«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОКОГНИТИВНЫХ ТЕХНОЛОГИЙ

**Автоматизация тестирования ПО**

Расчетно-пояснительная записка

курсового проекта по дисциплине

«Веб-разработка»

студент группы 221-361

**Комендант Максим Игоревич.**

Преподаватели:

асс. В.С. Мохначев

**Москва, 2024 г.**

**Введение**

Целью курсового проекта является изучение существующих принципов и подходов к автоматизации тестирования, приобретение навыков и изучение методов автоматизированного тестирования ПО в процессе выполнения заданий курсовой работы и получение навыков использования на начальном профессиональном уровне популярных средств автоматизации тестирования, формирования и выполнения автоматизированных тестов.

В ходе курсового проекта выполняются следующие задачи:

– изучение материалов лекционных и практических занятий из курса LMS «Введение в автоматизированное тестирование»;

– выполнение заданий по тестированию фронтенда *—* тестирование "LambdaTest Sample App", тестирование страницы расписания на сайте Мосполитеха, тестирование Яндекс.Маркета;

– выполнение индивидуального задания по тестированию фронтенда;

– выполнение заданий по тестированию бэкенда.

Для выполнения задач курсового проекта применяется ряд технологий и инструментов, а именно язык программирования Java, технология для созданий проектов Maven, паттерн PageObject и инструмент Selenium WebDriver[1] при написании автотестов для фронтенда, библиотеки RestAssured и AssertJ при написании автотестов для бэкенда, Data-классы и Lombok для обработки JSON данных, тестовый фрэймворк JUnit4[2] и JSON-схемы для валидации JSON.

**Задание 1. Тестирование списка дел "LambdaTest Sample App"**

Для выполнения первого задания необходимо написать автотесты к тест-кейсу, представленному в таблице 1.

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://lambdatest.github.io/sample-todo-app/> | На странице присутствует заголовок “LambdaTest Sample App”. |
| 2. Проверить, что присутствует текст: “5 of 5 remaining” |  |
| 3. Проверить, что первый элемент списка не зачеркнут | Применён класс “done-false”, CSS-стили проверять не надо. |
| 4. Поставить галочку у первого элемента | Элемент списка становится зачеркнутым (применяется класс “done-true)”, отображаемое число оставшихся элементов уменьшается на 1. |
| 5. Повторить шаги 3, 4 для остальных элементов списка |  |
| 6. Добавить новый элемент списка | Новый элемент списка не зачеркнут, отображаемое общее число и число оставшихся элементов увеличиваются на 1. |
| 7. Нажать на новый элемент списка | Элемент списка становится зачеркнутым, отображаемое число оставшихся элементов уменьшается на 1. |

Таблица 1 – Тест-кейс для первого задания

Получение элементов происходит в классе Lambda.java с помощью аннотации «@FindBy» и «xpath» пути каждого элемента на странице. Все тесты оформлены в классе LambdaTest.java.

Выполнение шагов 1-4 выполняется путем сравнения предполагаемых и актуальных текстовых и атрибутных характеристик элементов с помощью метода «Assert.assertEquals()».

Выполнение шага 5 выполняется путем обрамления предыдущих шагов циклом следующим образом:

for(Map.Entry<String, WebElement> entry : lambdaPage.getElements().entrySet()){

if(!entry.getKey().equals("First Item")){

Assert.assertEquals(entry.getValue().findElement(By.xpath("span"))

.getAttribute("class"), "done-false");

}

}

Результат запущенных тестов представлен на рисунке 1.

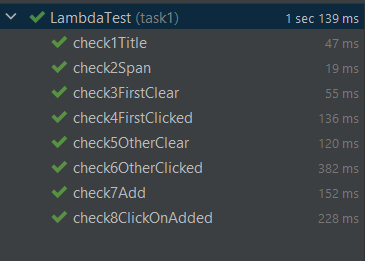


Рисунок 1 – Результат первого задания

**Задание 2. Тестирование страницы расписания на сайте Мосполитеха**

Для выполнения первого задания необходимо написать автотесты к тест-кейсу, представленному в таблице 2.

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://mospolytech.ru/> |  |
| 2. Нажать на кнопку Расписания |  |
| 3. В разделе “Расписания занятий” нажать “Смотрите на сайте” | Открывается страница поиска расписания в новой вкладке |
| 4. Ввести номер группы в поле поиска | В результатах поиска отображается только искомая группа |
| 5. Нажать на найденную группу в результатах поиска | Открывается страница расписания выбранной группы, текущий день недели выделен цветом (необходимо написать метод для определения текущего дня недели) |

Таблица 2 – Тест-кейс для второго задания

Получение элементов происходит в классе Mospolytech.java с помощью аннотации «@FindBy» и «xpath» пути каждого элемента на странице. Все тесты оформлены в классе MospolytechTest.java.

Шаги 1-2 выполняются путем вызова метода «click()» для соответствующих элементов, найденных при помощи «xpath».

Шаг 3 предполагает открытие странице в новой вкладке (рисунок 2), что добивается методом «switchTo(int page)», где «page» – номер вкладки в браузере :

public static String switchTo(int page){

Object[] windowHandles=driver.getWindowHandles().toArray();

driver.switchTo().window((String) windowHandles[page]);

return driver.getTitle();

}



Рисунок 2 – Открытие страницы в новой вкладке

Для выполнения шага 5 требуется написать метод для определения текущего дня недели. Метод «dayOfWeek()» из пакета Utils получает текущий день недели в английской локализации и с помощью словаря переводит его в русскую локализацию:

public static String dayOfWeek(){

TreeMap<String,String> engRus = new TreeMap<>();

engRus.put("Monday", "Понедельник");

engRus.put("Tuesday", "Вторник");

engRus.put("Wednesday", "Среда");

engRus.put("Thursday", "Четверг");

engRus.put("Friday", "Пятница");

engRus.put("Saturday", "Суббота");

engRus.put("Sunday", "Воскресенье");

Date d = new Date();

SimpleDateFormat dateFormat = new SimpleDateFormat("EEEE", Locale.US);

String str = dateFormat.format(d);

return engRus.get(str);

}

Результат запущенных тестов представлен на рисунке 3.

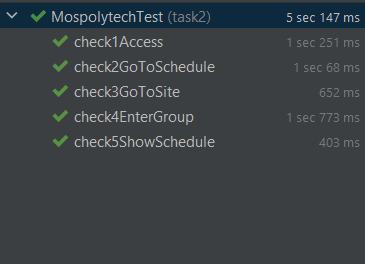


Рисунок 3 – Результат второго задания

**Задание 3. Тестирование Яндекс.Маркета (вариант 6)**

Для выполнения третьего задания необходимо написать автотесты к тест-кейсу варианта 6, представленному в таблице 3.

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://market.yandex.ru> | Открывается главная страница Яндекс.Маркета |
| 2. В меню “Каталог” выбрать категорию: Все для гейминга -> Xbox -> Игровые приставки | Открывается страница “Игровые приставки Xbox” |
| 3. Вывести в лог первые 5 найденных товаров (название и цену) |  |
| 4. Запомнить первую позицию из списка товаров (название и цену) |  |
| 5. У первого товара в списке нажать кнопку с белым сердечком (Добавить в избранное) | Кнопка “Добавить в избранное” меняется на “Удалить из избранного” (красное сердечко). Появляется всплывающее сообщение “Товар добавлен в избранное. Нажмите, чтобы перейти к списку.” |
| 6. На верхней панели нажать на кнопку “Избранное” | Открывается страница “Избранное”. На странице присутствует добавленный товар (название и цена соответствуют запомненным). |
| 7. Нажать на кнопку “Удалить из избранного” (красное сердечко) рядом с добавленным товаром. | Сверху появляется всплывающее сообщение:  “<Название товара> удалён из избранного  в списке 0 товаров”. |
| 8. Обновить страницу. | Товар не отображается на странице, отображается сообщение “Войдите в учётную запись”. |

Таблица 3 – Тест-кейс для третьего задания

Получение элементов происходит в классе Yandex.java с помощью аннотации «@FindBy» и «xpath» пути каждого элемента на странице. Все тесты оформлены в классе YandexTest.java.

Переход в категорию (шаг 2) осуществляется преждевременным наведением мыши на родительскую категорию, что было реализовано с помощью класса «Actions» из библиотеки «Seelnium»:

Actions action = Driver.createAction();

action.moveToElement(yandexPage.getSubCategory()).perform();

action.moveToElement(yandexPage.getCategory()).click().perform();

Для логирования первых 5 товаров (шаг 3, рисунок 4) в цикле инициировались переменные «title» и «price», содержащие соответственно название и цену товара. Поскольку рекламные блоки имеют в html-коде такой же класс, как и карточки товаров, используется блок «try-catch», обрабатывающий ошибку отсутствия элемента в древе «NoSuchElementException»:

try {

String title = product.findElement(By.xpath(".//\*[@data-auto=\"snippet-title\"]")).getText();

String price = product.findElement(By.xpath(".//\*[@data-auto=\"snippet-price-current\"]")).findElement(By.xpath("span")).getText();

price = price.replaceAll("\\D+", "");

System.out.println(title + " " + price);

}catch (NoSuchElementException e){

continue;

}

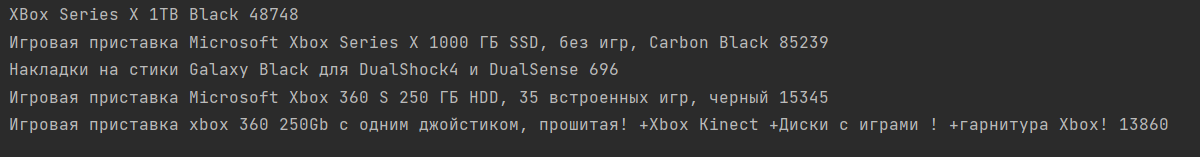


Рисунок 4 – Логирование первых 5 товаров

Аналогичная проблема решается циклом в шаге 4. Логирование сохраненного товара представлено на рисунке 5.

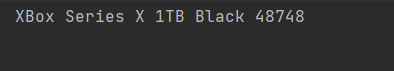


Рисунок 5 – Логирование первого товара

Чтобы сохранить данные о товаре, создан словарь TreeMap:

private final TreeMap<String, String> productMap = new TreeMap<>();

Шаги 6-7, кроме метода «Assert.assertEquals()», содержат метод «Assert.assertTrue», чтобы проверить наличие товара на странице «Избранное».

Результат запущенных тестов представлен на рисунке 4.

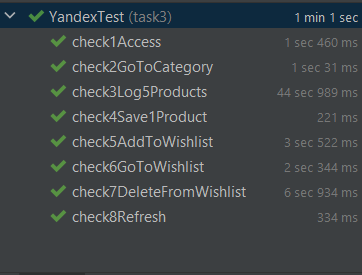


Рисунок 4 – Результат третьего задания

**Задание 4. Тестирование сайта LeetCode**

Для выполнения индивидуального задания был выбран сайт LeetCode (ссылка: <https://leetcode.com>).

Получение элементов происходит в классе LeetCode.java с помощью аннотации «@FindBy» и «xpath» пути каждого элемента на странице. Все тесты оформлены в классе LeetCodeTest.java.

Первый тест-кейс представлен в таблице 4.

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://leetcode.com> | Открывается главная страница LeetCode |
| 2. На странице найти блок с вопросами и перейти на страницу проблем | Открывается страница «Problems», соответствующая кнопка в навигационном меню выделяется цветом |
| 3. В поиске ввести «1. Two Sum» и нажать на первую проблему | Первая проблема в списке имеет название «1. Two Sum», при нажатии открывается страница проблемы |
| 4. Нажать на логотип для возвращения на главную страницу | Открывается главная страница |

Таблица 4 – Первый тест-кейс четвертого задания

Для проверки выделения цветом кнопки в навигационном меню с помощью метода «Assert.assertEquals()» проверяется атрибут «класс» кнопки с необходимым CSS-селектором.

Для ввода текста в поиск используется метод «sendKeys(param)», где param – строка, которую необходимо ввести в поиск:

leetcodePage.getInputProblems().sendKeys("1. Two Sum");

Второй тест-кейс представлен в таблице 5.

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://leetcode.com> | Открывается главная страница LeetCode |
| 2. На странице найти блок с названием «Start Exploring» и перейти на страницу исследования | Открывается страница «Explore», соответствующая кнопка в навигационном меню выделяется цветом |
| 3. В навигационном меню выбрать вкладку «Discuss» | Открывается страница «Discuss», соответствующая кнопка в навигационном меню выделяется цветом |
| 4. Ввести в поиске «Selenium» |  |
| 5. Проверить, что количество отображаемых статей меньше 10 | Количество отображаемых статей меньше 10 |
| 6. Вывести в лог первые 5 найденных статей (название и количество просмотров) |  |
| 7. Нажать на логотип для возвращения на главную страницу | Открывается главная страница |

Таблица 5 – Второй тест-кейс четвертого задания

Для проверки количества статей используется метод «Assert.assertTrue()» с необходимым условием. Чтобы посчитать количество статей, применяется метод «size()» для массива всех найденных элементов:

Assert.assertTrue(leetcodePage.getListDiscuss().findElements(By.xpath("//div[@class=\"topic-item-wrap\_\_2FSZ\"]")).size() < 10);

Для логирования первых 5 статей в цикле инициировались переменные «title» и «views», содержащие соответственно название и количество статей:

for(WebElement topic : leetcodePage.getListDiscuss()

.findElements(By.xpath("//div[@class=\"topic-item-wrap\_\_2FSZ\"]"))){

String title = topic.

findElement(By.xpath(".//div[@class=\"topic-title\_\_3LYM\"]")).getText();

String views = topic.

findElements(By.xpath(".//div[@class=\"no\_\_1erK\"]")).get(1).getText();

System.out.println(title + " " + views + " views");

}

Третий тест-кейс представлен в таблице 6

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: <https://leetcode.com> | Открывается главная страница LeetCode |
| 2. На странице найти блок с вопросами и перейти на страницу проблем | Открывается страница «Problems», соответствующая кнопка в навигационном меню выделяется цветом |
| 3. Установить количество отображаемых проблем с 50 до 20 | Количество отображаемых проблем равно 20 |
| 4. Установить фильтр сложности проблем на «Легко» | На странице отображаются проблемы только с сложностью «Легко» (число равно 20) |
| 5. Нажать на логотип для возвращения на главную страницу | Открывается главная страница |

Выполнение шага 4 не соответствует ожидаемому результату, так как кроме отфильтрованных проблем с сложностью «Легко» отображается «сезонная» проблема, имеющая сложность «Тяжело» (рисунок 5).

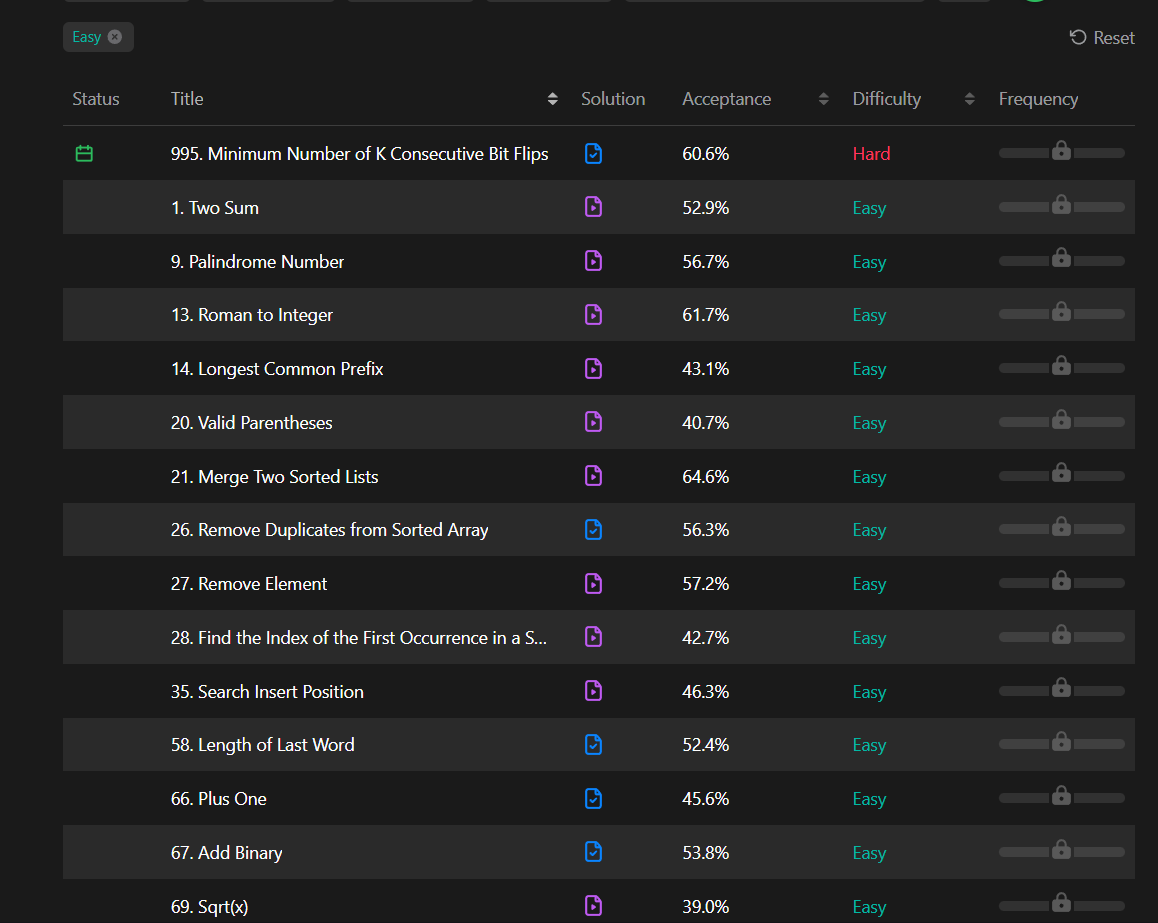


Рисунок 5 – Отфильтрованный список проблем на LeetCode

Так как тест показал отрицательный результат, был сделан скриншот с помощью класса «ScreenshotsOnFailure.java» из пакета utils. Сделанный скриншот был помещен в папку Screenshots.

Результат запущенных тестов для четвертого задания представлен на рисунке 6.

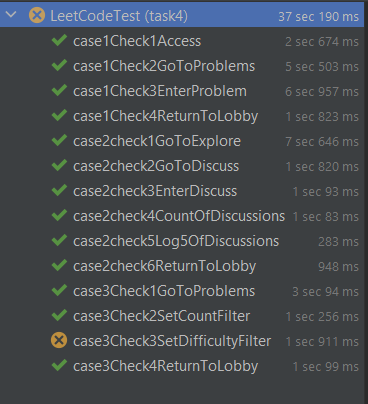


Рисунок 6 – Запущенные тесты четвертого задания

**Задание 5 – тестирование бэкенда**

Для выполнения задания 5 необходимо написать автотесты для открытого API (ссылка: <https://reqres.in/>), проверить ответы на соответствие JSON-схеме и убедиться, что при нажатии на кнопку на странице отправки образца запроса результат такой же, как и через API.

Для преобразования JSON-ответов в объекты, с которыми можно работать в Java, используются Data-классы из инструмента Lombok. Data-классы для списка пользователей, для отдельного пользователя, для создания пользователя, для списка ресурсов, для отдельного ресурса, для ошибки и для аутентификации пользователя реализованы в классах «UsersList.java», «SingleUser.java», «PostUser.java», «ResourceList.java», «ResourceSingle.java», «Error.java» и «AuthUser.java» соответственно. Тесты для пользователей, ресурсов и аутентификации представлены в «UsersTest.java», «ResourceTest.java» и «AuthTest.java» соответственно.

Для описания типичных методов отправки запросов создан класс «BaseTest.java», от которого наследуются остальные тестовые классы.

Сравнение данных происходит путем преобразования ответов в Data-классы и проверки их атрибутов методами «Assert.assertEquals()», «Assert.assertTrue()», «Assert.assertNull()» и «Assert.assertNotNull()».

Тест-кейсы API:

1. Тест-кейс 1

* Отправить GET-запрос на адрес https://reqres.in/api/users?page=2, проверить, что код ответа равен 200.
* В ответе проверить значение полей “page”, “page\_count”, “total”, “total\_pages”
* Проверить, что внутри массива “data” значения полей “id” не равны null
* Проверить, что в массиве есть элементы со следующими значениями полей: first\_name”=”Tobias”, “last\_name”=”Funke”.

1. Тест-кейс 2

* Отправить GET-запрос на адрес https://reqres.in/api/users/2, проверить, что код ответа равен 200.
* Проверить, что внутри поля “data” значения полей равны следующим:

"id": 2,

"email": "janet.weaver@reqres.in",

"first\_name": "Janet",

"last\_name": "Weaver",

"avatar": "https://reqres.in/img/faces/2-image.jpg"

1. Тест-кейс 3

* Отправить GET-запрос на адрес https://reqres.in/api/users/22, проверить, что код ответа 404.
* Проверить, что тело ответа пустое (содержит только пустые фигурные скобки {}).

1. Тест-кейс 4

* Отправить POST-запрос на адрес https://reqres.in/api/users, заполнив тело запроса следующим образом:

{

"name": "morpheus",

"job": "leader"

}

* Проверить, что код ответа равен 201
* Проверить, что в теле ответа значения полей “name” и “job” равны аналогичным полям из тела запроса.

1. Тест-кейс 5

* Отправить PUT-запрос на адрес https://reqres.in/api/users/2, заполнив тело запроса следующим образом:

{

"name": "morpheus",

"job": "zion resident"

}

* Проверить, что код ответа равен 200
* Проверить, что в теле ответа значения полей “name” и “job” равны аналогичным полям из тела запроса.

1. Тест-кейс 6

* Отправить PATCH-запрос на адрес https://reqres.in/api/users/2, заполнив тело запроса следующим образом:

{

"name": "morpheus",

"job": "zion resident"

}

* Проверить, что код ответа равен 200
* Проверить, что в теле ответа значения полей “name” и “job” равны аналогичным полям из тела запроса.

1. Тест-кейс 7

* Отправить DELETE-запрос на адрес https://reqres.in/api/users/2, проверить, что код ответа 204.
* Проверить, что тело ответа пустое (содержит только пустые фигурные скобки {}).

1. Тест-кейс 8

* Отправить GET-запрос на адрес https://reqres.in/api/unknown, проверить, что код ответа равен 200.
* В ответе проверить значение полей “page”, “page\_count”, “total”, “total\_pages”
* Проверить, что внутри массива “data” есть объекты.

1. Тест-кейс 9

* Отправить GET-запрос на адрес https://reqres.in/api/unknown/2, проверить, что код ответа равен 200.
* Проверить, что внутри поля “data” значения полей равны следующим:

{

"id": 2,

"name": " fuchsia rose”,

"year": 2001,

"color": " #C74375",

"pantone\_value": "17-2031"

}

1. Тест-кейс 10

* Отправить GET-запрос на адрес https://reqres.in/api/unknown/23, проверить, что код ответа 404.
* Проверить, что тело ответа пустое (содержит только пустые фигурные скобки {}).

1. Тест-кейс 11

* Отправить POST-запрос на адрес https://reqres.in/api/register, заполнив тело запроса следующим образом:

{

"email": "[eve.holt@reqres.in](mailto:eve.holt@reqres.in)",

"password": "pistol"

}

* Проверить, что код ответа равен 200
* Проверить, что в теле ответа значения полей “email” и “password” равны аналогичным полям из тела запроса.

1. Тест-кейс 12

* Отправить POST-запрос на адрес https://reqres.in/api/register, заполнив тело запроса следующим образом:

{

"email": "eve.holt@reqres.in"

}

* Проверить, что код ответа равен 400
* Проверить, что в теле ответа значения полей “error” равно "Missing password"

1. Тест-кейс 13

* Отправить POST-запрос на адрес https://reqres.in/api/login, заполнив тело запроса следующим образом:

{

"email": "eve.holt@reqres.in",

"password": "cityslicka"

}

* Проверить, что код ответа равен 200
* Проверить, что в теле ответа значения полей “email” и “password” равны аналогичным полям из тела запроса.4

1. Тест-кейс 14

* Отправить POST-запрос на адрес https://reqres.in/api/login, заполнив тело запроса следующим образом:

{

"email": "peter@klaven"

}

* Проверить, что код ответа равен 400
* Проверить, что в теле ответа значения полей “error” равно "Missing password"

Результаты запущенных тестов для пятого задания представлены на рисунках 7, 8 , 9.

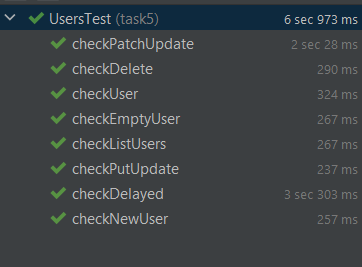


Рисунок 7 – Запущенные тесты для пользователей пятого задания

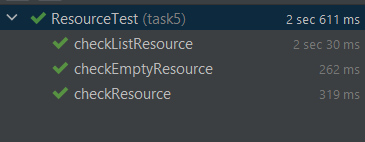


Рисунок 8 – Запущенные тесты для ресурсов пятого задания

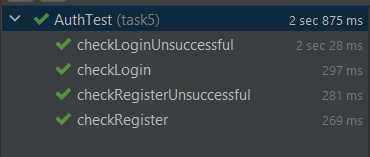


Рисунок 9 – Запущенные тесты для аутентификации пятого задания

Валидация JSON-схем осуществляется методом «JsonSchemaValidator.mathcesJsonSchemaInClasspath()»:

UserData newUser = checkStatusCodeGet("/register", user, 200)

.assertThat() .body(JsonSchemaValidator.matchesJsonSchemaInClasspath("resources/AuthSchema.json"))

Все JSON-схемы расположены в пакете resources. Пример JSON-схемы:

{

"$schema": "http://json-schema.org/draft-04/schema#",

"type": "object",

"properties": {

"id": {

"type": "integer"

},

"token": {

"type": "string"

}

},

"required": ["token"]

}

Сравнение веб-страницы и API реализована с помощью Selenium WebDriver и RestAssured одновремнно. Получение элементов происходит в классе WepPage.java с помощью аннотации «@FindBy» и «xpath» пути каждого элемента на странице. Все тесты оформлены в классе WebTest.java.

Тесты, связанные с изменением пользователя, а именно «checkNewUser», «checkPatchUpdate» и «checkPutUpdate» возвращают отрицательные результаты, так как время обновления пользователя через веб-страницу и API не совпадает, для каждого из этих тестов создается скриншот в папку screenshots. (рисунок 10)



Рисунок 10 – Отрицательный результат для теста checkPatchUpdate в связи с разницей во времени

Результат запущенных тестов для сравнения веб-страницы и API пятого задания представлен на рисунке 11.

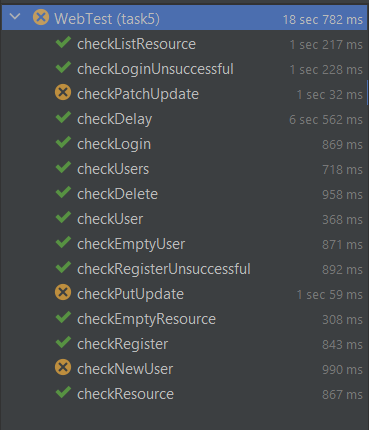


Рисунок 11 – Результат тестов сравнения веб-страницы и API пятого задания

**Теория тестирования ПО**

Проведение тестирования ПО гарантирует, что программное обеспечение работает корректно и соответствует заявленным требованиям, выявляет ошибки, дефекты и несоответствия в работе программы, улучшает качество программного продукта., обеспечивает безопасность и надежность работы ПО и повышает удовлетворенность пользователей.

Существует несколько видов тестирования, среди них выделяют следующие: модульное тестирование, интеграционное тестирование, системное тестирование., приемочное тестирование, нагрузочное тестирование, функциональное тестирование и нефункциональное тестирование. По степени автоматизации выделяют ручное и автоматизированные тестирование.

Автоматизированное тестирование имеет ряд преимуществ: ускорение процесса тестирования, повышение эффективности и точности тестирования, возможность повторного использования тестовых сценариев, автоматизация монотонных и рутинных задач, снижение вероятности человеческих ошибок и улучшение отчетности и трассируемости результатов тестирования [3].

Автоматизированные тестирование может проводиться разными способами и методами и содержать следующие стадии[4]:

* использование специализированных инструментов для создания и запуска автоматизированных тестов (например, Selenium, JUnit, Appium);
* написание скриптов и тестовых сценариев на языках программирования (например, Java, Python, JavaScript);
* интеграция автоматизированных тестов в процесс непрерывной интеграции и поставки (CI/CD);
* организация тестовых наборов, группировка и запуск тестов в пакетном режиме;
* мониторинг и анализ результатов автоматизированных тестов для выявления проблем и улучшения качества ПО.

**Заключение**

В ходе курсовой работы были изучены материалы лекционных и практических занятий из курса LMS «Введение в автоматизированное тестирование», был проведен анализ методов и средств автоматизированного тестирования программного обеспечения в ходе выполнения заданий по автоматизированном тестировании фронтенда и бэкенда.

Были выявлены основные цели проведения тестирования ПО, а также перечислены существующие виды тестирования. Выполнение задач соответствует методическим рекомендациям.

**Использованная литература**

1. Boni Garcia Hands-On Selenium WebDriver with Java // O'Reilly Media, Inc. 2022 (11), ISBN 109810997X, 9781098109974
2. Tudose, Catalin. JUnit in Action. Соединенные Штаты Америки, Manning, 2020.
3. Гребенюк Виктор Михайлович Oценка целесообразности внедрения автоматизированного тестирования // Вестник евразийской науки. 2013. №1 (14). URL: https://cyberleninka.ru/article/n/otsenka-tselesoobraznosti-vnedreniya-avtomatizirovannogo-testirovaniya (дата обращения: 24.06.2024).
4. Хыдыров Недир Какамырадович, Чарыев Шагулы Мурадниязович СПЕЦИФИКА СРЕДСТВ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ // Science Time. 2016. №4 (28). URL: https://cyberleninka.ru/article/n/spetsifika-sredstv-avtomatizirovannogo-testirovaniya (дата обращения: 24.06.2024).