МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

РАБОТА ПРОВЕРЕНА
Рецензент, руководитель отдела ИТ
компании ООО «ПРОАНАЛИТИКС»
_____/Ю.А.Сергеев/
«_____» _____2021 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.т.н., с.н.с.
_____/Б.М.Суховилов/
«____» _____2021 г.

# Разработка программного продукта для автоматизации покупки лимитированных товаров в интернет-магазинах

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ–09.04.03.2021.404.ВКР

Руководитель, к.т.н., доцент
_____/ В.А. Конов /
«_____» _____2021 г.

Автор
студент группы ЭУ-221
_____/ А.А.Корнев /
«_____» _____2021 г.

Нормоконтролер, ст.
преподаватель
_____/ Е.Н.Горных/
«_____» _____2021г.

Челябинск 2021

# АННОТАЦИЯ

Корнев А.А.,Разработка программного продукта для автоматизации покупки лимитированных товаров в интернет-магазинах.– Челябинск: ЮУрГУ, ЭиУ-221, 89 с.,48 ил., 14 табл., библиогр. список – 9 наим., 1прил.

Разработка информационной системы, автоматизирующей покупку лимитированных товаров в интернет магазинах.

Проанализированы аналогичные существующие информационные системы, выявлены их достоинства и недостатки, обоснована актуальность выбранной темы, сформулированы задачи.

Разработана структура приложения. Разработано настольное windows-приложение, позволяющее автоматизировать и ускорить покупку лимитированных товаров в интернет магазинах, по сравнению с аналогичной покупкой таких товаров человеком. Также разработан web-интерфейс, позволяющий удаленно управлять настольным приложением, или несколькими такими приложениями одновременно.

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

В настоящее время на рынке все чаще и чаще можно встретить товары, которые вызывают повышенный спрос у потребителей, и, также, часто эти товары выпускаются ограниченным тиражом и только один раз – лимитировано.

На момент написания этой записки, такими товарами можно считать игровые приставки PlayStation, видеокарты, а также большое количество одежды и обуви брендов Adidasи Nike.

Из-за большой конкуренции на такие товары, открывается целая ниша для разработки специального программного обеспечения – ботов для покупки лимитированных товаров. Бот полностью дублирует действия человека, но осуществляет процесс покупки гораздо быстрее.

На рынке программного обеспечения уже существуют решения для покупки товаров в интернет-магазинах, но большинство из них – закрытые проекты с уже сформированной базой пользователей, приобрести которые чаще всего получится, только если использовать другое специальное ПО.

Цель дипломного проекта: разработать программу дляавтоматизации покупки лимитированных товаров в интернет-магазинах, работающую как настольное приложение, так и с возможностью управления настольным приложением через веб-интерфейс.

Задачи дипломного проекта: анализ предметной области, выбор инструментальных средств, позволяющих работать на веб-сервере и сервере баз данных, разработка настольного приложения, а также web-интерфейса для удаленного управления им.

В дипломной работе разработано настольное, а также web-приложение, позволяющееавтоматизировать процесс покупки лимитированных товаров, продаваемых в интернет-магазинах. Приложение имеет универсальный характер и может быть доработано для использования с любым интернет-магазином.

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Техническое задание

Требуется разработать настольное приложение, реализующее покупку заданных товаров на выбранных сайтах, а также web-приложение, позволяющее удаленно управлять настольным приложением.

Основные требования к программе:

- авторизованный доступ как к настольному, так и web приложению;
- хранение локально данных пользователя, требуемых для покупки товаров, таких как адреса, карты, аккаунты сайтов, а также возможность добавить неограниченное количество таких данных и выбирать нужные для конкретных задач;
- уведомления об успешных покупках в заданный пользователем discordканал;
- возможность работы программы в заданное время, а именно – запуск задач по таймеру;
- возможность удаленного управления задачами на настольном приложении.

## 1.2 Анализ существующих разработок и выбор стратегии автоматизации

В настоящее время на рынке присутствует много программных средств, для автоматизации покупки товаров в интернет магазинах,ниже приведено описание некоторых из них.

**CYBERSOLE**

Данный продукт на текущее время является одним из лидеров на рынке ботов, заполучить его за цену, указанную на сайте, практически невозможно, более того, чтобы купить этого бота, придется использовать другое специальное программное обеспечение.

Cybersole, бот, впервые дебютировавший на рынке в феврале 2017 года, взял сообщество боттинга штурмом с его многочисленными особенностями и последовательным успехом, став очень успешным в мире боттинга. С розничной ценой £300 (~ 30 тысяч рублей) Cybersole включает в себя множество различных функций, что, возможно, делает его одним из самых успешных ботов на рынке. (в настоящее время доступен только для Windows).

Один из разработчиков Cybersole поделился тем, что в прошлом году его пользователи суммарно потратили более $30 миллионов, используя их ПО.

В настоящее время Cybersole поддерживает следующие платформы/сайты электронной коммерции [1]:

- Supreme.
- Shopify (Все сайты на платформе).
- Footsites (Footlocker, Champs, Footaction, Finishline, Eastbay).
- Mesh (Footpatrol, Size?, JD Sports, The Hip Store).
- Такие сайты, как Porter, Net-A-Porter, Offspring, Lacoste и многие другие.

Функции:

1. Неограниченные задачи

Cybersole позволяет выполнять неограниченное количество

одновременных задач. Хотя большинство пользователей не смогут выполнять большое количество задач из-за ограничений сервера и прокси, однако если это сделать, Cyber легко справится с этим.

2. Управление из мобильного приложения

Одной из самых уникальных функций Cybersole является возможность оформления заказа вдали от своего компьютера. Приложение Cybersole, доступное на устройствах iOS, позволяет пользователям создавать, управлять и редактировать задачи несколькими тапами по экрану. Кроме того, пользователи могут активировать мобильные уведомления при оформлении заказа или для того, чтобы узнать о необходимости решить капчу, что тоже можно сделать в приложении.

Ка́пча (от CAPTCHA — англ. Completely Automated Public Turing test to tell Computers and Humans Apart — полностью автоматизированный публичный тест Тьюринга для различения компьютеров и людей) — компьютерный тест, используемый для того, чтобы определить, кем является пользователь системы: человеком или компьютером. Термин появился в 2000 году. Основная идея теста: предложить пользователю такую задачу, которая с лёгкостью решается человеком, но крайне сложна и трудоёмка для компьютера. По состоянию на 2013 год, каждый день пользователями по всей планете вводится примерно 320 миллионов «капчей»

3. Поддержка прокси-серверов

В рамках программного обеспечения Cyber позволяет импортировать прокси-серверы в несколько кликов или покупать пылающие незапланированные прокси-серверы Cybersole через само приложение. После того как пользователь импортировал или приобрел прокси-серверы, он может протестировать их скорость, чтобы убедиться, что все они готовы к следующему выпуску.

4. Управление адресами для доставки

Cybersole поддерживает импорт, а также экспорт учетных записей для сайтов, которым требуется имя пользователя и пароль для покупки товаров на

сайте. Вы также можете создать несколько профилей оплаты и доставки для определенных сайтов и для настройки быстрых задач.

5. Своевременныепомощь и поддержка

Независимо от того, являетесь ли вы опытным пользователем бота или любителем, Cybersole предлагает круглосуточную поддержку через несколько платформ. Поддержка доступна в любое время через Discord или через систему тикетов поддержки. Существует также множество материалов, которые пользователи могут прочитать для общих настроек и устранения неполадок.

Интерфейс бота представлен ниже (Рисунок 1).

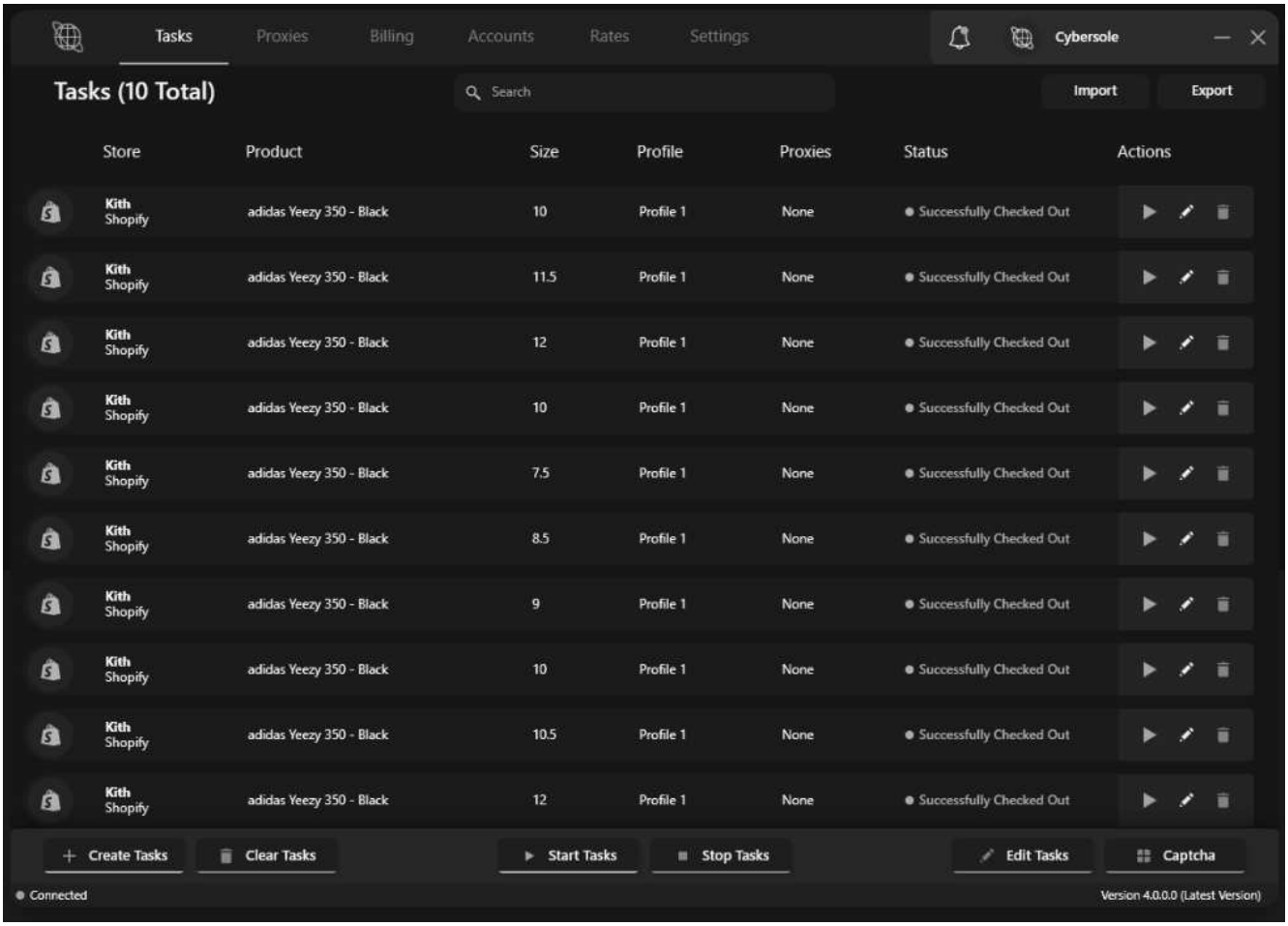



Рисунок 1 – Интерфейс ботаCybersole

На главной странице отображаются все текущие задачи, каждая из которых состоит из названия магазина, товара и его размера, профиля для заполнения адреса, прокси, а также текущего статуса задачи. Стоит сразу

отметить, что интерфейсы большинства ботов похожи, зачастую немного отличается только выбор цветовых решений и компоновка элементов управления.

**F3ATHER**

F3ather – бот, ориентированный только на один сайт– www.supremenewyork.com.

Свойства бота:

1. Совместим с Mac OS, Windows и Linux[2].
2. Веб-сайтбота является пустой страницей, поэтому необходимую информацию можно узнать только на других ресурсах.
3. Бот является лимитированным продуктом и даже перепродается, поэтому получить его не так просто.Официальная сезонная цена лицензии составляет 130 долларов, однако они перепродаются в диапазоне 450-600 долларов.
4. Ботявляется быстрой технологией и использует лучшие технологии на рынке.
5. Бот обладает одним из лучших интерфейсов, которыйинтуитивно понятен любому пользователю.
6. F3ather, широко известный как F3, – это исключительно бот для сайта Supreme, который в последнее время лидирует на рынке. Он отлично показывал себя а последних релизах, а также большую часть сезона Spring-Summer2019. F3 уже имел успехи в сезонеFall-Winter 2019, будучи одним из немногих, которые работали на самом первом релизе, а также успешно покупал товары всех последующих релизов.
7. Бот, как и большинство поддерживает прокси, одновременную покупку разных товаров, экспорт и импорт данных, а также многие другие функции.

Интерфейс бота представлен ниже (Рисунок 2).

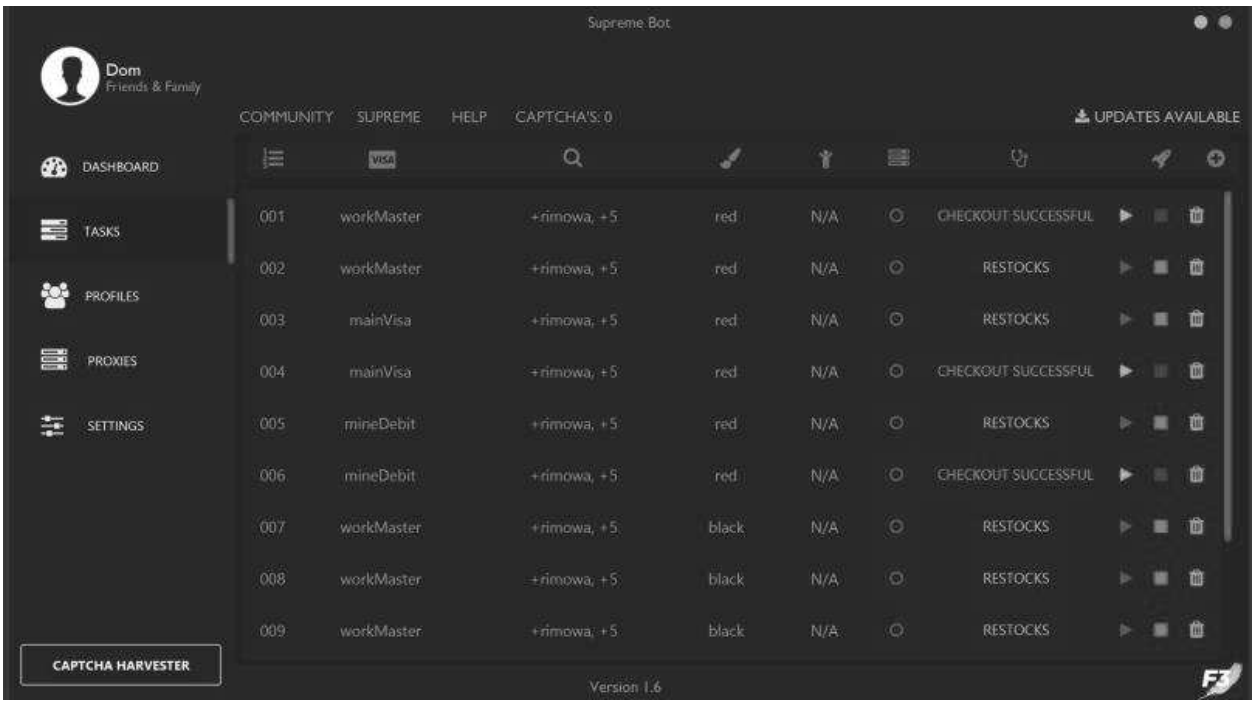



Рисунок 2 – Интерфейс ботаF3ather

**NIKE SHOE BOT**

Более 3 лет является одним из ведущих лидеров в сфере сникер-ботов.

Nike Shoe Bot имеет серьезный функционал — более 140 сайтов. Несомненно, увеличит шансы на сложных релизах, отлично проявляет себя на Shopify сайтах и Supreme. В прошлом году после обновлений успешен на многих релизах и входит в топ ботов по всему миру.В данном обзоре это единственный продукт, который можно купить в любое время на официальном сайте, а его розничная цена – 500 долларов в год [3].

Nike Shoe Bot – один из старейших ботов Nike. Он имеет простой и удобный интерфейс, который позволяет пользователям легко покупать одновременно несколько товаров. NSB поддерживает более ста розничных сайтов. Он постоянно обновляется и поставляется вместе с сервером Discord.

Nike Shoe Bot доступен как на компьютерах Mac, так и на компьютерах Windows. Пользовательский интерфейс очень прост и понятен: вкладка "Биллинг" предназначена для сохранения адресов, вкладка "задачи" - для управления задачами, а вкладка "прокси" - для списков прокси.

Самое замечательное в прокси-серверах Nike Shoe Bot заключается в том, что можно выделить список прокси-серверов для конкретной задачи, а это означает, что вы можете отслеживать релизы на разных сайтах и использовать лучшие прокси-серверы для каждого из них.

Каждая задача в Nike Shoe Bot может выполняться одновременно с любой другой, также доступен поиск товаров по ключевым словам.

Nike Shoe Bot также отлично подходит, если вас нет дома во время старта продаж. Каждую задачу можно настроить на определение время.

Бот имеет удобный инструмент для решения капчи.

В боте также есть несколько режимов, например, быстрый и медленный, полностью имитирующий человека.

Наконец, Nike Shoe Bot имеет хороший активный сервер Discord, где можно найти ответы на любые вопросы по использованию бота.

Интерфейс бота представлен ниже (Рисунок 3).

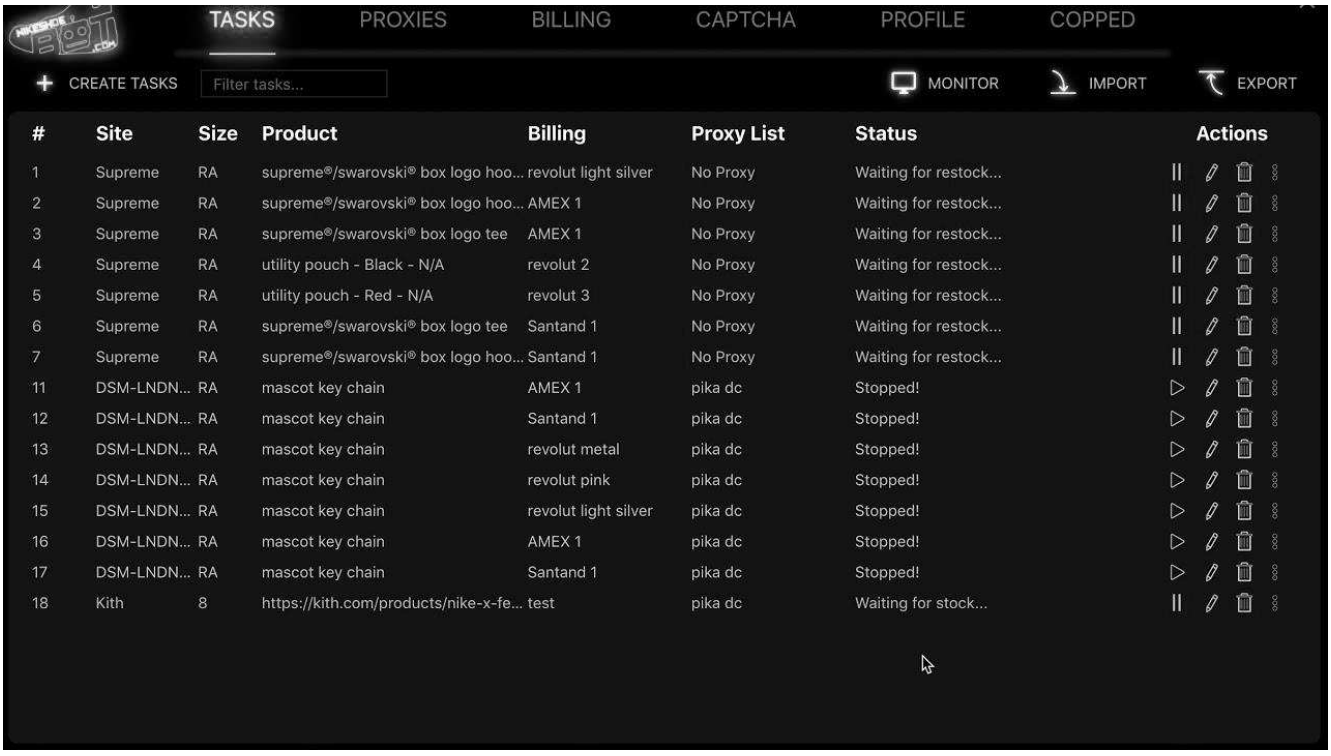



Рисунок 3 – Интерфейс ботаNike Shoe Bot

**FLARE AIO**

FlareAIOBotодин из лучших кроссовочных ботов для использования на Snipes – он также работает на сайтах Footlocker. Бот не имеет огромной

популярности, его используют очень ограниченное число людей из-за его специфичного набора сайтов.Из всех ботов, рассмотренных в этой работе, FLARE AIO – единственный не имеет графического интерфейса, а управляется через CLI - command-line interface [4].

1. Многопоточность

Flare AIO Bot был разработан не только для тех, кто носит кроссовки, он также нацелен на тех, кто их перепродает, и поэтому он должен иметь поддержку функций, представляющих интерес для таких пользователей. Многопоточность - одна из ключевых особенностей, которую ищут реселлеры в боте. Она позволяет бот одновременно выполнять несколько задач.

2. Мониторинг наличия товара

Flare AIO Bot автоматизирует задачу проверки наличия кроссовок на сайтах, которые он поддерживает. Это достигается за счет использования API на сайтах и отправки запросов через них, имитирующих браузеры. Имитация является здесь ключевым словом.

3. Решение капчи

Любой бот не может полностью остановить появление капчи на сайтах, однако не все боты могут помочь в ей решении.Flare может сделать процесс их решения быстрее, а также уменьшить их возникновение.

4. Ценообразование

Бот распродан, и цена на него прямо сейчас не отображается. Однако, можно быть уверенным, что она будет в районе 800 долларов.

Интерфейс бота представлен ниже (Рисунок 4).

### 1.3 Сравнительный анализ

В таблице 1 приведено сравнение функций существующих систем и разрабатываемой системы.

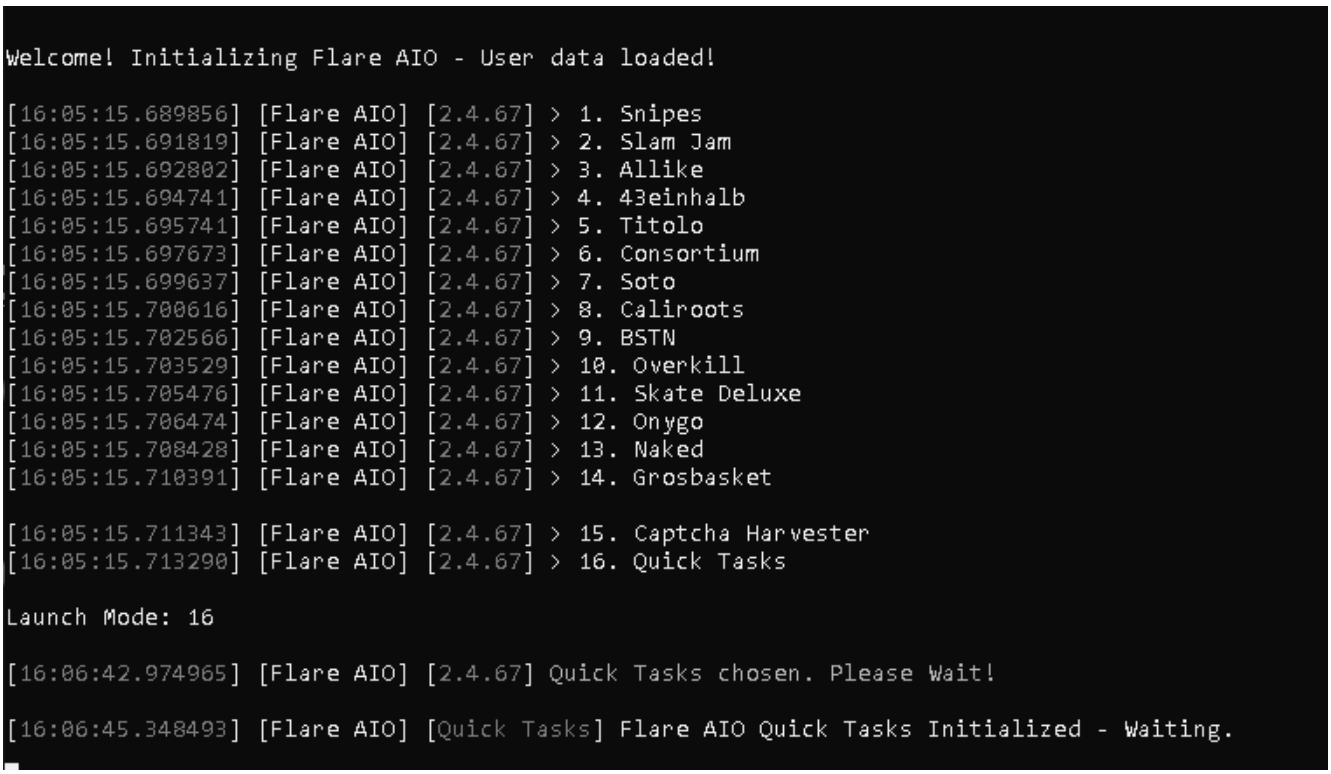



Рисунок 4 – Интерфейс ботаFlare

Таблица 1 – Сравнение возможностей систем

| **Свойство** | **Cybersole** | **F3ather** | **Nike Shoe Bot** | **Flare AIO** | **Разрабатываемая система** |
|---|---|---|---|---|---|
| Сайты | Supreme, Shopify (Всесайтынаплатформе), Footsites (Footlocker, Champs, Footaction, Finishline, Eastbay), Mesh (Footpatrol, Size?, JD Sports, The Hip Store), Такиесайты, как Porter, Net-A-Porter, Offspring, Lacoste имногиедруг ие | Supreme | Supreme, Footlocker EU, CA, AU and US EastBay, ChampsSports, Footaction, KidsFootlocker, LadyFootlocker, BSTN, CourirFR, CourirES, CourirBE, также, больше 100 сайтовнаплатформе Shopify | BSTN Overkill SJS SOTO Consortium Titolo 43einhalbC aliroots Allike | Adidas.ru Brandshop.ru Nike.com Wildberries.ru |

Продолжение таблицы 1

| | | | | | |
|---|---|---|---|---|---|
| Цена (в рублях) | 30 000 при покупке + 10 000 каждые полгода | 12 000 при покупке + 10000 каждые полгода | 45 000 в год | 60 000 в год | - |
| Возможность покупки в любое время | Нет | Нет | Да | Нет | - |
| Интерфейс | GUI | GUI | GUI | CLI | GUI |

Исходя из данных в таблице, можно сделать вывод, что ни одна из рассмотренных систем не поддерживает российские сайты, такие как adidas.ru, brandshop.ru, nike.com, wildberries.ru. Также, все продукты стоят существенных денег, но тем не менее, большинство из них нельзя купить в любое время.

### 1.4 Выбор инструментария для решения задачи

Поскольку разрабатываемая система содержит как настольное, так и web-приложение: выбраны следующие средства для разработки:

1. Язык программирования на клиенте – JavaScript + JQuery Framework + Vue.js.

2. JQueryупозволяет значительно упростить взаимодействие JavaScript и HTML. Библиотека помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими [5]. Также библиотека JQuery предоставляет удобный API для работы с AJAX.

3. С помощью Vue.js можно легко построить всю логику интерфейсов приложения, а также хранения данных на клиенте.

4. Серверный язык программирования – PHP. PHP довольно легок в написании, так как его синтаксис схож с синтаксисом C или perl. В языке PHP нет очень строгой типизации и совсем отсутствует необходимость по освобождению памяти или выделению памяти [6].

5. База данных на сервере – MySQL. Преимущество этой БД в хорошей связи с PHP. Поддержка MySQL входит в стандартную сборку PHP, и можно быть

уверенным, что проблем обращения к серверу MySQL из PHP-скриптов не будет. Также эта БД присутствует на большинстве хостингов, что делает максимально простым работу с ней[7]. Для хранения данных в настольном приложении используется файловая база данныхNeDB.

**Вывод по разделу один**

В разделе представлено техническое задание, приведены примеры существующих программ, определены возможности программ, выявлены преимущества и недостатки. По проведенному анализу были определены инструментальные средства для разработки.

# 2РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

Реализация приложения разделена на две части, настольное приложение и web-приложение. Каждая из этих частей также разделена на две части, клиентскую и серверную.

## 2.1 Архитектура программного продукта

Ниже представлена архитектура разрабатываемого продукта (Рисунок 5) и ее описание (Таблица 2, Таблица 3). Покупку на сайте совершает настольное приложение, оно может работать как самостоятельно, так и управляться извне (при выборе режима арів нем). За удаленное управление отвечает связка из web-интерфейса и сервера, который может работать по двум протоколам – restи websocket.

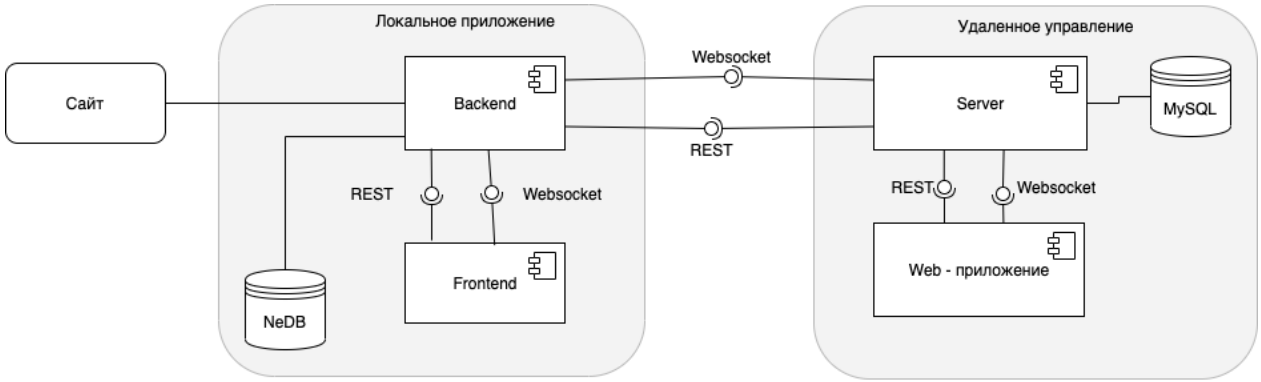



Рисунок 5– Архитектура системы

Таблица 2 – Описание компонентов

| Название | Описание |
|---|---|
| Локальное приложение - Backend | Backend локального приложения, которое обеспечивает покупку заданного товара на сайте. |
| Локальное приложение - Frontend | Интерфейс локального приложения, которое обеспечивает покупку заданного товара на сайте. |
| NeDB | Библиотека, обеспечивающая хранение данных на компьютере пользователя. |
| Сайт | Интернет-магазин, на котором происходит покупка заданного товара. |
| Server | Удаленный сервер, на котором развернуто web-приложение. |
| Web – приложение | Приложение, позволяющее удаленно управлять настольным приложением. |
| MySQL | База данных, хранящая данные, добавленные через web-приложение. |

Таблица 3 – Описание взаимодействий

| **Инициатор** | **Исполнитель** | **Назначение** | **Способ взаимодействия** |
|---|---|---|---|
| Server | Web-Приложение | Получение данных для выполнения задач, карт, аккаунтов и адресов. | REST |
| Server | Web-Приложение | Инициация запуска задач на покупку товаров. | Websocket |
| Web-Приложение | Server | Передача логов в процессе работы приложения. | Websocket |
| Server | Backend | Инициация запуска задач с удаленного приложения на локальном приложении. | Websocket |
| Backend | Server | Передача логов в процессе работы локального приложения. | Websocket |
| Backend | Server | Получение данных для задач, созданных удаленно через web-приложение. | REST |
| Backend | Frontend | Передача логов в процессе работы локального приложения. | Websocket |
| Frontend | Backend | Получение данных для задач, созданных локально через интерфейс настольного приложения. | REST |

## 2.2Алгоритм работы программного продукта

Ниже представлена диаграмма последовательности (Рисунок 6). Диаграмма последовательности (с англ. sequence diagram) — диаграмма, на которой показано взаимодействие объектов (обмен между ними сигналами и сообщениями), упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления.

Описание действий:

1 Инициирует вход – пользователь вводит свой логин и пароль от системы и нажимает кнопку «Войти»

1.1 auth(login, password) – веб приложение передает логин и пароль на сервер, где в дальнейшем произойдет их проверка.

1.1.1 check(login, password) : flag– сервер возвращает результат проверки логина и пароля, ошибку, если они не верны, либо токен доступа в обратном случае.

1.2 goToTaskPage() – клиентская часть, после получения токена переадресовывает пользователя на страницу с задачами приложения.

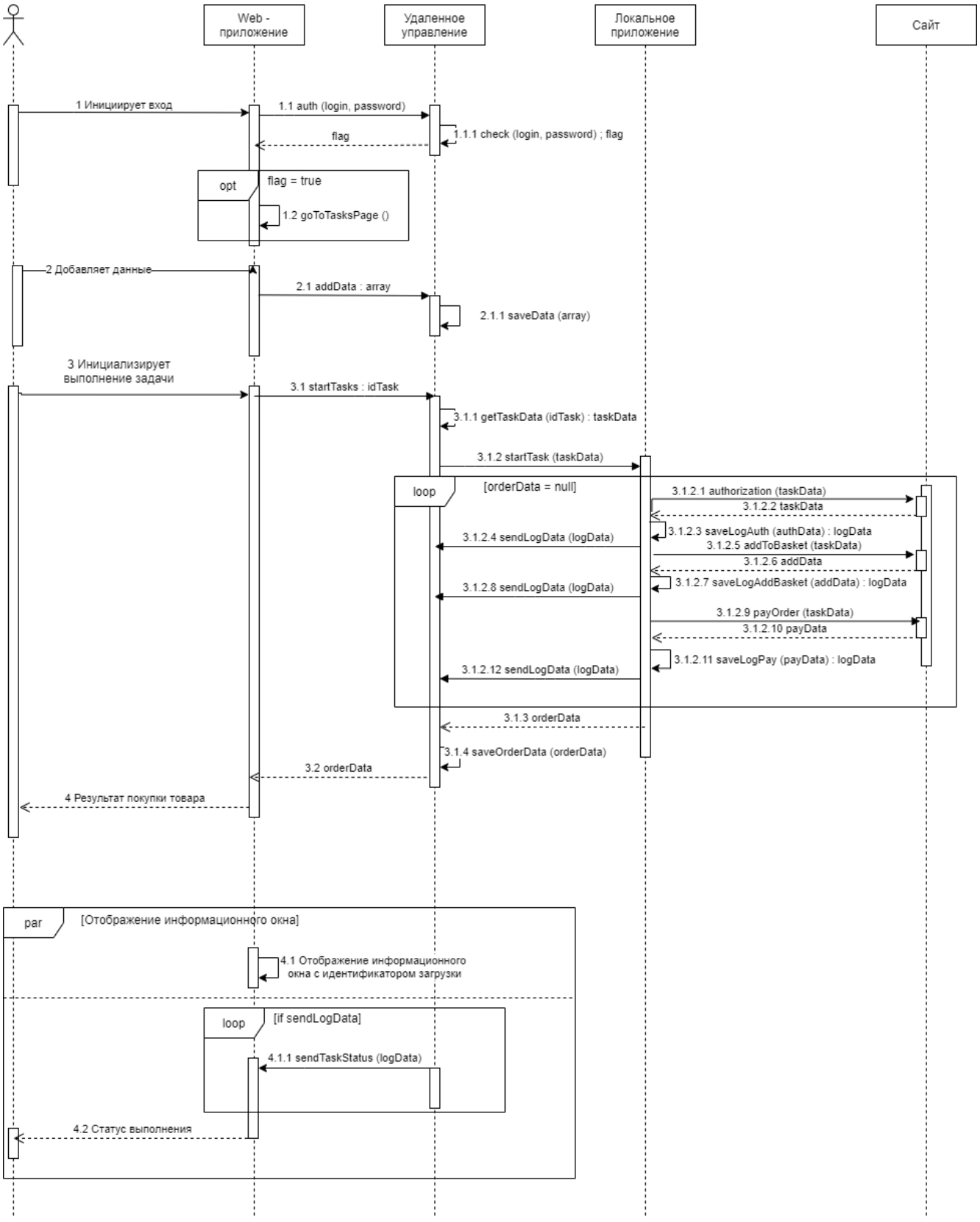



Рисунок 6 – Диаграмма последовательности

2 Добавляет данные – пользователь добавляет в приложение данные, которые в дальнейшем будут использоваться в задачах, а именно – адреса, карты, аккаунты и прокси.

2.1 addData: array– клиентская часть приложения отправляет запрос на серверную часть, запрос содержит введенные пользователем данные.

2.1.1 saveData(array) – серверная часть записывает переданные с клиентской данные в базу.

3 Инициализирует выполнение задачи – пользователь запускает задачу путем нажатия на соответствующий элемент управления на странице задач.

3.1 startTasks : idTask – клиентская часть передает серверу idзадачи, которую нужно запустить.

3.1.1 getTaskData (idTask) : taskData – серверная часть, получив idзадачи, получает из базы данных все нужные для нее параметры.

3.1.2 startTask (taskData) – сервер передает параметры для задачи в локальное приложение.

3.1.2.1 authorization (taskData) – локальное приложение, получив данные для выполнения задачи, обращается к выбранному в задаче сайту с целью прохождения процедуры авторизации.

3.1.2.2 taskData – сайт возвращает в приложение данные об авторизации, например, сессионные куки пользователя.

3.1.2.3 saveLogAuth (authData) : logData – локальное приложение генерирует логи о том, что авторизация пройдена.

3.1.2.4 sendLogData (logData) – логи о том, что авторизация пройдена передаются на сервер удаленного управления.

3.1.2.5 addToBasket (taskData) – локальное приложение делает запрос на сайт с целью добавления выбранного товара в корзину.

3.1.2.6 addData – сайт возвращает в приложение данные о добавленном товаре, например, сумму покупок в корзине и количество товаров в ней. Также эта информация может быть полезна для проверки товаров в корзине, так как у

пользователя могут быть уже добавлены в корзину товары до работы приложения, а поскольку в задаче может быть задана только покупка одного товара на один цикл работы приложения, товары, отличающиеся от выбранного нужно удалить.

3.1.2.7 saveLogAddBasket (addData) : logData – локальное приложение генерирует логи о том, что товар добавлен в корзину.

3.1.2.8 sendLogData (logData) – логи о том, что товар добавлен в корзину передаются на сервер удаленного управления.

3.1.2.9 payOrder (taskData) – локальное приложение делает запрос на сайт с целью получения ссылки на оплату товара, либо сразу делает запрос на оплату, передавая в запросе данные заказа и карты.

3.1.2.10 payData – сайт возвращает ссылку для оплаты товара, либо статус платежа.

3.1.2.11 saveLogPay (payData) : logData – локальное приложение генерирует логи о том, что товар оплачен.

3.1.2.12 sendLogData (logData) – логи о том, что товар оплачен передаются на сервер удаленного управления.

3.1.3 orderData – данные заказа передаются из локального приложения на сервер удаленного управления, откуда в дальнейшем они будут переданы на клиентскую часть.

3.1.4 saveOrderData (orderData) – данные заказа сохраняются в базу данных.

3.2 orderData – данные заказа передаются на клиентскую часть системы удаленного управления.

4 Результат покупки товара – пользователь получает в интерфейсе данные созданного программой заказа.

4.1 Отображение информационного окна с идентификатором загрузки – на клиентской стороне системы удаленного управления, у каждой задачи есть поле

для отображения текущего статуса задачи. В него записывается текущий лог в реальном времени.

4.1.1 sendTaskStatus (logData) – Приложение, получая лог, определяет, к какой задаче он относится и выводит его в соответствующий блок в списке задач.

4.2 Статус выполнения – пользователь получает текущий статус задачи в интерфейсе программы.

## 2.2Настольное приложение

### 2.2.1 Клиентская часть

Клиентская часть настольного приложения написана на стеке фреймворков Node-WebKit, Vue.js (включая VueXи VueRouter), Bootstrap.

Node-WebKit — комбинация Node.js и встроенного браузера WebKit. Код JavaScript выполняется в особом окружении, из которого есть доступ и к стандартному API браузеров, и к Node.js.

Логика интерфейсов построена на связке Vue.js, VueXи VueRouter. Из них Vue.js отвечает за отрисовку данных на страницах, VueX за хранение данных, VueRouter за маршрутизацию по страницам приложения.

Для визуального оформления использован фреймворк Bootstrap.

**Структура**

Приложения, написанные на Vue,состоят из компонентов. Компоненты — это блоки кода, подходящие для многократного использования, которые могут включать в себя и описание внешнего вида частей приложения, и реализацию возможностей проекта[8]. Каждый компонент – отдельный файл, состоящий из трех блоков: верстки, скриптов и стилей. Каждый компонент может быть использован внутри другого компонента, при этом уровень вложенности не ограничена.

Ниже приведен пример структуры компонентов клиентской части данного приложения (Рисунок 7).

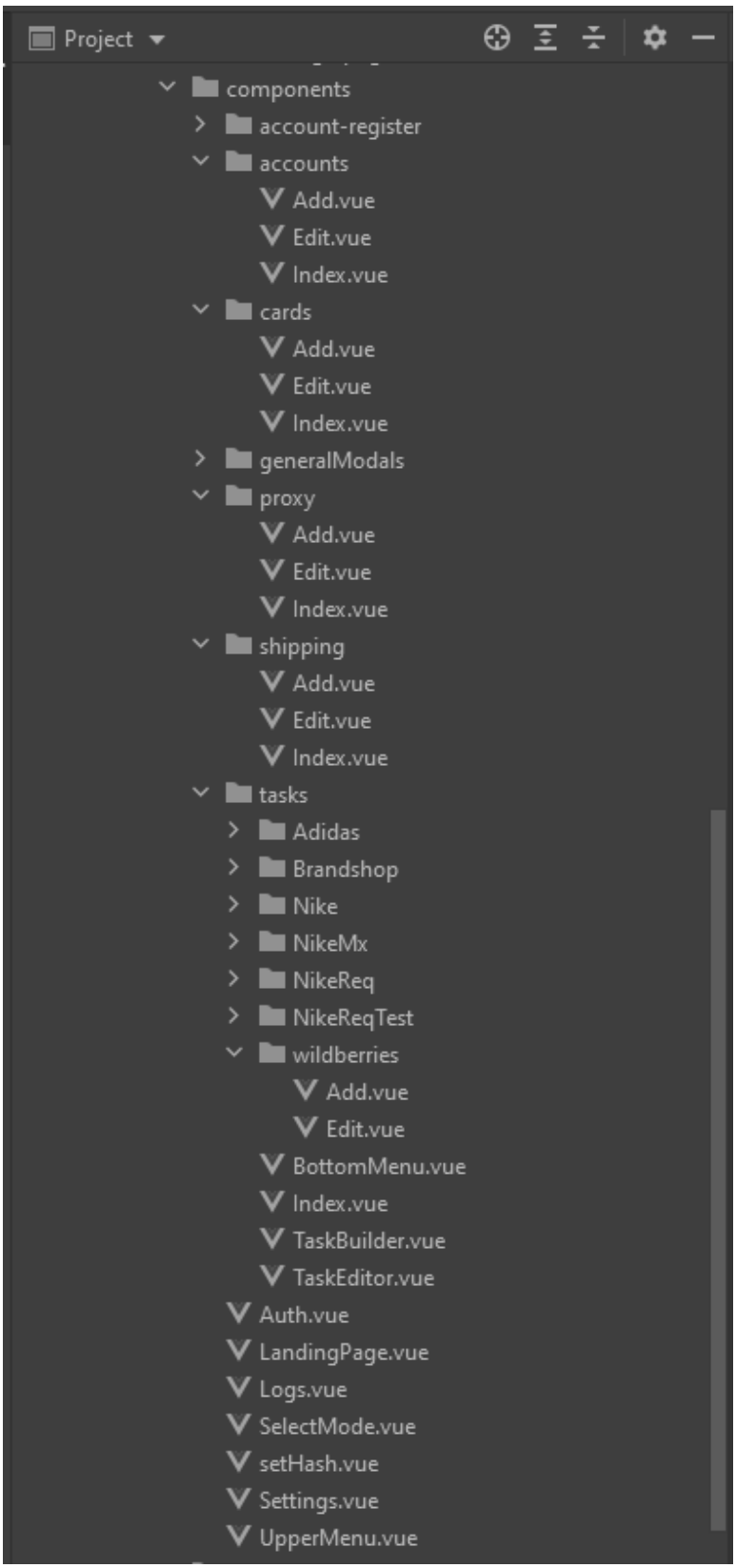



Рисунок 7 – Структура компонентов клиентской части

Однако структура не ограничивается только .vueфайлами, поскольку в проекте использованы VueXи VueRouter, для них тоже созданы директории и файлы. Ниже представлена их структура (Рисунок 8).

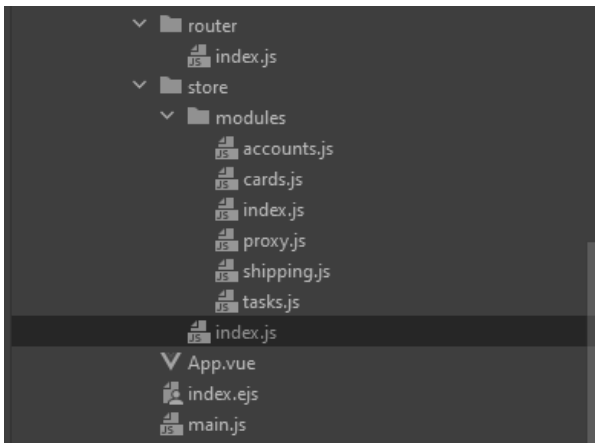



Рисунок 8 – Структура файлов VueXи VueRouter

**Страницы настольного приложения**

**Страница авторизации**

При загрузке приложения пользователь попадает на страницу авторизации, которая содержит форму с двумя полями – логином и паролем.


Рисунок 9 – Страница авторизации

**Страница выбора режима работы**

У приложения есть два режима работы – локальный, при котором все данные берутся из локального хранилища приложения, а также режим api, при котором данные берутся с api удаленного сервера. После успешной авторизации

пользователь попадает на страницу выбора режима которая состоит из двух кнопок:local–первый режим и api– второй режим (Рисунок 10).


Рисунок 10 – Страница выбора режима работы

При выборе режима api на следующем шаге пользователю будет предложено ввести hashсервера – специальный ключ, который идентифицирует компьютер, на котором установлено локальное приложение. Hashпозволяет удаленному серверу вернуть в приложение только те задачи, которые созданы для него. Пример интерфейса для ввода отображен ниже (Рисунок 11).

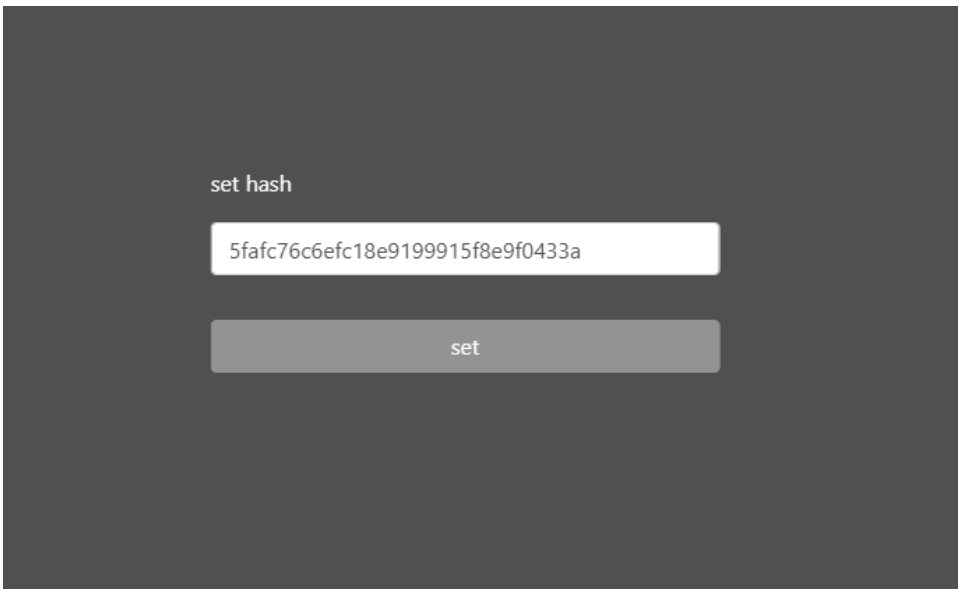



Рисунок 11 – Страница ввода ключа сервера

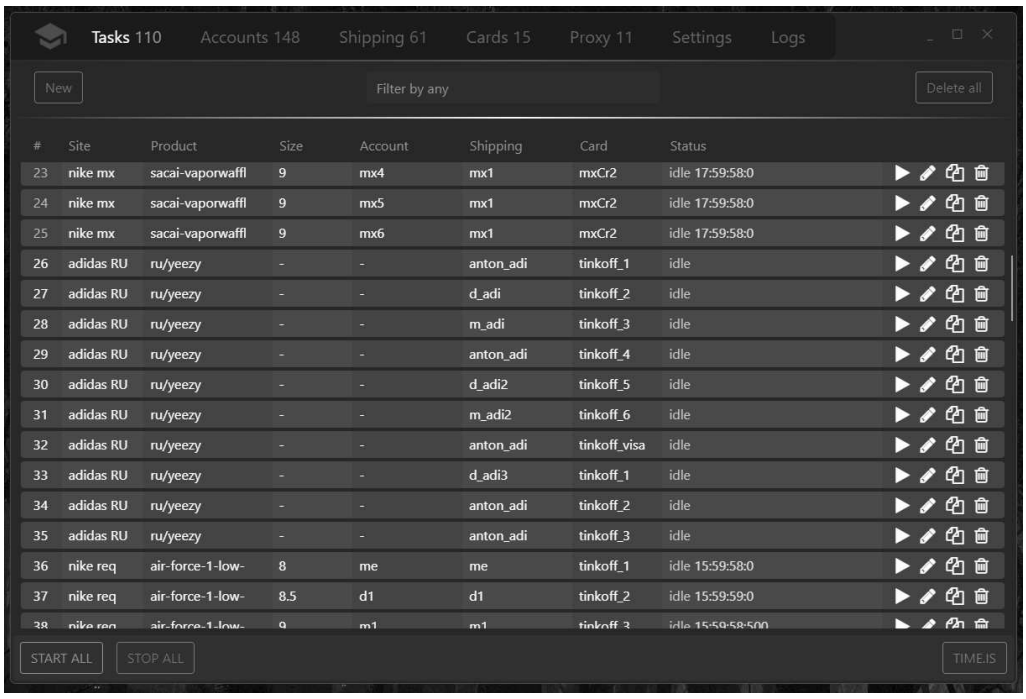
**Страница задачи**

После прохождения авторизации и выбора режима работы пользователь попадает на главную страницу приложения, на которой отражаются текущие созданные задачи, а также элементы управления ими (Рисунок 12).

Одна строка в таблице с задачами содержит номер задачи, название сайта, сокращенную ссылку на товар, размер товара, аккаунт, карту для оплаты товара, адрес для доставки, текущий статус задачи и кнопки для управления задачей – старт/стоп, редактирование, копирование, удаление. Копирование позволяет значительно упросить взаимодействие с задачами, благодаря тому, что не нужно постоянно вводить все настройки заново, а можно просто скопировать уже готовую задачу и поменять в ней только какую-то часть настроек.

Также в верхней части приложения размещено меню для доступа к остальным страницам приложения – аккаунтам, картам, адресам, прокси, настройкам и логам.

В нижней части страницы расположены кнопки для запуска и остановки всех задач одновременно, а также кнопка для открытия дополнительного окна с точным временем, которое часто нужно во время старта продаж товаров (Рисунок 13).




| # | Site | Product | Size | Account | Shipping | Card | Status |
|---|---|---|---|---|---|---|---|
| 23 | nike mx | sacai-vaporwaffl | 9 | mx4 | mx1 | mxCr2 | idle 17:59:58:0 |
| 24 | nike mx | sacai-vaporwaffl | 9 | mx5 | mx1 | mxCr2 | idle 17:59:58:0 |
| 25 | nike mx | sacai-vaporwaffl | 9 | mx6 | mx1 | mxCr2 | idle 17:59:58:0 |
| 26 | adidas RU | ru/yeezy | - | - | anton_adi | tinkoff_1 | idle |
| 27 | adidas RU | ru/yeezy | - | - | d_adi | tinkoff_2 | idle |
| 28 | adidas RU | ru/yeezy | - | - | m_adi | tinkoff_3 | idle |
| 29 | adidas RU | ru/yeezy | - | - | anton_adi | tinkoff_4 | idle |
| 30 | adidas RU | ru/yeezy | - | - | d_adi2 | tinkoff_5 | idle |
| 31 | adidas RU | ru/yeezy | - | - | m_adi2 | tinkoff_6 | idle |
| 32 | adidas RU | ru/yeezy | - | - | anton_adi | tinkoff_visa | idle |
| 33 | adidas RU | ru/yeezy | - | - | d_adi3 | tinkoff_1 | idle |
| 34 | adidas RU | ru/yeezy | - | - | anton_adi | tinkoff_2 | idle |
| 35 | adidas RU | ru/yeezy | - | - | anton_adi | tinkoff_3 | idle |
| 36 | nike req | air-force-1-low- | 8 | me | me | tinkoff_1 | idle 15:59:58:0 |
| 37 | nike req | air-force-1-low- | 8.5 | d1 | d1 | tinkoff_2 | idle 15:59:59:0 |
| 38 | nike req | air-force-1-low- | 9 | m1 | m1 | tinkoff_3 | idle 15:59:58:500 |

Рисунок 12 – Страница задачи

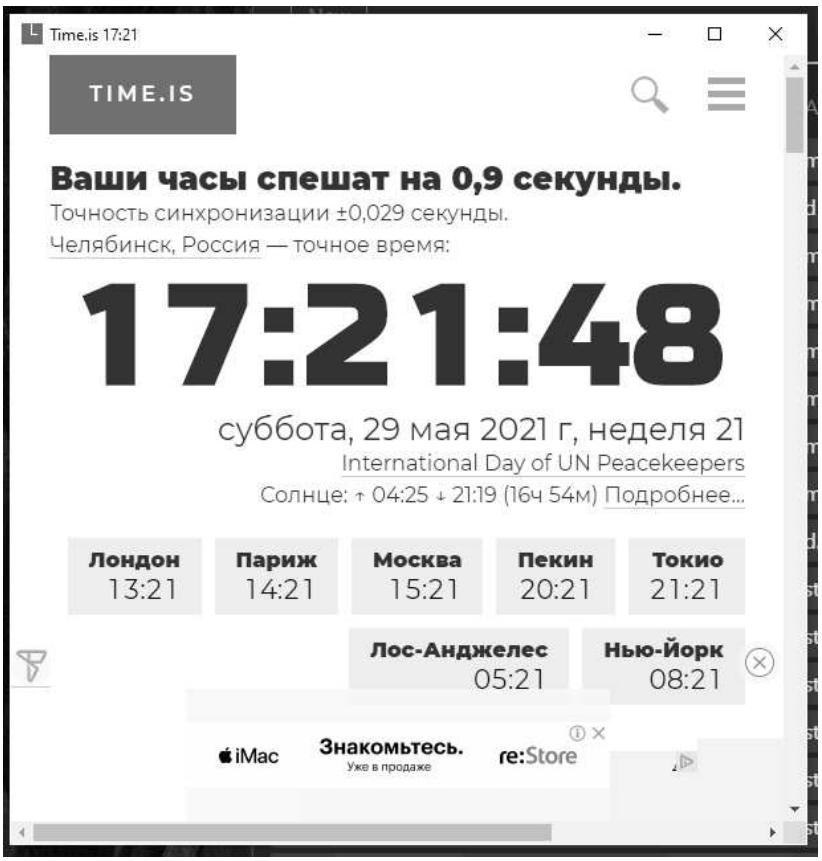



Рисунок 13 – Точное время в программе

Для создания задачи нужно нажать на кнопку «New», в верхней части страницы, далее откроется окно (Рисунок 14). При создании задачи можно выбрать сайт, для которого она создается, ссылку на товар, размер товара, аккаунт сайта, адрес, на который приложение оформит заказ, карту, которой он будет оплачен, количество задач, которое будет создано с введенными настройками, включить/отключить поддержку прокси для задачи, а также выбрать время старта задачи и время отправки заказа. Все настройки, описанные выше формируются динамически в зависимости от выбранного на первом этапе сайта.

**Страница аккаунты**

Данная страница отображает аккаунты, с помощью которых программа может авторизоваться на выбранных сайтах. В таблице на странице отображены следующие параметры – использован ли аккаунт в задаче, название аккаунта, e-mail, а также кнопки для управления аккаунтами – редактирование, авторизация

на сайте, удаление. При нажатии на кнопку для авторизации на сайте, приложение откроет в браузере сайт, перейдет на страницу авторизации и авторизуется на нем (Рисунок 15).

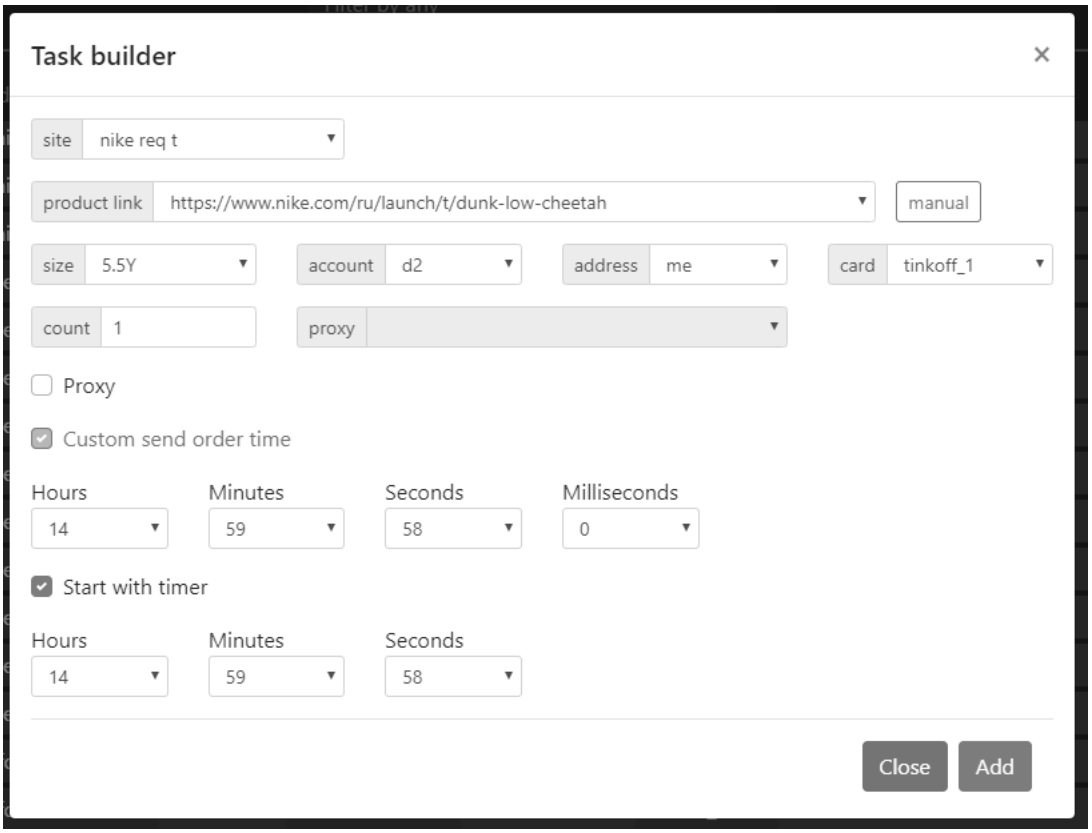



Рисунок 14 – Форма добавления задач

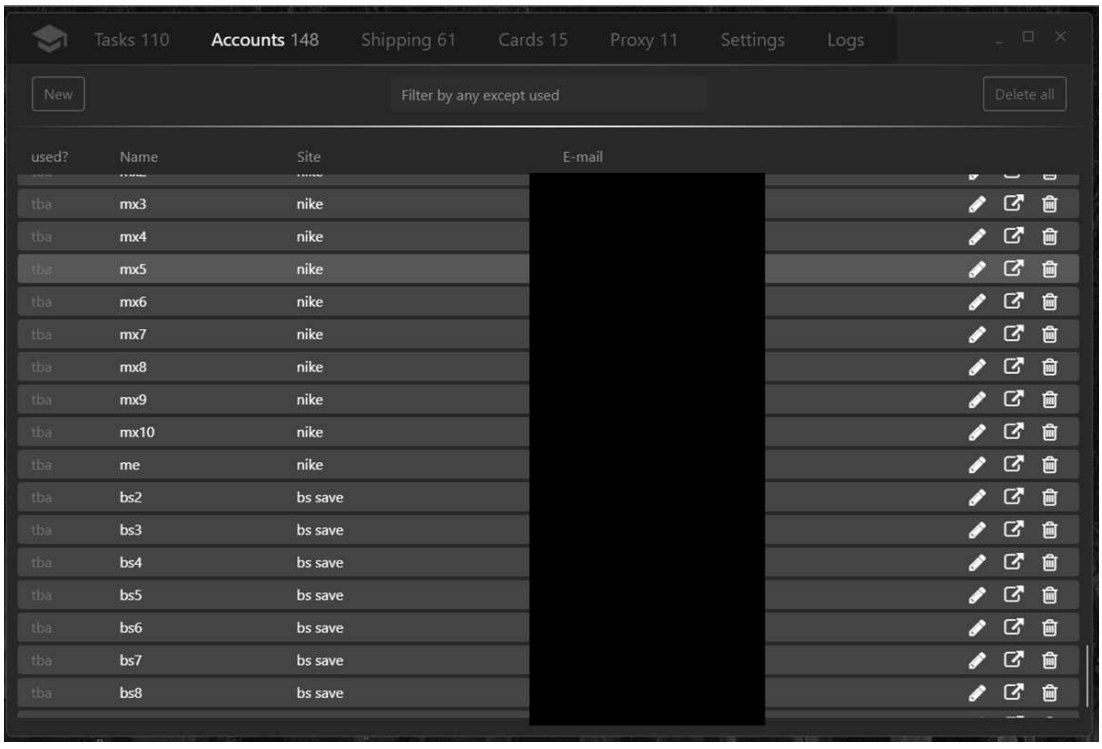



Рисунок 15 – Страница «Аккаунты»

Для создания аккаунта нужно нажать на кнопку «New», в верхней части страницы, далее нажатии откроется окно (Рисунок 16). При создании задачи можно выбрать сайт, для которого добавляется аккаунт, название аккаунта, e-mail, пароль. Для каждого аккаунта можно выбрать адрес и карту, которые будут использоваться по умолчанию, то есть – при создании задачи, после выбора аккаунта, при условии, что у него заданы адрес или карта по умолчанию, они автоматически выберутся в окне создания задачи.

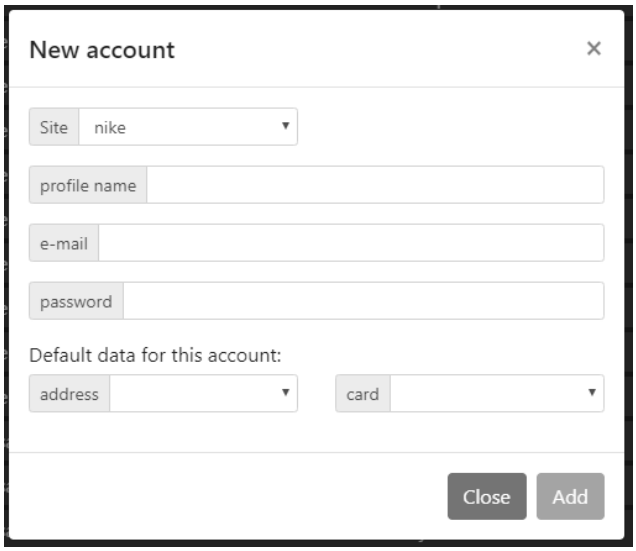



Рисунок 16 – Форма добавления нового аккаунта

**Страница адреса**

Данная страница отображает адреса, которые программа будет вводить на сайтах на этапе оформления заказа (Рисунок 17). В таблице на странице отображены следующие параметры – название адреса, имя получателя, фамилия получателя, город получателя, адрес получателя, а также кнопки для управления адресом – редактирование, копирование, удаление.

Для создания нового адреса нужно нажать на кнопку «New», в верхней части страницы, далее откроется окно (Рисунок 17). При создании адреса можно добавить его название, фамилию, имя, отчество, индекс, город, улицу, номер дома, номер квартиры, номер телефона, e-mail, а также данные паспорта и инн. В данной форме нет обязательных полей, так как каждый сайт подразумевает разный набор данных для оформления заказа, а для каких-то сайтов адрес может не понадобиться вообще.

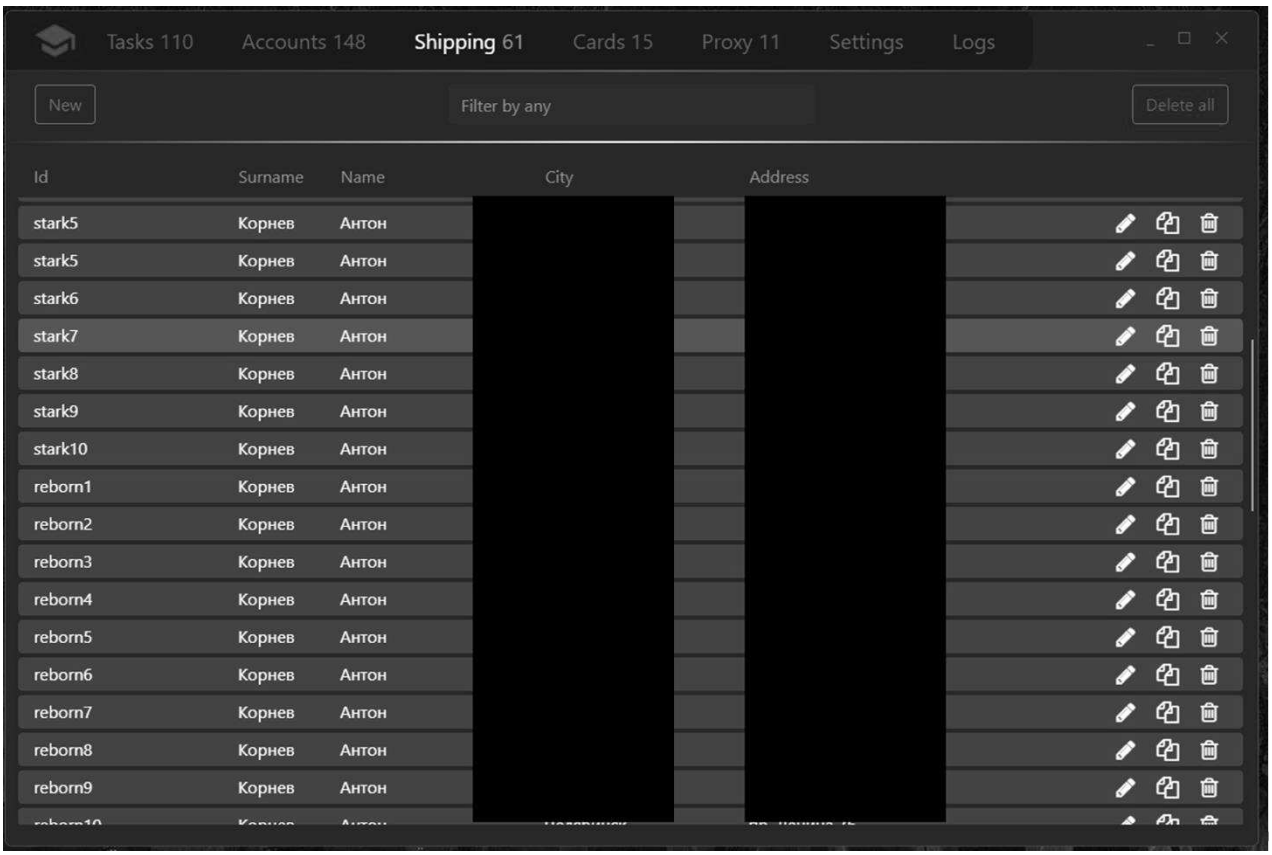



Рисунок 17 – Страница «Адреса»


Рисунок 18 – Форма добавления адреса

**Страница карты**

Данная страница отображает карты, которые программа будет вводить на сайтах на этапе оформления заказа (Рисунок 19). В таблице на странице отображены следующие параметры – название карты, номер, срок действия, владелец карты, а также кнопки для управления картой – редактирование, и удаление.

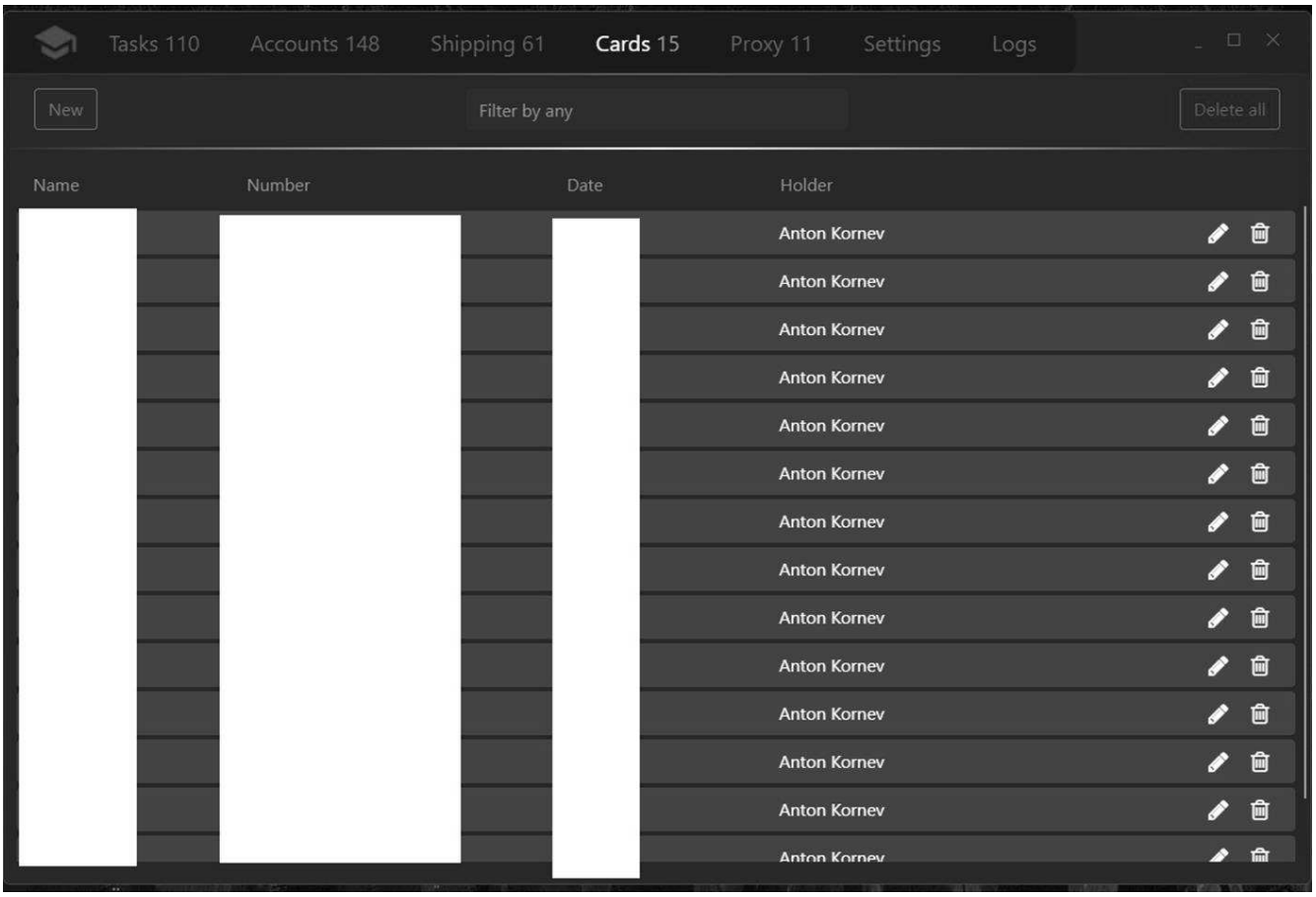



Рисунок 19 – Страница «Карты»

Добавление карты происходит идентично с предыдущими страницами программы.

**Страница прокси**

Для некоторых сайтов может потребоваться использование прокси, на этой странице можно их добавить/удалить/изменить (Рисунок 20).

Для добавления прокси нужно нажать на кнопку «New», в верхней части страницы, далее откроется окно (Рисунок 21).

Стоит отметить, что прокси провайдеры зачастую предоставляют прокси сразу большим списком единого формата «ip:port[:login:password]», где каждая новая строка – новый адрес для подключения к прокси серверу. Поэтому имеет смысл сделать в программе добавление сразу нескольких прокси из списка за одну итерацию, а не за количество итераций, равное количеству прокси. После ввода в поле одной или нескольких строк, кнопка для добавления будет разблокирована, а после её нажатия все строки будут переданы на серверную часть приложения, где уже будут обработаны построчно и записаны в хранилище данных.

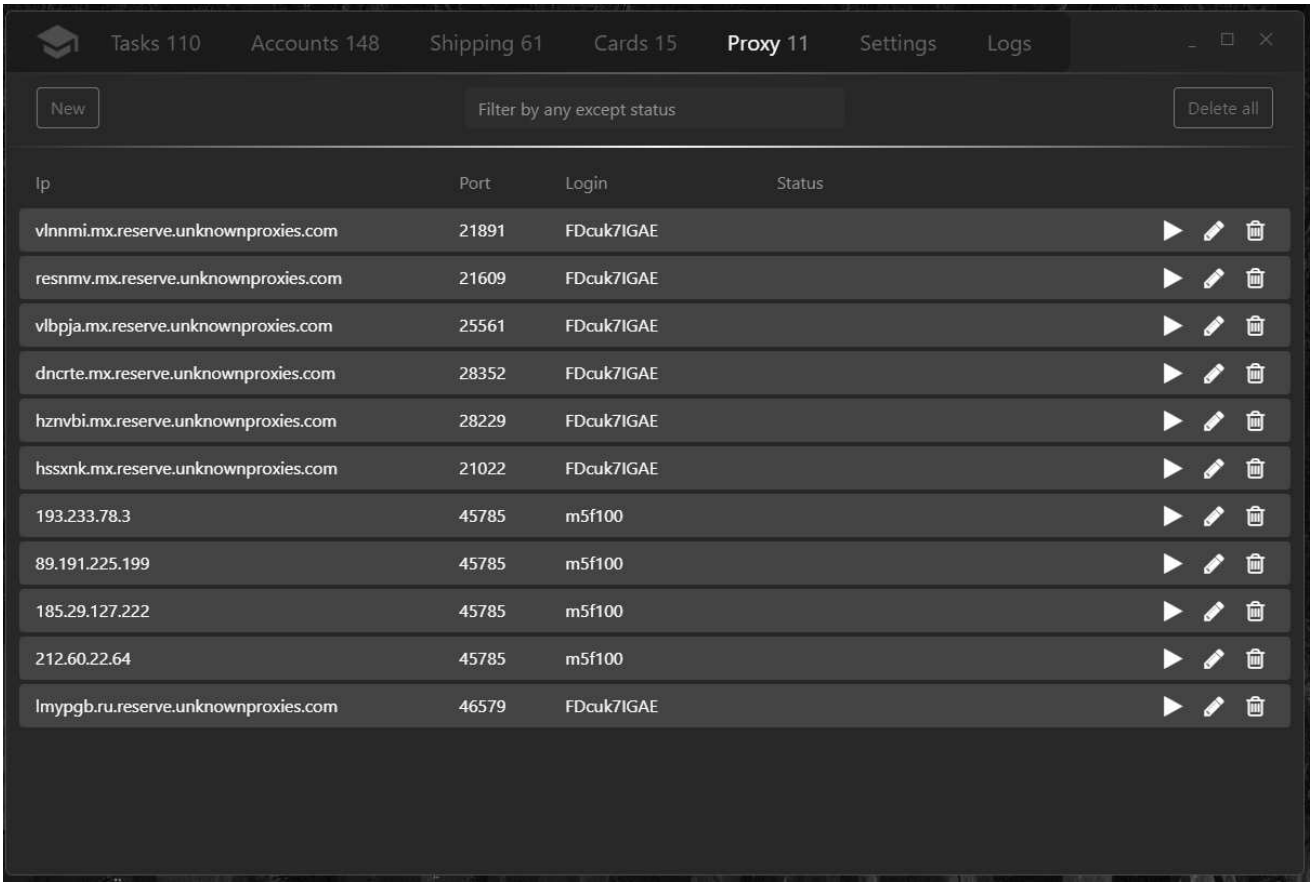



Рисунок 20 – Страница «Прокси»


Рисунок 21 – Форма добавления прокси

Для редактирования прокси не подойдет способ, описанный выше, потому что редактирование в данном случае подразумевает изменение одной записи, а не нескольких сразу, поэтому окно редактирования выглядит следующим образом:


Рисунок 22 – Форма редактирования прокси

**Страница настройки**

Программа для работы требует некоторые настройки, которые вынесены в отдельную страницу, т.к. используются повсеместно в программе (Рисунок 23).

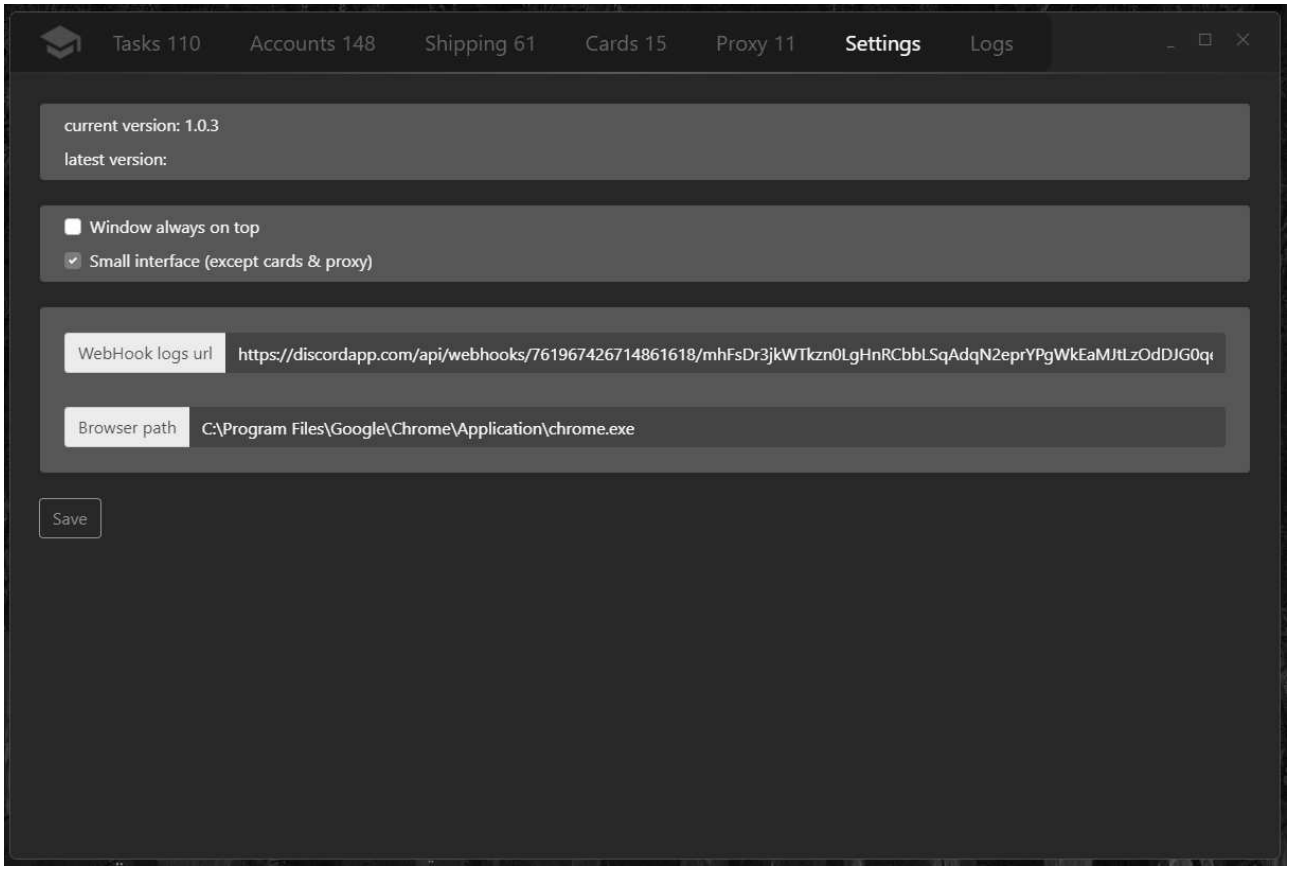



Рисунок 23 – Страница «Настройки»

На странице отображены текущая и последняя версии приложения, настройки размера интерфейса, настройка закрепления окна приложения поверх всех окон компьютера, а также ссылка для отправки уведомлений в discordи пусть к браузеру на компьютере.

**Страница логи**

Каждая задача генерирует свои логи, которые отображаются на странице задачи, но там можно увидеть только последний лог, а историю увидеть нельзя. Для этого создана данная страница, все логи, которые когда-либо были отражены в задаче, попадают сюда и хранятся в интерфейсе до завершения сессии работы с программой, либо до их очистки пользователем (Рисунок 24).

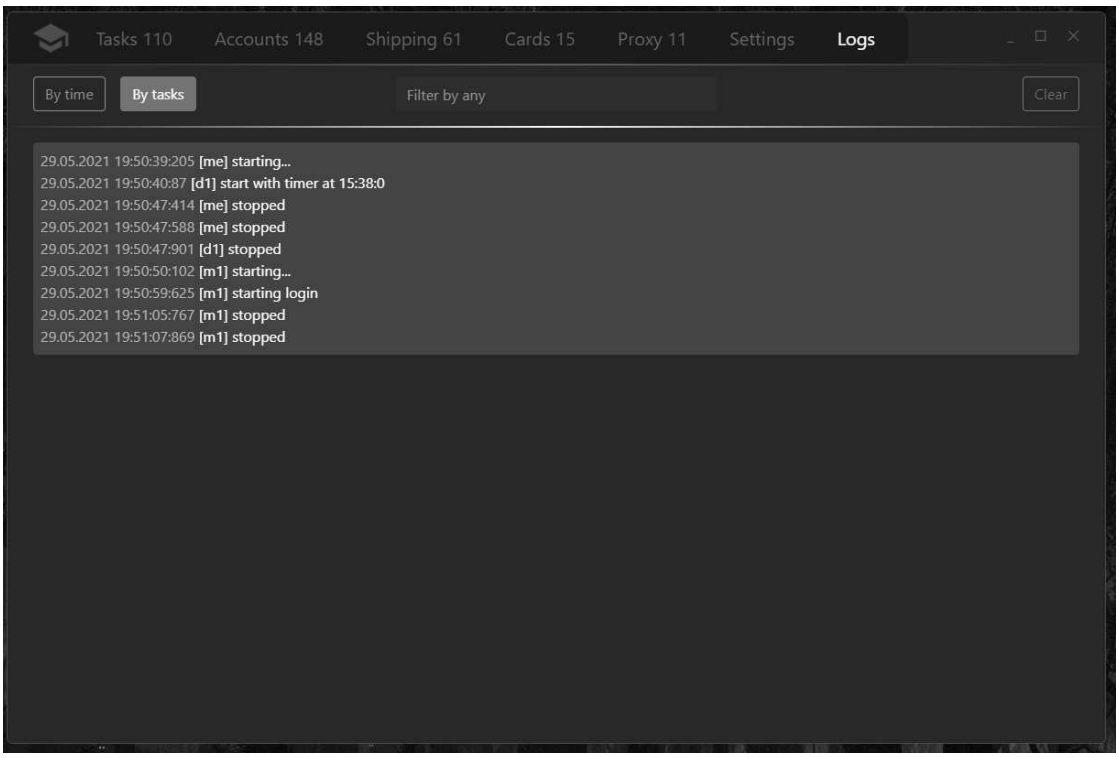



Рисунок 24 – Страница «Логи»

Запись лога состоит из времени, названия аккаунта задачи, а также текущего значения лога. Доступно два варианта отображения, с сортировкой по времени и сортировкой по задачам, на рисунке выше отображен пример с сортировкой по времени, когда все логи со всех задач идут списком по времени. На ниже приведен пример сортировки логов по задачам, где логи каждой задачи группируются в блоки, внутри которых они уже отсортированы по времени (Рисунок 25).

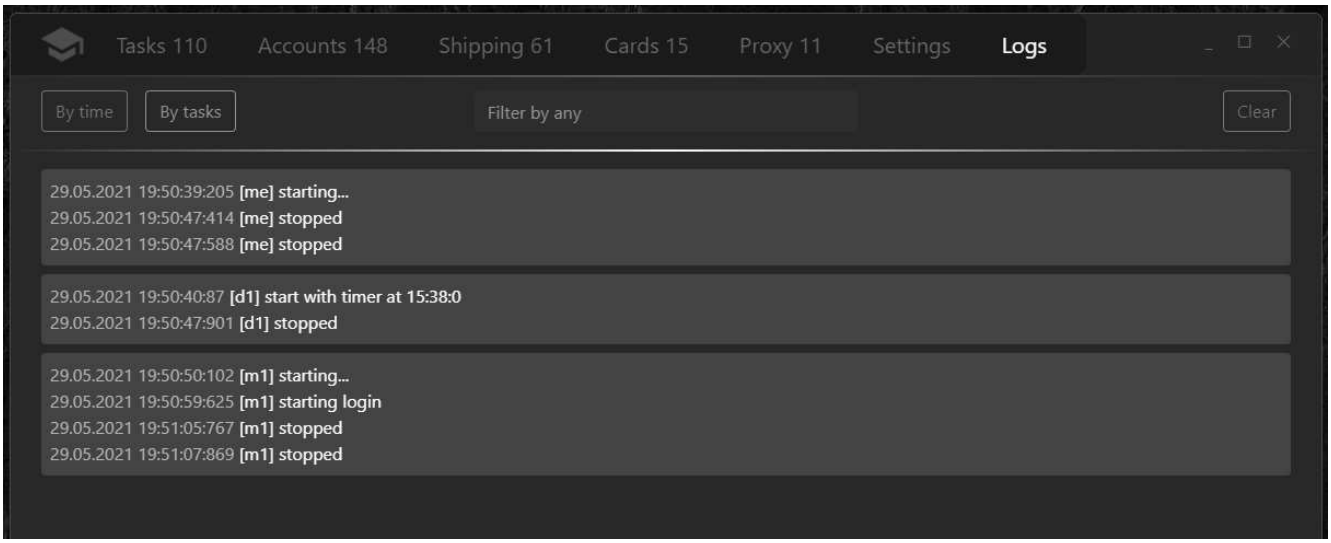



Рисунок 25 – Страница «Логи»

### 2.2.2 Серверная часть

**Архитектура**

Node-WebKitпредоставляет возможность использовать Node.js в качестве серверной части приложения.

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel, low.js и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Поскольку используется клиент-серверная архитектура, требовалось реализовать RESTapi на сервере настольного приложения,для этого выбран фреймворк Express.js, однако это apiдолжно реализовывать не только получение данных по запросу клиентской части, но и мгновенное обновление данных на клиенте, а именно – статусы задач нужно обновлять мгновенно, без дополнительных запросов с клиента. Для это решено использовать механизм вебсокетов, который также поддерживается фреймворком Express.js.

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

WebSocket разработан для воплощения в веб-браузерах и веб-серверах, но он может быть использован для любого клиентского или серверного приложения. Протокол WebSocket — это независимый протокол, основанный на протоколе

TCP. Он делает возможным более тесное взаимодействие между браузером и веб-сайтом, способствуя распространению интерактивного содержимого и созданию приложений реального времени.

Для некоторых задач требуется браузер, которым можно было бы управлять удаленно, для этих целей выбрана библиотека Puppeteer.

Библиотека puppeteer для Node.js позволяет автоматизировать работу с браузером Google Chrome. В частности, с помощью puppeteer можно создавать программы для автоматического сбора данных с веб-сайтов, так называемые веб-скраперы, имитирующие действия обычного пользователя. В подобных сценариях может применяться браузер без пользовательского интерфейса, так называемый «Headless Chrome». Используя puppeteer, можно управлять и браузером, который запущен в обычном режиме, что особенно полезно при отладке программ.

**Структура**

Структура файлов на сервере разделена на несколько логических частей – непосредственно классы, которые отвечают за реализацию покупки товаров на сайтах, модули, отвечающие за связь с клиентом средствами RESTapi, модули для работы с базой данных, сервисные модули для работы дополнительного функционала приложения, таких как, например, отправка сообщений в discord.

Ниже отображена структура файлов в серверной части клиентского приложения (Рисунок 26).

Файл bg.jspреализует входную точку в приложение, при его загрузке запускаются RESTapiи Websocketсервер. Кодизэтогофайлапредставленвлистингениже.

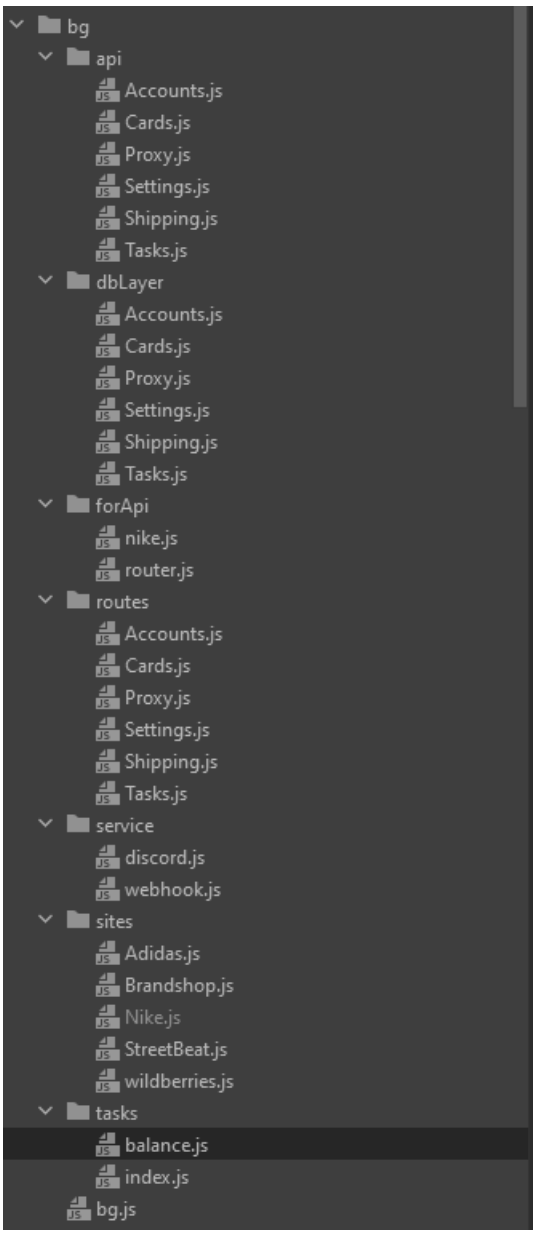



Рисунок 26 – Структура файлов web-приложения

Листинг 1

```
import Balance from './tasks/balance'
import ApiRouter from './forApi/router'

let myDiscord = require('./service/discord');

const express = require('express')
const app = express()
const port = 3005

let expressWs = require('express-ws')(app)

app.use(express.json())

app.use('/accounts', require('./routes/Accounts'));
app.use('/shipping', require('./routes/Shipping'));
app.use('/settings', require('./routes/Settings'));
app.use('/proxy', require('./routes/Proxy'));
```

Окончаниелистинга 1

```
app.use('/cards', require('./routes/Cards'));
app.use('/tasks', require('./routes/Tasks'));

myDiscord.showMeInDiscordApp();

app.ws('/', function(ws, req, next) {
  ws.on('message', async function(data) {
       let action = JSON.parse(data);

       //api
       if (action.mode === 'fromDashboard'){
         let router = new ApiRouter(action.type, action.data, ws);
         await router.selectAction();
       }
       //local
       else {
         let balance = new Balance(action.type, action.taskId, ws);
         await balance.selectAction();
       }

  })
})
app.listen(port, () => {})
```

В папке tasksнаходятся два файла, их задачу и содержание ниже рассмотрим подробнее.

Файл balance.jsотвечает за запуск функций, соответствующих выбранному в задаче сайту, структура представлена ниже (Рисунок 27).

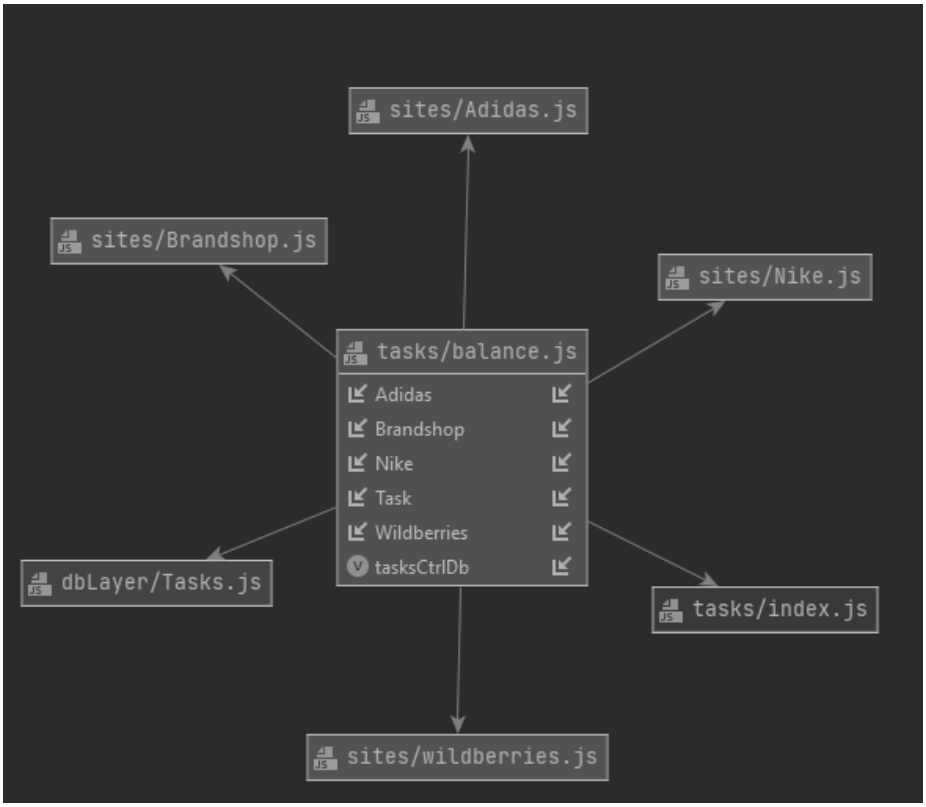



Рисунок 27 – Структура классов

Получая на входе значение действия и id сайта, он запустит нужную функцию. Кодэтогофайлаприведенвлистингениже.

Листинг 2

```
import Task from "./index";

let tasksCtrlDb = require('../dbLayer/Tasks');

import Brandshop from "../sites/Brandshop";
import Adidas from "../sites/Adidas";
import Nike from "../sites/Nike";
import Wildberries from "../sites/Wildberries";

let Tasks = [];

class Balance{

  constructor(actionType, taskId, ws) {

    this.actionType = actionType;
    this.taskId = taskId;
    this.ws = ws;
  }

  async selectAction(){
    let Ex = this;
switch (this.actionType){



     case "start": {

        let site = await tasksCtrlDb.getTaskSite(this.taskId);
        await Ex.start(site, Ex.taskId, Ex.ws);
        break;
      }
      case "stop": {

        Tasks.forEach(async function(item, i, arr) {
          if(await item.getTaskId() === Ex.taskId){
            await item.stopOne();
          }
        });
        break;
      }
      case "stopAll": {

        Tasks.forEach(async function(item, i, arr) {
          await item.stopOne();
        });
        break;
      }
      case "startAll": {

        let TasksList = await tasksCtrlDb.get();
        for (let i = 0; i < TasksList.length; i++) {
          Ex.start(TasksList[i].data.site, TasksList[i]._id, Ex.ws);
        }
        break;
      }
```

```
  }
};

async start(site, taskId, ws){

  switch (site) {

    case 1:{

      let task = await new Nike(taskId, ws);
      await Tasks.push(task);
      await task.start(site);
      break;
    }
    case 2:{

      let task = await new Nike(taskId, ws);
      await Tasks.push(task);
      await task.start(site);
      break;
    }
    case 3:{

      let task = await new Nike(taskId, ws);
      await Tasks.push(task);
      await task.start(site);
      break;
    }

    case 55:{

      let task = await new Nike(taskId, ws);
      await Tasks.push(task);
      await task.start(site);
      break;
    }

    case 4:{

      let task = await new Adidas(taskId, ws);
      await Tasks.push(task);
      await task.work();
      break;
    }
    case 5:{

      let task = await new Brandshop(taskId, ws);
      await Tasks.push(task);
      await task.safe();
      break;
    }
    case 6:{

      let task = await new Brandshop(taskId, ws);
      await Tasks.push(task);
      await task.Fast();
      break;
    }
    case 8:{
```

Окончаниелистинга 2

```
            let task = await new Wildberries(taskId, ws);
            await Tasks.push(task);
            await task.work();
            break;
          }

        }
      }
    }

export default Balance
```

Файл index.jscодержит базовый класс Task, который является родительским для всех остальных классов, реализующих задачи, структура представлена ниже (Рисунок 28). Класс связан с файлами, реализующими работу с базой данных, а также с такими библиотеками, как fs, pathи jquery.

fs – стандартный модуль Node.js, дающий разработчику средства для работы с файловой системой.

path – модуль Node.js, является встроенным и предоставляет набор функций для работы с путями в файловой системе.

jquery – набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX.

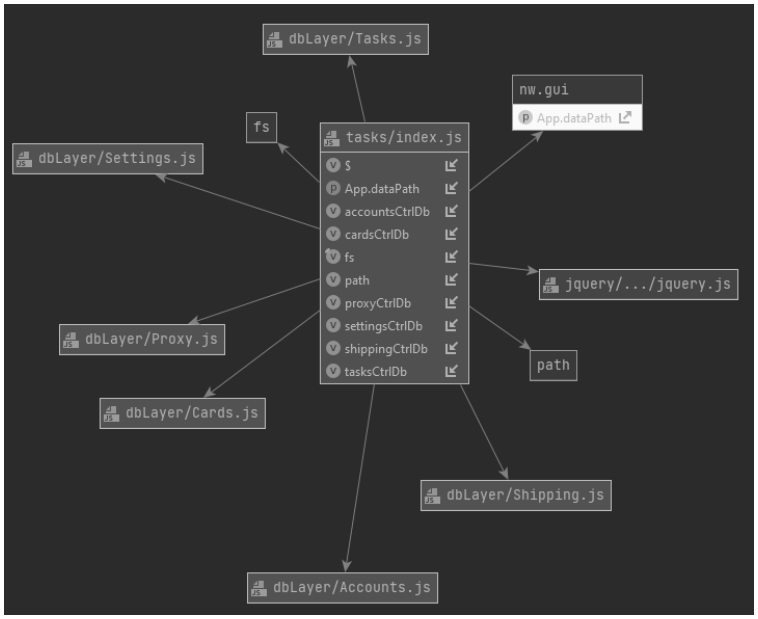



Рисунок 28 – Структура файлов и библиотек

Код файла приведен в листинге ниже.

Листинг 3

```
const fs = require('fs');
let path = require('path')
let $ = require('jquery');

let settingsCtrlDb = require('../dbLayer/Settings');
let accountsCtrlDb = require('../dbLayer/Accounts');
let shippingCtrlDb = require('../dbLayer/Shipping');
let proxyCtrlDb = require('../dbLayer/Proxy');
let cardsCtrlDb = require('../dbLayer/Cards');
let tasksCtrlDb = require('../dbLayer/Tasks');

Date.prototype.ddmmyyyy_hhmmssms = function() {
  var mm = this.getMonth() + 1;
  var dd = this.getDate();
  var hh = this.getHours();
  var min = this.getMinutes();
  var sec = this.getSeconds();
  var milisec = this.getMilliseconds();

  return [(dd>9 ? '' : '0') + dd,
      (mm>9 ? '' : '0') + mm,
this.getFullYear()].join('.')
    + " " +
    [ (hh > 9 ? '' : '0') + hh,
      (min > 9 ? '' : '0') + min,
      (sec > 9 ? '' : '0') + sec,
      milisec,
    ].join(':');
};

Date.prototype.hhmmssms = function() {
  var hh = this.getHours();
  var min = this.getMinutes();
  var sec = this.getSeconds();
  var milisec = this.getMilliseconds();

  return [(hh > 9 ? '' : '0') + hh,
      (min > 9 ? '' : '0') + min,
      (sec > 9 ? '' : '0') + sec,
      milisec,
    ].join(':');
};

class Task {

  constructor(id, ws){

    let p = new $.Deferred();

    (async () => {

      this.stopped = false;

      this.taskId = id;
      this.theWs = ws;
      this.logsWebhookUrl = null;
      this.browser = null;
      this.shedule = null;
```

```
this.accountName = null;

this.colors = {
        red: "#f63434",
        green: "#04cba1",
        blue: "#6da5fd",
        yellow: "#fdbf08",
      }

      this.winnersLetters = [
        'WINNER',
        'COPPED',
        'SAVED PAIR FOR YOU',
        'SUCCESS',
        'GOOD DAY FOR COP',
        'CONGRATS',
      ]

      if (!fs.existsSync(path.join(require('nw.gui').App.dataPath, 'magisters-
local-data/logs/'))){

        fs.mkdirSync(path.join(require('nw.gui').App.dataPath, 'magisters-local-
data/logs/'));
      }

      this.taskArray = await tasksCtrlDb.getOne(this.taskId);

      if(this.taskArray.data.account) this.accountArray = await
accountsCtrlDb.getOne(this.taskArray.data.account);
      if(this.taskArray.data.shipping) this.shippingArray = await
shippingCtrlDb.getOne(this.taskArray.data.shipping);
      if(this.taskArray.data.card) this.cardsArray = await
cardsCtrlDb.getOne(this.taskArray.data.card);
      if(this.taskArray.data.proxy.id) this.proxyArray = await
proxyCtrlDb.getOne(this.taskArray.data.proxy.id);

      this.logsWebhookUrl = await settingsCtrlDb.getWebhookUrl();
      this.browserPath = await settingsCtrlDb.getBrowserPath();

      this.accountName = this.accountArray ? this.accountArray.data.name : '-';

      p.resolve(this);

      })();

    return p.promise();
  };

  async inputField(page, selector, text){

    if(text !== "" && text !== null){
      let field = await page.$eval(selector, el => el.value);
      if(field !== text){

        await page.$eval(selector, el => el.value = '');
        await page.click(selector);
        await page.type(selector, text, { delay: 0 });
}
    }
  };
```

```
log(msg, color) {

let Ex = this;

    if (Ex.stopped)
      return;

    Ex.theWs.send(JSON.stringify({
      type: "log",
      value: msg,
      taskId: Ex.taskId,
      color: color,
      time: new Date().ddmmyyyy_hhmmssms(),
      accountName: Ex.accountName,
    }))

    let utc = new Date().toJSON().slice(0,10).replace(/-/g,'-');
    let log = new Date().ddmmyyyy_hhmmssms() + " [" + Ex.accountName + "] : " +
msg + "\n";
    //логивфайл
fs.appendFile(path.join(require('nw.gui').App.dataPath, 'magisters-local-
data/logs/') + utc + '.txt', `\r\n${log}`, (err) => {
      if (err) throwerr;
    });

    //логи в дискорд
/*if (Ex.logsWebhookUrl){

      try{
        let request = new XMLHttpRequest();
        request.open("POST", Ex.logsWebhookUrl);
        request.setRequestHeader('Content-type', 'application/json');
        let params = {
          username: "Logger",
          avatar_url: "",
          content: log
        }
        request.send(JSON.stringify(params));
      }
      catch (e) {

      }
    }*/
  };

  getTaskId(){
    let p = new $.Deferred();
    p.resolve(this.taskId);
    return p.promise();
  }

  getRandomMs(min, max) {
    return Math.random() * (max - min) + min;
  }

  async stopOne(){

    let p = new $.Deferred();
    if (this.browser){
      await this.browser.close();
```

Окончаниелистинга 3

```
      this.log("stopped", this.colors.red);
      p.resolve(true);
    }
    else if (this.shedule){
      this.shedule.cancel();
      this.log("stopped", this.colors.red);
      p.resolve(true);
    }
    else {
      this.log("stopped", this.colors.red);
    }
    this.stopped = true;
    return p.promise();
}

}

exportdefaultTask
```

### 2.2.3Поддерживаемые сайты

В настоящий момент поддерживаются следующие сайты:

- Adidas.ru

- Brandshop.ru

- Nike.com

- Wildberries.ru

### 2.2.4Описание работы программы на примере сайтаbrandshop.ru

Рассмотрим процесс покупки товаров на примере сайта brandshop.ru. Для данного сайта в приложении есть два режима – быстрый, без использования браузера, который показывает среднее время покупки товара около 5 секунд. Второй режим частично использует браузер для покупки, при таком подходе можно получить среднее время покупки товара примерно 17 секунд.

Первый режим реализован с помощью запросов к серверной стороне сайта, приложение сначала авторизуется на сайте с помощью логина и пароля, заданных пользователем, затем сохраняет данные сессии, которые дальше используются во всех запросах.

Следующий шаг – проверка, если ли у авторизованного пользователя какие-либо товары в корзине, поскольку мы должны купить только заданный товар, не пустую корзину следует очистить перед созданием заказа.

После проверки и очистки корзины, приложение переходит на страницу товара, собирает оттуда нужные параметры для добавления товара в корзину и делает запрос в серверной части сайта, чтобы добавить товар в корзину.

Теперь нужно перейти в корзину, где требуется выбрать метод доставки и метод оплаты, а затем создать заказ. После создания заказа приложение получает ссылку на его оплату, которая будет открыта в небольшом окне поверх нашего приложения, где автоматически будут введены все данные для оплаты, а пользователю только останется нажать кнопку «Оплатить».

Второй режим, работающий через браузер, делает все то же самое, только управляя процессом не через запросы к серверу сайта, а эмитируя действия обычного человека в браузере, эмулируются нажатия на кнопки сайта, ввод в текстовые поля и т.д.

Таким образом мы получаем два режима, один из которых гарантирует стабильность при небольшой скорости, а второй, наоборот, скорость при меньшей стабильности.

Исходный код класса, отвечающего за покупку товара на описанном выше сайте, представлен в приложении А.

### 2.2.5 Описание структуры хранения данных

Поскольку персональные данные пользователя хранятся локально на его компьютере, для хранения данных используется файловая база данных NeDB.

NeDB (Node.js Embedded Database) — встраиваемая база данных для NodeJS, реализующая подмножество MongoDB API. Эта легкая NoSQL СУБД написана на чистом JavaScript, не имеет бинарных зависимостей и, помимо NodeJS, может использоваться в NW.js, Electron или прямо в браузере.

NeDB обеспечивает хранение данных в простом файле на диске в json-формате, который похож на коллекции в MongoDB.

Всего для хранения используются 6 файлов, список которых представлен на рисунке ниже (Рисунок 29).

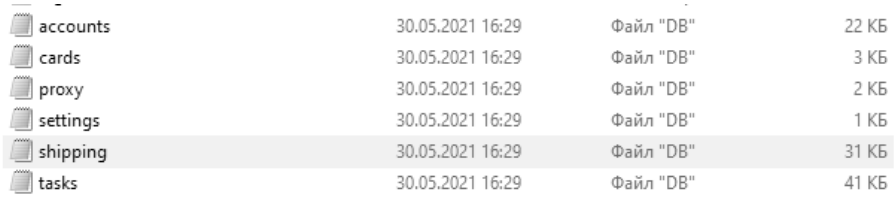



Рисунок 29 – Хранение пользовательских данных

Каждый файл хранит одну сущность, где новая запись идет с новой строки, а благодаря тому, что данные хранятся в jsonформате, мы получаем очень гибкую структуру, в которой не нужно заранее конфигурировать поля для хранения данных, как это пришлось бы делать, например, при использовании любой реляционной базы данных. Каждая строка в файле содержит 2 поля, первое – id – уникальный идентификатор записи, а также поле data, которое является массивом с неограниченным уровнем вложенности и динамическим структурой. Проще говоря, туда можно записать любое кол-во данных в json формате.

## 2.3WEB-приложение

### 2.3.1 Клиентская часть

Web-приложение, как и настольное, написано на Vue.js, VueXи VueRouter), а такжеBootstrap.

**Структура**

Структура также похожа на структуру настольного приложения, однако небольшие отличия в неё все же имеются. В компоненте, отвечающем за логику задач упрощена логика добавления и изменения, а именно сокращено количество компонентов. Для создания задачи теперь используется только один Vuекомпонент, то же самое касается и редактирования. Ниже представлена структура файлов настольного приложения (Рисунок 30, Рисунок 31).

**Страницы web-приложения**

**Страница авторизации**

При загрузке приложения пользователь попадает на страницу авторизации, которая содержит форму с двумя полями – логином и паролем.

Пример данной страницы отображен ниже (Рисунок 32).

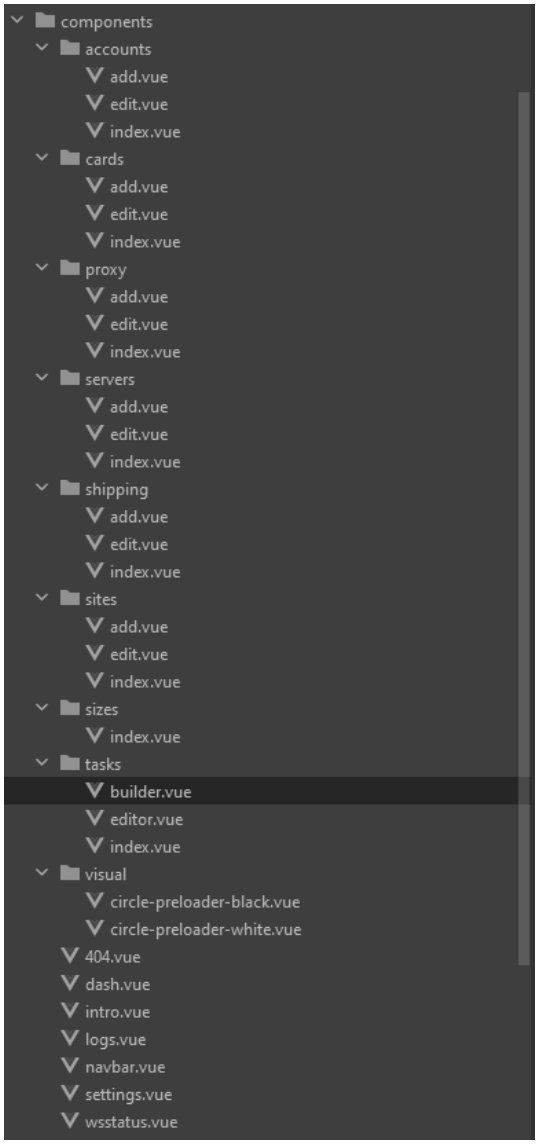



Рисунок 30 – Структура файлов компонентов


Рисунок 31 – Структура jsфайлов


Рисунок 32 – Авторизация web-приложения

**Панель управления**

Визуально панель управления разделена на две части: управление задачами и управление серверами. Под сервером в данном случае понимается любой компьютер, на котором будет установлено локальное приложение.

**Блок управления серверами**

Чтобы взаимодействовать с удаленным настольным приложением, нужно добавить его в список серверов в web-приложении, сделать это можно при помощи специальной формы (Рисунок 33).


Рисунок 33 – Форма добавления сервера

Для добавления сервера нужен только его ip и название, которое может быть любым. После добавления сервера будет сгенерирован специальный ключ – hash сервера, который будет использоваться в настольном приложении для его идентификации.

После добавления сервер будет отображен в списке серверов слева в панели управления (Рисунок 34).


Рисунок 34 – Список серверов

Также вверху приложения отображается текущий статус подключения к websocketсерверу, в данном случае показан пример успешного подключения, а в случае отсутствия соединения с сервером будет отображён статус (Рисунок 35).

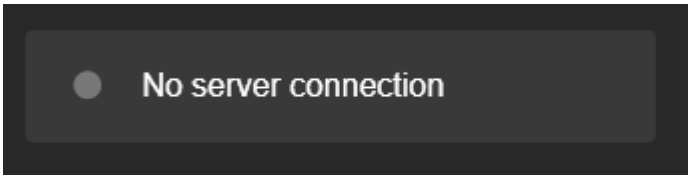



Рисунок 35 – Статус подключения к серверу

**Страница задачи**

На этой странице отображены задачи, созданные для подключенных серверов. Стоит отметить, что задачи выполняются только в локальном приложении, а web-интерфейс нужен только, чтобы управлять им удаленно. Внешний вид страницы представлен ниже (Рисунок 36).

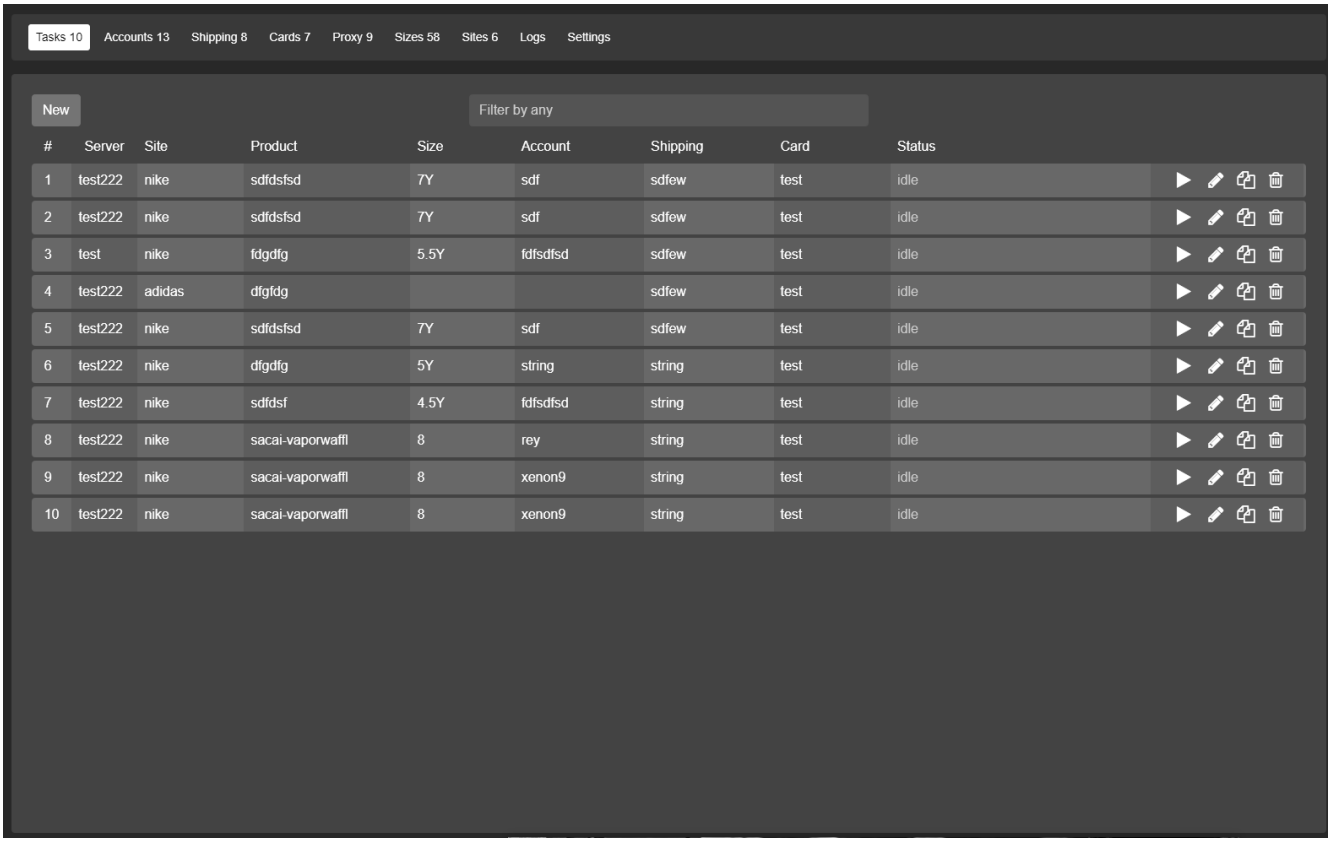



Рисунок 36 – Страница «задачи»

Для создания задачи нужно нажать на кнопку «New», в верхней части страницы. При нажатии откроется окно, изображенное на рисунке ниже. При создании задачи можно выбрать сайт, для которого она создается, сервер, ссылку на товар, размер товара, аккаунт сайта, адрес, на который приложение оформит заказ, карту, которой он будет оплачен, количество задач, которое будет создано с введенными настройками, включить/отключить поддержку прокси для задачи, а также выбрать время старта задачи и время отправки заказа. Все настройки, описанные выше формируются динамически в зависимости от выбранного на первом этапе сайта. Интерфейс для добавления представлен ниже (Рисунок 37).

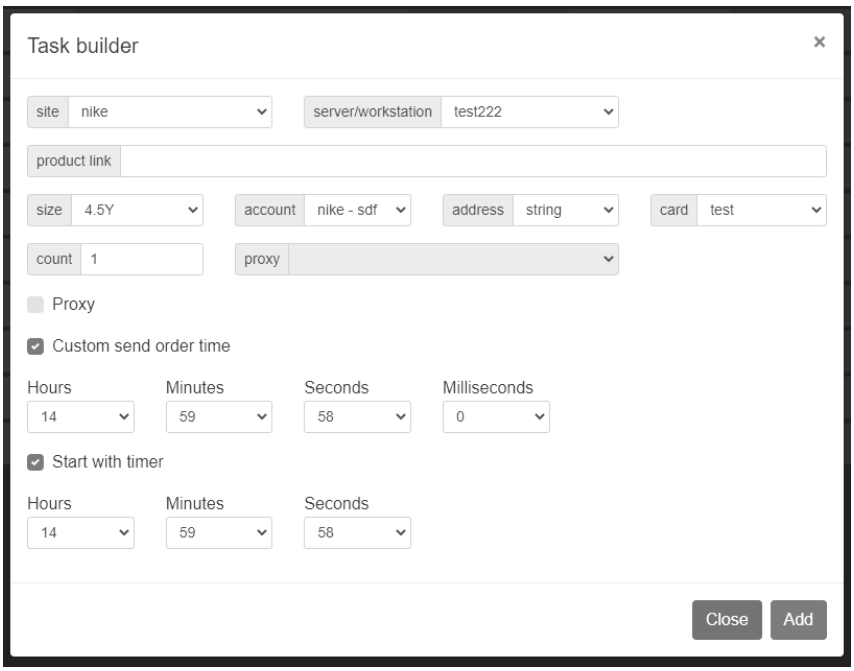



Рисунок 37 – Форма добавления задач

Можно сказать, что процесс добавления идентичен этому же процессу в настольном приложении, поэтому далее будет показан только интерфейс страниц. Этот вывод справедлив для всех страниц, кроме страницы сайты и страницы размеры товаров, так как такой страницы в настольном приложении нет.

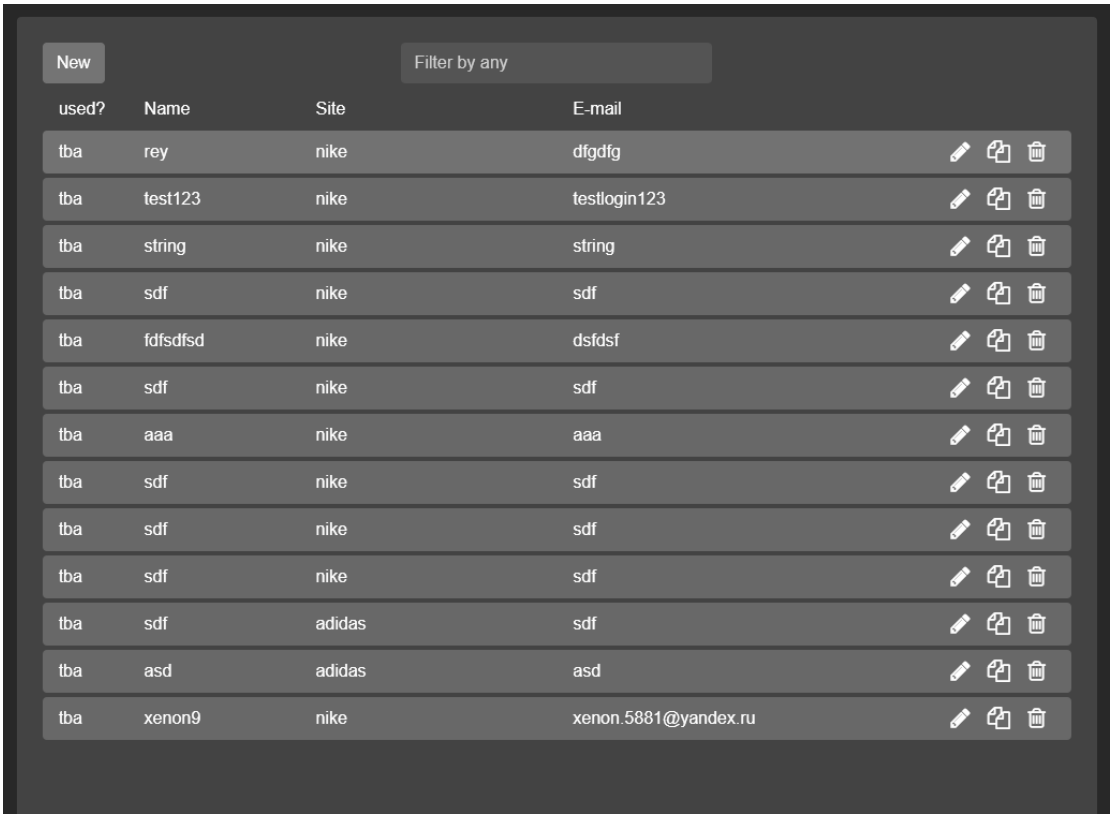



Рисунок 38– Страница «аккаунты»

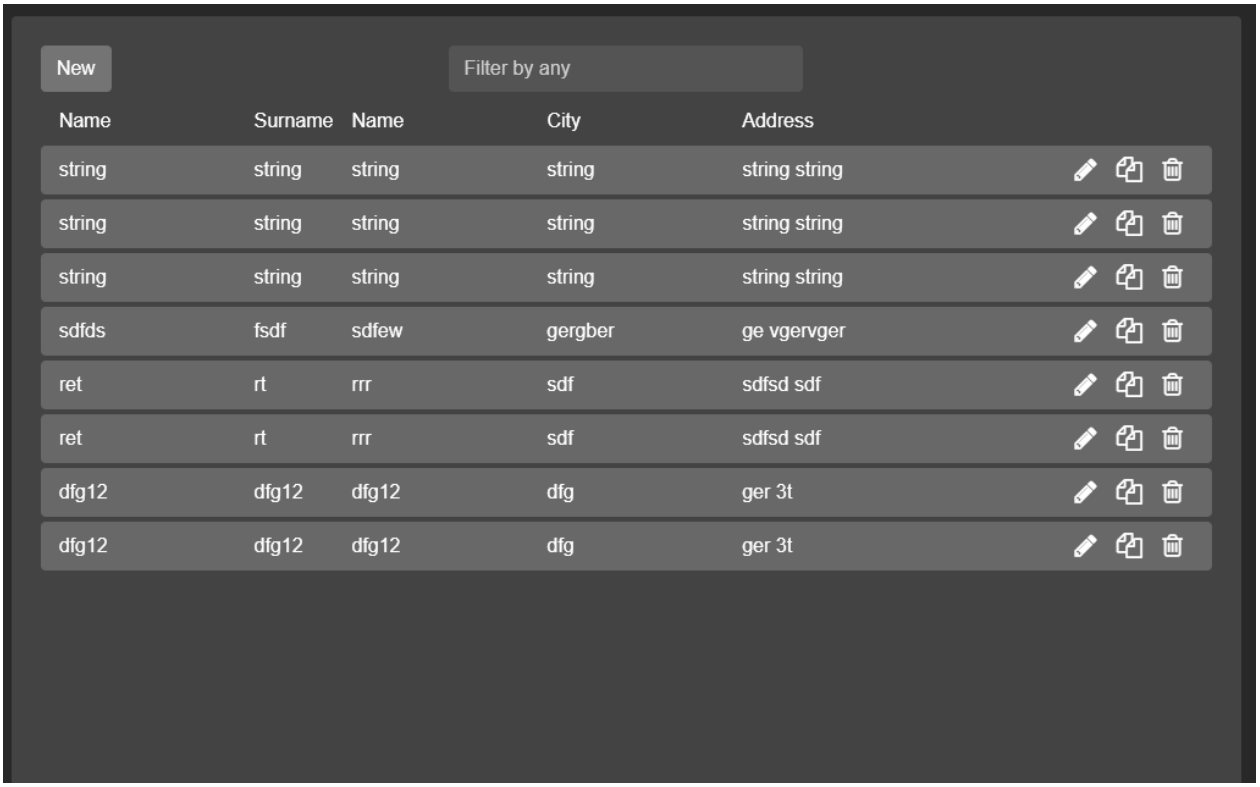



Рисунок 39 – Страница «адреса»

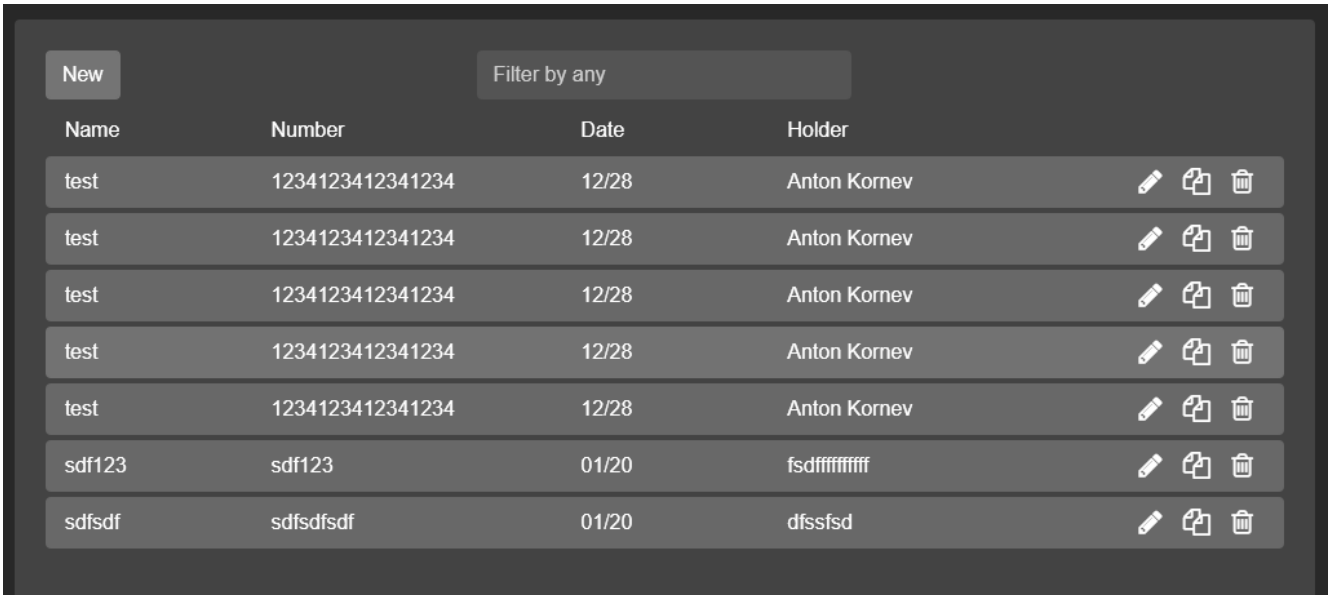



Рисунок 40 – Страница «карты»

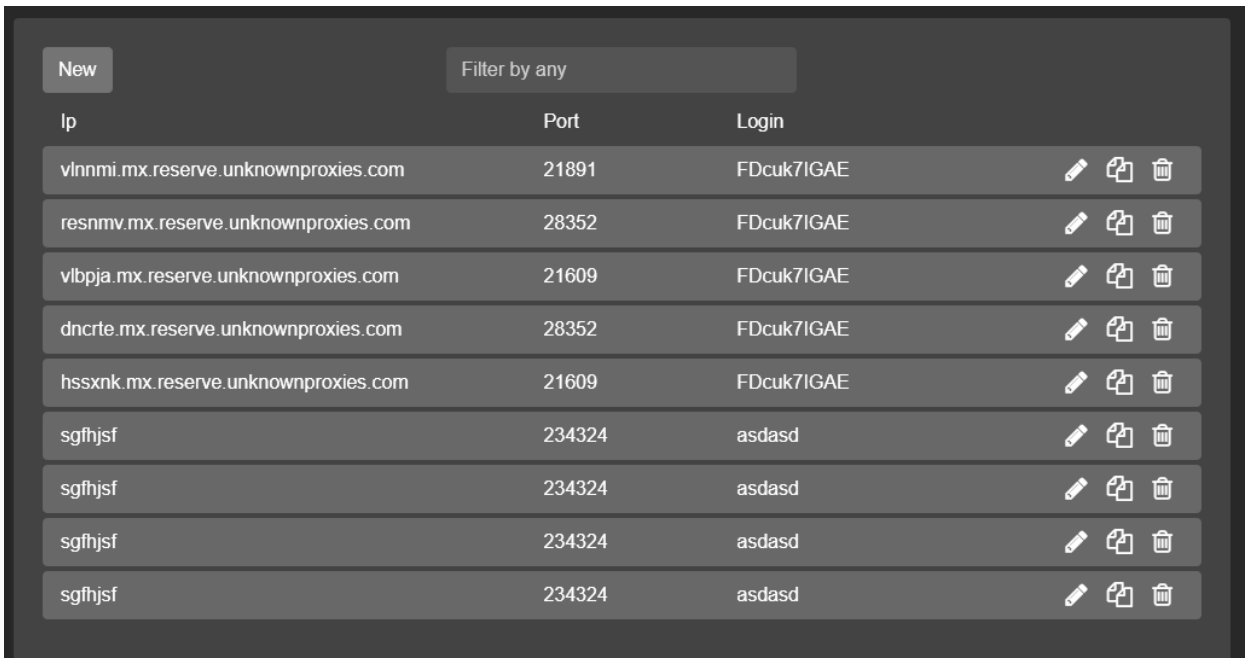



Рисунок 41 – Страница «прокси»

**Страница размеры товаров**

Да данной странице отображены размеры товаров, которые можно будет выбрать при создании задач. На текущий момент на этой форме отсутствуют интерфейсы для добавления, редактирования и удаления, а все эти операции производятся администратором напрямую через базу данных.

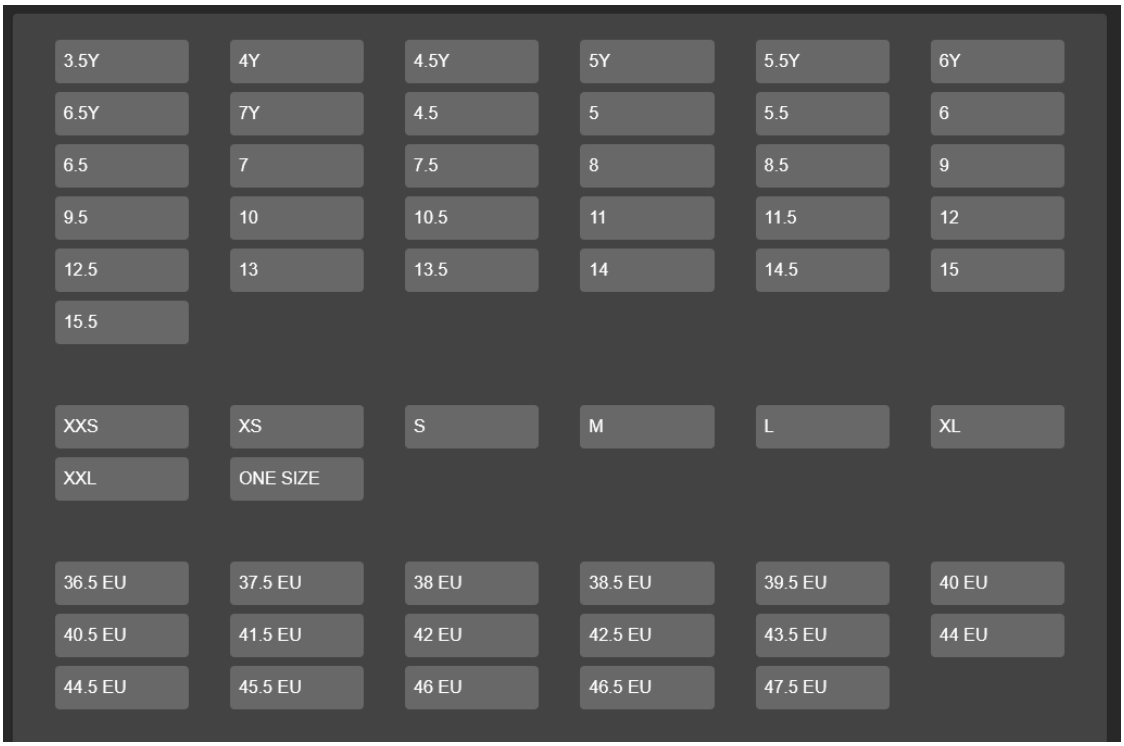



Рисунок 42 – Страница «размеры»

**Страница сайты**

Страницы нужна, чтобы добавить в систему поддержку нового сайта, также тут можно включить или отключить сайт. Под включением понимается то, возможно ли будет выбрать этот сайт на этапе создания задачи.


Рисунок 43 – Страница «сайты»

### 2.3.2 Серверная часть

Серверная сторона web-приложения написана на языке php с использованием таких фреймворков, как Symfony, Ratchet, Api-platform, для хранения данных используется база данных mysql, размещенная на VPS сервере, который в свою очередь привязан к доменному имени, через которое осуществляется доступ к клиентской части приложения.

Symfony — свободный фреймворк, написанный на PHP.Symfony предлагает быструю разработку и управление веб-приложениями, позволяет легко решать рутинные задачи веб-программиста. Работает только с PHP 5 и выше. Имеет поддержку множества баз данных (MySQL, PostgreSQL, SQLite или любая другая PDO-совместимая СУБД). Информация о реляционной базе данных в проекте должна быть связана с объектной моделью. Это можно сделать при помощи ORM инструмента. Symfony поставляется с двумя из них: Propel и Doctrine. Symfony бесплатен и публикуется под лицензией MIT [9].

Ratchet – phpбиблиотека, позволяющая использовать web-сокеты. В данном проекте мы будем использовать его прямо внутри Symfony сервера.

Api-platform – мощный инструмент, с помощью которого на основе Symfonyможно реализовать полноценное REST API, документацию, а также Swagger UI с возможностью протестировать работу наших эндпоинтов.

**Создание сервера Symfony**

Symfony представляет таблицы базы данных в разрезе объектов, которые в терминологии Symfony называются сущностями. На старте проекта у нас как может уже быть спроектированная база данных, так может и не быть её, Symfony может ускорить процесс создания сервера в любом случае. В данном проекте база данных была спроектирована первой, поэтому на первом шаге нам нужно подключить её в созданный шаблонный проект Symfony. Делается это с помощью редактирования конфигурационного файла .env в корневой директории проекта.

Подключение к базе данных выглядит следующим образом

```
DATABASE_URL="mysql://anton:vt27t3bt23462vv@81.124.110.30:3306/db_name?serverVersion=5.3"
```

После добавления в проект базы данных и выполнения нескольких консольных команд, мы получаем готовые сущности Symfony, которые полностью отражают все таблицы, поля, а также связи базы данных. Созданные сущности отражены ниже (Рисунок 44).

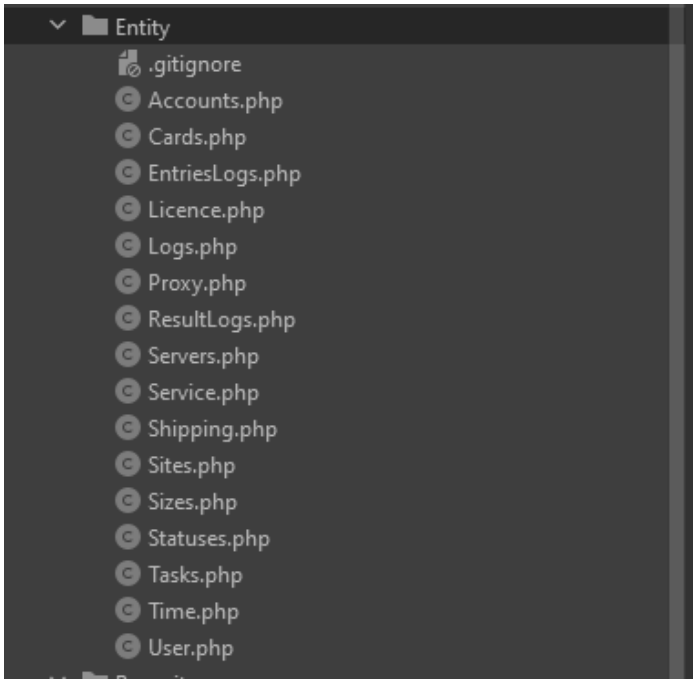



Рисунок 44 – Сущности

Каждая сущность состоит из набора полей симметричных полям в базе данных, а также геттеров и сеттеров для этих полей, в листинге ниже представлен пример исходного кода сущности Sites–сайты.

Листинг 4

```
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use ApiPlatform\Core\Annotation\ApiResource;
use Symfony\Component\Serializer\Annotation\Groups;

/**
 * Sites
 *
 * @ORM\Table(name="sites")
 * @ORM\Entity
*/
class Sites
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     * @ORM\Column(name="name", type="string", length=100, nullable=false)
     */
    private $name;

    /**
     * @var bool
     *
     * @ORM\Column(name="enabled", type="boolean", nullable=false)
     */
    private $enabled;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }

    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }
```

Окончаниелистинга4

```
    public function getEnabled(): ?string
    {
        return $this->enabled;
    }

    public function setEnabled(string $enabled): self
    {
        $this->enabled = $enabled;

        return $this;
}
}
```

На данном этапе мы уже имеем готовый сервер, но не имеем интерфейса, по которому сможем к нему обращаться и получать данные.Можно было бы написать такой интерфейс вручную методами Symfony, но в данном проекте мы используем еще один фреймворк, речь о котором пойдет дальше.

**Подключение Api-platform**

Api-platform позволяет буквально за пару строк, добавленных в проект, получить полный набор CRUD (create, read, update, detele)методов для сущности Symfony. При этом также создается точка входа в RESTAPIприложение вместе с информативным Swagger UI интерфейсом, в котором можно наглядно увидеть все эндпоинты и даже протестировать их без каких-либо дополнительных инструментов.

Настройка Api-platform происходит с помощью нотаций – комментариев специального формата, которые воспринимаются как Symfony, так и Api-platform.

Пример нотации приведен в листинге ниже.

Листинг 5

```
/**
 * Sites
 *
 * @ORM\Table(name="sites")
* @ORM\Entity
*/
```

Чтобы сделать сущность Symfony частью RESTAPI, нужно импортировать библиотеку ApiResource Api-platform в файл с сущностью, а также добавить в нотацию тег @ApiResource с параметрами внутри, или без них. Пример тега приведен в листинге ниже.

Листинг 6

```
* @ApiResource(
 *      itemOperations={
 *          "get",
 *          "delete",
 *          "patch"
 *      }
 * )
```

На этом наша первая сущность готова к работе, за относительно небольшое количество шагов мы получили полный набор методов для манипуляции нашей сущностью, а также целую тестовую среду для работы с ней.

При запуске приложения и переходе по адресу http://127.0.0.1:8000/api мы увидим интерфейс Swagger UI, который наглядно подтверждает, что все сделано верно. Интерфейс Swagger UI представлен ниже (Рисунок 45).

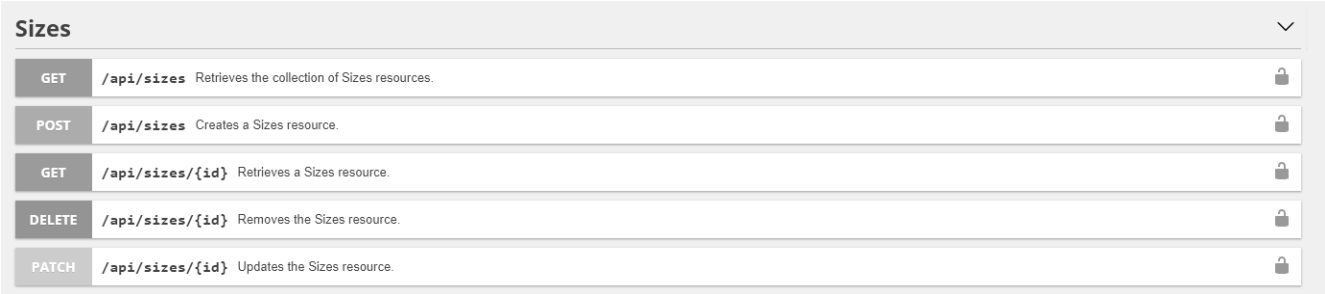



Рисунок 45 – Swagger UI

При нажатии на одну их строк открывается интерфейс, в котором можно протестировать метод. На рисунке ниже приведен пример данных, возвращенных GETметодом сущности размеры (Рисунок 46).

То же самое нужно проделать для всех остальных сущностей. В результате мы получаем готовое RESTAPI, в котором остается только настроить права доступа и авторизацию.

В данном разделе рассмотрен случай, когда база данных создана до создания сервера, однако Symfony может быть использован и в обратном порядке, через консольные команды можно создать нужные сущности, которые потом будут конвертированы в таблицы базы данных.

Также, по умолчанию Symfony отдает данные постранично, но эта функция отключена, так как показалась лишней в данном проекте.

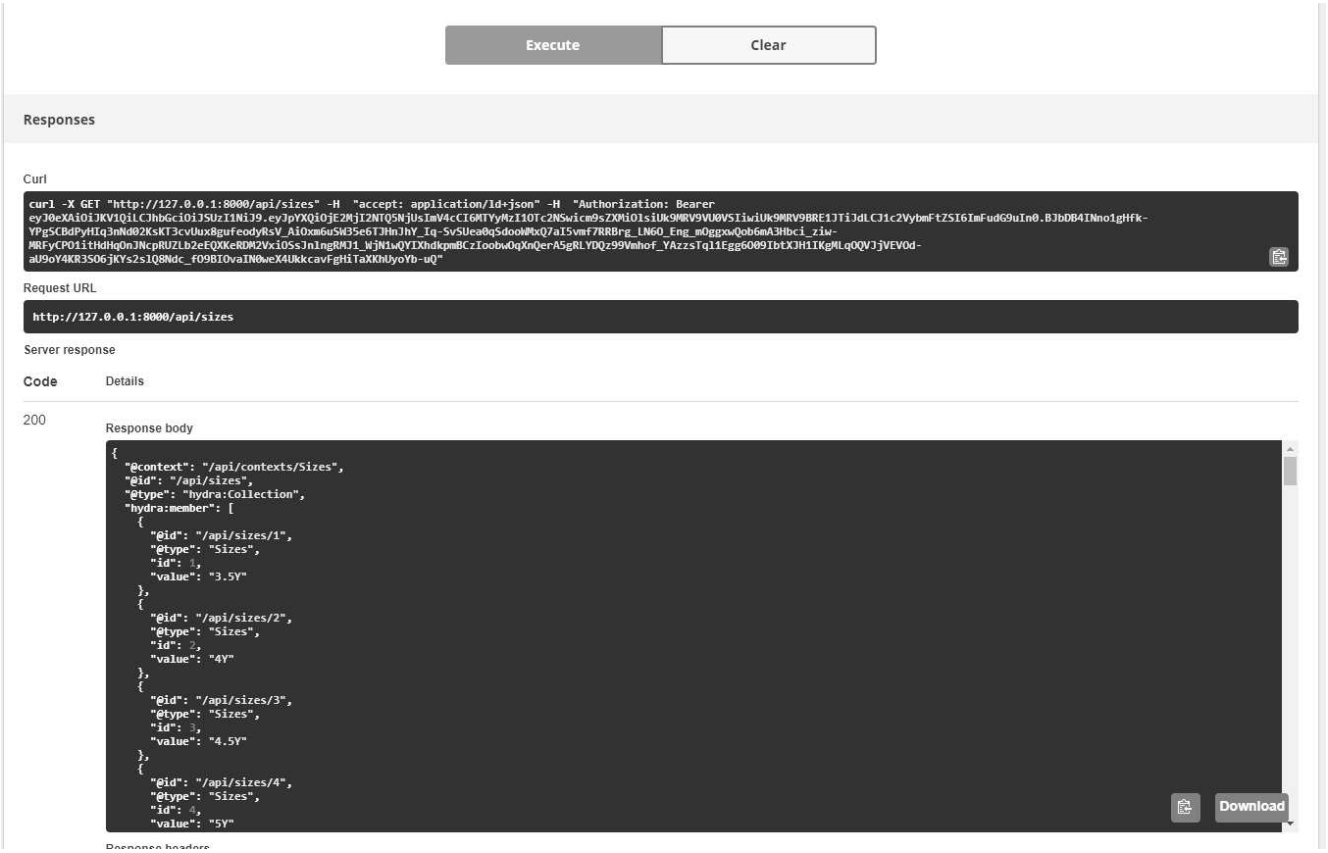



Рисунок 46 – Тестирование в Swagger UI

### Добавление Ratchet

Ratchet устанавливается в проект с Symfony, также он запускается и контролируется изнутри сервера Symfony. Symfony поддерживает пользовательские команды – то есть, можно задать серверу формулировку команды, а также алгоритм, который будет выполняться при выполнении этой команды. Для этих целей в проекте есть специальная директория – Command.

Для реализации запуска сервера создан файл WebsocketServerCommand.php, его содержание приведено в листинге ниже.

Листинг 7

```
<?php
namespace App\Command;

use Ratchet\Http\HttpServer;
use Ratchet\Server\IoServer;
use Ratchet\WebSocket\WsServer;
use App\Websocket\MessageHandler;
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
```

Окончаниелистинга7

```
class WebsocketServerCommand extends Command
{
    protected static $defaultName = "run:websocket-server";

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $port = 3001;
        $output->writeln("Starting server on port " . $port);
        $server = IoServer::factory(
            new HttpServer(
                new WsServer(
                    new MessageHandler()
                )
            ),
            $port
        );
        $server->run();
        return 0;
    }
}
```

Сервер будет запущен на 3001 при исполнении консольной команды «phpbin/console run:websocket-server».

Следующим шагом нужно настроить обработку сообщений, для этого в проекте создана директория Websocket, в которой создан файл MessageHandler.php, в котором и описана реализация обработки сообщений. Егосодержаниеприведеновлистингенижe.

Листинг 8

```
<?php
namespace App\Websocket;

use App\Entity\Logs;
use Exception;
use Ratchet\ConnectionInterface;
use Ratchet\MessageComponentInterface;
use SplObjectStorage;

use App\Entity\User;
use Doctrine\ORM\EntityManagerInterface;

class MessageHandler implements MessageComponentInterface
{
    protected SplObjectStorage $connections;

    public function __construct()
    {
        $this->connections = new SplObjectStorage;
    }

    public function onOpen(ConnectionInterface $conn)
    {
        $this->connections->attach($conn);
    }
```

```
    public function onMessage(ConnectionInterface $from, $msg)
    {
        echo $from->remoteAddress . "\n";

        $data = json_decode($msg);

        echo $msg . "\n";

        foreach($this->connections as $connection)
        {
            if($connection === $from)
            {
                continue;
            }
            $connection->send($msg);
        }
    }

    public function onClose(ConnectionInterface $conn)
    {
        $this->connections->detach($conn);
    }

    public function onError(ConnectionInterface $conn, Exception $e)
    {
        $this->connections->detach($conn);
        $conn->close();
    }
}
```

В настоящий момент сервер работает самым примитивным образом – рассылает входящие сообщения всем клиентам, кроме отправителя.

**Реализация авторизации**

Для авторизации и управления пользователями в целом, выбрана базовая сущность Symfony, которая добавляется в проект буквально за 3 консольных команды, «из коробки» эта сущность уже включает в себя такие поля, как id пользователя, имя пользователя, пароль, который сразу шифруется, почту, а также флаг активности пользователя, но он не использован в данном проекте.

Для хэширования паролей используется функция argon2i, она не требует отдельного поля для хранения соли, так как хранит все в одном поле, предназначенном для пароля.

Ниже представлен пример того, в каком виде хранится пароль в базе.

$argon2id$v=19$m=65536,t=4,p=1$MEVRZzVBbXo3SUptUlpVTg$iAnyYtHuX54V+xQFRvMeP//XTRf4Q+ptymdTauhXo0c

Строку можно разделить на следующие части:

argon2i: используемый алгоритм;

v=19: версия;

m=65536,t=4,p=1: параметры алгоритма (стоимость памяти, затраты времени и используемые потоки);

MEVRZzVBbXo3SUptUlpV: автоматически генерируемая соль;

Tg$iAnyYtHuX54V+xQFRvMeP//XTRf4Q+ptymdTauhXo0c: Фактический hash.

При добавлении сущности, автоматически создается эндпоинт в api, к которому пользователь может обратиться для авторизации, передав свои логин и пароль. При успешной авторизации, сервер в ответ вернет jwtключ для доступа к остальным методам в api.Время действия ключа можно задать в специальном конфигурационном файле, пример которого приведен ниже (Рисунок 47).

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    token_ttl: 604800
```

Рисунок 47 – Параметры jwtключей

В данном случае время жизни ключа ограничено 604800 секундами, что равно одной неделе.

### 2.3.3 Описание структуры базы данных

Для хранения данных в web-приложении создана база данных MySQL, которая расположена на том же VPSсервере, что и весь проект. Структура базы данных представлена на ниже (Рисунок 48).

В базе данных создано 11 таблиц,которые содержат задачи, размеры товаров, карты, время, прокси, пользователей, сервера сайты, аккаунты, адреса доставки, описание каждой из которых приведено ниже.

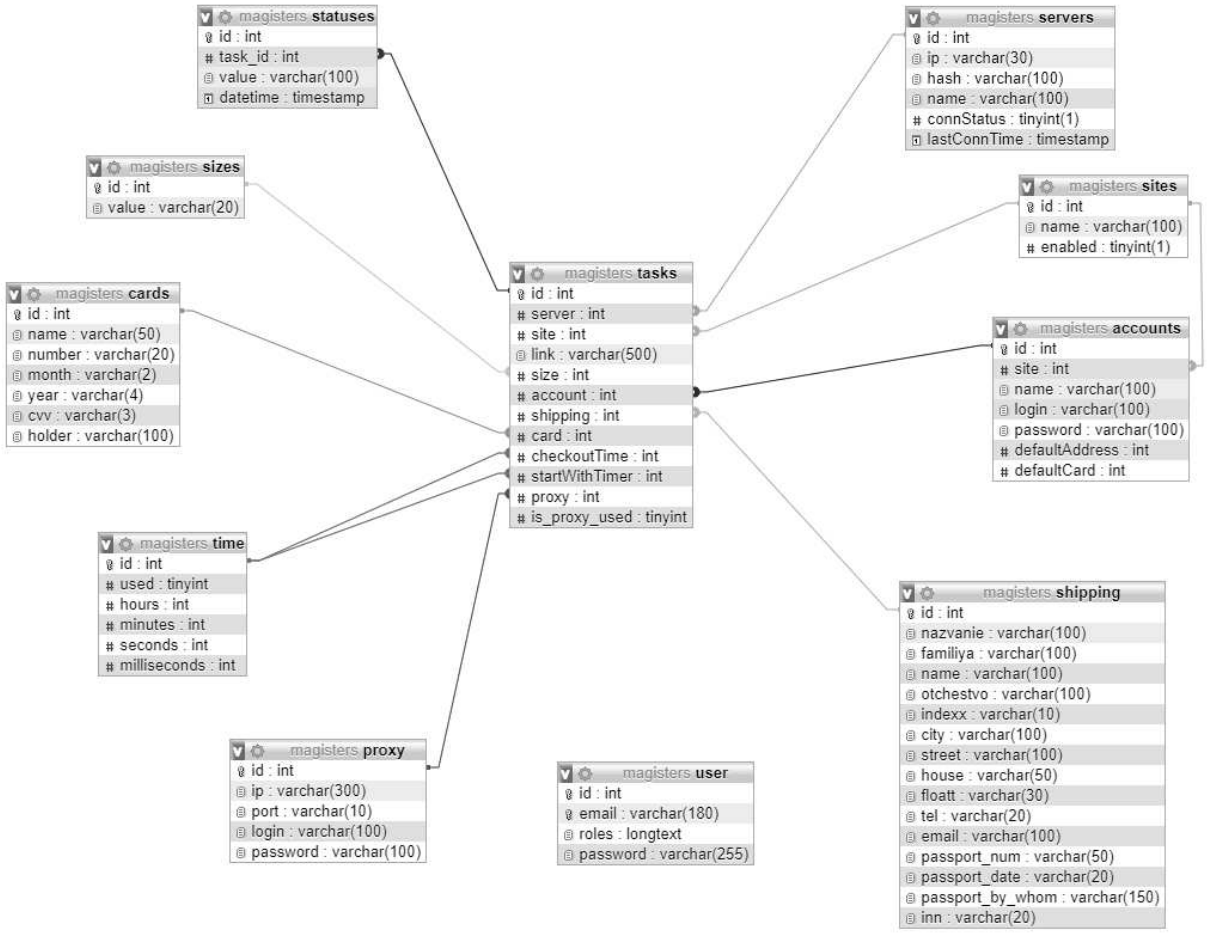



Рисунок 48 – База данных

**Таблица statuses**

В таблице хранится история статусов задач.

Таблица 4– Описание полей таблицы shops

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| task_id | int | Id задачи |
| value | varchar(100) | Значение статуса |
| datetime | timestamp | Метка времени |

**Таблица sizes**

В таблице хранятся размеры заказываемых товаров.

Таблица 5– Описание полей таблицы sizes

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| value | varchar(20) | Значение размера |

**Таблица cards**

В таблице хранятся карты, которыми можно оплатить заказ.

Таблица 6– Описание полей таблицы cards

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| name | varchar(50) | Имя карты |
| number | varchar(20) | Номер карты |
| month | varchar(2) | Месяц |
| year | varchar(4) | Год |
| cvv | varchar(3) | CVV |
| holder | varchar(100) | Имя держателя карты |

**Таблица time**

В таблице хранятся временные метки, которые используются для работы программы по заданному времени.

Таблица 7– Описание полей таблицы time

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| used | bool | Признак использования времени |
| hours | int | Часы |
| minutes | int | Минуты |
| seconds | int | Секунды |
| milliseconds | int | Миллисекунды |

**Таблица proxy**

В таблице хранятся прокси, которые могут быть использованы для задач.

Таблица 8– Описание полей таблицы proxy

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| ip | varchar(300) | Ip |
| port | varchar(10) | Порт |
| login | varchar(100) | Логин |
| password | varchar(100) | Пароль |

**Таблица user**

В таблице хранятся пользователи.

Таблица 9– Описание полей таблицы user

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| email | varchar(180) | email |
| roles | longtext | Роли пользователя |
| password | varchar(255) | Пароль |

**Таблица servers**

В таблице хранятся данные компьютеров, на которых установлена локальная программа, которой и нужно управлять.

Таблица 10– Описание полей таблицы servers

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| ip | varchar(30) | Ip сервера |
| hash | varchar(100) | Hash сервера |
| name | varchar(100) | Имя сервера |
| connStatus | bool | Статус соединения |
| lastConnTime | timestamp | Последнее время соединения |

**Таблица sites**

В таблице хранится список сайтов, с которыми может работать настольная программа.

Таблица 11– Описание полей таблицы sites

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| name | varchar(180) | Название сайта |
| enabled | bool | Признак включения сайта |

**Таблица accounts**

В таблице хранится список аккаунтов от сайтов.

Таблица 12– Описание полей таблицы accounts

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| site | varchar(180) | Id сайта |
| name | bool | Название сайта |
| login | varchar(100) | Логин |
| password | varchar(100) | Пароль |
| defaultAddress | int | Id адреса по умолчанию |
| defaultCard | int | Id карты по умолчанию |

**Таблица shipping**

В таблице хранятся адреса, которые могут быть использованы для задач.

Таблица 13– Описание полей таблицы shipping

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| nazvanie | varchar(300) | Название адреса |
| familiya | varchar(10) | Фамилия |
| name | varchar(100) | Имя |
| otchestvo | varchar(100) | Отчество |
| indexx | varchar(10) | Индекс |
| city | varchar(100) | Город |
| street | varchar(100) | Улица |
| house | varchar(50) | Дом |
| floatt | varchar(30) | Квартира |
| tel | varchar(20) | Телефон |
| email | varchar(100) | E-mail |
| passport_num | varchar(50) | Номер паспорта |
| passport_date | varchar(20) | Дата выдачи паспорта |
| passport_by_whom | varchar(150) | Кем выдан паспорт |
| inn | varchar(20) | Инн |

**Таблица tasks**

В таблице хранятся задачи.

Таблица 14– Описание полей таблицы tasks

| Поле | Тип | Описание |
|---|---|---|
| id | int | Ключевое поле |
| server | int | Id сервера |
| site | int | Id сайта |
| link | varchar(500) | Ссылка на товар |
| size | int | Id размера |
| account | int | Id аккаунта |
| shipping | int | Id адреса |
| card | int | Id карты |
| checkoutTime | int | Id времени создания заказа |
| startWithTimer | int | Id времени запуска задачи |
| proxy | int | Id прокси |
| is_proxy_used | bool | Признак использования прокси |

**Вывод по разделу два**

В разделе представлены описания реализации настольного и web-приложения, описаны поддерживаемые системой сайты, структуры хранения данных обоих приложений, а также описаны текущие интерфейсы систем.

Стоить отметить, что в отличии от настольного приложения, система удаленного управления в текущем виде не может предоставлена пользователям, так как не отвечает современным требованиям безопасности, в частности, в вопросе хранения личных данных.

# ЗАКЛЮЧЕНИЕ

Цель дипломного проекта – разработкаприложениядля автоматизации покупки лимитированных товаров в интернет-магазинах.

Для решения поставленной задачи использовались программные инструменты – Node-WebKit, Vue, Bootstrap, Puppeteer, Symfony, DoctrineORM, Api-platform, Ratchet, MySQL, NeDB,.

В ходе выполнения дипломного проекта решены задачи:

- проектирование программного продукта;
- выбор программных средств и инструментов для разработки;
- проектирование настольного приложения;
- проектирование web-интерфейса для управления удаленным приложением;
- разработка сервера для обработки запросов между локальным и web приложениями.

В результате создано приложение, которое отвечает всем заданным ему требованиям. Таким образом, поставленные задачи выполнены, а цель дипломного проекта достигнута.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. Бот cybersole [Электронный ресурс] – https://cybersole.io/
2. Бот f3ather[Электронный ресурс] – https://f3ather.io/
3. Бот NIKESHOEBOT[Электронный ресурс] – https://www.nikeshoebot.com/
4. Бот FLAREAIO[Электронный ресурс] – https://www.flarebots.com/
5. Каслдайн Э., Шарки К. - Изучаем jQuery (2-е изд.), 2012. 402 с.
6. Котеров,Д. PHP в подлиннике / Д. КотеровД., А. Костарев. –Спб.: БХВ-Петербург, 2005. 1120 с.
7. Гарсиа-Молина, Г. Системы баз данных. Полныйкурс = DatabaseSystems: TheCompleteBook/ Г. Гарсиа-Молина, Дж. Ульман, Дж. Уидом. –М.: Вильямс, 2003. 1088 с.
8. Vue.js в действии | Листуон Бенджамин, Хэнчетт Эрик, 2019. 304 с.
9. Fabien Potencier, Symfony 5: The Fast Track, 2019. 324 с.

# ПРИЛОЖЕНИЕ А

## Код класса «Brandshop»

```
letschedule = require('node-schedule');
constpuppeteer = require('puppeteer')
importTask from "../tasks/index";
let request = require('request');
let $ = require('jquery');

class Brandshop extends Task{

async start(modeId){

let Ex = this;
let startTime = Ex.taskArray.data.startWithTimer ?
Ex.taskArray.data.startWithTimer : null;

if (startTime && startTime.used){

      Ex.shedule = schedule.scheduleJob({
        hour: startTime.hours,
        minute: startTime.minutes,
        second: startTime.seconds,
      }, async function(){

        switch (modeId){

          case 5:{
            await Ex.safe();
            break;
          }
          case 6:{
            await Ex.Fast();
            break;
          }
        }
      });
      Ex.log(`start with timer at
${startTime.hours}:${startTime.minutes}:${startTime.seconds}`, Ex.colors.blue)

    }
    else{

      switch (modeId){

        case 5:{
          await Ex.safe();
          break;
        }
        case 6:{
          await Ex.Fast();
          break;
        }
      }
    }
  }

  async safe(){

    let Ex = this;
```

```
Ex.log("starting...", this.colors.blue);

if(this.taskArray.data.proxy.used){

      Ex.browser = await puppeteer.launch({
        headless: false,
        slowMo: 40,
        /*ignoreDefaultArgs: [
          '--enable-automation'
        ],
        args: [
          '--proxy-server=' + this.proxyArray.data.ip + ":" +
this.proxyArray.data.port,
        ],*/
        executablePath: Ex.browserPath
      });
    }
    else {


       Ex.browser = await puppeteer.launch({
         headless: false,
         slowMo: 40,
         /*ignoreDefaultArgs: [
           '--enable-automation'
         ],*/
         /*args: [
           '--no-proxy-server',
           `--window-size=1200,800`
         ],*/
         defaultViewport: null,
         executablePath: Ex.browserPath
       });

    }

    let page = await Ex.browser.newPage();
    if(this.taskArray.data.proxy.used){

      await Ex.log("authenticating proxy", Ex.colors.blue);
      await page.authenticate({
        username: this.proxyArray.data.login,
        password: this.proxyArray.data.password
      });
      await Ex.log("proxy authenticated", Ex.colors.blue);
    }
    //                          Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36
    //await page.setUserAgent('Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36');

    await page.setViewport({width: 1200, height: 800});
    await page.goto("https://brandshop.ru/login");

    await page.waitForSelector("#login-form");
    await page.waitForSelector("#login-form [name=\"email\"]");
    await Ex.log("login", Ex.colors.blue);
    await page.click("#login-form [name=\"email\"]");
    await page.type("#login-form [name=\"email\"]", this.accountArray.data.login,
{ delay: 20 });
```

```
    await page.waitForSelector("#login-form [name=\"password\"]");
    await page.click("#login-form [name=\"password\"]");
    await page.type("#login-form [name=\"password\"]",
this.accountArray.data.password, { delay: 20 });
    await page.click("#login-form [type=\"submit\"]");


    await page.waitForSelector("nav");

    const cookies0 = await page.cookies();
    let sessionCookie = await
JSON.stringify(cookies0).match(/PHPSESSID","value":"([^"]+)/);
    //let upCookie = await JSON.stringify(cookies0).match(/up","value":"([^"]+)/);

    let phpSession = sessionCookie[1];
    //let up = upCookie[1];

    //alert('PHPSESSID=' + phpSession + '; currency=RUB; up=' + up + ';');

//проверка наличия товаров в корзине и их удаление

try {
      var options = {
        'method': 'GET',
        'url': 'https://brandshop.ru/xhr/cart/',
        'headers': {
          'cookie': 'PHPSESSID=' + phpSession + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'Referer': 'https://brandshop.ru/'
        }
      };
      request(options, async function (error, response) {

        //alert(JSON.stringify(response.body));

        if (error) throw new Error(error);

        let cart_array = JSON.parse(response.body);

        if (cart_array.products.length){

          Ex.log("cart is not empty", Ex.colors.red);

          cart_array.products.forEach(function(item, i, arr) {

            var options = {
              'method': 'POST',
              'url': 'https://brandshop.ru/cart/',
              'headers': {
                'Cookie': 'PHPSESSID=' + phpSession + '; currency=RUB;',
                'Host': 'brandshop.ru',
                'Referer': 'https://brandshop.ru/'
              },
              formData: {
                'remove': item.key
              }
            };
            request(options, async function (error, response) {
              if (error) throw new Error(error);
```

```
              await Ex.log("item " + parseInt(i+1) + " deleted: " + response.body,
Ex.colors.green);
            });
          });
        }
      });
    }
    catch (e) {
      alert(e.message)
    }


    await page.goto(Ex.taskArray.data.link);
    await page.waitForSelector(".product-size");

    const [elementHandle] = await page.$x( './/div[@class="product-
size"]/div[contains(text(), \"' + Ex.taskArray.data.size + '\")]' );
    const propertyHandle = await elementHandle.getProperty('outerHTML');
    const propertyValue = await propertyHandle.jsonValue();


    let html = await page.content();

    let product_id_regex = await html.match(/data-product-id="([^"]+)/);
    let option_value_id_regex = await propertyValue.match(/data-option-value-
id="([^"]+)/);
    let option_regex = await html.match(/name="([^"]+)" id="product-size"/);
    let option_id_regex = await propertyValue.match(/data-option-id="([^"]+)/);

    let product_id = product_id_regex[1];
    let option_value_id = option_value_id_regex[1];
    let option = option_regex[1];
    let option_id = option_id_regex[1];


    /*app.Logger(num, "safe product_id = " + product_id, "text-success");
    app.Logger(num, "safe option_value_id = " + option_value_id, "text-success");
    app.Logger(num, "safe option = " + option, "text-success");
    app.Logger(num, "safe option_id = " + option_id, "text-success");*/

    /*alert(product_id);
    alert(option_value_id);
    alert(option);
    alert(option_id);*/

    try {

      options = {
        'method': 'POST',
        'url': 'https://brandshop.ru/index.php?route=checkout/cart/add',
        'headers': {
          'cookie': 'PHPSESSID=' + phpSession + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'Referer': 'https://brandshop.ru/'
        },
        formData: {
          'quantity': 1,
          'product_id': product_id,
          'option_value_id': option_value_id,
[option]: option_id
```

```
}
      };

      await request(options, async function (error, response) {
        if (error) await Ex.log("added to cart error", Ex.colors.red);

        //alert(JSON.stringify(response));

        await Ex.log("added to cart", Ex.colors.green);
        await page.goto("https://brandshop.ru/checkout/");
      });
    }
    catch (e) {
      alert(e.message)
    }

    //return;

    /*if (await app.tasks[num].page.$x( ".//div[contains(text(), \"" +
app.tasks[num].size + "\")]" ) !== null){
        let elements = await app.tasks[num].page.$x(".//div[contains(text(), \"" +
app.tasks[num].size + "\")]");
        await elements[0].click();
        await app.Logger(num, "size selected", "text-success");

        await app.tasks[num].page.click(".btn-cart");
        await app.Logger(num, "added to cart", "text-success");

        await app.Logger(num, "checkout", "text-primary");
        await app.tasks[num].page.goto("https://brandshop.ru/checkout/");
    }
    else {
        await app.Logger(num, "size is not available, add manually", "text-
danger");
    }*/

    /*app.tasks[num].page.evaluate(_ => {
        window.scrollBy(0, window.innerHeight);
    });*/


    //page.url()


    /*alert(app.tasks[num].page.url());

//ждём попадания на страницу чекаута
while (true){

if (app.tasks[num].page.url() === "https://brandshop.ru/checkout/"){

            break;
        }
        else {
            await app.tasks[num].page.waitFor(100);
        }
    }*/

    //checkout-products
```

```
    await page.waitForSelector(".checkout-products");
    Ex.log("checkout", Ex.colors.blue);

    options = {
      'method': 'POST',
      'url':
'https://brandshop.ru/index.php?route=checkout/checkout/setshippingmethod',
      'headers': {
        'cookie': 'PHPSESSID=' + phpSession + '; currency=RUB;',
        'Host': 'brandshop.ru',
        'Referer': 'https://brandshop.ru/'
      },
      formData: {
        'shipping_method': Ex.taskArray.data.ship
      }
    };
    await request(options, async function (error, response) {
      if (error) throw new Error(error);
      Ex.log("shipping selected: " + response.body, Ex.colors.green);
    });

    options = {
      'method': 'POST',
      'url':
'https://brandshop.ru/index.php?route=checkout/checkout/setpaymentmethod',
      'headers': {
        'cookie': 'PHPSESSID=' + phpSession + '; currency=RUB;',
        'Host': 'brandshop.ru',
        'Referer': 'https://brandshop.ru/'
      },
      formData: {
        'payment_method': 'payture'
      }
    };
    await request(options, async function (error, response) {
      if (error) throw new Error(error);
      await page.reload();
      Ex.log("payment selected: " + response.body, Ex.colors.green);
    });


    await page.evaluate(_ => {
      window.scrollBy(0, window.innerHeight);
    });

    await page.waitForSelector(".checkout-products button[type=\"submit\"]");
    await page.click(".checkout-products button[type=\"submit\"]");

    //карта
    await page.waitForSelector(".number input[name=\"CardNumber\"]");
    await Ex.log("card input", Ex.colors.blue);
    await page.click(".number input[name=\"CardNumber\"]");
    await page.type(".number input[name=\"CardNumber\"]",
Ex.cardsArray.data.number, { delay: 0 });

    await page.waitForSelector(".date input[name=\"DateTo\"]");
    await page.click(".date input[name=\"DateTo\"]");
    await page.type(".date input[name=\"DateTo\"]", Ex.cardsArray.data.month +
Ex.cardsArray.data.year, { delay: 20 });
```

```
   await page.waitForSelector(".cvv input[name=\"SecureCode\"]");
    await page.click(".cvv input[name=\"SecureCode\"]");
    await page.type(".cvv input[name=\"SecureCode\"]", Ex.cardsArray.data.cvv, {
delay: 20 });


    await page.waitForSelector(".holder > [name=\"CardHolder\"]");
    await page.click(".holder > [name=\"CardHolder\"]");
    await page.type(".holder > [name=\"CardHolder\"]", Ex.cardsArray.data.holder,
{ delay: 0 });

    await page.waitForSelector(".payBlock > .pay");
    await page.click(".payBlock > .pay");

    await Ex.log("input code from your bank", Ex.colors.green);

  }

  Fast(){

    //app.tasks[num].phpSessionCookie = '31429109b1d53c967f1ae1bd5db1c7bf';
    //app.tasks[num].upCookie = '01';

    //app.tasks[num].inWork = true;

    let Ex = this;
    let phpSessionCookie = null;
    let cookie = null;

    Ex.log("starting fast", this.colors.blue);

    function login() {

      //alert(Ex.stopped);

      if(Ex.stopped)
        return;

      //alert(Ex.accountArray.data.login);
      //alert(Ex.accountArray.data.password);

      let dfd = new $.Deferred();

      let options = {
        'method': 'POST',
        'url': 'https://brandshop.ru/login/',
        'headers': {
          'authority': 'brandshop.ru',
          'method': 'POST',
          'accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.9',
          'cache-control': 'max-age=0',
          'origin': 'https://brandshop.ru',
          'referer': 'https://brandshop.ru/login/',
          'Host': 'brandshop.ru',
          //'cookie': 'currency=RUB; up=' + app.cloudflareCookie.up + ';
__cfduid=' + app.cloudflareCookie.__cfduid + '; cf_clearance=' +
app.cloudflareCookie.cf_clearance + ';',
          //'user-agent': app.cloudflareCookie.user_agent
```

```
        },
formData: {
          'redirect': '/',
'email': Ex.accountArray.data.login,
          'password': Ex.accountArray.data.password
        }
      };

      //alert(JSON.stringify(options));
      //return;


      /*if(app.tasks[num].proxy_on){
        options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/

      request(options, async function (error, response) {
        if (error) throw new Error(error);

        alert(JSON.stringify(response));
        //return;

        /*if (response.statusCode === 503 && response.body.includes("Just a
moment...")){
          if(!app.tasks[num].stopped) app.Logger(num, "cloudflare detected",
"text-primary");

          //alert(111);

          await cloudFlareSkipper(num);

          //alert(222);

          await brandShopFastMode(num);

          //alert(333);

          return;
        }*/
        if (response.statusCode === 302){
          if(!Ex.stopped) Ex.log("logged in", Ex.colors.green);
        }
        else if(!Ex.stopped) {
          Ex.log("login error", Ex.colors.red);
          return;
        }

        //alert(JSON.stringify(response.headers["set-cookie"]));

        //let sessionCookie = JSON.stringify(response.headers["set-
cookie"]).match(/PHPSESSID=([^;]+)/);
        //let upCookie = JSON.stringify(response.headers["set-
cookie"]).match(/up=([^;]+)/);

        cookie = response.headers["set-cookie"].join('; ');

        //alert(cookie);

        //app.Logger(num, "sessionCookie = " + sessionCookie[1], "text-primary");
```

```
        //app.Logger(num, "upCookie = " + upCookie[1], "text-primary");

        //phpSessionCookie = sessionCookie[1];

        //alert(phpSessionCookie);
        //upCookie = upCookie[1];

        dfd.resolve();
      });

      return dfd.promise();
    }

    function clear_cart() {

      //alert('PHPSESSID=' + app.tasks[num].phpSessionCookie + '; currency=RUB;
up=' + app.tasks[num].upCookie + ';');

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

//проверка наличия товаров в корзине и их удаление
let options = {
        'method': 'GET',
        'url': 'https://brandshop.ru/xhr/cart/',
        'headers': {
          'cookie': /*'PHPSESSID=' + phpSessionCookie + '; currency=RUB;'*/
cookie,
          //'cookie': 'PHPSESSID=31429109b1d53c967f1ae1bd5db1c7bf; currency=RUB;
up=01;',
          'Host': 'brandshop.ru',
          'referer': 'https://brandshop.ru/'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/


      //alert(JSON.stringify(options));


      try{


        request(options, function (error, response) {
          if (error) throw new Error(error);

          //alert(JSON.stringify(error));
          //alert(JSON.stringify(response));


          alert(JSON.stringify(response.body));


          let cart_array = JSON.parse(response.body);
          if (cart_array.products.length){
```

```
            if(!Ex.stopped) Ex.log("cart is not empty", Ex.colors.red);

            cart_array.products.forEach(function(item, i, arr) {

              var options = {
                'method': 'POST',
                'url': 'https://brandshop.ru/cart/',
                'headers': {
                  'Cookie': /*'PHPSESSID=' + phpSessionCookie + ';
currency=RUB;'*/ cookie,
                  'Host': 'brandshop.ru',
                  'referer': 'https://brandshop.ru/'
                },
                formData: {
                  'remove': item.key
                }
              };
              /*if(app.tasks[num].proxy_on){
                  options['proxy'] = "http://" + app.tasks[num].proxy.data.login +
":" + app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
              }*/
              request(options, function (error, response) {
                if (error) throw new Error(error);
                if (JSON.parse(response.body).response === "success"){
                  if(!Ex.stopped) Ex.log("item " + parseInt(i+1) + " deleted: " +
response.body, Ex.colors.green);
                }

              });
            });
          }
          dfd.resolve();
        });

        return dfd.promise();


      }
      catch (e){
        alert(e.message)
      }

    }

    function addToCart() {


      //alert("addToCart");

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      var options = {
        'method': 'GET',
        'url': Ex.taskArray.data.link,
        'headers': {
          'Cookie': /*'PHPSESSID=' + phpSessionCookie + '; currency=RUB;'*/
cookie,
```

```
          'Host': 'brandshop.ru',
'referer': 'https://brandshop.ru/'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {
        if (error) throw new Error(error);

        let html = response.body;

        Ex.log(html, Ex.colors.red);

        let product_id_regex = html.match(/data-product-id="([^"]+)/);
        let option_regex = html.match(/name="([^"]+)" id="product-size"/);
        let option_value_id_regex = html.match(new RegExp(`data-option-
id="(\\d+)"\\sdata-option-value-id="(\\d+)">${Ex.taskArray.data.size}<`));
        let option_id_regex = html.match(new RegExp(`data-option-
id="(\\d+)"\\sdata-option-value-id="(\\d+)">${Ex.taskArray.data.size}<`));

        if (!product_id_regex || !option_regex || !option_value_id_regex ||
!option_id_regex){
          Ex.log("add to cart error", Ex.colors.red);
          Ex.stopped = true;
          return;
        }

        let product_id = product_id_regex[1];
        let option = option_regex[1];
        let option_id = option_id_regex[1];
        let option_value_id = option_value_id_regex[2];


        /*app.Logger(num, "fast product_id = " + product_id, "text-success");
        app.Logger(num, "fast option_value_id = " + option_value_id, "text-
success");
        app.Logger(num, "fast option = " + option, "text-success");
        app.Logger(num, "fast option_id = " + option_id, "text-success");*/


        options = {
          'method': 'POST',
          'url': 'https://brandshop.ru/index.php?route=checkout/cart/add',
          'headers': {
            'cookie': /*'PHPSESSID=' + phpSessionCookie + '; currency=RUB;'*/
cookie,
            'Host': 'brandshop.ru',
            'referer': 'https://brandshop.ru/'
          },
          formData: {
            'quantity': 1,
            'product_id': product_id,
            'option_value_id': option_value_id,
            [option]: option_id
          }
        };
        /*if(app.tasks[num].proxy_on){
```

```
            options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
        }*/
        request(options, function (error, response) {
          if (error) Ex.log("add to cart error", Ex.colors.red);
          if(!Ex.stopped) Ex.log("added to cart", Ex.colors.green);
          dfd.resolve();
        });
      });

      return dfd.promise();
    }

    function CheckCart() {

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();


      var options = {
        'method': 'GET',
        'url': 'https://brandshop.ru/xhr/cart/',
        'headers': {
          'cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'referer': 'https://brandshop.ru/'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {
        if (error) throw new Error(error);

        //alert(JSON.stringify(JSON.parse(response.body)));

        let cart_array = JSON.parse(response.body);
        if(!Ex.stopped) Ex.log(cart_array.products.length + " item in cart",
Ex.colors.red);

        dfd.resolve();
      });


      return dfd.promise();
    }

    function goToCheckout() {

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      var options = {
        'method': 'GET',
        'url': 'https://brandshop.ru/checkout/',
```

```
        'headers': {
          'Cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'referer': 'https://brandshop.ru/'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {
        if (error) throw new Error(error);
        if(!Ex.stopped) Ex.log("checkout", Ex.colors.blue);

        dfd.resolve();
      });

      return dfd.promise();
    }

    function setShippingMethod() {

      //alert('alert(app.tasks[num].brandShopShipVariant); ' +
app.tasks[num].brandShopShipVariant);

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      //alert(app.tasks[num].shipVariant);

      let options = {
        'method': 'POST',
        'url':
'https://brandshop.ru/index.php?route=checkout/checkout/setshippingmethod',
        'headers': {
          'cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'referer': 'https://brandshop.ru/'
        },
        formData: {
          'shipping_method': Ex.taskArray.data.ship
        }
      };

      //alert(JSON.stringify(options));

      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {

        //alert(JSON.stringify(JSON.parse(response.body)));


        if (error) throw new Error(error);
        if (!Ex.stopped){
          if (JSON.parse(response.body).success === "true") {
```

```
            Ex.log("shipping selected", Ex.colors.green);
          }
          else{
            Ex.log("error shipping select: " +
JSON.stringify(JSON.parse(response.body)), Ex.colors.red);
          }
        }
        dfd.resolve();
      });

      return dfd.promise();
    }

    function setPayingMethod() {


      //alert("setPayingMethod");

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      let options = {
        'method': 'POST',
        'url':
'https://brandshop.ru/index.php?route=checkout/checkout/setpaymentmethod',
        'headers': {
          'cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
          'Host': 'brandshop.ru',
          'referer': 'https://brandshop.ru/'
        },
        formData: {
          'payment_method': 'payture'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {
        if (error) throw new Error(error);

        //alert(JSON.stringify(response));

        if (!Ex.stopped){
          if (JSON.parse(response.body).success === "true") {
            Ex.log("payment selected", Ex.colors.green);
          }
          else{
            Ex.log("error payment select: " +
JSON.stringify(JSON.parse(response.body)), Ex.colors.red);
          }
        }

        dfd.resolve();
      });
      return dfd.promise();
}
```

```
functiongetPayingLink() {

if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      var options = {
        'method': 'GET',
        'url': 'https://brandshop.ru/checkout/',
        'headers': {
          'Cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
          'referer': 'https://brandshop.ru/'
        }
      };
      /*if(app.tasks[num].proxy_on){
          options['proxy'] = "http://" + app.tasks[num].proxy.data.login + ":" +
app.tasks[num].proxy.data.password + "@" + app.tasks[num].proxy.data.url;
      }*/
      request(options, function (error, response) {
        if (error) throw new Error(error);

        let checkoutData = response.body.match(/request-data.+?value="([^"]+)/);

        var options = {
          'method': 'POST',
          'url': 'https://brandshop.ru/index.php?route=payment/payture/send',
          'headers': {
            'Cookie': 'PHPSESSID=' + phpSessionCookie + '; currency=RUB;',
            'referer': 'https://brandshop.ru/'
          },
          formData: {
            'Data': checkoutData[1]
          }
        };
        request(options, function (error, response) {
          if (error){
            Ex.log("error: " + JSON.stringify(error), Ex.colors.red);
          }

          let link = JSON.parse(response.body).success;
          //if(!app.tasks[num].stopped) app.Logger(num, "link: " + link, "text-
success");
          dfd.resolve(link);
        });
      });
      return dfd.promise();
    }

    function inputPayingData(link) {

      if(Ex.stopped)
        return;

      let dfd = new $.Deferred();

      let payment_window = window.open(link, "",
"scrollbars=1,width=600,height=600");

      payment_window.eval(
```

```
"window.onload = function() {" +
        "let form = document.getElementById(\"card-one\");" +
        "let card = form.getElementsByClassName(\"onlyCardNum\");"+
        "card[0].value = '" + Ex.cardsArray.data.number + "';" +
        "document.getElementById(\"date\").value = '" + Ex.cardsArray.data.month +
"/" + Ex.cardsArray.data.year +"';"+
        "document.getElementById(\"cvv\").value = '" + Ex.cardsArray.data.cvv +
"';"+
        "let formOfCardHolder = form.getElementsByClassName(\"holder\"); "+
        "let CardHolder = formOfCardHolder[0].getElementsByTagName(\"input\"); "+
        "CardHolder[0].value = '" + Ex.cardsArray.data.holder + " ';" +
        "};"
      );

      dfd.resolve();

      return dfd.promise();
    }

    login()
      .then(clear_cart)
      .then(addToCart)
      .then(CheckCart)
      .then(goToCheckout)
      .then(setShippingMethod)
      .then(setPayingMethod)
      .then(getPayingLink)
      .then(inputPayingData)

      .then(function () {

        //app.tasks[num].inWork = false;
        if(!Ex.stopped) Ex.log("done. finish payment", Ex.colors.green);
      });
  }

  async cloudFlareSkipper(){



    /*app.isCloudflareSkippes = true;

    //alert("cloudFlareSkipper " + num);

    app.tasks[num].inWork = true;
    await app.Logger(num, "cloudflare worker starting...", "text-primary");

    if(app.tasks[num].proxy_on){

      app.tasks[num].browser = await puppeteer.launch({
        args: ['--proxy-server=' + app.tasks[num].proxy.url],
        headless: false,
        slowMo: 60,
      });
    }
    else {
      app.tasks[num].browser = await puppeteer.launch({
        headless: false,
slowMo: 60,
      });
```

```
}

    app.tasks[num].page = await app.tasks[num].browser.newPage();

    if(app.tasks[num].proxy_on){

      await app.Logger(num, "authenticating proxy", "text-primary");
      await app.tasks[num].page.authenticate({
        username: app.tasks[num].proxy.login,
        password: app.tasks[num].proxy.password
      });
      await app.Logger(num, "proxy authenticated", "text-primary");
    }

    await app.tasks[num].page.setViewport({width: 1200, height: 800});
    await app.tasks[num].page.goto("https://brandshop.ru/login");
    await app.tasks[num].page.waitForSelector("#login-form");

    const cookies0 = await app.tasks[num].page.cookies();

    //alert(await app.tasks[num].browser.userAgent());


    let __cfduidCookie = await
JSON.stringify(cookies0).match(/__cfduid","value":"([^"]+)/);
    let cf_clearanceCookie = await
JSON.stringify(cookies0).match(/cf_clearance","value":"([^"]+)/);
    let upCookie = await JSON.stringify(cookies0).match(/up","value":"([^"]+)/);


    app.cloudflareCookie.__cfduid = __cfduidCookie[1];
    app.cloudflareCookie.cf_clearance = cf_clearanceCookie[1];
    app.cloudflareCookie.up = upCookie[1];
    app.cloudflareCookie.user_agent = await app.tasks[num].browser.userAgent();

    app.tasks[num].browser.close();
    await app.Logger(num, "session copied", "text-success");

    app.isCloudflareSkippes = false;*/

  }

}

export default Brandshop
```