К.А. Манов, А.А. Тропин, Ю.В. Бидуля

Тюменский государственный университет, г. Тюмень

УДК 004.4

СЕРВИС-АГРЕГАТОР ДЛЯ ПОИСКА ТОВАРОВ В ИНТЕРНЕТ-МАГАЗИНАХ ОДЕЖДЫ

Аннотация. В статье представлена разработка сервиса-агрегатора для поиска товаров из каталогов различных сайтов Интернет-магазинов с возможностью ведения собственного списка сохраненных товаров и выбора наиболее выгодного предложения. Средствами разработки являются фреймворки Djungo и VueJS.

Ключевые слова: веб-сервис, Restful API, фреймворки, Django, VueJS.

В настоящее время магазины одежды активно размещают свои площадки в Интернете. Уже созданы крупные интернет-магазины, предлагающие клиентам большой выбор товаров. Помимо стандартных функций покупки данные интернет-ресурсы добавляют возможность сохранять понравившиеся варианты в список желаний, получить список рекомендаций исходя из предпочтений, оценить подобранный образ в онлайн-примерочных, и даже заказать доставку из любой точки мира прямо до двери.

Компания Data Insight проводила анализ рынка интернет-магазинов и покупателей [1]. Согласно итогам исследования, 34% опрошенных покупают товары в Интернете, потому что это экономит время. При этом, естественно, покупатель заинтересован приобрести вещь с максимальной скидкой и не обязательно в конкретном Интернет-магазине. Таким образом, имеется потребность, с одной стороны, найти товар с наибольшей

выгодой и, с другой стороны, не затратив при этом слишком много времени.

Решение этой проблемы предлагают *агрегаторы* – инструменты поиска товаров сразу по нескольким магазинам. Они представляют собой сообщества в социальных сетях, каналы в мессенджерах, и порой отдельные сайты – их функционал зачастую не уступает интернет-магазинам, за исключением одного – для совершения покупки необходимо перейти на сайт магазина. Создание такого сервиса приведет к экономии времени на поиск товаров за счет систематизации показа предложений по поисковому запросу, автоматизации процесса поиска и обновления информации о товарах из интернет-магазинов и сохранения закладок в едином списке.

Целью данной работы является разработка веб-сервиса, предлагающая осуществить поиск товаров из каталогов различных сайтов Интернет-магазинов, с возможностью ведения собственного списка сохраненных товаров и выбора наиболее выгодного предложения.

Для достижения цели требовалось решить следующие задачи:

- на стороне сервера: парсинг каталогов, сохранение данных о товарах в базу данных, реализация API для доступа из клиентских десктопных и мобильных приложений.
- на стороне клиента: отображение передаваемых из API данных на странице каталога, разработка форм для работы с приложением.

Анализ существующих решений

Создаваемый проект не является новинкой на рынке, уже успешно существуют сайты, сообщества в соцсетях и каналы в мессенджерах. Примерами таких проектов являются Clouty.ru, Justbutik.ru, Elemor.ru, Navisale.ru. К возможностям этих сервисов можно отнести следующие:

- Наличие собственного мобильного приложения;

- Поиск товаров по фото;
- Регистрация через социальные сети, а также возможность привязки сразу несколько аккаунтов соцсетей;
- Возможность объединить группу вещей в единый гардероб;
- Сохранение вещей в список избранного;
- Список рекомендаций, из которого можно подобрать похожую вещь;
- Список брендов каталогизирован и вынесен на отдельную страницы;
- Возможность ведения списка избранного;
- Группа в социальных сетях и канал в Telegram, куда выкладываются товары с большой скидкой;
- Бот в Telegram, который предлагает товары исходя из нужных размеров и искомой категории вещей.

Средства разработки

Разработка программной части проекта производилась на языке программирования Python с использованием веб-фреймворков Django и Django REST Framework. Пользовательский интерфейс сервиса разработан на веб-фреймворке Vue JS и фреймворке для статической генерации страницы Nuxt JS.

Использование Vue JS дает следующие преимущества [2]:

- SSR - server-side rendering. Библиотека Nuxt JS позволяет сгенерировать статичное содержимое на сервере. В таком случае у проекта повышаются показатели SEO, а также уменьшается время до отображения контента, что может удержать пользователей на мобильных устройствах и/или при плохом соединении.
- Гибкий набор компонентов для адаптирования под различные размеры устройств. Для создания мобильной версии достаточно изменить конфигурацию компонентов.

- Конечный проект в виде SPA — разновидности веб-приложений, которые загружаются и работают в одной HTML-странице. Остальные страницы и весь прочий контент подгружается динамически через JavaScript. В результате обеспечивается высокая скорость загрузки, так как все ресурсы загружаются лишь однажды и меняется лишь контент.

Предпосылками для выбора Django послужили следующие факторы [3]:

- Большинство инструментов для разработки приложения уже включены в фреймворк и их не нужно подключать отдельными библиотеками. К таким инструментами относятся панель администратора, формы, регистрация/авторизация пользователя и некоторые инструменты работы с БД;
- Включена защита от угроз злоумышленников, таких как SQL-инъекции и подделки межсайтовых запросов;
- Поддерживается REST Framework, это инструмент для удобного построения API;
- Приложения Django позволяют разделить проект на несколько частей, а также интегрировать в него готовые решения – к примеру, авторизацию пользователя через ВКонтакте.

Для получения каталогов магазинов была использована CPA-сеть Admitad, позволяющая получать каталоги магазинов в формате CSV.

Разработка клиентской части сервиса

1. Инициализация проекта

На начальном этапе был создан проект Nuxt JS командой npx create-nuxt-app в командной строке. Далее были заданы параметры создаваемого проекта. В качестве режима работы Nuxt был выбран параметр Universal SSR; добавлен пакет Axios – библиотека для работы с HTTP запросами. Для поддержания единого стиля кода использовался ESLint – инструмент

статического анализа кода для выявления проблемных шаблонов, обнаруженных в коде JavaScript [6].

Настройка конфигурации Nuxt находится в файле nuxt.config.js и служит для установки параметров окружения env, подключения плагинов для работы с LocalStorage, установки параметров для Axios, и настройки мета-тэгов в head.

Особо важно для проекта правильно настроить Axios, управляющий запросами API. Из настроек окружения берется ссылка на API и устанавливается токен для авторизации, если он имеется. Это необходимо для разграничения доступа к методам Django приложения. Таким образом, случайный посетитель не сможет получить по API данные о сохраненных товарах другого пользователя.

Создание глобального хранилища данных Vuex

Global store (глобальное хранилище) – глобальный объект JavaScript, хранящий данные всех компонентов приложения. Хранилище содержит в себе:

1. Состояние (state)
2. Геттеры (getters)
3. Мутации (mutations)
4. Действия (actions)

Приведенные далее примеры кода проекта являются частями модуля, описывающего хранилище для данных по товарам магазинов. Модули позволяют разбить общее хранилище на части, при этом сохраняя в едином виде.

Состояние – это те данные, которые будут отображаться пользователю путем помещения их в компоненты. И каждый компонент получает данные из состояния Vuex, а также отправляет данные обратно в состояние при их изменении. Пример состояния:

```
  state: {
    brands: [],
    category: [],
    shops: [],
    products: [],
    product_equality: 0,
    favorite_product: [],
  },
```

В эти поля размещаются данные о товарах, получаемые с API.

Геттеры – это часть хранилища, которые возвращают компонентам данные текущего состояния хранилища

Действия – методы, которые будут вызывать изменения в хранилище, но самостоятельно состояния не меняют – для этого нужно зафиксировать мутацию. Действия могут служить для получения запроса от сервера, а также могут вызываться из компонента методом dispatch – например, чтобы сохранить понравившийся товар в список «Избранное». Пример действия:

```
  actions: {
       ...
    async setBrands({commit}) {
      await this.$axios.$get(`/brands/`).then(data =>
commit('SET_BRANDS', data));
    },
    async setShops({commit}) {
      await this.$axios.$get(`/shops/`).then(data =>
commit('SET_SHOPS', data));
    },
    async getExistFavoriteProduct({commit}) {
      await this.$axios.$get('/favoriteitems/').then(data =>
commit('SET_FAVORITE_PRODUCT', data));
    },
    async addFavoriteProduct({commit}, payload) {
      await this.$axios.$put(`/favoriteitems/${payload}/`).then(data
=> commit('SET_FAVORITE_PRODUCT', data));
    },
    async delFavoriteProduct({commit}, payload) {
      await
this.$axios.$delete(`/favoriteitems/${payload}/`).then(data =>
commit('SET_FAVORITE_PRODUCT', data));
    }
}
```

В данном примере метод setBrands и setShops получают список магазинов из API, getExistFavoriteProduct – список уже добавленных в избранное товаров, addFavoriteProduct – отправляет запрос на добавление товара список избранных, а delFavoriteProduct – запрос на его удаление.

Мутации – методы, единственная задача которых — изменения состояния, то есть мутации не производят никаких вычислений. Использование мутаций невозможно в асинхронном режиме. Очевидный пример использования асинхронных методов — это получение данных с сервера, в ином случае в хранилище запишется невалидное состояние. по этой причине нужно использовать Действия. Пример мутаций:

```
mutations: {
    SET_PRODUCTS: (state, payload) => {
      state.product_equality = payload.count;
      state.products = payload.results
    },
    SET_SHOPS: (state, payload) => {
      const shops = payload.map(element => ({...element, selected:
false, key: 'shops'}));
      state.shops = shops.sort(sortByName)
    },
    SET_CATEGORY: (state, payload) => {
      state.category = payload
    },
    SET_BRANDS: (state, payload) => {
      const brands = payload.map(element => ({...element, selected:
false, key: 'brands'}));
      state.brands = brands.sort(sortByName)
    },
    SET_FAVORITE_PRODUCT: (state, payload) => {
      state.favorite_product = payload[0].items;
    },
  }
```

Создание страниц

Страницы в Vue являются также компонентами, но дополнительно участвуют в маршрутизации и собирают в себе все прочие компоненты, которые составляют тот конечный вид страницы, который видит

пользователь. В сервисе реализовано 4 страницы – главная страница, страница каталога, личный кабинет и страница авторизации.

Компонент – элемент Vue, содержащий в себе части пользовательского интерфейса, которые объединяясь на страницах и составляют общий вид приложения. Компоненты являются центральным элементом фреймворка, включая в себя одновременно и верстку в HTML, и CSS стили, и логику JavaScript. Компоненты могут быть как обособленным элементом, так и частью большого элемента. Главной целью использования компонентов является возможность их переиспользования, как с целью динамического обновления информации на странице, так и для поддержания принципа Don't Repeat Yourself.

На примере конфигуратора поиска можно рассмотреть элементы, из которых состоит компонент. Компонент состоит из тегов Template, Script и Style. Основной функционал задается в части Script, где указываются импорты всех используемых в компоненте компонентов и путь к ним. Также он содержит список объектов, которые в себе содержат методы по работе с шаблоном:

1. Name. хранит название компонента, к которому будут обращаться при вызове на странице:

```
name: "filters"
```

2. Data. Хранит данные и прочее переменные, необходимые для отображения в компоненте – в данном случае это набор параметров, установленных в фильтрах:

```
data: () => ({
  selectedFilters: [],
}),
```

3. Computed (вычисляемые свойства). В этой функции описывается изменение свойств, которые меняют свое значение при каждом изменении значения какого-либо другого свойства, отслеживаемого в данной функции:

```
computed: {
      ...mapState('products', ['brands', 'shops'])
    },
```

4. Methods (методы). Методы служат для обработки событий, совершаемых пользователем на странице. Следующие методы обрабатывают выбор пользователем какого-либо параметра в фильтре:

```
    methods: {
      ...mapActions('products', ['setProducts']),
      ...mapActions('filters', ['setProductFilter']),
      async searchProducts() {
        const filters = [...this.brands, ...this.shops];
        await this.setProductFilter(filters.filter(element =>
element.selected === true));
        await this.setProducts({page: 1});
      },
```

5. Watch. Методы-наблюдатели необходимы для реагирования на изменение данных – выполняет функцию в случае, если изменилось какое-либо свойство компонента. В данном случае выбор пользователем определенного магазина и/или бренда вызовет функцию поиска товара, который уточнит выдачу в зависимости от выбора:

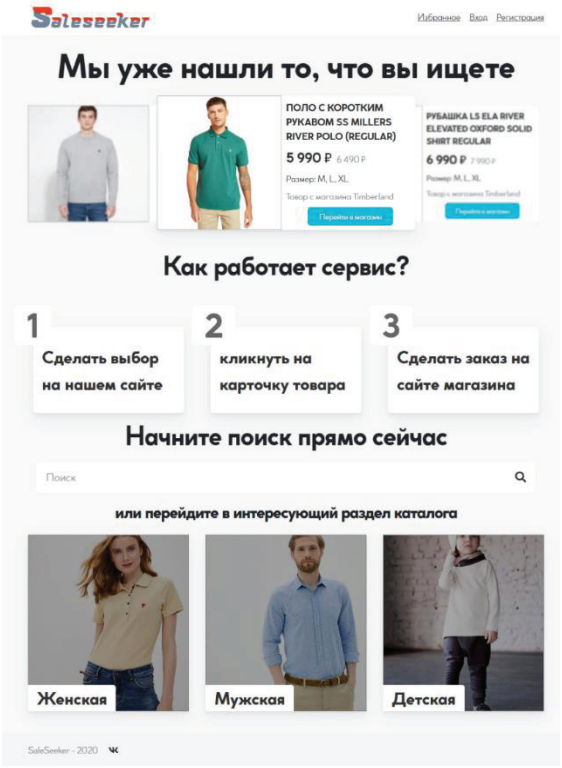
```
    watch: {
          brands: {
            deep: true,
            handler(){
                this.searchProducts()
            }
          },
          shops: {
            deep: true,
            handler(){
                this.searchProducts()
            }
          }
```

Разработка пользовательского интерфейса сервиса

Первоначально создание форм производилось в инструменте для прототипирования интерфейсов Figma. В этом графическом редакторе

можно «собрать» интерфейс без написания HTML разметки, с последующим импортом использованных стилей в CSS файле.

Для удобства была использованы две CSS-библиотеки для создания интерфейсов – Bootstrap 4 и Shard UI. На главной странице приложения расположены блоки с примерами товаров, краткой инструкцией по использованию сервиса, строка для поиска и блоки для перехода на страницу каталога (Рисунок 1).



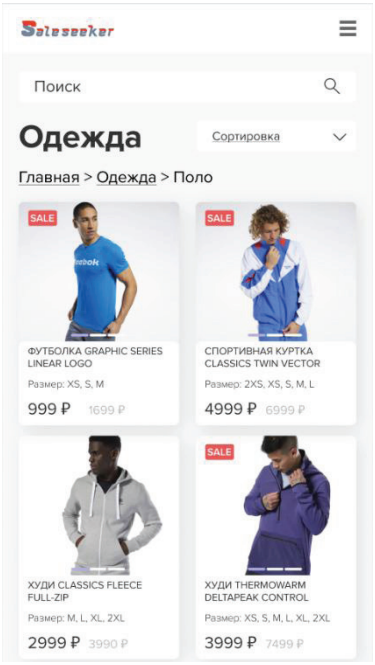


Рис. 1. Главная страница сервиса в десктопной и мобильной версиях.

После набора запроса в поисковой строке, или нажатия на один из блоков пользователь попадает на страницу каталога. Здесь расположены поисковая строка, конфигуратор с фильтрами, позволяющий сортировать выдачу по магазинам, категориям, брендам и цене; выпадающий список, предназначенный для сортировки выдачи относительно цены и по размеру скидки; а также карточки товаров и пагинация для перехода между страницами с товарами.

На странице личного кабинета можно увидеть персональные данные, оставленные при регистрации, а также поле смены пароля. Ниже

расположены сохраненные в раздел «Избранное» товары, с возможностью их сортировки по категории.

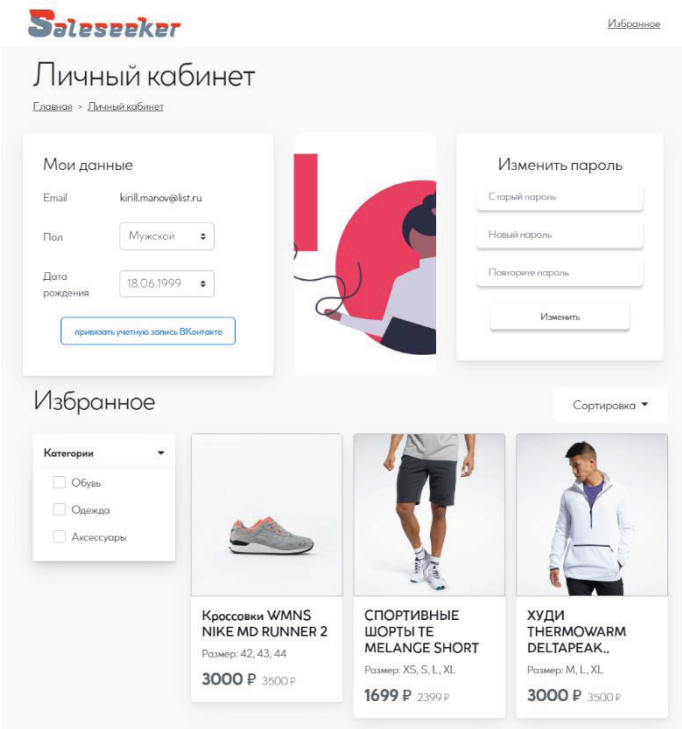


Рис. 2. Личный кабинет пользователя с сохраненными элементами в избранном.

Данные в программной части поставляются через API, что предоставляет возможность масштабирования проекта через разработку Telegram-бота и бота для сообщества ВКонтакте, которые увеличат аудиторию проекта. В планах по дальнейшему развитию проекта предполагается настройка поисковой оптимизации сайта; реализация функции поиска по картинке; создание Telegram-бота сервиса.

СПИСОК ЛИТЕРАТУРЫ

1. Data Insight – Онлайн-рынок одежды и обуви [Электронный ресурс.]. —Режим доступа: http://www.datainsight.ru/sites/default/files/DI-OnlineFashion1h2019.pdf. (Дата обращения:23.05.2020)

2. VueJS. The Progressive JavaScript Framework [Электронный ресурс.]. — Режим доступа: https://vuejs.org (Дата обращения:27.05.2020)
3. Django framework [Электронный ресурс.]. — Режим доступа: https://www.djangoproject.com/ (Дата обращения:30.05.2020).
4. Википедия – ESLint [Электронный ресурс]. — Режим доступа: https://en.wikipedia.org/wiki/ESLint (Дата обращения: 20.05.2020)