авторизованного пользователя, тесты с невалидными данными методов – пропущенный или

невалидный токен пользователя. Также присутствует тест проверки обновления email адреса, тесты с разными ошибками при его обновлении.

OptionsTests включает в себя методы тестирования добавочной информации о системе. Здесь тестируются методы получения социальный сетей, связанных с проектом, а также методы управления ими (добавление новых сетей, редактирование, удаление).

2.3.3 Класс AccountTests

Методы API тестирования используют для тестов сервис для автоматического тестирования API Rest-Assured. Данный сервис позволяет относительно просто формировать запросы к API с помощью обычных Java методов.

Разберём на примере один из методов класса AccountTests:

Метод authenticateCredentialsOK принимает объект AccountCredentialsDTO, используемый как хранилище входных данных аккаунта. Основная задача метода – авторизовать пользователя и возвращать токен аккаунта для последующего тестирования.

```
public String authenticateCredentialsOK(AccountAuthenticationDTO
dto) {
        String token = given()
                .spec(Configuration.getRequestSpec())
                .body(dto).when()
                .post(Configuration.getDomain() +
"/api/account/authenticate")
                .then()
                .statusCode(200)
                .body("code", equalTo(200))
                .extract()
                .path("object");

        System.out.println("Token: " + token);
        return token;
               }
```

Листинг 2.7 – Метод authenticateCredentialsOK

Примечание – Источник: собственная разработка.

Здесь метод given() является целым блоком функций и используется для определения параметров запроса к методу API.

Метод spec() определяет спецификацию (основные параметры) запроса. В каждом тесте к API вызывается данный метод, в который мы передаём созданный в Configuration RequestSpecification (метод Configuration.getRequestSpec()).

Метод body() устанавливает контент для тела запроса, который будет парситься соответствующим парсером, установленным в RequestSpecification.

Метод when() – второй блок, определяющий HTTP метод, с которым будет проводиться запрос к API (GET, POST, PUT, DELETE). Каждый метод запроса в данном блоке имеет как минимум один параметр – URL запроса.

Метод then() – третий и последний блок, определяющий действия, которые необходимо выполнить с результатом запроса. Здесь проходит проверка на его получение, валидность данных, соответствие http кодов ошибок.

К примеру, в данном примере мы проверяем, является ли код ответа от сервера (метод statusCode) 200 (OK), вернул ли сервер json объект с параметром «code» и значением 200 (метод body), и, если вышеперечисленные требования верны, возвращаем (метод extract) из всего теста значение объекта с путём (метод path) «object».

2.3.4 Результаты теста

В данном случае тесты будут запускаются параллельно и независимо друг от друга, значение имеет лишь первый метод, при котором получается токен для выполнения дальнейших нескольких методов.

Рисунок 2.3 – Итоги теста в IntelliJ IDEA

Примечание – Источник: собственная разработка.

2.4 Итоги тестирования API приложения

В результате мануального и автоматизированного тестирования были выявлены следующие отличия полученного результата от ожидаемого:

Таблица 2.2 – Отчёт о тестировании

№	Описание	Тест-кейс	Шаги воспроизведения	Ожидаемый результат	Актуальный результат

1	Возможно обновить email вводом пустого значения	14	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"email": ""
			}
				Ответ сервера 400	Ответ сервера 200

2	Возможно обновить email вводом невалидного значения email	15	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"email": "string"
			}
				Ответ сервера 400	Ответ сервера 200

Примечание – Источник: собственная разработка.

В целом, две найденные ошибки можно обобщить в одну, поскольку они связаны с невалидной обработкой запроса.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы изучены современные методы автоматизации тестирования пользовательского интерфейса веб-приложений и API серверного ПО, и на основании полученных знаний, а также проведенного анализа тестируемой системы, выбраны подходы, позволяющие выполнить поставленные задачи. В соответствии с выбранными методиками подобраны инструментальные среды – JUnit и TestNG, архитектура API REST, средство для автоматизации работы браузера Selenium Webdriver.

В процессе выполнения курсовой работы решены следующие задачи:

— написание сценариев работы пользователя и, на их основе, создание последовательностей тестов, симулирующих работу пользователя;

— написание ПО для автоматического тестирования основных функций веб-сайта с проверкой всех вариаций действий пользователя;

— написание ПО для автоматического тестирования всех доступных API методов с проверкой всех возможных событий, указанных разработчиком в документации к методам;

— валидация запросов со стороны клиента системы;

— составление отчётов тестирования с подробной информацией о всех найденных ошибках.

В силу того, что критерием оценки достижения цели является готовый набор документации для проведения тестирования, а также функционирующее ПО для его автоматизации, возможно говорить о том, что поставленная цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – Минск: Четыре четверти, 2017. – 312 с.
2. Selenium 2.0 и WebDriver [Электронный ресурс] / Проект портала Software-Testing.Ru. – Software-Testing.Ru, 2012. – Режим доступа: https://selenium2.ru/docs/webdriver.html. – Дата доступа: 20.05.2019.
3. Официальный сайт проекта Selenium [Электронный ресурс] – 2017. – Режим доступа: http://www.seleniumhq.org. – Дата доступа: 21.05.2019.
4. TestNG Documentation [Электронный ресурс] – TestNG.Org, – Режим доступа: https://testng.org/doc/documentation-main.html. – Дата доступа: 20.05.2019.
5. JUnit - Test Framework [Электронный ресурс] – TutorialSpoints. – Режим доступа: https://www.tutorialspoint.com/junit/junit_test_framework.htm. – Дата доступа: 22.05.2019.
6. Тестирование программы, JUnit [Электронный ресурс] – Java-online.ru, 2017. – Режим доступа: http://java-online.ru/blog-junit.xhtml. – Дата доступа: 22.05.2019.
7. Освоение тестирования REST API [Электронный ресурс] – Software-testing.ru, 2017. – Режим доступа: http://http://software-testing.ru/library/testing/general-testing/2518-rest-api-testing. – Дата доступа: 23.05.2019.
8. Статья «Java Test Setup Example» [Электронный ресурс] – 2017. – Режим доступа: https://wiki.saucelabs.com/display/DOCS/Java+Test+Setup+Example. – Дата доступа: 21.05.2019.
9. Статья «Selenium Tutorial – WebDriver Basics» [Электронный ресурс] – ToolsQA, 2017. – Режим доступа: http://toolsqa.com/selenium-tutorial. – Дата доступа: 15.05.2019.
10. Data Transfer Object (Объект передачи данных) [Электронный ресурс] / Справочник «Паттерны проектирования» – 2016. – Режим доступа: http://design-pattern.ru/patterns/data-transfer-object.htmll. – Дата доступа: 25.05.2019.

ПРИЛОЖЕНИЕ 1

Тестовые случаи

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
1	Запуск сайта	Запуск локального хоста сайта в браузере	1. Открыть локальный хост сайта	1. В браузере открылась страница сайта	Критический	Smoke	+	+
2	Начальная страница	Проверка наличия иконок на странице	1. Открыть сайт на начальной странице 2. Запрос к api отправлен автоматически (/api/web/social) 3. Результатом запроса является массив социальных сетей (VK, Instagram, Email) с иконками и ссылками	1. Ниже заголовка страницы располагаются 3 иконки социальных сетей: Почта, Instagram, VK	Средний	Critical-path	+	+
3	Начальная страница	Проверка функциональности иконки почты	1. Нажать на иконку с изображением конверта	1. Открылся почтовый клиент	Средний	Critical-path	+	-
4	Начальная страница	Проверка функциональности иконки Instagram	1. Нажать на иконку с изображением логотипа Instagram	1. Открылся профиль @irmwork в Instagram	Средний	Critical-path	+	+
5	Начальная страница	Проверка функциональности иконки VK	1. Нажать на иконку с изображением иконки VK	1. Открылась страница группы IRM Studio в социальной сети VK	Средний	Critical-path	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
6	Навигация сайта	Проверка наличия навигации	1. Открыть сайт	1. В верхней части страницы располагается блок, полностью замощающий страницу по ширине	Высокий	Smoke	+	+
7	Навигация сайта	Проверка наличия пунктов для логина и регистрации	1. Открыть сайт	1. Навигация содержит разделы"Login", "Register"	Высокий	Smoke	+	+
8	Навигация сайта	Проверка функциональности элемента "Login"	1. Нажать на "Login"	1. Открылась страница логина	Высокий	Smoke	+	+
9	Навигация сайта	Проверка функциональности элемента "Register"	1. Нажать на "Register"	1. Открылась страница регистрации с соответствующей формой	Высокий	Smoke	+	+
10	Страница регистрации	Проверка наличия текстбокса "First name"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "First name"	Высокий	Smoke	+	+
11	Страница регистрации	Проверка наличия текстбокса "Second name"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "Second name"	Высокий	Smoke	+	+
12	Страница регистрации	Проверка наличия текстбокса "Login"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "Login"	Высокий	Smoke	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
13	Навигация сайта	Проверка наличия текстбокса "Email"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "Email"	Высокий	Smoke	+	+
14	Навигация сайта	Проверка наличия текстбокса "Password"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "Password"	Высокий	Smoke	+	+
15	Навигация сайта	Проверка наличия текстбокса "Repeat password"	1. Открыть страницу регистрации	1. На странице есть текстбокс с подписью "Repeat password"	Высокий	Smoke	+	+
16	Навигация сайта	Проверка наличия кнопки с текстом "Sign in"	1. Открыть страницу регистрации	1. На странице есть кнопка с текстом "Sign in"	Средний	Critical-path	+	+
17	Страница регистрации	Проверка поля First name на валидность	1. Ввести валидное значение от 2 до 24 символов в поле First name 2. Нажать Sign in	2. Значение допустимо и ошибок не возникло	Высокий	Smoke	+	+
18	Страница регистрации	Проверка поля First name на обязательность	1. Оставить поле First name пустым 2. Нажать Sign in	2. Значение допустимо и ошибок не возникло	Средний	Critical-path	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
19	Страница регистрации	Проверка поля First name с невалидными значениями и значениями из классов эквивалентности	1. Ввести 1 символ в поле First name и нажать Sign in 2. Ввести 2 символа в поле First name и нажать Sign in 3. Ввести 23 символа в поле First name и нажать Sign in 4. Ввести 24 символа в поле First name и нажать Sign in 5. Ввести 25 символов в поле First name и нажать Sign in.	1. Сообщение об ошибке 2. Значение допустимо, регистрация успешна 3. Значение допустимо, регистрация успешна 4. Значение допустимо, регистрация успешна 5. Значение допустимо, в строке 24 символа, регистрация успешна	Средний	Critical-path	+	+
20	Страница регистрации	Проверка поля First name путём ввода пробелов	1. Ввести пробелы в поле First name и нажать Sign in	1. Сообщение об ошибке	Низкий	Extended	+	-
21	Страница регистрации	Проверка поля First name путём ввода только спецсимволов	1. Ввести только от 2 спецсимволов в поле First name и нажать Sign in	1. Значение допустимо, регистрация успешна	Низкий	Extended	+	+
22	Страница регистрации	Проверка поля Second name на валидность	1. Ввести валидное значение от 2 до 24 символов в поле Second name	2. Значение допустимо и ошибок не возникло	Средний	Critical-path	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
23	Страница регистрации	Проверка поля Second name на обязательность	1. Оставить поле Second name пустым **2. Нажать Sign in**	2. Значение допустимо и ошибок не возникло	Средний	Critical-path	+	+
24	Страница регистрации	Проверка поля Second name путём ввода пробелов	1. Ввести пробелы в поле Second name и нажать Sign in	1. Сообщение об ошибке	Низкий	Extended	+	-
25	Страница регистрации	Проверка поля Second name с невалидными значениями и значениями из классов эквивалентности	1. Ввести 1 символ в поле Second name и нажать Sign in 2. Ввести 2 символа в поле Second name и нажать Sign in 3. Ввести 23 символа в поле Second name и нажать Sign in 4. Ввести 24 символа в поле Second name и нажать Sign in 5. Ввести 25 символов в поле Second name и нажать Sign in.	1. Сообщение об ошибке 2. Значение допустимо, регистрация успешна 3. Значение допустимо, регистрация успешна 4. Значение допустимо, регистрация успешна 5. Значение допустимо, в строке 24 символа, регистрация успешна	Средний	Critical-path	+	+
26	Страница регистрации	Проверка поля Second name путём ввода только спецсимволов	1. Ввести только от 2 спецсимволов в поле Second name и нажать Sign in	1. Значение допустимо, регистрация успешна	Низкий	Extended	+	+
27	Страница регистрации	Проверка поля Login на валидность	1. Ввести от 4 до 24 символов в поле Login 2. Нажать Sign in	2. Значение допустимо и ошибок не возникло	Высокий	Smoke	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
28	Страница регистрации	Проверка поля Login на обязательность	1. Оставить поле Login пустым 2. Нажать Sign in	2. Сообщение об ошибке	Высокий	Critical-path	+	+
29	Страница регистрации	Проверка поля Login с невалидными значениями и значениями из классов эквивалентности	1.Ввести 3 символа в поле Login и нажать Sign in 2.Ввести 4 символа в поле Login и нажать Sign in 3. Ввести 23 символа в поле Login и нажать Sign in 4. Ввести 24 символа в поле Login и нажать Sign in 5. Ввести 25 символов в поле Login и нажать Sign in.	1. Сообщение об ошибке 2. Значение допустимо, регистрация успешна 3. Значение допустимо, регистрация успешна 4. Значение допустима, регистрация успешна 5. Значение допустимо, в строке 24 символа, регистрация успешна	Средний	Critical-path	+	+
30	Страница регистрации	Проверка поля Login путём ввода пробелов	1. Ввести пробелы в поле Login и нажать Sign in	1. Сообщение об ошибке	Средний	Critical-path	+	-
31	Страница регистрации	Проверка поля Login путём ввода только спецсимволов	1. Ввести спецсимолы в поле Login и нажать Sign in	1. Сообщение об ошибке	Низкий	Extended	+	-
32	Страница регистрации	Проверка поля Login путём ввода туже имеющегося в системе логина	1. Ввести уже имеющийся в системе логин и нажать Sign in	1. Сообщение об ошибке	Высокий	Extended	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
33	Страница регистрации	Проверка поля Email на валидность	1. Ввести валидный email в соответствующее поле 2. Нажать Sign in	2. Значение допустимо и ошибок не возникло	Высокий	Smoke	+	+
34	Страница регистрации	Проверка поля Email на обязательность	1. Оставить поле Email пустым 2. Нажать Sign in	2. Сообщение об ошибке	Высокий	Critical-path	+	+
35	Страница регистрации	Проверка поля Email с невалидными значениями и значениями из классов эквивалентности	1.Ввести email без "@" и нажать Sign in 2. Ввести email с "@@" и нажать Sign in 3. Ввести email без домена и нажать Sign in 4. Ввести email без значения до символа "@" и нажать Sign in	1. Сообщение об ошибке 2. Сообщение об ошибке 3. Сообщение об ошибке 4. Сообщение об ошибке	Средний	Critical-path	+	-
36	Страница регистрации	Проверка поля Email путём ввода пробелов	1. Ввести пробелы в поле Email и нажать Sign in	1. Сообщение об ошибке	Средний	Critical-path	+	+
37	Страница регистрации	Проверка поля Email путём ввода туже имеющегося в системе email	1. Ввести уже имеющийся в системе email и нажать Sign in	1. Сообщение об ошибке	Высокий	Extended	+	+
38	Страница регистрации	Проверка полей Password и Repeat password на валидность	1. Ввести от 6 до 24 символов в поле Password 2. Ввести от 6 до 24 символов в поле Repeat password 3. Нажать Sign in	3. Значения допустимы и ошибок не возникло	Высокий	Smoke	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
39	Страница регистрации	Проверка полей Password и Repeat password на обязательность	1. Оставить поля Password и Repeat password пустыми 2. Нажать Sign in	2. Сообщение об ошибке	Высокий	Critical-path	+	+
40	Страница регистрации	Проверка поля Password на обязательность	1. Ввести значение в поле Repeat password 2. Оставить поле Password пустым 3. Нажать Sign in	3. Сообщение об ошибке	Средний	Critical-path	+	+
41	Страница регистрации	Проверка поля Repeat password на обязательность	1. Ввести значение в поле Password 2. Оставить поле Repeat password пустым 3. Нажать Sign in	3. Сообщение об ошибке	Средний	Critical-path	+	+
42	Страница регистрации	Проверка полей Password и Repeat password с разными значениями	1. Ввести от 6 до 24 символов в поле Password 2. Ввести от 6 до 24 других символов в поле Repeat password	3. Сообщение об ошибке	Высокий	Smoke	+	-
43	Страница регистрации	Проверка поля Password с невалидными значениями и значениями из классов эквивалентности	1. Ввести 5 символов в поле Password и нажать Sign in 2. Ввести 6 символов в поле Password и нажать Sign in 3. Ввести 7 символов в поле Password и нажать Sign in 4.Ввести 23 символа в поле Password и нажать Sign in	1. Сообщение об ошибке 2. Значение допустимо, регистрация успешна 3. Значение допустимо, регистрация успешна 4. Значение допустимо, регистрация успешна	Низкий	Extended	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
44	Страница логина	Проверка наличия поля Login	1. Нажать Login в навигации сайта	1. На странице есть текстбокс с подписью "Login"	Высокий	Smoke	+	+
45	Страница логина	Проверка наличия поля Password	1. Нажать Login в навигации сайта	1. На странице есть текстбокс с подписью "Password"	Высокий	Smoke	+	+
46	Страница логина	Проверка наличия кнопки Log in	1. Нажать Login в навигации сайта	1. На странице есть кновка с подписью "Log in"	Высокий	Smoke	+	+
47	Страница логина	Логин с валидными данными	1. Ввести валидное значение в поле Login 2. Ввести валидное значение в поле Password 3. Нажать Log in	3. Логин успешен	Высокий	Smoke	+	+
48	Страница логина	Логин с неверным логином	1. Ввести неверное значение в поле Login 2. Ввести валидное значение в поле Password	3. Ошибка входа	Высокий	Smoke	+	+
49	Страница логина	Логин с неверным паролем	1. Ввести от 6 до 24 символов в поле Password 2. Ввести от 6 до 24 символов в поле Repeat password	3. Ошибка входа	Высокий	Smoke	+	+
50	Страница логина	Логин с пустыми полями логина и пароля	1. Нажать Log in"	1. Ошибка входа	Средний	Critical-path	-	-

ПРИЛОЖЕНИЕ 2

Тестовые случаи API

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

1	POST /api/Account/authenticate	Логин с валидными данными	Header:
			"Accept": "application/json-patch+json”
			Тело запроса: {
			"login": "string",
			"password": "string" }
				Код 200
				Тело ответа: {
				"code": 200,
				"meta": "string",
				}
					Критический	Smoke	+	+

2	POST /api/Account/authenticate	Логин с пропуском поля логина	Header: "Accept": "application/json-patch+json” Тело запроса: { "login": "", "password": "string" }	Код 400, данные не валидны	Средний	Critical-path	+	+
3	POST /api/Account/authenticate	Логин с пропуском поля пароля	Header: "Accept": "application/json-patch+json” Тело запроса: { "login": "string", "password": "" }	Код 400, данные не валидны	Средний	Extended	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

4	POST /api/Account/authenticate	Логин с пустыми полями	Header:
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "",
			"password": ""
			}
				Код 400, данные не валидны	Высокий	Critical-path	+	+

5	POST /api/Account/authenticate	Логин с неверными или несуществующими данными	Header: "Accept": "application/json-patch+json” Тело запроса: { "login": "string", "password": "string"}	Код 404	Высокий	Critical-path	+	+

6	POST /api/Account/register	Регистрация с валидными значениями	"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "string",
			"password": "string",
			"email": "string"
			}
				Код 200
				Тело ответа:
				{
				"code": 0,
				"meta": "string",
				"object": "string"
				}
					Критический	Smoke	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

7	POST /api/Account/register	Регистрация с пропуском поля логина	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "",
			"password": "string",
			"email": "string"
			}
				Код 400, данные не валидны	Средний	Critical-path	+	+

8	POST /api/Account/register	Регистрация с пропуском поля пароля	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "string",
			"password": "",
			"email": "string"
			}
				Код 404	Средний	Critical-path	+	+

9	POST /api/Account/register	Регистрация с пропуском поля email	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "string",
			"password": "string",
			"email": ""
			}
				Код 400, данные не валидны	Средний	Extended	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

10	POST /api/Account/register	Регистрация с уже существующим логином	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "string",
			"password": "string",
			"email": "string"
			}
				Код 409	Высокий	Critical-path	+	+

11	POST /api/Account/register	Регистрация с уже существующим email	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "string",
			"password": "string",
			"email": "string"
			}
				Код 410	Высокий	Critical-path	+	+

12	POST /api/Account/register	Регистрация с пустыми полями	Метод: POST
			/api/Account/register
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"login": "",
			"password": "",
			"email": ""
			}
				Код 400, данные не валидны	Средний	Critical-path	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

13	POST/api/Account/email	Обновление существующего email	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			
			Тело запроса:
			{
			"email": "string@string"
			}
				Код 200	Высокий	Smoke	+	+

14	POST/api/Account/email	Обновление email с пустым значнием	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			
			Тело запроса:
			{
			"email": ""
			}
				Код 400, данные не валидны	Низкий	Extended	+	-

15	POST/api/Account/email	Обновление существующего email с невалидным email (без @)	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			
			Тело запроса:
			{
			"email": "string"
			}
				Код 400, данные не валидны	Низкий	Extended	+	-

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении

16	POST/api/Account/email	Обновление существующего email с невалидным email (с @@)	Метод: POST
			/api/Account/email
			"Accept": "application/json-patch+json”
			Тело запроса:
			{
			"email": "string@@string"
			}
				Код 400, данные не валидны	Низкий	Extended	+	+

17	GET /api/Account/getAccountInfo	Получение информации о текущем пользователе	Метод: GET
			/api/Account/getAccountInfo
			
			Header:
			"Accept": "application/json-patch+json”,
			"Authentication: Bearer {token}"
				Код 200
				Тело ответа:
				{
				"code": 200,
				"meta": null,
				"object": {
				"id": {token},
				"login": {login},
				"permission": {permission},
				"code": {code}
				}
				}
					Средний	Critical path	+	+

18	/api/Account/getAccountInfo	Получение информации о пользователе с неверным токеном	Метод: GET /api/Account/getAccountInfo Header: "Accept": "application/json-patch+json”, "Authentication: Bearer {invalid token}"	Код 401	Средний	Critical path	+	+

№	Модуль	Описание	Шаги	Ожидаемый результат	Приоритет теста	Уровень тестирования	Автоматизация	Отметка о прохождении
19	GET /api/Account/getAccountInfo	Получение информации о пользователе с пустым токеном	Метод: GET /api/Account/getAccountInfo Header: "Accept": "application/json-patch+json”, "Authentication: Bearer {}"	Код 402	Средний	Critical path	+	+

20	GET /api/Account/getProfileInfo	Получение информации о текущем пользователе	Метод: GET /api/Account/getAccountInfo
			
			Header:
			"Accept": "application/json-patch+json”,
			"Authentication: Bearer {token}"
				Код 200
				Тело ответа:
				{
				"code": 200,
				"meta": null,
				"object": {
				"id": {token},
				"login": {login},
				"permission": {permission},
				"code": {code}
				}
				}
					Средний	Critical path	+	+

21	GET /api/Account/getProfileInfo	Получение информации о пользователе с неверным токеном	Метод: GET /api/Account/getAccountInfo Header: "Accept": "application/json-patch+json”, "Authentication: Bearer {invalid token}"	Код 402	Средний	Critical path	+	+