Модули

**Основные**

Mainscript – частично готово

ClientsScript – частично готово

Авторизация – частично готово

**Синхронизация**Основная информация  
Вознаграждения  
Товары  
Заказы

**Утилиты**

Капча – иногда не работает

Удаление корзины – частично готово

Написание отзывов – готово, но не внедрено

**Действия**

Заказ корзины

**База данных  
  
Общая архитектура.**Так, смотри, какая общая идея проекта, он нужен для заказа корзин, отслеживание состояние совершенных заказов, состояние аккаунтов, синхронизация данных и тд.Пока что в конфиге.

**Адрес** (откуда должен знать про последний заказанный адрес).   
(мини план:

“Orders.  
Логика смены адреса  
По очереди или рандомный?   
как подставлять адрес”)

**Введение:**

Так, один из функций высшего модуля — это **подставлять нужный адрес** в инпут конфиг.   
Он смотрит (сколько раз был использован адрес из ordersConfig.js за последние {x} дней (параметр changeAddressFrequencyByDay)

на ordersData.json (…CRM\data\OrdersData\ordersData.json) в поле **deliveryAddress  
пример:** "deliveryAddress": "Данияр Ермек Рахимович",

**Логика смены адреса**:  
Если он повторялся более {2} раз за последние {x} (параметр хранится в ordersConfig.js поле  changeAddressFrequency: 2),

дней он **меняет** адрес на другой (в случайном порядке)

**Откуда брать адрес**:  
Адрес находится в C:\Users\Didar1520\Docks\CRM\data\adressBook\adressList.JSON  
поле FullName (то есть нам не нужны для этой логики все данные, поле FullName (ФИО) хватает).   
  
**Если адрес не повторялся,** то оставляет текущий адрес в ordersConfig.js (C:\Users\Didar1520\Docks\CRM\scripts\actions\ ordersConfig.js)

Поле - defaultAdress

**Тестирование**:  
Пока что просто запускаем функцию adressManager (нужно создать таковой внутри папки utils (C:\Users\Didar1520\Docks\CRM\scripts\actions\utils).   
  
Запускаем через высший модуль.   
  
Остальные функций не запускаем (например, смену аккаунта и тд, в целом не запускаем пока хромиум и папитир), просто протестируем и выведем в консоль какой адрес по умолчанию стоял и подходит ли он нам или же меняем, если меняем записываем в соответствующий файл (меняем) и выводим в консоль.   
И напиши, как тестировать, что запустить и тд.   
  
  
  
1) orderManager.js был написан тобой (как и весь проект) и был написан некорректно, я просто могу тебе дать к нему инструкцию каким он должен быть (по этому инструкций ты тоакже можешь задавать вопросы, цикл вопросов может быть бесконечным пока ты не останешься без вопросов)   
2) я тебе скину файл, только не знаю насколько он корректный, возможно его тоже придеться улучшить  
3) мы хотим их брать только из параметров, вообще я планирую сделать эти параметры общими для всей программы и перенести модуль синхронизаций просто в модуль utils ведь нету никакой логики в таком делений на разные части.   
  
  
Объединение модулей и изменение конфигураций

Я сделал много изменений, по этому тебе придется исправить много импортов, и названия переменных   
  
1) теперь передача аккаунта для синхронизаций данных (парсинг данных профиля аккаунта или заказанных товаров) осуществляется с одного места и все является одним модулем, теперь синхронизация это просто утилита и часть программы   
2) вот тебе пока что код некоторых файлов (расположение выше):  
  
  
1) функция синхронизаций просто стало утилитой и не надо в мейн скрипте или где либо еще делать особый акцент на разный функционал, теперь syncData это просто утилита который можно запустить для синхронизаций данных как и утилита для отзывов и тд.   
поэтому нету смысла в action="syncAccount", если нужно запустить синхронизацию это просто придет в инпутконфиге  
2) включается в inputConfig.json:   
например:  
"Account": "Lebron@gmail.com",

"sync": "orders, reviews, mainInfo" (тут написано что нужно синхронизировать эту информацию по очереди)  
а остальные поля по типу корзины вовсе не приходят или могут быть пустыми.   
3) нужно в целом убрать эту логику ведь у нас сейчас другая архитектура   
  
  
Правки:  
я получил сообщение: Получен ответ от Puppeteer скрипта: { status: false, message: 'Нет sync/cartLink. Нечего делать.' }  
хотя в инпутконфиге у меня такие данные:  
…..  
  
что означает что нужно оформить 2 заказа и сделать синхронизацию данных ("orders, reviews, mainInfo

) для LebronJameskz@mail.ru

Если подробно то первую корзину он оформляет с аккаунта [didar1520@gmail.com](mailto:didar1520@gmail.com) а затем закрывает браузер и открывает с профиля леброна (или создает если такогового нету, все это есть в browserManager) и затем оформляет заказ с аккаунта леброн и затем не закрывая браузер, поскольку для синхронизаций данных запрошен тот же аккаунт делает синхронизацию данных.   
  
вот что он должен был сделать и часть этого функционала (по логике и бест практис) должен был быть в ордер менеджере.   
  
И поскольку у нас еще не готова функция оформление заказа, он просто должен в консоли показать что получил параметры из какого то места и что он якобы оформил заказ и все прошло успешно.   
И соответсвенно поскольку логика авторизаций у нас есть, просто выполнить его, показать в консоли данные, затем написать что заказ оформлен и закрыть и переключится на другой аккаунт в инпуте и дедать то что он должен.   
И затем опять после успешной авторизаций и консоль лога он должен опять с имитировать синхронизацию

, просто остаться на том аккаунте если синхронизация на том же и показать консоль лог об успещшной синхронизаций.   
пока что у нас будут консоль логи и после тестирований подключим функционал

(ps не забывай про вопрос для уточнения если они есть, они очень важны)

Мы с тобой пару дней просидели над кодом.   
Мне просто нужен был код который кликает на элементе дропдауна и пару дней мы не смогли его написать.   
но в итоге мы его написали.   
посмотри в чем отличие неправильных вариантов от правильных и почему там клик не работал.   
вот тебе неправильные варианты:  
(async function() {

const wait = (ms) => new Promise(resolve => setTimeout(resolve, ms));

// Функция симуляции клика с использованием PointerEvent и scrollIntoView

function simulateMouseClick(element) {

console.log("Simulating click on element:", element);

element.scrollIntoView({ behavior: 'smooth', block: 'center', inline: 'center' });

const rect = element.getBoundingClientRect();

const clickX = rect.left + rect.width / 2;

const clickY = rect.top + rect.height / 2;

const options = {

bubbles: true,

cancelable: true,

clientX: clickX,

clientY: clickY,

view: window

};

// Отправляем события pointerover, pointerdown, pointerup и click

element.dispatchEvent(new PointerEvent('pointerover', options));

element.dispatchEvent(new PointerEvent('pointerdown', options));

element.dispatchEvent(new PointerEvent('pointerup', options));

element.dispatchEvent(new PointerEvent('click', options));

}

// 1. Находим и кликаем по инпуту валюты (.select-currency .search-input)

const currencyInput = document.querySelector('.select-currency .search-input');

if (!currencyInput) {

console.error('Элемент .select-currency .search-input не найден.');

return;

}

console.log("Кликаем по инпуту валюты");

currencyInput.focus();

simulateMouseClick(currencyInput);

// Ждем 2 секунды

await wait(2000);

// 2. Имитация ввода текста "USD" по символам с задержкой

async function simulateHumanTyping(element, text, delay = 150) {

element.focus();

element.value = ""; // очищаем инпут

for (let char of text) {

element.dispatchEvent(new KeyboardEvent('keydown', { key: char, bubbles: true }));

element.value += char;

element.dispatchEvent(new Event('input', { bubbles: true }));

element.dispatchEvent(new KeyboardEvent('keyup', { key: char, bubbles: true }));

await wait(delay);

}

}

await simulateHumanTyping(currencyInput, 'USD', 150);

// Ждем 2 секунды для появления выпадающего списка

await wait(2000);

// 3. Ожидание появления элемента дропдауна с нужными классами и data-val="USD"

async function waitForElement(selector, timeout = 5000) {

const startTime = Date.now();

while (Date.now() - startTime < timeout) {

const el = document.querySelector(selector);

if (el) return el;

await wait(100);

}

throw new Error(`Элемент ${selector} не найден в течение ${timeout} мс`);

}

const dropdownSelector = 'div.item.gh-dropdown-menu-item.grouped-item.selected[data-val="USD"]';

try {

const dropdownItem = await waitForElement(dropdownSelector, 5000);

console.log("Найден целевой элемент:", dropdownItem);

// 4. Выполняем клик по найденному элементу дропдауна

simulateMouseClick(dropdownItem);

} catch (error) {

console.error(error);

}

})();  
  
(async function() {

// Функция ожидания

function sleep(ms) {

return new Promise(resolve => setTimeout(resolve, ms));

}

// Функция симуляции pointer-событий (mousedown, mouseup, click)

function simulatePointerClick(element) {

if (!element) return;

const pointerDown = new PointerEvent("pointerdown", {

bubbles: true,

cancelable: true,

pointerType: "mouse",

view: window

});

element.dispatchEvent(pointerDown);

const pointerUp = new PointerEvent("pointerup", {

bubbles: true,

cancelable: true,

pointerType: "mouse",

view: window

});

element.dispatchEvent(pointerUp);

const clickEvent = new PointerEvent("click", {

bubbles: true,

cancelable: true,

pointerType: "mouse",

view: window

});

element.dispatchEvent(clickEvent);

console.log("Pointer events dispatched on element:", element);

}

// 1. Находим инпут для поиска валюты

const input = document.querySelector(".select-currency .search-input");

if (!input) {

console.error("Элемент .select-currency .search-input не найден");

return;

}

// Кликаем по инпуту для установки фокуса (используем pointer-события)

input.focus();

simulatePointerClick(input);

console.log("Инпут для поиска валюты кликнут и получил фокус");

await sleep(500);

// 2. Имитируем набор текста "USD" по буквам с задержками

input.value = ""; // очищаем значение, если нужно

const text = "USD";

for (const char of text) {

// Эмулируем keydown

const keydownEvent = new KeyboardEvent("keydown", { key: char, bubbles: true });

input.dispatchEvent(keydownEvent);

// Добавляем символ и генерируем событие input

input.value += char;

input.dispatchEvent(new Event("input", { bubbles: true }));

console.log(`Введена буква: ${char}`);

await sleep(300); // задержка между нажатиями

// Эмулируем keyup

const keyupEvent = new KeyboardEvent("keyup", { key: char, bubbles: true });

input.dispatchEvent(keyupEvent);

await sleep(100);

}

console.log("Полное значение 'USD' введено");

// 3. Ждем пару секунд, чтобы дропдаун обновился и нужный элемент появился

await sleep(2000);

// 4. Ищем целевой элемент и выводим его размеры для отладки

const target = document.querySelector(".selected:nth-child(2) .cloned-label");

if (target) {

const rect = target.getBoundingClientRect();

console.log("BoundingRect целевого элемента:", rect);

if (rect.width === 0 || rect.height === 0) {

console.error("Целевой элемент имеет нулевые размеры – возможно, он не видим или скрыт");

} else {

console.log("Целевой элемент виден, эмулируем клик через PointerEvent");

}

simulatePointerClick(target);

console.log("Клик по целевому элементу выполнен");

} else {

console.error("Целевой элемент .selected:nth-child(2) .cloned-label не найден");

}

})();

// Функция для эмуляции человеческого ввода с задержками

function humanDelay(min = 200, max = 500) {

return new Promise(resolve => setTimeout(resolve, Math.random() \* (max - min) + min));

}

(async function() {

try {

// Находим контейнер выбора валюты

const currencyDropdown = document.querySelector('.select-currency');

// 1. Находим и кликаем на основной триггер дропдауна

const trigger = currencyDropdown.querySelector('.gh-form-control'); // Измененный селектор

if (!trigger) throw new Error('Dropdown trigger not found');

// Эмулируем полный цикл клика

const clickEvents = ['mousedown', 'mouseup', 'click'];

for(let event of clickEvents) {

trigger.dispatchEvent(new MouseEvent(event, {

bubbles: true,

cancelable: true,

view: window

}));

}

console.log('Currency dropdown opened');

await humanDelay(300, 500);

// 2. Вводим текст в поисковое поле

const searchInput = currencyDropdown.querySelector('input.search-input');

searchInput.focus();

// Эмуляция человеческого ввода

const inputEvents = ['keydown', 'input', 'keyup'];

searchInput.value = 'USD';

for(let event of inputEvents) {

searchInput.dispatchEvent(new Event(event, {bubbles: true}));

}

console.log('USD typed in search');

await humanDelay(500, 800);

// 3. Находим и кликаем на опцию USD

const usdOption = currencyDropdown.querySelector('div.item[data-val="USD"]');

if(!usdOption) throw new Error('USD option not found');

// Полная эмуляция клика

for(let event of clickEvents) {

usdOption.dispatchEvent(new MouseEvent(event, {

bubbles: true,

cancelable: true,

view: window

}));

}

console.log('USD selected');

// 4. Проверяем обновление значения

await humanDelay(300, 500);

const hiddenInput = document.getElementById('CurrentCurrencyCode');

if(hiddenInput.value !== 'USD') {

hiddenInput.value = 'USD';

hiddenInput.dispatchEvent(new Event('change', {bubbles: true}));

console.log('Value force-updated');

}

// 5. Закрываем дропдаун

document.activeElement.blur();

console.log('Dropdown closed');

// Финальная проверка

setTimeout(() => {

console.log('Final currency value:', document.getElementById('CurrentCurrencyCode').value);

}, 1000);

} catch(error) {

console.error('Error:', error);

}

})();  
  
  
а вот это рабочий код, правда в отличий от остальных он не выполняется в консоли а в puppeteer