Программное решение Spotify_website_UI_testing.
Техническая документация

1. Общие сведения

Программное решение "Spotify_website_UI_testing", далее – "программное решение", "ПР", разработано в учебных целях и не предназначено для какого-либо коммерческого использования.

Настоящее ПР предназначено для частичной автоматизации тестирования пользовательского интерфейса (UI), фронтенд части веб-сайта стримингового музыкального сервиса "Spotify" по заранее заданным сценариям поведения.

Программное решение выполнено в интегрированной среде разработки IntelliJ IDEA на языке программирования Java с использованием библиотеки автоматизированного управления веб-браузерами "Selenium WebDriver" и фреймворка для автоматизации тестирования программного обеспечения "TestNG"; сборщиком проекта выступает Maven. В качестве рабочего веб-браузера используется "Chrome" и утилита доступа и взаимодействия с ним "ChromeDriver".

2. Структура программного решения

"Spotify_website_UI_testing" имеет иерархическую структуру в которой модули ПР разделены на общую "main" и тестовую "test' части.

В общем "main" модуле расположена вся логика работы программного решения: классы, описывающие веб-элементы графического интерфейса, представленных в DOM принимаемых веб-страниц с которыми взаимодействует ПР, а также классы и методы для обработки и действий над поступающей информацией.

В тестовом "test" модуле расположены классы, описывающие тестовые сценарии по которым происходит автоматизированная работа веб-браузера. Исходные данные, необходимые для доступа в аккаунт, пути расположения утилиты доступа к веб-драйверу и прочая информация необходимая для тестирования расположена так же в этом модуле.

Отдельно, в корневом каталоге ПР, распложены xml-файлы, описывающие тестовые последовательности для TestNg и внешние зависимости для конфигурирования и сборки pom проекта для Maven.

Структура каталогов программного решения представлена ниже:

```
Spotify_website_UI_testing
    src
        main
            java
                pageObjects
                utils
        test
            java
                negativeScenarios
                parentTestClass
                smokeTests
            resources
```

3. Описание тестовых классов и последовательности работы тестовых сценариев программного решения

В настоящее программное решение включено 4 тестовых сценария, распределённых на две категории в зависимости от производимого вида тестирования.

В каталоге "smokeTests" расположено 3 тестовых сценария для быстрого смоук-тестирования некоторых основных функций веб-сайта, описанные в тестовых классах:

- BaseTestScenario
- GettingSongsFromPlaylist
- ProfilePlaylistsTests

В каталоге "negativeScenarios" расположен один тестовый сценарий для негативного тестирования формы доступа к аккаунту сервиса Spotify, описанный в классе:

- WrongLoginTestCases

Все тестовые сценарии являются логически взаимосвязанными наборами законченных тест-кейсов и могут запускаться как по отдельности из своих тестовых классов, так и в составе – с учётом последовательностей выполнения, описанных в xml-файлах тестового фреймворка TestNG.

Применяемые в данном программном решении аннотации @BeforeSuite, @BeforeTest, @BeforeClass, @Test, @AfterClass, @AfterTest, @AfterSuite, @DataProvider являются аннотациями фреймворка TestNG если не указано иное.

3.1 Базовый тестовый класс TestBasics

Все вышеперечисленные тестовые сценарии описаны в одноимённых .java классах и являются производными классами от базового класса TestBasics.java, находящегося в каталоге "parentTestClass".

Базовый класс TestBasics.java в своём составе имеет поля и методы, общие для производных от него классов тестовых сценариев и содержит:

- статическое поле *driver* типа WebDriver, предназначенное для передачи вебдрайвера в используемые экземпляры тестовых классов;
- статические поля *BROWSER_TYPE, DRIVER_PATH, PAGE_URL, USER_NAME, PASSWORD* типа String предназначены для задания значений типа веб-браузера, пути расположения утилиты вебдрайвера, адреса стартовой страницы открываемого веб-сайта с которого начинается работа по тестированию, действующих и корректных имени пользователя и пароля для входа в аккаунт сервиса Spotify соответственно.

Метод *somePreparingActions()*, подписанный аннотацией @BeforeSuite, предназначен для вывода информации о старте работы тестового набора.

3.2 Описание тестовых сценариев для смоук-тестирования в каталоге "smokeTests"

3.2.1 BaseTestScenario

Тестовый сценарий BaseTestScenario описан в классе BaseTestScenario.java и представляет собой базовый сценарий поведения пользователя сервиса Spotify и включает в себя следующие шаги:

- открытие стартовой страницы веб-сайта сервиса Spotify;
- проверка наличия стандартных плейлистов, составленных сервисом Spotify (далее – СП) на стартовой странице;
- поиск одного из СП на странице поиска;
- просмотр содержимого найденного СП и выбор любой музыкальной композиции (песни) из списка для проигрывания;
- вход в аккаунт (учётную запись) пользователя сервиса Spotify после нажатия кнопки воспроизведения выбранной песни;
- проверка соответствия названия выбранной в СП песни и автоматически воспроизводимой после входа пользователя в систему сервиса Spotify;
- переход на страницу пользователя и проверка имени пользователя;
- выход из аккаунта и проверка того, что пользователь вышел из системы сервиса Spotify.

В составе класса есть поля типа String в которых заданы значения имени искомого стандартного плейлиста (поле *DESIRED_PLAYLIST*) и количество песен в искомом СП (поле *AMOUNT_OF_SONGS*). В данном тестовом сценарии происходит работа с плейлистом "Metal Essentials" количество песен в котором равно 100.

В класс включены поля, описывающие веб-элементы DOM-структуры соответствующих веб-страниц, принимаемых браузером во время работы тестового сценария:

startPage, spotifyPlayList, laftSideBar, searchPage, loginPage, playerPanel, topMenu, profilePage, cookiesMenu. Классы, описывающие данные поля находятся в каталоге "pageObjects" модуля "main". Подробное описание классов данных полей приведено далее в соответствующем разделе.

Метод *activateTestPages()*, подписанный аннотацией @BeforeTest, предназначен для активации всех веб-элементов тех веб-страниц, на которых проходит тестовый сценарий. Также данный метод выводит информацию о состоянии веб-драйвера перед началом тестирования.

Метод *enterToPage()*, подписанный аннотацией @BeforeClass, предназначен для открытия стартовой страницу веб-сайта и содержит следующую последовательность инструкций:

- *driver.get(PAGE_URL)* – переход на стартовую веб-страницу сервиса Spotify для начала тестирования. Адрес веб-страницы указан в передаваемом аргументе PAGE_URL. Переход осуществляется на веб-страницу с адресом ***https://open.spotify.com***;
- *cookiesMenu.closeCookies()* – закрытие всплывающего элемента, предупреждающего об использовании веб-сайтом cookie;
- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка открытой веб-страницы ожидаемому названию "Spotify – Web Player: Music for everyone".

Тестовый метод *getAmountOfStartPagePlaylists()*, подписанный аннотацией @Test, предназначен для проверки наличия стандартных плейлистов Spotify на стартовой странице веб-сайта. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта стартовая веб-страница сервиса Spotify.

- *boolean playlistPresence = false* – создание локальной логической переменной, указывающей на наличие раздела стандартных плейлистов Spotify на стартовой странице веб-сайта. Изначально задано отрицательное состояние;
- *playlistPresence = startPage.searchPlaylists()* – поиск СП на стартовой странице и передача истинности наличия/отсутствия раздела с СП переменной;
- *if(playlistPrecese == true)* – если наличие стандартных плейлистов Spotify на стартовой веб-странице подтверждено, то происходит выполнение следующих действий:
  - *startPage.showAllPlaylists()* – раскрытие всех СП, представленных на стартовой веб-странице;
  - *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;
  - *Assert.assertEquals(startPage.amountOfPlaylists(), expectedResult)* – проверка соответствия количества стандартных плейлистов Spotify ожидаемому количеству 10;
  - *startPage.clickOnPlaylist()* – клик по веб-элементу карты желаемого СП и переход на веб-страницу с этим плейлистом;
  - *spotifyPlaylist.getPlaylistName()* – получение названия выбранного стандартного плейлиста, дальнейший вывод на консоль названия;
  - *spotifyPlaylist.getNumberOfSongs()* – получение количестве песен в выбранном стандартного плейлисте, дальнейший вывод на консоль количества;
  - *Assert.assertTrue(driver.getCurrentUrl().contains("адрес_страницы"))* – проверка нахождения выбранного СП по стандартному адресу для плейлистов (https://open.spotify.com/playlist/).
- else – в противном случае вывод на консоль надписи о том, что стандартных плейлистов Spotify на стартовой странице не обнаружено.

Тестовый метод *playlistVerifying()*, подписанный аннотацией @Test, предназначен для проверки работы поиска стандартного плейлиста Spotify и просмотра найденного СП. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта стартовая веб-страница сервиса Spotify.

- *leftSidebar.goToSearchPage()* – клик по элементу поиска, находящемуся на левой боковой панели и переход на веб-страницу поиска;
- *Assert.assertEquals(driver.getCurrentUrl(), expectedResult)* – проверка соответствия адреса текущей открытой веб-страницы ожидаемому адресу "https://open.spotify.com/search";
- *searchPage.searchPlaylist(DESIRED_PLAYLIST)* – ввод в строку поиска названия стандартного плейлиста, указанного в передаваемом аргументе *DESIRED_PLAYLIST,* и поиск желаемого СП;
- serachPage.enterToPlaylist(DESIRED_PLAYLIST) – поиск наиболее подходящего под запрос результата из найденных и переход на веб-страницу желаемого СП.
- *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;
- String name = *spotifyPlaylist.getPlaylistName()* – создание локальной переменной name и присвоение ей названия текущего плейлиста;
- String amount = *spotifyPlaylist.getNumberOfSongs()* – создание локальной переменной amount и присвоение ей количества песен текущего плейлиста;
- Вывод на консоль названия текущего плейлиста и количества песен, содержащихся в нём;
- *Assert.assertEquals(amount, AMOUNT_OF_SONGS)* – проверка фактического количества песен на соответствие ожидаемому для данного стандартного листа Spotify;
- *Assert.assertEquals(name, DESIRED_PLAYLIST)* – проверка фактического названия плейлиста на соответствие ожидаемому для данного стандартного листа Spotify;
- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка текущей веб-страницы на которой расположен желаемый плейлист стандартному шаблону заголовка для СП который состоит:

  название_плейлиста | Spotify Playlist.

Тестовый метод *pressRandomSongToPlay(),* подписанный аннотацией @Test, предназначен для проверки работы перехода на страницу входа в аккаунт пользователя через нажатие кнопки воспроизведения песни из текущего стандартного плейлиста Spotify. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта страница стандартного плейлиста Spotify.

- *spotifyPlaylist.ascendToHeader()* – поднятие в верх страницы;
- spotifyPlaylist.clickToPlayRandomSong() – случайный выбор песни из СП и нажатие на кнопку воспроизведения песни. Во всплывшем попап-окне нажать кнопку перехода на страницу входа в аккаунт;
- *Thread.sleep(1000)* – задержка совершения дальнейших действий длительностью в 1 секунду;
- *Assert.assertTrue(driver.getCurrentURL().contains("адрес_страницы_входа"))* – проверка перехода на веб-страницу входа в аккаунт путём выявления адреса открытой страницы (https://accounts.spotify.com/).

Тестовый метод *loginToAccount(),* подписанный аннотацией @Test, предназначен для входа в учётную запись пользователя и проверки того, что вход был совершён в тот стандартный плейлист Spotify который был определён и открыт ранее. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: совершён переход на страницу входа в аккаунт из СП.

- *loginPage.setLogin(LOGIN_NAME)* – в поле ввода имени пользователя вписать адрес электронной почты от зарегистрированного в системе сервиса Spotify аккаунта;

- *loginPage.setPassword(PASSWORD)* – в поле ввода пароля вписать корректный пароль от зарегистрированного в системе сервиса Spotify аккаунта;

- *loginPage.setCheckboxOff()* – чекбокс поля "Запомнить меня" перевести в выключенное положение;

- *loginPage.loginEnter()* – нажать на кнопку входа в аккаунт;

- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;

- *Assert.assertTrue(driver.getCurrentURL().contains("адрес_плейлиста"))* – проверка перехода на веб-страницу, содержащую выбранный стандартный плейлист Spotify путём выявления соответствия адресу открытой веб-страницы для СП (https://open.spotify.com/playlist);

- *Assert.assertEquals(spotityPlaylist.getPlaylistName(), DESIRED_PLAYLIST)* – проверка соответствия названия плейлиста, открытого после входа в аккаунт, искомому.

Тестовый метод *checkTheSong()*, подписанный аннотацией @Test, предназначен для проверки соответствия названия воспроизводимой песни после входа в учётную запись той, которая была выбрана в СП перед входом в аккаунт. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: вход в учётную запись должен быть осуществлён после выбора любой музыкальной композиции в найденном стандартном плейлисте Spotify.

- *playerPanel.getCurrentSongPlayer()* – вывод на консоль информации о воспроизводимой песне;

- *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;

- *playerPanel.pauseCurrentSong()* – постановка на паузу воспроизводимой песни;

- *Assert.assertEquals(playerPanel.getCurrentSongPlayer(), expectedResult)* – проверка соответствия названия воспроизводимой песни, указанной в поле плеера, названию заранее выбранной музыкальной композиции из СП перед входом в аккаунт. Ожидаемое значение expectedResult принимается при помощи инструкции *spotifyPlaylist.getSongForPlay()*.

Тестовый метод *profileNameVerifying()*, подписанный аннотацией @Test, предназначен для проверки соответствия имени пользователя указанного в верхнем горизонтальном меню тому, которое указано на странице пользователя. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: должен быть совершён вход в аккаунт.

- *topMenu.enterToProfilePage()* – переход на страницу пользователя;

- *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;

- *driver.navigate().refresh()* – обновление открытой веб-страницы;

- *Assert.assertTrue(driver.getCurrentUrl().contains("адрес_домена/user/"))* – проверка того, что в текущий момент открыта страница пользователя;

- *Assert.assertEquals(driver.getTitle(), "Spotify – имя_пользователя")* – проверка того, что заголовок веб-страницы пользователя содержит имя пользователя. Имя пользователя принимается путём вызова инструкции *profilePage.getProfileName()*;

- Вывод имени пользователя на консоль путём вызова инструкции *profilePage.getProfileName()*;

- *Assert.assertEquals(profilePage.getProfileName(), expectedResult)* – проверка соответствия имени пользователя, указанного на странице пользователя тому ко-

торое указано в значке верхнего горизонтального меню. *expectedResult* принимается путём вызова инструкции *topMenu.getUserName()*.

Метод *afterTestCasesActions()*, подписанный аннотацией @AfterClass, предназначен для выхода из учётной записи сервиса Spotify и проверки, что выход из аккаунта выполнен; содержит следующую последовательность инструкций:

- *topMenu.logOutfromAccount()* – выход из аккаунта путём нажатия кнопки "Выйти" из всплывающего подменю верхнего горизонтального меню пользователя;
- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;
- *Assert.assertEquals(driver.getCurrentUrl(), "https://open.spotify.com")* – проверка того, что после выхода из учётной записи пользователя был осуществлён переход на стартовую страницу веб-сайта;
- *Assert.assertTrue(playerPannel.signUpButtonPresented())* – проверка того, что после выхода из аккаунта на веб-странице присутствует блок с кнопкой "Зарегистрироваться";
- Вывод на консоль сообщения о том, что все тест-кейсы в данном тестовом классе завершены.

Метод *quitWebDriver()*, подписанный аннотацией @AfterTest, предназначен для выгрузки экземпляра утилиты вебдрайвера. После данного метода происходит завершение работы тестового класса BaseTestScenario.

3.2.2 GettingSongsFromPlaylist

Тестовый сценарий GettingSongsFromPlaylist описан в тестовом классе GettingSongsFromPlaylist.java и представляет собой сценарий поведения пользователя, ищущего определённый стандартный плейлист Spotify и ознакамливающегося с полным перечнем содержащихся в нём музыкальных композиций; включает в себя следующие шаги:

- открытие стартовой страницы веб-сайта сервиса Spotify;
- поиск одного из СП на странице поиска;
- детальный просмотр содержимого открытого СП.

В составе класса есть поля типа String в которых заданы значения названия искомого стандартного плейлиста (поле *DESIRED_PLAYLIST*) и количество песен в искомом СП (поле *AMOUNT_OF_SONGS*). В данном тестовом сценарии происходит работа с плейлистом "Rock Classics" количество песен в котором равно 200.

В класс включены поля, описывающие веб-элементы DOM-структуры соответствующих веб-страниц, принимаемых браузером во время работы тестового сценария: *spotifyPlayList, laftSideBar, searchPage, cookiesMenu*. Классы, описывающие данные поля находятся в каталоге "pageObjects" модуля "main". Подробное описание классов данных полей приведено далее в соответствующем разделе.

Метод *activateTestPages()*, подписанный аннотацией @BeforeTest, предназначен для активации всех веб-элементов веб-страниц, участвующих в тестовом сценарии. Также данный метод выводит информацию о состоянии веб-драйвера перед началом тестирования.

Метод *enterToPage()*, подписанный аннотацией @BeforeClass, предназначен для открытия стартовой страницы веб-сайта и содержит следующую последовательность инструкций:

- *driver.get(PAGE_URL)* – переход на стартовую веб-страницу сервиса Spotify для начала тестирования. Адрес веб-страницы указан в передаваемом аргументе PAGE_URL. Переход осуществляется на веб-страницу с адресом ***https://open.spotify.com***;

- *cookiesMenu.closeCookies()* – закрытие всплывающего элемента, предупреждающего об использовании веб-сайтом cookie;

- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка открытой веб-страницы ожидаемому названию "Spotify – Web Player: Music for everyone".

Тестовый метод *playlistVerifying()*, подписанный аннотацией @Test, предназначен для поиска желаемого стандартного плейлиста Spotify и перехода на его веб-страницу проверки. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта стартовая веб-страница сервиса Spotify.

- *leftSidebar.goToSearchPage()* – клик по элементу поиска, находящемуся на левой боковой панели и переход на веб-страницу поиска;

- *Assert.assertEquals(driver.getCurrentUrl(), expectedResult)* – проверка соответствия адреса текущей открытой веб-страницы ожидаемому адресу "https://open.spotify.com/search";

- *searchPage.searchPlaylist(DESIRED_PLAYLIST)* – ввод в строку поиска названия стандартного плейлиста, указанного в передаваемом аргументе *DESIRED_PLAYLIST,* и поиск желаемого СП;

- serachPage.enterToPlaylist(DESIRED_PLAYLIST) – поиск наиболее подходящего под запрос результата из найденных и переход на веб-страницу желаемого СП.

- *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;

- String name = *spotifyPlaylist.getPlaylistName()* – создание локальной переменной name и присвоение ей названия текущего плейлиста;

- String amount = *spotifyPlaylist.getNumberOfSongs()* – создание локальной переменной amount и присвоение ей количества песен текущего плейлиста;

- Вывод на консоль названия найденного желаемого плейлиста и количества песен, содержащихся в нём;

- *Assert.assertEquals(amount, AMOUNT_OF_SONGS)* – проверка фактического количества песен на соответствие ожидаемому для данного стандартного листа Spotify;

- *Assert.assertEquals(name, DESIRED_PLAYLIST)* – проверка фактического названия плейлиста на соответствие ожидаемому для данного стандартного листа Spotify;

- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка текущей веб-страницы на которой расположен желаемый плейлист стандартному шаблону заголовка для СП который состоит:

название_плейлиста | Spotify Playlist.

Тестовый метод *gettingAllSongs(),* подписанный аннотацией @Test, предназначен для детального ознакомления с выбранным плейлистом и получения списка всех песен, содержащихся в нём. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта страница стандартного плейлиста Spotify.

- *spotifyPlaylist.getFullListOfSongs()* – потрековая прокрутка плейлиста и получение названий всех музыкальных композиций. Вывод списка всех песен СП на консоль.

Метод *afterTestCasesActions()*, подписанный аннотацией @AfterClass, предназначен для оповещения о том, что все тест-кейсы в данном тестовом классе завершены; содержит следующую инструкцию:

- Вывод на консоль сообщения о том, что все тест-кейсы в данном тестовом классе завершены.

Метод *quitWebDriver()*, подписанный аннотацией @AfterTest, предназначен для выгрузки экземпляра утилиты вебдрайвера. После данного метода происходит завершение работы тестового класса BaseTestScenario.

3.2.3 ProfilePlaylistsTests

Тестовый сценарий ProfilePlaylistsTests описан в тестовом классе ProfilePlaylistsTests.java и представляет собой сценарий поведения пользователя, вошедшего в сервис и создавшего новый плейлист, сохраняемый в своей учётной записи; включает в себя следующие шаги:

- открытие стартовой страницы веб-сайта сервиса Spotify;
- вход в учётную запись пользователя;
- подсчёт количества имеющихся у пользователя плейлистов;
- создание нового плейлиста;
- подсчёт нового количества плейлистов пользователя;
- переименование созданного плейлиста;
- поиск и добавление музыкальных композиций по имени исполнителя в созданный плейлист;
- включение воспроизведения случайной песни в созданном плейлисте;
- проверка соответствия названия воспроизводимой песни.
- выход из аккаунта и проверка того, что пользователь вышел из системы сервиса Spotify.

В класс включены поля, описывающие веб-элементы DOM-структуры соответствующих веб-страниц, принимаемых браузером во время работы тестового сценария: *loginPage, topMenu*, *cookiesMenu*, *playerPanel, leftSidebar, playlistPage*. Классы, описывающие данные поля находятся в каталоге "pageObjects" модуля "main". Подробное описание классов данных полей приведено далее в соответствующем разделе.

Метод *activateTestPages()*, подписанный аннотацией @BeforeTest, предназначен для активации всех веб-элементов веб-страниц, участвующих в тестовом сценарии. Также данный метод выводит информацию о состоянии веб-драйвера перед началом тестирования.

Метод *enterToPage()*, подписанный аннотацией @BeforeClass, предназначен для открытия стартовой страницы веб-сайта и содержит следующую последовательность инструкций:

- *driver.get(PAGE_URL)* – переход на стартовую веб-страницу сервиса Spotify для начала тестирования. Адрес веб-страницы указан в передаваемом аргументе PAGE_URL. Переход осуществляется на страницу с адресом ***https://open.spotify.com***;
- *cookiesMenu.closeCookies()* – закрытие всплывающего элемента, предупреждающего об использовании веб-сайтом cookie;
- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка открытой веб-страницы ожидаемому названию "Spotify – Web Player: Music for everyone".

Тестовый метод *logInToAccount()*, подписанный аннотацией @Test, предназначен для входа в учётную запись пользователя сервиса и проверки того, что зарегистрированный пользователь вошёл в аккаунт. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта стартовая веб-страница сервиса Spotify.

- *topMenu.EnterToLoginPage()* – переход на страницу входа в аккаунт путём нажатия кнопки "Войти" в верхнем горизонтальном меню;
- *Thread.sleep(1000)* – задержка совершения дальнейших действий длительностью в 1 секунде;

- *Assert.assertTrue(driver.getCurrentURL().contains("адрес_домена/"))* – проверка перехода на веб-страницу входа в аккаунт путём проверки содержимого URL открытой веб-страницы на наличие адреса (адрес https://accounts.spotify.com/);

- *loginPage.setLogin(LOGIN_NAME)* – в поле ввода имени пользователя вписать адрес электронной почты от зарегистрированного в системе сервиса Spotify аккаунта;

- *loginPage.setPassword(PASSWORD)* – в поле ввода пароля вписать корректный пароль от зарегистрированного в системе сервиса Spotify аккаунта;

- *loginPage. setCheckboxOff ()* – чекбокс поля "Запомнить меня" перевести в выключенное положение;

- *loginPage.loginEnter()* – нажать на кнопку входа в аккаунт;

- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;

- *Assert.assertTrue(playerPanel.getPlayerBar().isDisplayed())* – проверка входа в учётную запись пользователя путём проверки наличия панели управления плеером в нижнем горизонтальном меню. Вывод состояния наличия панели управления плеером на консоль.

Тестовый метод *createNewPlaylist()*, подписанный аннотацией @Test, предназначен для создания нового плейлиста в учётной записи пользователя. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: должен быть осуществлён вход пользователя в аккаунт.

- *int amountOfPlaylists = leftSidebar.getAmountOfPlaylists()* – создание локальной переменной и сохранение в неё значения количества плейлистов пользователя до создания нового плейлиста. Вывод этого значения в консоль;

- *leftSidebar.createNewPlaylist()* – создание нового плейлиста путём нажатия кнопки "+" в левом сайдбаре с последующим нажатием кнопки "Создать плейлист" во всплывшем подменю;

- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;

- *Assert.assertEquals(leftsidebar.getAmountOfPlaylists(), amountOfPlaylists+1)* – проверка создания нового плейлиста путём сравнивания нынешнего количества плейлистов в списке аккаунта с предыдущим, которое должно быть на 1 меньше. Вывод результатов сравнения в консоль.

Тестовый метод *renameNewPlaylist()*, подписанный аннотацией @Test, предназначен для переименования созданного плейлиста в учётной записи пользователя. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: должен быть осуществлён вход в пользователя в аккаунт и создан новый пользовательский плейлист.

- *String oldTitle = playlistPage.getPlaylistTitle()* – создание локальной переменной и сохранение в неё строкового значения изначального названия созданного плейлиста (по умолчанию это "Мой плейлист №…");

- *String newTitle = "User_Playlist_2020-" + (int)(Math.random()*2024)* – создание локальной переменной и сохранение в неё строкового значения для нового названия созданного плейлиста, полученного путём сложения шаблона со случайно полученным значением;

- *playlistPage.renameOfPlaylist(newTitle)* – переименование созданного плейлиста;

- *Thread.sleep(2000)* – задержка совершения дальнейших действий длительностью в 2 секунды;

- *Assert.assertNotEquals(oldTitle, playlistPage(getPlaylistTitle))* – проверка переименования созданного плейлиста. Вывод прежнего и нынешнего названий созданного плейлиста в консоль;

- *leftSidebar.goToStartPage()* – переход на стартовую страницу.

Тестовый метод *addSongsToNewPlaylist()*, подписанный аннотацией @Test, предназначен для проверки добавления новых треков в созданный плейлист и возможности их воспроизведения. Тест-кейс состоит их следующих шагов, описанный в последовательности инструкций:

Предусловие: должен быть пустой плейлист в котором отсутствуют музыкальные композиции.

- *String bandName = "Some_music_band"* – создание локальной переменной и сохранение в ней имя исполнителя / название коллектива, песни которого будут искаться и добавляться в пустой плейлист;
- *leftSidebar.openNewestEmptyPlaylist()* – переход на веб-страницу созданного плейлиста путём нажатия в левом сайдбаре на кнопку с названием самого первого сверху плейлиста в перечне пользовательских плейлистов;
- *playlistPage.searchBand(bandName)* – после открытия созданного пустого плейлиста поиск содержимого об исполнителе путём написания названия исполнителя в поле поиска;
- *playlistPage.chooseFromSearchResult(bandName)* – выбор исполнителя в результатах поиска и переход к списку песен этого исполнителя путём нажатия на строку, содержащую в себе категорию "Исполнитель" ("Artist");
- *playlistPage.addSongsToPlaylist()* – добавление всех музыкальных композиций, представленных в перечне треков, в созданный плейлист путём нажатия кнопки "Добавить" напротив каждой песни из перечня;
- *Thread.sleep(1000)* – задержка совершения дальнейших действий длительностью в 1 секунду;
- *playlistPage.getSongsAmount()* – получение количества песен в текущем плейлисте и вывод этого значения в консоль;
- *playlistPage.PlaySongFromCurrentPlaylist()* – выбор любой песни из плейлиста и её воспроизведение. Вывод названия воспроизводимой песни в консоль;
- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;
- *Assert.assertEquals(playlistPage.getPlayingSongName(), playerPanel.getCurrentSongPlayer())* – проверка соответствия названия текущей воспроизводимой песни в панели плеера с ранее случайно выбранной песней на предыдущем шаге;
- *playlistPage.compareSongName(playerPanel.getCurrentSongPlayer()*) – вывод результатов проверки соответствия песен на консоль.

Метод *afterTestCasesActions()*, подписанный аннотацией @AfterClass, предназначен для выхода из учётной записи сервиса Spotify и проверки, что выход из аккаунта выполнен; содержит следующую последовательность инструкций:

- *topMenu.logOutfromAccount()* – выход из аккаунта путём нажатия кнопки "Выйти" из всплывающего подменю верхнего горизонтального меню пользователя;
- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;
- *Assert.assertEquals(driver.getCurrentUrl(), "https://open.spotify.com")* – проверка того, что после выхода из учётной записи пользователя был осуществлён переход на стартовую страницу веб-сайта;
- *Assert.assertTrue(playerPannel.signUpButtonPresented())* – проверка того, что после выхода из аккаунта на веб-странице присутствует блок с кнопкой "Зарегистрироваться";
- Вывод на консоль сообщения о том, что все тест-кейсы в данном тестовом классе завершены.

Метод *quitWebDriver()*, подписанный аннотацией @AfterTest, предназначен для выгрузки экземпляра утилиты вебдрайвера. После данного метода происходит завершение работы тестового класса ProfilePlaylistsTests.

3.3 Описание тестовых сценариев для негативного типа тестирования в каталоге "negativescenarios"

3.3.1 WrongLoginTestCases

Тестовый сценарий WrongLoginTestCases описан в тестовом классе WrongLoginTestCases.java и представляет собой сценарий поведения пользователя, пытающегося войти в систему, указывая различные комбинации некорректных данных для входа в аккаунт; состоит из следующих шагов:

- открытие стартовой страницы веб-сайта сервиса Spotify;
- переход на страницу входа в учётную запись пользователя;
- попытка входа в аккаунт при вводе некорректных данных для входа;
- обратный переход на стартовую веб-страницу сервиса Spotify.

В класс включены поля, описывающие веб-элементы DOM-структуры соответствующих веб-страниц, принимаемых браузером во время работы тестового сценария: *loginPage, topMenu*, *cookiesMenu*, *playerPanel*. Классы, описывающие данные поля находятся в каталоге "pageObjects" модуля "main". Подробное описание классов данных полей приведено далее в соответствующем разделе.

Метод *activateTestPages()*, подписанный аннотацией @BeforeTest, предназначен для активации всех элементов тех веб-страниц, участвующих в тестовом сценарии. Также данный метод выводит информацию о состоянии веб-драйвера перед началом тестирования.

Метод *enterToPage()*, подписанный аннотацией @BeforeClass, предназначен для открытия стартовой страницы веб-сайта и содержит следующую последовательность инструкций:

- *driver.get(PAGE_URL)* – переход на стартовую веб-страницу сервиса Spotify для начала тестирования. Адрес веб-страницы указан в передаваемом аргументе PAGE_URL. Переход осуществляется на веб-страницу с адресом ***https://open.spotify.com***;
- *cookiesMenu.closeCookies()* – закрытие всплывающего элемента, предупреждающего об использовании веб-сайтом cookie;
- *Assert.assertEquals(driver.getTitle(), expectedResult)* – проверка соответствия заголовка открытой веб-страницы ожидаемому названию "Spotify – Web Player: Music for everyone".

Тестовый метод *logInToAccount(String username, String password)*, подписанный аннотацией @Test, предназначен попытки входа в учётную запись пользователя сервиса, используя различные комбинации некорректных данных для входа. Тест-кейс состоит из следующих шагов, описанных в последовательности инструкций:

Предусловие: открыта стартовая веб-страница сервиса Spotify.

- *topMenu.enterToLoginPage()* – переход на веб-страницу входа в учётную запись путём нажатия кнопки "Войти" в верхнем горизонтальном меню;
- *loginPage.setLogin(userName)* – заполнить поле ввода имени пользователя вписать тестовыми данными;
- *loginPage.setPassword(password)* – заполнить поле ввода пароля тестовыми данными;
- *loginPage.setCheckboxOff()* – чекбокс поля "Запомнить меня" перевести в выключенное положение;
- *loginPage.loginEnter()* – нажать на кнопку входа в аккаунт;
- *Thread.sleep(1000)* – задержка совершения дальнейших действий длительностью в 1 секунду;
- Boolean warningBanner = loginPage.getWarningBanner().isDisplayed() – создание локальной переменной логического типа и присвоение ей значения, опове-

щающего о появлении/отсутствии веб-элемента на странице – предупреждающего баннера о вводе некорректных данных для входа в аккаунт;

- if(warningBanner == true)
    - System.out.println("Invalid credentials message") – при появлении на странице в форме ввода данных предупреждающего баннера, в консоль выводится надпись о появлении этого веб-элемента со значениями userName и password;
- Assert.assertTrue(warningBanner, "Warning banner is not displayed") – проверка появления предупреждающей надписи при вводе некорректных данных;
- driver.navigate.refresh() – обновление веб-страницы для повторного ввода некорректных данных другой конфигурации.

Метод setWrongCredentials(), подписанный аннотацией @DataProvider, предназначен для передачи данных в тест-кейс, описанный тестовом методе loginToAccount() и передаёт в него следующие конфигурации имени пользователя и пароля:

№ пп	userName	password
1	wrong@userName	wrongPassword
2	существующий пользователь	пустое поле
3	пустое поле	существующий пароль
4	существующий пользователь	существующий пароль + 1 символ в конце
5	существующий пользователь	1 символ в начале + существующий пароль

Метод setWrongCredentials() вызывается столько раз, сколько передано некорректных конфигураций данных для входа.

Метод *returnToStartPage()*, подписанный аннотацией @AfterClass, предназначен для проверки возможности возвращения на стартовую веб-страницу сервиса Spotify после вода некорректных данных для входа и отсутствии возможности войти в систему; содержит следующую последовательность инструкций:

- loginPage.pressSpotifyLogo() – переход на стартовую страницу сервиса путём нажатия на элемент значка логотип "Spotify";
- *Thread.sleep(3000)* – задержка совершения дальнейших действий длительностью в 3 секунды;
- *Assert.assertEquals(driver.getCurrentUrl(), "https://open.spotify.com")* – проверка того, что переход на стартовую страницу веб-сайта был осуществлён;
- *Assert.assertTrue(playerPannel.signUpButtonPresented())* – проверка того, что после выхода из аккаунта на веб-странице присутствует блок с кнопкой "Зарегистрироваться";
- Вывод на консоль сообщения о том, что все тест-кейсы в данном тестовом классе завершены.

Метод *quitWebDriver()*, подписанный аннотацией @AfterTest, предназначен для выгрузки экземпляра утилиты вебдрайвера. После данного метода происходит завершение работы тестового класса WrongLoginTestCases.

3.4 Описание файлов управления группами тестов типа testNG.xml

В настоящем программном решении разработано два файла управления тестами в которых описана последовательность запуска тестовых сценариев в соответствии с выполняемой функциональностью: работа с учётной записью и работа с плейлистами.

Файл loginTests.xml объединяет в себе запуск тестовых сценариев, описанных в тестовых классах BaseTestScenario и WrongLoginTestCases.

Файл playlistsTests.xml объединяет в себе запуск тестовых сценариев классов ProfilePlaylistsTests и GettingSongsFromPlaylist.

4. Модуль "main". Описание работы методов и алгоритмов, используемых для работы тестовых сценариев

Рабочий модуль "main" включает в себя .java классы, обеспечивающие логику программного решения и взаимосвязь выполнения тестов.

В состав модуля "main" входят каталоги `pageObjects` и `utils`.

В каталоге pageObjects находятся классы, содержащие локаторы для веб-элементов DOM, принимаемых браузером веб-страниц с которыми происходит взаимодействие во время автоматизированного тестирования веб-сайта и методы с алгоритмами для работы с этими веб-элементами.

Каталог utils содержит классы для установки вебдрайвера и обработки поступающих из внешних источников.

4.1 Каталог "pageObjects"

Каталог "pageObjects" содержит в себе файл класса PageObjectBasics и подкаталоги accountPages, mainPages, sidebarsAndMenus.

Для определения и инициализации веб-элементов подключен PageFactory, входящий в состав библиотеки Selenium. Все используемые в работе веб-элементы помечены аннотацией @FindBy. В большинстве случаев в качестве локаторов для веб-элементов выбрано определение элементов по Xpath как наиболее стабильный и неизменчивый со временем. В одном случае использован локатор, определяющий веб-элемент по CSS.

4.1.1 PageObjectBasics

Класс PageObjectBasics является суперклассом для всех файлов пакета pageObjects и предназначен для задания основных членов, общих для всех дочерних классов.

В своём составе суперкласс имеет поля типов WebDriver и Actions из библиотеки Selenium для задания которых составлен конструктор, принимающий на вход экземпляр вебдрайвера. Задание PageFactory для отложенной инициализации вебэлементов также предусмотрено в конструкторе.

Суперкласс содержит в себе следующие методы:

- *boolean checkWebElementIsPresented(WebElement webElement);*
- *boolean checkWebElementIdDisplayed(WebElement webElement)*

  предназначены для определения наличия в DOM открытой страницы и отображения веб-элемента;
- *WebElement waitForVisibility(WebElement webElement);*
- *WebElement waitForVisibility(List<WebElement> listOfElemenst);*
- *WebElement waitForClicable(WebElement webElement)*

  предназначены для задания явного ожидания появления веб-элементов в DOM и дальнейшей работе с ними;
- *WebDriver getDriverInstance()*

  предназначен для получения экземпляра вебдрайвера, при необходимости.

4.1.2 Пакет "accountPages"

Пакет "accountPages" содержит в себе классы, описывающие взаимодействие с веб-страницами которые тем или иным образом относятся к учётной записи и действиям пользователя веб-сайта сервиса Spotify.

Все классы пакета являются дочерними по отношению к суперклассу PageObjectBasics и наследуют от него все перечисленные в разделе 4.1.1 элементы.

accountPages.LoginPage

Класс LoginPage содержит в себе логику для работы на веб-странице входа в аккаунт. Экземпляры, созданные на основе этого класса входят в состав тестовых сценариев BaseTestScenario, ProfilePlaylistsTests, WrongLoginTestCases. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *userInputField* – веб-элемент формы ввода имени пользователя;
- *passwordInputFiled* – веб-элемент формы ввода пароля для входа в аккаунт;
- *loginButton* – кнопка входа в учётную запись;
- *loginCheckbox* – чекбокс для запоминания данных входа;
- *warningBanner* – баннер красного цвета который появляется в форме входа в аккаунт, предупреждающий о вводе некорректных данных для входа;
- *spotifyLogo* – элемент хэдера страницы при нажатии на который происходит возврат на стартовую веб-страницу сервиса Spotify. В качестве локатора для данного веб-элемента выбрано определение по CSS так как этот векторный графический элемент по Xpath определить не возможно.

Методы класса:

- *void setUserName(String loginEmail){*
  userInputFiled.clear();
  userInputFiled.sendKeys(loginEmail);
  } – ввод имени пользователя;

- *void setPassword(String loginEmail){*
  passwordInputFiled.clear();
  passwordInputFiled (loginPassword);
  } – ввод пароля от аккаунта;

- *void setCheckboxOff(){*
  loginCheckbox.click();
  } – выключение чекбокса;

- *void loginEnter(){*
  loginButton.click();
  } – нажатие на кнопку входа в аккаунт;

- *WebElement getWarningBanner(){*
  return warningBanner;
  } – обнаружение предупреждающего баннера на форме ввода данных;

- *void pressSpotifyLogo(){*
  spotifyLogo.click()
  } – нажатие на значок логотипа Spotify для перехода на стартовую веб-страницу сайта сервиса.

accountPages.PlaylistPage

Класс PlaylistPage содержит в себе логику для работы на веб-странице плейлиста, составленного пользователем. Экземпляр, созданный на основе этого класса входит в состав тестового сценария ProfilePlaylistsTests. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *playlistTitle* – название плейлиста;
- *titleInputFiled* – поле ввода для изменения названия плейлиста;
- *saveChangesButton* – кнопка сохранения изменений в поле выполнения действий над плейлистом;
- *searchSongsInputFiled* – поле ввода для поиска музыкальных композиций или исполнителей;
- *List<WebElement> searchingResults* – блок с найденными результатами запроса;

- *List<WebElement> addSongButton* – кнопки добавления выбранных песен в плейлист. Находятся в составе блока наёденных по запросу песен;

- *List<WebElement> playSongButton* – кнопки воспроизведения желаемой песни. Находятся в составе блока песен плейлиста;

- *List<WebElement> songNames* – текстовые поля названий песен. Находятся в составе блока песен плейлиста.

Private String *playingSongName* – текстовая переменная принимающая название воспроизводимой песни. Необходима для сравнения названия желаемой и фактически воспроизводимой песни.

Методы класса:

- *String getPlaylistTitle(){*
  return playlistTitle.getText();
  } – получение названия текущего плейлиста;

- *void renameOfPlaylist(String newTitle){*
  playlistTitle.click();
  titleInputField.clear();
  titleInputField.sendKeys(newTitle);
  saveChamgesButton.click();
  } – переименование текущего плейлиста;

- *void searchBand(String bandName){*
  searchSongInputField.click();
  searchSongInputField.clear();
  searchSongInputField.sendKeys(bandName);
  } – поиск желаемого исполнителя;

- *void chooseFromSearchResultList(){*
  String searchAttr;
  for(int i = 0; i < searchingResults.size(); i++){
  searchAttr = searchingResults.get(i).getAttribute("textContent")
  if(searchAttr.contains(searchQuery) && searchAttr.contains("Artist")){
  searchingResults.get(i).click();
  break;}
  }
  } – выбор раздела откуда будет производиться добавление песен в плейлист среди результатов (появившихся ссылочных карточек) осуществляется путём перебора результатов поиска и нахождения такой карточки, которая содержит в себе названия исполнителя и слово "Исполнитель" ("Artist). В методе создаётся локальная переменная searchAttr в которую заносится получаемый текстовый атрибут и сравнивается с требуемым. При нахождении соответствия происходит нажатие по выбранной карточке и выход из цикла;

- *void addSongsToPlaylist(){*
  for(int i = 0; i < addSongButton.size(); i++){
  actions.moveToElement(addSongButton.get(i)).pause(500).perform();
  addSongButton.get(i).click}
  } – добавление в плейлист всех открывшихся песен после перехода по ссылке с карточки исполнителя;

- void getSongsAmount() – метод вывода в консоль количества песен в текущем плейлисте путём получения размера списка веб-элементов playSongButton.size();

- *void playSongFromCurrentPlaylist(){*
  Random ran – new Random();
  int randomSong = ran.nextInt(playSongButton.size();

actions.moveToElement(playSongButton.get(randomSong)).
pause(500).click.perform();
playingSongName =songNames.get(randomSong).getText();

} – воспроизведение любой песни из текущего плейлиста. Создаётся локальная переменная randomSong типа int куда заносится случайно сгенерированное число в диапазоне до величины количества песен в плейлисте. Выбор песни для воспроизведения осуществляется путём передачи значения randomSong в индекс списка песен. Далее в переменную класса playingSongName заносится название воспроизводимой песни. Вывод в консоль информации о названии воспроизводимой песни и её порядковому номеру в плейлисте;

- String getPlayingSongName(){
return playingSong;} – *получение названия воспроизводимой песни для проведения ассертов в тестовых методах;*

- void compareSongName(String titleOfPlayingSongInPlayer){
boolean comp = false;
if(playingSongName.equals(titleOfPlayingSongInPlayer)){
comp = true;
инструкции для вывода результатов на консоль;
}

} – проверка соответствия выбранной для воспроизведения песни названию песни, указанному в панели плеера. В метод передаётся название воспроизводимой песни, взятое из панели плеера. Далее происходит сравнение значений переданного в метод и полученного до этого playingSongName. При совпадении содержимого обоих переменных выводится в консоль сообщение с названием песни по плейлисту и название, указанное в панели плеера.

accountPages.ProfilePage

Класс ProfilePage содержит в себе логику для работы на веб-странице учётной записи вошедшего пользователя. Экземпляр, созданный на основе этого класса, входит в состав тестового сценария BaseTestScenario. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *profileNameButton* – элемент с текущим именем пользователя.

Метод класса:

- String getProfileName(){
return profileNameButton.getAttribute("textContent");
} – метод для получения имени пользователя. Предназначен для дальнейших ассертов тестовом классе.

4.1.3 Пакет "mainPages"

Пакет "mainPages" содержит в себе классы, описывающие взаимодействие с основными веб-страницами, доступ к которым открыт как авторизовавшемся пользователю сервиса, так простому посетителю веб-сайта сервиса Spotify.

Все классы подкаталога являются дочерними по отношению к суперклассу PageObjectBasics и наследуют от него все перечисленные в разделе 4.1.1 элементы.

mainPages.SearchPage

Класс SearchPage содержит в себе логику для работы на веб-странице поиска. Экземпляры, созданные на основе этого класса, входят в состав тестовых сценариев BaseTestScenario и GettingSongsFromPlaylist. В состав класс входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *searchInputFiled* – поле ввода поиска по сервису Spotify;

- *List<WebElement> plylistSearchEntity* – список полученных в результате поиска ссылочных карточек в подразделе "Playlists" ("Плейлисты").

Методы класса:

- *void searchPlaylist(String playlistQuery){*
  actions.moveToElement(searchInputField).sendKeys(playlistQuery).perform();
  } – поиск желаемого стандартного плейлиста Spotify путём передачи строки запроса с интересующим плейлистом в поле поиска;

- *void enterToPlaylist(String desiredPlaylist){*
  String tagAttribute = desiredPlaylist + "Spotify";
  for(int i = 0; i < playlistSearchEntity.size(); i++){
  if(playlistSearchEntity.get(i).getAttribute("textContent"). equals(tagAttribute))
  {
  playlistSearchEntity.get(i).click();
  break;
  }
  }
  } – выбор наиболее подходящего варианта из списка карточек плейлистов. Осуществляется путём сопоставления значения атрибута "textContent" элементов списка с содержимым локальной переменной в которой задано требуемое значение. При нахождении совпадения производится взаимодействие с этим веб-элементом для перехода на веб-страницу желаемого СП.

mainPages.SpotifyPlaylist

Класс SpotifyPlaylist содержит в себе логику для работы на веб-странице стандартного плейлиста сервиса Spotify. Экземпляры, созданные на основе этого класса, входят в состав тестовых сценариев BaseTestScenario и GettingSongsFromPlaylist. В состав класс входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *playlistName* – имя открытого плейлиста;
- *footer* – футер веб-страницы;
- *List<WebElement> playButton* – список, содержащий в себе веб-элементы кнопок воспроизведения музыкальных композиций в плейлите. Количество данных элементов равно количеству треков, содержащихся в плейлисте;
- *List<WebElement> namesOfSongs* – список, содержащий в себе веб-элементы названий песен в плейлисте. Количество данных элементов равно количеству треков, содержащийся в плейлисте;
- *popupEnterButton* – кнопка перехода на страницу входа в аккаунт, находящаяся во всплывающем после попапе;

List<WebElement> listOfSongs – список всех треков, содержащихся в плейлисте;

Private static String *songForPlay* – переменная, принимающая значение названия предлагаемой для воспроизведения песни. Необходима для сравнения названия воспроизводимой песни.

Методы класса:

- *String getPlaylistName(){*
  return playlistName.getText()
  } – получение названия открытого плейлиста;

- *String getNumberOfSongs(){*
  actions.moveToElement(footer).pause(500).perform();
  return playButton.get(playButton.size()-1).getAttribute("textContent"))
  } – получение количества песен в плейлисте путём выявления номера последней песни. Так как в DOM открытой веб-страницы не отображены все позиции плейлиста, а динамически подгружаются в процессе вертикальной прокрутки, то для получения последнего трека необходимо спуститься до фу-

тера веб-страницы с текущим плейлистом. После этого из веб-элемента playButton в последнем треке извлекается порядковый номер путём просмотра атрибута "textContent";

- void ascendToHeader()
{actions.moveToElement(playlistName).perform()
} – поднятие к верху веб-страницы;

- void clickToPlayRandomSong(){
Random random = new Random();
int playSongNumber = random.nextInt(namesOfSongs.size());
songForPlay = namesOfSongs.get(playSongNumber).getText();
playbutton.get(playSongNumber).click();
popupEnterButton.click();
} – постановка песни из открытого плейлиста на воспроизведение после входа в аккаунт путём случайного выбора номера трека. Получение названия песни для воспроизведения в поле класса songForPlay понадобится для ассертов названия выбранной для воспроизведения песни перед входом в аккаунт с фактически воспроизводимой песней;

- String getSongForPlay(){return songForPlay} – получение названия музыкальной композиции перед воспроизведением для ассертов в тестовом классе BaseTestScenario;

Описание алгоритма работы метода *getFullListOfSongs()*

Как уже было сказано, в DOM открытой веб-страницы не отображён полный список треков, содержащихся в плейлисте и их отображение происходит по мере вертикальной прокрутки к низу страницы. Как показали наблюдения, DOM отображает, в среднем, от 25 до 50 позиций треков и по мере прокрутки добавляются новые позиции в конец блока, а какое-то количество старых – удаляется. Для получения информации об отображённых в DOM веб-страницы треках предусмотрен список *listOfSongs*.

Для получения всех треков и отображения названий и номеров музыкальных композиций в консоль в классе разработан метод *getFullListOfSongs()*, описание и принцип алгоритма которого представлен далее:

```
public void getFullListOfSongs(){
    int counter = 0;
    String songNumber, ariaRowIndex, maxAriaRowIndex;
    songNumber = ariaRowIndex = maxAriaRowIndex = null;
    boolean pointer = false;
```

//counter предназначен для контроля количества позиций в блоке треков, отображаемых в DOM страницы в конкретный момент. songNumber принимает порядковый номер песни, указанный в веб-элементе playButton (элемент входит в состав веб-элемента трека). Переменная ariaRowIndex принимает атрибут "ariaRowIndex" веб-элементов listOfSongs треков; данная переменная нужна для отслеживания состояния просматриваемого трека. maxAriaRowIndex – принимает значение атрибута "ariaRowIndex" последнего трека в плейлисте.

```
        actions.moveToElement(footer).pause(1000).perform();
  maxAriaRowIndex = listOfSongs.get(listOfSongs.size()-1).getAttribure("airaRowIndex");
        actions.moveToElement(playlistName).pause(1000).perform;
```

//для получения значения последнего в плейлисте трека необходимо спустится вниз, до футера страницы. После этого обновится блок DOM, содержащий список треков плейлиста. Получить искомое значение атрибута из последнего в List<WebElement> listOfSongs элемента. Вернуться на верх веб-страницы. Блок списка треков в DOM обновится в первоначальное состояние.

```
        for(int i = 0; i < listOfSongs.size(); i++){
```

```
                counter++;
                songNumber = playlistButton.get(i).getText();
                System.out.print(counter + "\t" + songNumber + " - ");
                System.out.println(namesOfSongs.get(i).getText());
                }
```
//Получение информации о первоначально содержащихся в DOM блоке треков музыкальных композиций и вывод в консоль их индекса в списке, номера трека и названия песни.

```
        actions.moveToElement(listOfSongs.get(listOfSongs.size()-1)).perform();
```

//Переход на последний трек из нынешнего состава треков в DOM

//При прокрутке веб-страницы и переходе на позицию последней отображённой на текущий момент в блоке треков DOM песни блок обновляется, при этом последняя позиция становится промежуточной. В данной промежуточной позиции у веб-элемента playButton текст отсутствует, что позволяет, проитерировав обновлённый список треков, отбросить повторяющиеся треки и в консоль вывести только те которые вошли обновлённый список. Повторение данного механизма происходит до тех пор пока атрибут "ariaRowIndex" текущего трека веб-элемента listOfSongs не будет больше значения maxAriaRowIndex.
Данный механизм описан в блоке do-while:

```
        do{
            System.out.println("Size of list is : " + listOfSongs.size());
            //Позволяет в консоли отследить обновлённый список треков
            counter = 0; pointer = false;
            for(int i = 0; i < listOfSongs.size(); i++){
                if(pointer == false){
                    counter++;
                    songNumber = playButton.get(i).getText();
                    pointer = songNumber.equals("");
```
//в каждой итерации блока do происходит проверка на соответствие номера песни пустой строке. Достижение истинности означает, что в текущей вариации списка достигнуты новые треки, которых не было в предыдущей вариации listOfSongs. Соответственно пока значение указателя не истинно – номера и песни в консоль не выводятся.
```
                }
                else{
                    counter++;
                    songNumber = playButton.get(i).getText();
                    ariaRowIndex=listOfSongs.get(i).getAttribute("ariaRowIndex");

                    System.out.print(counter + "\t" + songNumber + " - ");
                    System.out.println(namesOfSongs.get(i).getText());
```
//Текущее значение ariaRowIndex принимается по атрибуту трека и сравнивается с последним значением в плейлисте, принятым в начале работы метода.
```
                }
            }
            actions.moveToElement(listOfSongs.get(listOfSongs.size()-1)).perform();
        } while(ariaRowIndex.equals(maxAriaRowIndex) != true);
}
```

mainPages.startPage

Класс StartPage содержит в себе логику для работы на стартовой странице веб-сайта сервиса Spotify. Экземпляр, созданный на основе этого класса, входит в состав тестового сценария BaseTestScenario. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *List<WebElement>* popularSections – список блоков популярных разделов;
- *showAllButton* – кнопка раскрытия содержимого популярных разделов;

- *List<WebElement> cardsFromPopularSections* – список, содержащий веб-элементы ссылочных карточек из популярных разделов.

Методы класса:

- *boolean searchPlaylistSection(){*
  for(int i = 0; i < popularSections.size(); i++){
  if(popularSections.get(i).getText().equals("Spotify Playlists"))
  return true;
  }
  return false;
  } – проверка нахождения секции "Плейлисты Spotify" на стартовой веб-странице;

- void showAllPlaylists(){
  for(int i = 0; i < popularSections.size(); i++){
  if(popularSections.get(i).getText().equals("Spotify Playlists")){
  popularSections.get(i).click();
  break;}
  }
  } – выбор секции "Плейлисты Spotify" и её открытие;

- *void clickOnPlaylist(){*
  for(int i = 0; i < cardsFromPopularSections.size(); i++){
  if(cardsFromPopularSections.get(i).getAttribute("textContent")).
  equals("Rock Classics")){
  actions.moveToElement(cardsFromPopularSections.get(i).
  pause(500)).perform();
  cardsFromPopularSections.get(i).click();
  break;
  }

//Первоначально осуществляется поиск плейлиста "Rock Classics". Если по атрибуту textContent искомый плейлист найден, то происходит клик по ссылочной карточке этого плейлиста и осуществляется переход на его страницу.

  if(i == cardsFromPopularSections.size()-1){
  Random random = new Random();
  int chosenNumber = random.nextInt(cardsFromPopularSections.size());
  actions.moveToElement(cardsFromPopularSections.get(chosenNumber).
  pause(500)).perform();
  cardsFromPopularSections.get(chosenNumber).click();
  }

//Если плейлиста "Rock Classics" в списке не обнаружено, то цикл доходит до последней итерации и происходит выбор любого плейлиста из списка и осуществляется переход на его страницу.

  }
  } – выбор плейлиста Spotify и его открытие;

- *int getAmountOfPlaylists(){*
  return cardsFromPopularSections.size();
  } – получение информации о количестве представленных на странице карточек с плейлистами.

4.1.3 Пакет "sidebarsAndMenus"

Подкаталог "sidebarsAndMenus" содержит в себе классы, описывающие элементы сайдбаров, меню и панели плеера, которые появляются при загрузке любой веб-страницы сайта музыкального сервиса Spotify.

Все классы подкаталога являются дочерними по отношению к суперклассу PageObjectBasics и наследуют от него все перечисленные в разделе 4.1.1 элементы.

sidebarsAndMenus.CookiesMenu

Класс CookiesMenu содержит в себе логику для закрытия блока, предупреждающего об использовании веб-сайтом файлов cookie, а так же проверку его отображении на веб-странице. Экземпляры, созданные на основе этого класса, входят в тестовые сценарии WrongLoginTestCases, BaseTestScenario, GettingSongsFromPlaylist, ProfilePlaylistTests. В состав класса входят следующие приватные поля типа WebElement, подписанное аннотацией @FindBy:

- *closeCookiesButton* – элемент закрытия панели, содержащий предупреждение об использовании cookies.

Методы класса:

- *void closeCookies(){*

```
if(checkWebelementDisplayed(closeCookiesButton) == true)
        closeCookiesButton.click();
else
        System.out.println("Cookies are already closed");
```

} – проверка отображения панели о cookies. В начале вызывается метод суперкласса PageObjectBasics checkWebElementDisplayed() для проверки отображения веб-элемента. При его наличии происходит нажатие для закрытия панели. При его отсутствии (необходимо при возвращении на стартовую страницу в тестовых сценариях) - в консоль выводится сообщение о том, что элемент уже закрыт.

sideBarsAndMenus.LeftSidebar

Класс LeftSidebar содержит в себе логику для взаимодействия с левой боковой панелью, которая присутствует на каждой открытой веб-странице сайта Spotify. Экземпляры, созданные на основе этого класса, входят в тестовые сценарии GettingSongsFromPlaylist, ProfilePlaylistTests. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *buttonStartPage* – кнопка перехода на стартовую страницу веб-сайта;
- *buttonSearchPage* – кнопка перехода на страницу поиска веб-сайта;
- *createPlaylistOfRolder* – кнопка для создания открытия подменю создания плейлиста или папки;
- *createPlaylistSubmenuButton* – кнопка "Создать плейлист".

Методы класса:

- *void goToStartPage(){*

```
buttonStartPage.click();}
```

- переход на стартовую страницу веб-сайта;

- *void goToSearchPage(){*

```
buttonSearchPage.click();}
```

- переход на веб-страницу поиска на сайте;

- *void createNewPlaylist(){*

```
createPlaylistOrFolderButton.click();
createPlaylistSubmenuButton.click();
```

} – нажатие по указанным веб-элементам для создания нового плей-ласта пользователя;

- *void openNewestEmptyPlaylist(){*
listOfPlaylists.get(0).click();
} – переход на страницу верхнего в списке боковой панели плейлиста;

- *int getAmountOfPlaylists(){*
return Integer.parseInt(listOfPlaylists.get(0).getAttribute("ariaSetSize"));
} – получение количества плейлистов в учётной записи пользователя путём просмотра атрибута ariaSetSize у веб-элемента listOfPlaylists из блока плейлистов.

sidebarsAndMenus.PlayerPanel

Класс PlayerPanel содержит в себе логику для работы с панелью проигрывателя, находящуюся в нижней части экрана и доступную зарегистрированному пользователю только после входа в учётную запись. Экземпляры, созданные на основе этого класса, входят в тестовые сценарии BaseTestScenario, ProfilePlaylistTests, WrongLoginTestCases. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *playerBar* – панель управления плеера;
- *playerPlayPauseButton* – кнопка воспроизведения-паузы;
- *currentSong* – блок, отображающий название воспроизводимой песни;
- *signUpButton* – кнопка "Зарегистрироваться", отображается когда не осуществлён вход в аккаунт.

Методы класса:

- *void pauseCurrentSong(){*
playerPlayPauseButton.click();} – нажатие кнопки паузы-воспроизведеия;

- *String getCurrentSongPlayer(){*
return currentSong.getText();} – получение названия текущей песни;

- *WebElement getPlayerBar(){*
return playerBar;} – возвращает опрашиваемый веб-элемент для дальнейших с ним действий;

- *boolean signUpButtonPresented(){*
return checkWebElementPresented(signUpButton);
} – проверка наличия веб-элемента в DOM текущей открытой страницы путём запуска метода checkWebElementPresented() из суперкласса.

sidebarsAndMenus.TopMenu

Класс TopMenu содержит в себе логику для взаимодействия с верхним горизонтальным меню, служащей для различных действий с аккаунтом пользователя. Экземпляры, созданные на основе этого класса, входят в тестовые сценарии BaseTestScenario, ProfilePlaylistTests, WrongLoginTestCases. В состав класса входят следующие приватные поля типа WebElement, подписанные аннотацией @FindBy:

- *loginButton* – кнопка для перехода на веб-страницу входа в аккаунт;
- *userWidgetButton* – кнопка выпадающего меню действий вошедшего в учётную запись пользователя;
- *userWidgetMenuProfileButton* – кнопка перехода в профайл пользователя в выпавшем подменю;
- *userWidgetMenuLogoutButton* – кнопка выхода из учётной записи в выпавшем подменю.

Методы класса:

- *void EnterToLoginPage(){*
loginButton.click()} –переход на страницу входа в аккаунт пользователя;

- void enterToProfilePage(){
userWidgetButton.click();
userWidgetMenuProfileButton.click();}–переход на страницу пользвателя;

- String getUserName(){
return userWidgetButton.getAttribute("ariaLabel")} – получение имени пользователя путём просмотра атрибута веб-элемента userWidgetButton;

- void logOutFromAccount(){
userWidgetButton.click();
userWidgeMenuLogoutButton.click();
} – выход из учётной записи пользователя.

4.2 Каталог "utils"

Каталог "utils" содержит в себе файлы классов TestDataReader и WebDriverSingleton, которые предназначены для вспомогательных действий во время работы программного решения.

4.2.1 TestDataReader

Класс TestDataReader предназначен для чтения внешних данных, необходимых для настройки и конфигурации работы программного решения, которые содержатся в файле testData.properties.

В составе класса поля, входящих в стандартную библиотеку Java типов – FileInputStream для считывания данных из внешних текстовых файлов и Properties для работы с полученными данными.

Требуемое по ключу значение считывается и передаётся в статическом методе:

```
static String readProperty(String propertyKey){
    try{
        fileInputStream = new FileInputStream("путь_к_файлу.properties");
        properties = new Properties();
        properties.load(fileInputStream);
        if(fileInputStream != null){
            fileInputStream.close();
            fileInputStream = null;
        }
    } catch(IOException exc){System.out.println(exc);}
return properties.getProperty(propertyKey);
}
```

Метод предусматривает обработку исключения ошибок ввода-вывода.

4.2.2 WebDriverSingleton

Класс WebDriverSingleton разработан для настройки единственного экземпляра вебдрайвера используемого в работе веб-браузера и передачи этого экземпляра всем тестовым классам.

В составе класса есть статический метод

static WebDriver setupDriver(String browserType, String driverPath) который в качестве аргументов принимает значение названия рабочего веб-браузера и значение адреса нахождения утилиты вебдрайвера в постоянной памяти рабочего компьютера. При отсутствии значения переменной веб-драйвера создаётся его экземпляр по входным параметрам, настраиваются таймауты неявных ожиданий, загрузки страницы и скриптов; тут же задаётся размер рабочего окна открываемого веб-браузера.

5 Данные для работы программного решения и тестирования работы веб-сайта

Необходимые исходные данные для работы и тестирования хранятся во внешнем текстовом файле testData.properties, который хоть и расположен в каталоге "resources" модуля "test", но не является неотъемлемой частью программного решения и может храниться вне каталогов с данным программным решением – в постоянной памяти рабочего компьютера. Для этого в классе TestDataReader необходимо в переменной fileInputStream указать другой путь к конфигурационному файлу.

Исходные данные для тестирования, такие как тип веб-браузера, путь расположения утилиты веб-драйвера в памяти компьютера, адрес открываемой веб-страницы, имя пользователя, пароль представлены в .properties файле в текстовом формате "ключ-значние". Значения считываются при помощи статического метода *String readProperty(String propertyKey)* класса TestDataReader.

6 Файлы конфигурации и управления тестами

Программным решением предусмотрены файлы управления тестовыми сценариями фреймворка TestNg в формате .xml:

1. loginTests.xml задаёт исполнение теста на проверку работы веб-сайта, имитируя действия пользователя по входу в учётную запись и некоторую работу по использованию сервиса Spotify. Содержит два тестовых сценария, описанных в тестовых классах
   - smokeTests.BaseTestScenario;
   - negativeScenarios.WrongLoginTestCases.

2. plsaylistsTests.xml задаёт исполнение теста на проверку работы веб-сайта, имитируя действия пользователя по работе с плейлистами: просмотр существующих стандартных плейлистов Spotify, созданию пользовательских плейлистов, управление ими, добавление песен. Содержит два тестовых сценария, описанных в тестовых классах:
   - smokeTests.ProfilePlaylistsTests;
   - smokeTests.GettingSongsFromPlaylist.

7 Использование программного решения Spotify_website_UI_testing

Для работы со всеми тестовыми сценариями, предусмотренными настоящим программным решением пользователю необходимо иметь зарегистрированную учётную запись в сервисе Spotify, либо зарегистрировать аккаунт, пройдя на веб-страницу:

https://spotify.com/signup

или нажать на кнопку "Зарегистрироваться" на стартовой странице веб-сайта сервиса Spotify.

Данные для входа в учётную запись сервиса Spotify:

- электронная почта на которую зарегистрирован аккаунт;
- пароль для входа

вписать в файл testData.properties в соответствующие поля "user_name" и "password".

Также в поле "driver_path" файла testData.properties прописать путь доступа к утилите веб-дайвера в памяти на компьютере пользователя.

При использовании в работе настоящего программного решения веб-браузера, отличного от Chrome, в поле "browser_type" файла testData.properties указать название рабочего веб-браузера (например – Firefox).