**<https://ci2.timelysoft.org>**

**Test plan**

**Version 1.0**

**23.03.2023**

**Test Plan Identifier:** [Test Case 83178](https://ci2.timelysoft.org:8080/tfs/DefaultCollection/Navigator.CRM/_workitems/edit/83178): Check 'Платформа' filter (all options)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата вноса изменений | Автор | Функционал | Утверждено | Версия документа |
| 23.03.2023 | Ханайым Белекова | Check 'Платформа' filter (all options) |  | #1.0 |

**Введение**

Тест план написан для того чтоб проверить и Check 'Платформа' filter (all options), сайта Navigator. Цель этого тест-плана – проверить работу сайта , работу от создания клиента до работы дропдауна Платформы чтобы убедиться, что она правильно работает и возвращает ожидаемые результаты.

**Test Objects** = **Features To Be Tested**

<https://ci2.timelysoft.org:8080/tfs/DefaultCollection/Navigator.CRM/_workitems/edit/83042> -сайт под тестированием

### **Тест-кейсы**

#### 1. Основная проверка ввода

В этом тесте будет проверена основная функциональность проверки ввода формы.

##### **Тест-кейс 1.1: Проверка обязательных полей**

<https://ci2.timelysoft.org:8080/tfs/DefaultCollection/Navigator.CRM/_testManagement?_a=tests&Length=2&planId=82226&suiteId=83406>

* Описание теста: Проверка ввода обязательных полей.
* Шаги теста:
  1. Оставьте обязательные поля пустыми в форме.
  2. Отправьте форму.
* Ожидаемый результат: Должно появиться сообщение, информирующее пользователя о том, что поле обязательно для заполнения.
* Фактический результат: [Записать фактический результат]

##### **Тест-кейс 1.2: Проверка формата адреса электронной почты**

* Описание теста: Проверка ввода недопустимого адреса электронной почты.
* Шаги теста:
  1. Введите недопустимый адрес электронной почты (например, abc) в форму.
  2. Отправьте форму.
* Ожидаемый результат: Должно появиться сообщение, информирующее пользователя о том, что адрес электронной почты недопустим.
* Фактический результат: [Записать фактический результат]

#### 2. Расширенная проверка ввода

В этом тесте будет проверена расширенная функциональность проверки ввода формы.

##### **Тест-кейс 2.1: Проверка формата номера телефона**

* Описание теста: Проверка ввода недопустимого номера телефона.
* Шаги теста:
  1. Введите недопустимый номер телефона (например, 1234567890) в форму.
  2. Отправьте форму.
* Ожидаемый результат: Должно появить
* Фактический результат: [Записать фактический результат]

##### **Тест-кейс 2.2: Проверка формата даты**

* Описание теста: Проверка ввода недопустимого формата даты.
* Шаги теста:
  1. Введите недопустимый формат даты (например, 30/02/2022) в форму.
  2. Отправьте форму.
* Ожидаемый результат: Должно появиться сообщение, информирующее пользователя о том, что формат даты недопустим.
* Фактический результат: [Записать фактический результат]

### **Результаты тестирования**

<https://ci2.timelysoft.org:8080/tfs/DefaultCollection/Navigator.CRM/_testManagement?_a=tests&Length=2&planId=82226&suiteId=83406>

Записать результаты тестирования для каждого тест-кейса, указав фактический результат.

### **Выводы**

На основе результатов тестирования сделать выводы о качестве валидации формы , также проверить работу дропдауна “платформа” и дать комментарии если это необходимо.

**1 тест кейс (тест по проверке обязательных полей)**

В данном примере мы определяем три тест-кейса для проверки обязательных полей формы создания автоуровня. В каждом тест-кейсе мы создаем объект formData, содержащий значения полей формы, и передаем его в функцию validateAutoLevelForm(), которая проверяет обязательные поля и возвращает true или false в зависимости от того, заполнены ли все обязательные поля. Затем мы используем библиотеку assert для проверки ожидаемого результата. Если функция validateAutoLevelForm() возвращает true для объекта formData, который заполнен правильно, то тест-кейс считается пройденным. Если функция возвращает false, то тест-кейс считается неудачным

const assert = require('assert');

const { validateAutoLevelForm } = require('./autoLevelFormValidation'); // импортирование функции валидации формы

describe('Auto level create form validation', function() {

describe('validateAutoLevelForm()', function() {

it('should return false when name field is empty', function() {

const formData = {

name: '',

description: 'This is a test description'

};

const result = validateAutoLevelForm(formData);

assert.strictEqual(result, false);

});

it('should return false when description field is empty', function() {

const formData = {

name: 'Test Auto Level',

description: ''

};

const result = validateAutoLevelForm(formData);

assert.strictEqual(result, false);

});

it('should return true when all fields are filled', function() {

const formData = {

name: 'Test Auto Level',

description: 'This is a test description'

};

const result = validateAutoLevelForm(formData);

assert.strictEqual(result, true);

});

});

});

2 **Тест по проверке формата адреса электронной почты**

В данном примере мы создаем два тест-кейса для проверки правильности формата адреса электронной почты. В каждом тест-кейсе мы вызываем функцию validateEmailFormat() с тестовыми данными (валидный и невалидный форматы электронной почты) и проверяем, возвращает ли функция ожидаемый результат (true для валидного формата, false для невалидного формата) с помощью библиотеки assert. Если функция validateEmailFormat() возвращает true для валидного адреса электронной почты, то тест-кейс считается пройденным. Если функция возвращает false, то тест-кейс считается неудачным.

function validateEmailFormat(email) {

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return emailRegex.test(email);

}

console.log(validateEmailFormat('example@mail.com')); // true

console.log(validateEmailFormat('examplemail.com')); // false

**3 тест по проверке формата даты**

В данном примере мы используем библиотеку assert из Node.js для проверки результатов наших тестов. Мы определяем функцию isValidDate(), которая принимает строку с датой и возвращает true, если дата соответствует формату "YYYY-MM-DD", и false в противном случае. Для проверки формата даты мы используем регулярное выражение dateRegex, которое проверяет, что в дате есть четыре цифры, затем символ "-", затем две цифры, затем опять символ "-", затем еще две цифры.

Затем мы используем функцию describe() для группировки тестов, и функцию it() для определения отдельных тест-кейсов. В первом тест-кейсе мы проверяем, что функция isValidDate() возвращает true для корректного формата даты. Во втором тест-кейсе мы проверяем, что функция isValidDate() возвращает false для некорректных форматов даты.

Для каждого тест-кейса мы используем функцию assert.strictEqual() для сравнения ожидаемого результата с фактическим. Если ожидаемый результат и фактический результат совпадают, то тест проходит успешно. Если они не совпадают, то тест завершается с ошибкой и выводится сообщение об ошибке.

const assert = require('assert');

function isValidDate(dateString) {

const dateRegex = /^\d{4}-\d{2}-\d{2}$/;

return dateRegex.test(dateString);

}

describe('isValidDate', function() {

it('should return true for valid date format (YYYY-MM-DD)', function() {

assert.strictEqual(isValidDate('2023-03-24'), true);

});

it('should return false for invalid date format', function() {

assert.strictEqual(isValidDate('03/24/2023'), false);

assert.strictEqual(isValidDate('2023-03-24 12:30:00'), false);

assert.strictEqual(isValidDate('24-03-2023'), false);

});

});

**4 тест по проверке формата номера телефона**

В данном примере мы используем библиотеку assert из Node.js для проверки результатов наших тестов. Мы определяем функцию isValidPhoneNumber(), которая принимает строку с номером телефона и возвращает true, если номер соответствует формату, и false в противном случае. Для проверки формата номера мы используем регулярное выражение phoneRegex, которое проверяет, что строка начинается с символа "+", за которым следует одна или более цифр от 1 до 9, и затем может быть до 14 цифр от 0 до 9.

Затем мы используем функцию describe() для группировки тестов, и функцию it() для определения отдельных тест-кейсов. В первом тест-кейсе мы проверяем, что функция isValidPhoneNumber() возвращает true для корректного формата номера телефона. Во втором тест-кейсе мы проверяем, что функция isValidPhoneNumber() возвращает false для некорректных форматов номеров телефонов.

Для каждого тест-кейса мы используем функцию assert.strictEqual() для сравнения ожидаемого результата с фактическим. Если ожидаемый результат и фактический результат совпадают, то тест проходит успешно. Если они не совпадают, то тест завершается с ошибкой и выводится сообщение об ошибке.

const assert = require('assert');

function isValidPhoneNumber(phoneNumber) {

const phoneRegex = /^\+?[1-9]\d{1,14}$/;

return phoneRegex.test(phoneNumber);

}

describe('isValidPhoneNumber', function() {

it('should return true for valid phone number format', function() {

assert.strictEqual(isValidPhoneNumber('1234567890'), true);

assert.strictEqual(isValidPhoneNumber('99999999999999'), true);

assert.strictEqual(isValidPhoneNumber('+11234567890'), true);

});

it('should return false for invalid phone number format', function() {

assert.strictEqual(isValidPhoneNumber('abc'), false);

assert.strictEqual(isValidPhoneNumber('123456789012345'), false);

assert.strictEqual(isValidPhoneNumber('00000000000000'), false);

assert.strictEqual(isValidPhoneNumber('+0000000000000'), false);

});

});

**5.** **Тест кейсы пройдены**

После прохождения тест кейсов на иконке search появились Дропдауны с фильтрами

Платформа выдала мне Чекбокс с CRM, IOS,Android etc . там выбрала нужный чекбокс но позже он не применился к фильтрам

**После чего было решено сделать:**

if(typeof checkbox[0] !== 'undefined' && typeof checkbox[0].title !== 'undefined'){

return checkbox;

}