

TEAM: Deprecated Desperados

Pinion

(пиньон)

Браузерное расширение и приложение для моментального поиска товаров в
Яндекс.Маркет по изображению

Содержание отчета

Проект	4
Архитектура и технологии	6
Браузерное расширение	8
Архитектура	8
Структура браузерного расширения	9
Описание интерфейса браузерного расширения	10
WEB-приложение	13
Архитектура	13
Описание интерфейса приложения	13
Сервер	17
Архитектура	17
API	17
Инфраструктура	23
Сборка, линтеры	23
CSS codestyle	23
CI	25
Тестирование	26
Документация	27
Командная работа	28
Личные результаты и достижения	29
Итоги	31

Проект

Изначальная идея проекта состояла в том, чтобы дать пользователю возможность моментально(буквально в несколько кликов) находить в Яндекс.Маркете понравившиеся ему вещи с абсолютно любого сайта. Концепт включал в себя расширение для браузера, при активации которого, на изображениях, находящихся в данный момент на странице, с помощью специального API, разработанного в Яндекс распознавались и подсвечивались предметы, идентичные или имеющие схожие внешние характеристики с товарами, размещенными на данный момент на Яндекс.Маркете. После чего пользователь получал возможность перейти на страницу данного товара и приобрести его. Также пользователь мог внести понравившийся товар в собственную, гибко кастомизированную коллекцию желаемых товаров. Это позволяло приобрести товар в дальнейшем, а также делиться найденными результатами с другими пользователями.

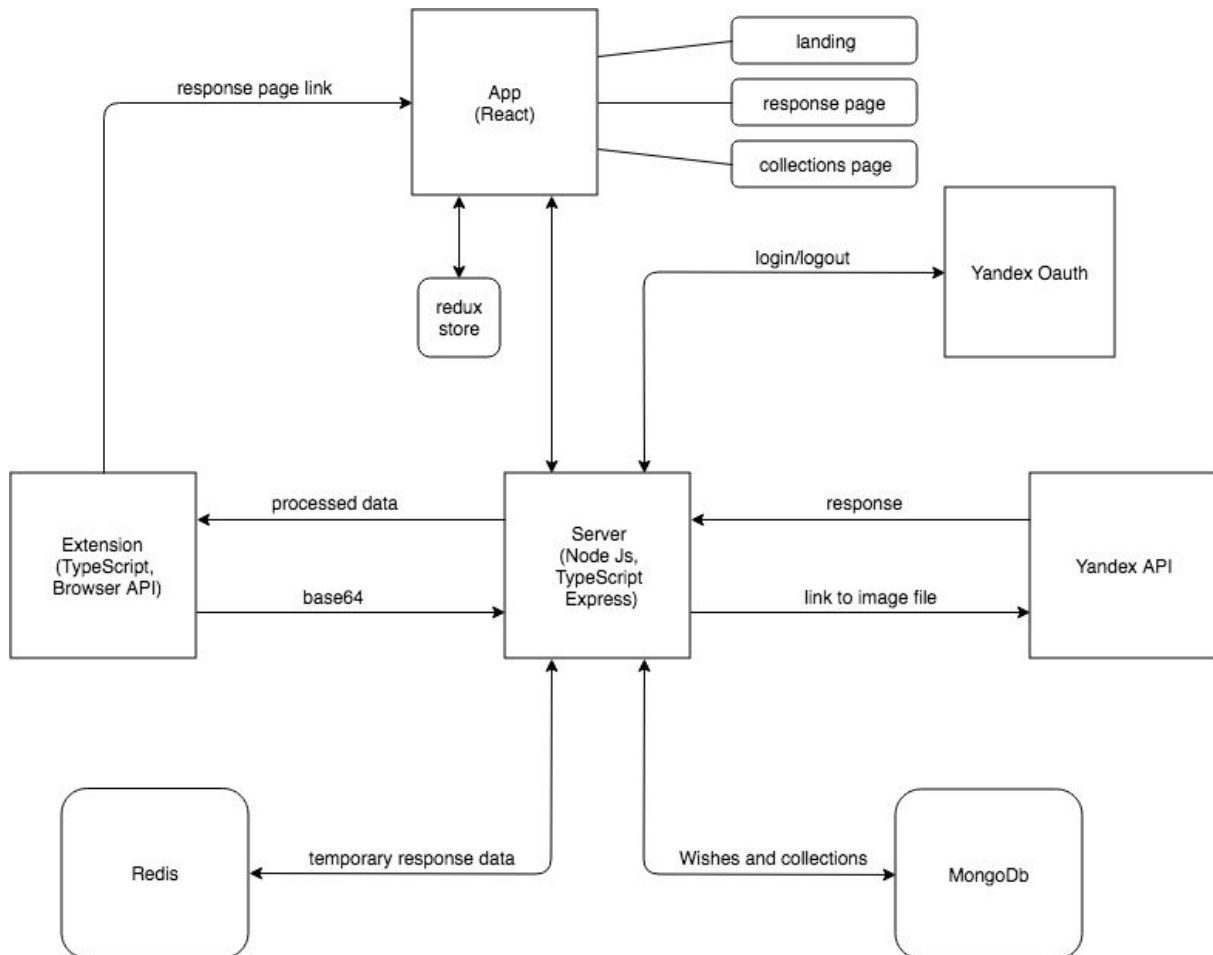
Основная идея проекта - сократить время между спонтанно возникшим желанием пользователя приобрести понравившийся продукт и самим его приобретением.

В процессе предварительных тестов и обсуждений командой были выявлены явные технические трудности в реализации первоначального концепта расширения. Были внесены изменения в техническую и визуальную реализацию проекта. Было принято решение предоставить пользователю выделять понравившийся предмет на странице браузера и отправлять его на распознавание. После этого было составлено изначальное MVP проекта, а сам проект был разделен на три части:

1. Расширение (extension) для браузера
2. Приложение (SPA)
3. Сервер

Предполагалось одновременное развитие всех трех составных частей проекта с возможностью для каждого участника команды поучаствовать в равной мере в разработке каждого из них с целью получения дополнительного опыта. Что и было реализовано в полной мере.

Архитектура и технологии



Функционал проекта реализован следующим образом:

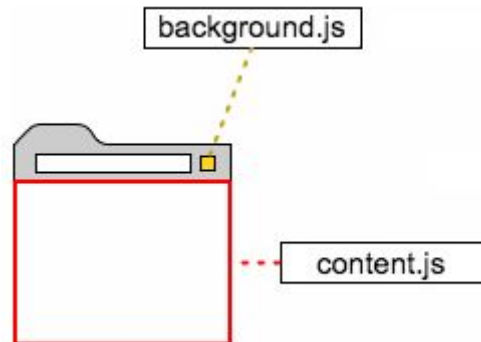
1. Пользователь устанавливает расширение в свой браузер
2. Нажимает на иконку расширения (или hotkeys)
3. Следуя инструкциям выделяет на экране понравившийся предмет
4. Нажимает иконку "OK" или Enter
5. Расширение конвертирует выделенную область в base64 и отправляет на сервер
6. Сервер сохраняет изображение в виде файла и формирует ссылку на него
7. Сервер отправляет ссылку на изображение на ручку яндекса и получает ответ
8. Сервер формирует из ответа данные в виде карточек, сохраняет эти данные в Redis со сроком истечения в час и отправляет в расширение первые пять карточек и сгенерированную ссылку на страницу выдачи в приложении
9. Сервер удаляет файл изображения

10. Расширение получает данные и отрисовывает первые пять релевантных карточек в виде слайдера и дает возможность занести карточки в коллекции, либо перейти в приложение
11. При переходе в приложение по сформированной сервером ссылке, приложение запрашивает у сервера данные по полной выдаче
12. Сервер извлекает данные из **Redis** и передает в приложение
13. Приложение отрисовывает данные
14. На странице выдачи возможно занести каждую карточку в любую из коллекций, а также создавать свои коллекции и переходить на страницу коллекций. Эти данные передаются на сервер и заносятся в базу данных (**MongoDb**)
15. При переходе на страницу коллекций идет запрос на сервер данных пользователя
16. Сервер обрабатывает входящий запрос и извлекает из базы данных информацию о пользователе и сохраненных им коллекциях и карточках
17. На странице коллекций есть возможность добавлять, редактировать и удалять коллекции, а также перемещать карточки между ними. Также реализована возможность удобного поиска по карточкам и названиям коллекций и возможность делиться продуктами в соцсетях
18. При клике на карточку на любой из страниц происходит переход на страницу продукта в Яндекс.Маркет
19. Авторизация происходит на сервере через Яндекс.Паспорт

Браузерное расширение

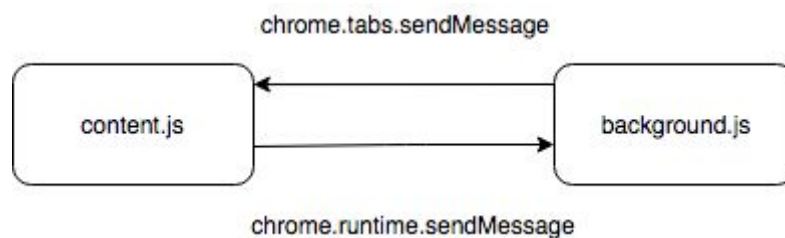
Архитектура

Браузерное расширение состоит из двух частей. Это бэкграунд скрипт (background.js, далее БС) и контент скрипт расширения (content.js, далее КС).



В БС происходит взаимодействие с браузерным API и бэкенд сервером. В КС непосредственное взаимодействие с веб страницей.

Общение БС и КС происходит посредством отправки сообщений.



Браузерное API предоставляет методы для подписки и отправки сообщений. При инициализации приложения БС и КС подписываются на сообщения и в зависимости от типа сообщения, происходит то или иное действие. При этом КС может слушать только те сообщения которые шлет БС, и наоборот.

Структура браузерного расширения


- app/ - взаимодействие с веб страницей(КС)
- api/ - директория со скриптами взаимодействия с бэкендом;
- components/ - директория компонентов;
- store/ - директория с данными приложения;
- styles/ - компоненты стилей (шрифты и переменные);
- utils/ - директория ф-ций помощников приложения;
- event.ts - обработчики событий;
- index.ts - точка входа;
- style.scss - общие стили;
- assets/ - директория со статическими файлами;
- tests/ - директория с файлами тестов;
- vendors/ - содержит скрипты для взаимодействия с браузерами(БС);
 - chrome/ - специфические скрипты для chrome браузера;
 - common/ - общие скрипты для всех браузеров;
 - firefox/ - специфические скрипты для firefox браузера;
- manifest.json - конфигурационный файл приложения;

В зависимости от браузера (параметр браузера передается в прм скрипте) собирается необходимая сборка расширения chrome или firefox.



Описание интерфейса браузерного расширения

Запуск

Включение осуществляется


- нажатием на иконку  в панели инструментов браузера;
- комбинацией клавиш: *Ctrl + Shift + 9*, *Command+Shift+9(mac)*;

Выключение осуществляется

- повторным нажатием на иконку  в панели инструментов браузера;
- комбинацией клавиш: *Ctrl + Shift + 9*, *Command+Shift+9(mac)*;
- в текущей вкладке, при включении PI в другой вкладке;
- нажатием кнопки  в навигационной панели;

Режим работы

При включении пользователь переходит в режим работы расширения:

- текущее окно браузера затемняется
- исчезает scroll страницы
- всплывает сообщение 
- пользователь получает возможность выделения области

Область выделения(ОВ)

Пользователь может перемещать и изменять размеры ОВ.

Минимальный размер стороны ОВ - 50px;

При задании ОВ меньше минимального пользователь получает сообщение



и получает возможность повторного выделения области.

Для удобства пользования навигационной панелью, ОВ имеет ограничение(40px) в нижней части окна браузера

Навигационная панель

Навигационная панель состоит из двух кнопок



- выключение PI



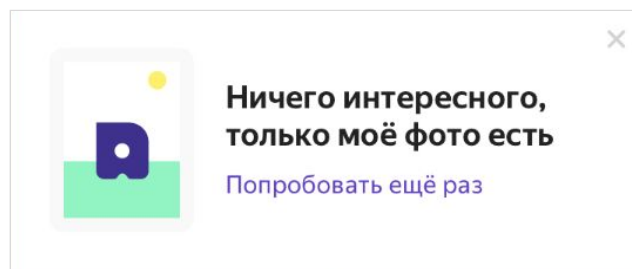
- включение поиска по ОБ

Процесс поиска

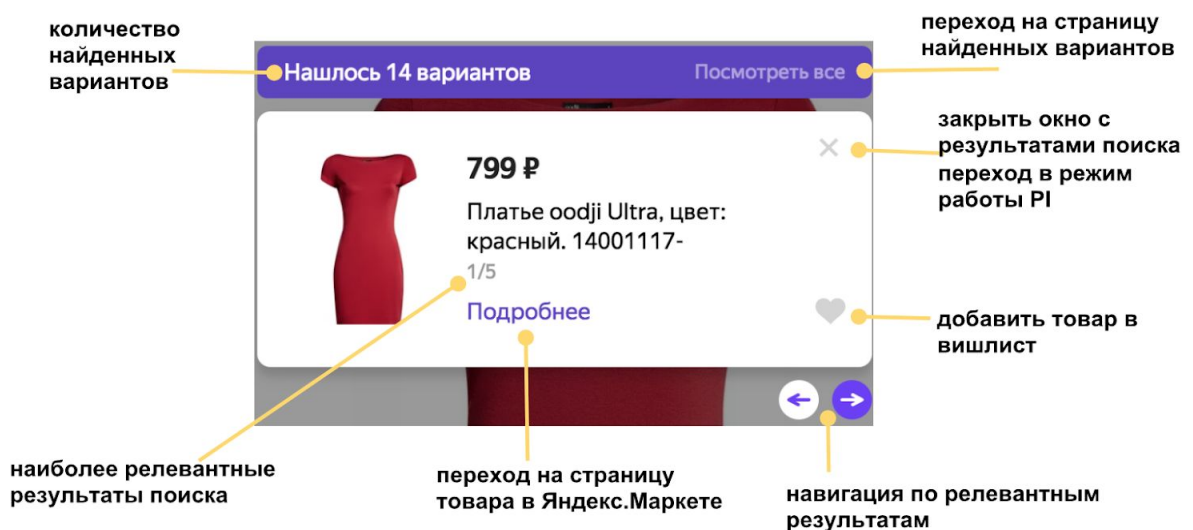
Процесс поиска сопровождается появлением лоадера, размер которого адаптирован под область выделения.

Результат поиска

При отсутствии результатов поиска появляется тултип



При наличии результатов поиска появляется тултип





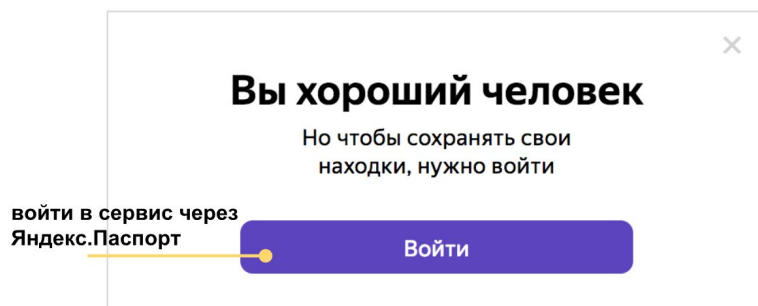
– заглушка, которая добавляется при отсутствии изображения на Яндекс.Маркете.

Добавление товара в коллекцию

Добавление товара в вишлист осуществляется нажатием кнопки



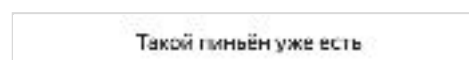
Если пользователь не залогинен в сервисе pinion, появляется тултип:



Если пользователь залогинен в сервисе pinion, осуществляется добавление товара в вишлист сервиса и появляется сообщение:



Если пользователь уже добавил товар в вишлист сервиса появляется сообщение:



WEB-приложение

Архитектура

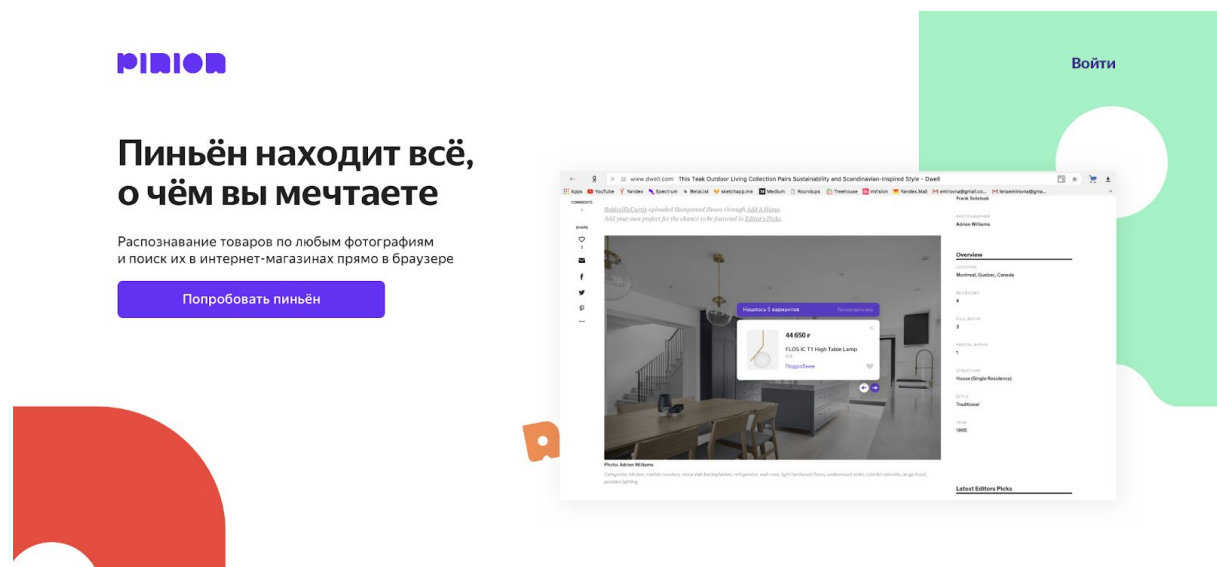
Приложение являет собой полноценный **Single Page Application** на базе **React/Redux**. В качестве сборки взят **Create React App Rewired**, расширенный поддержкой **Sass** и **Inline SVG**. Основные данные хранятся в **Redux Store** и гибко используются с помощью селекторов и библиотеки **Reselect**. В приложении обеспечена возможность сохранения, редактирования и удаления, а также фильтрации по наименованиям карточек и коллекций.

Дополнительные библиотеки:

- Classnames - для переопределения классов и унификации компонентов
- React-s-alert - для попапов (стили модифицированы под приложение)
- React-text-truncate - обрезание текста

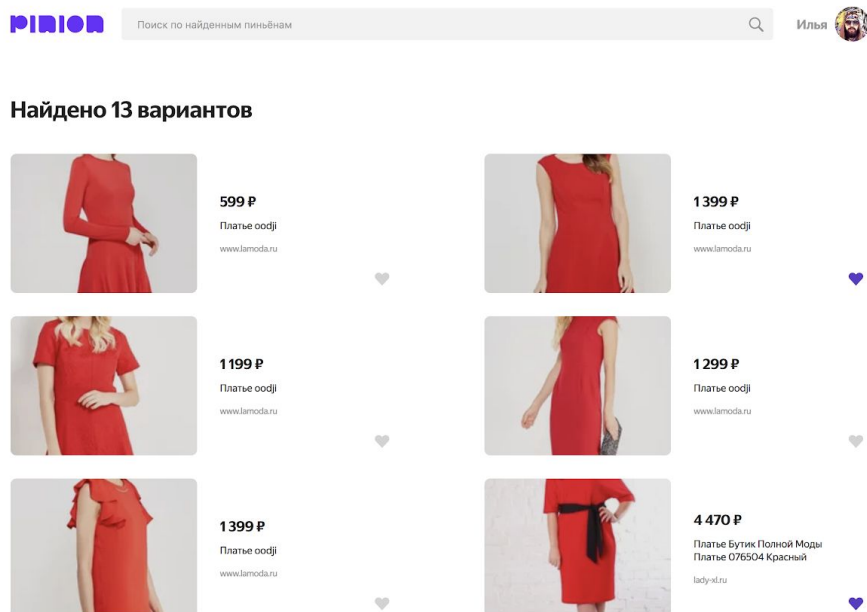
Описание интерфейса приложения

При инициализации приложения, пользователь попадает на стартовую страницу с описанием, с которой он может залогиниться, а также скачать браузерное расширение **Pinion**.




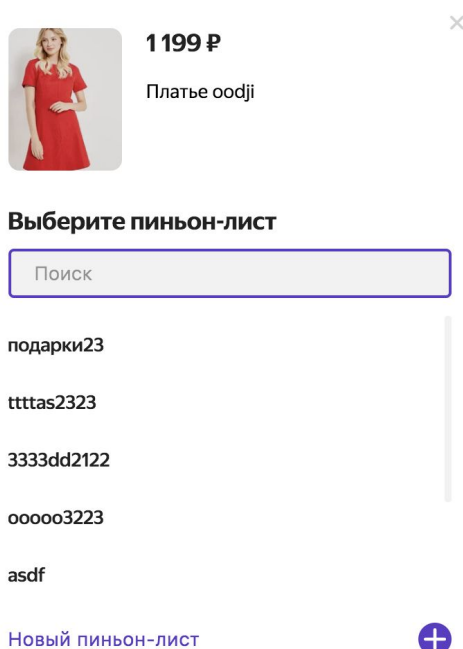
В случае перехода из расширения, пользователь попадает на страницу выдачи, где из данных, обработанных на сервере, сформированы карточки товара.

С каждой карточки можно перейти на страницу товара в маркете, или добавить товар в свои коллекции.



В верхней части страницы расположена строка поиска, позволяющая отфильтровать найденные товары по названию.

При нажатии на , появляется модальное окно с возможностью добавления карточки в коллекцию



В окне можно отфильтровать коллекции по названию, добавить карточку в любую из них, а также создать новую коллекцию буквально в два клика.

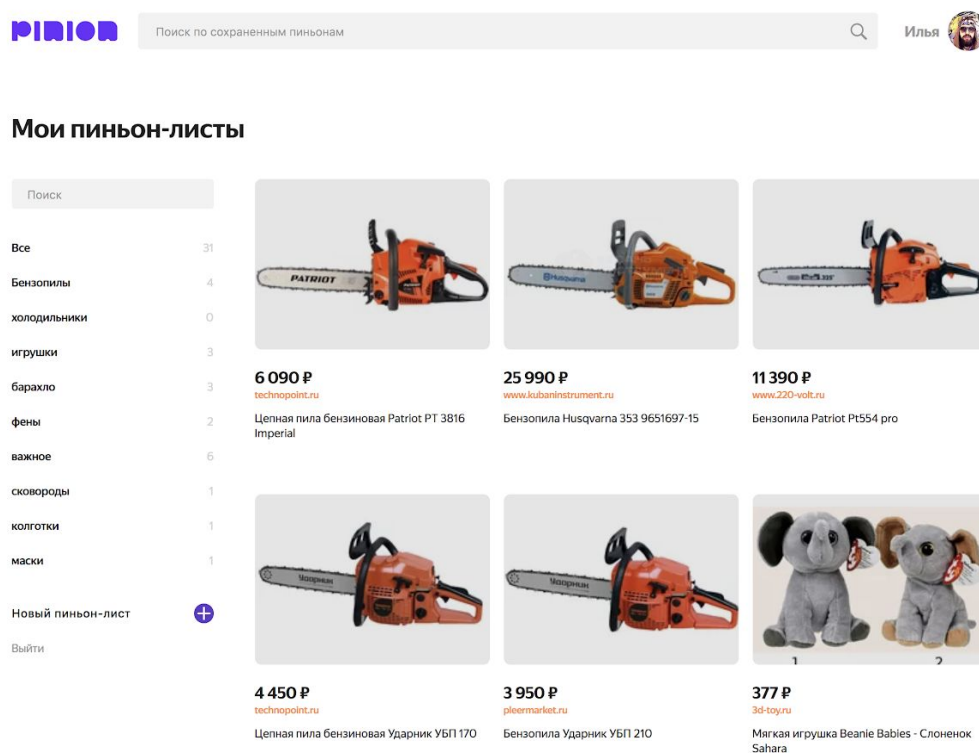
Все действия пользователя и ошибки выводятся всплывающими алертами в правом

верхнем углу экрана:

Пиньон сохранен!

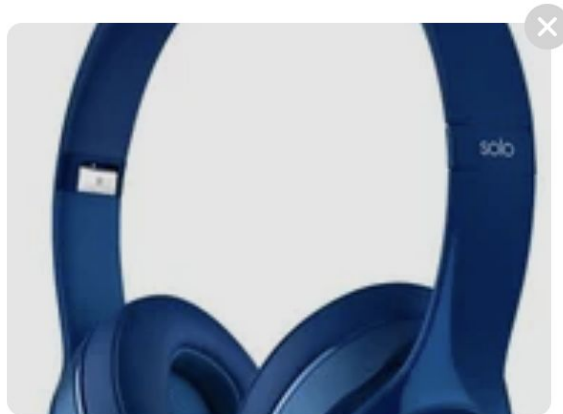
При нажатии на  пользователь может удалить карточку из своих списков.

Справа от строки поиска расположены имя и аватар пользователя или, в случае отсутствия авторизации, кнопка “войти”. При клике на аватар, пользователь переходит на личную страницу с сохраненными карточками.



The screenshot displays the Pinion application interface. At the top, there is a search bar with the text "Поиск по сохраненным пиньонам" and a magnifying glass icon. To the right of the search bar is the user's name "Илья" and a profile picture. Below the search bar, the section "Мои пиньон-листы" is visible. On the left side, there is a sidebar with a search filter "Поиск" and a list of categories with their respective counts: "Все" (31), "Бензопилы" (4), "холодильники" (0), "игрушки" (3), "баракло" (3), "фены" (2), "важное" (6), "сковороды" (1), "колготки" (1), "маски" (1), "Новый пиньон-лист" (indicated by a plus icon), and "Выйти". The main area shows a grid of six items. The first five items are chainsaws: "Цепная пила бензиновая Patriot PT 3816 Imperial" (6 090 P), "Бензопила Husqvarna 353 9651697-15" (25 990 P), "Бензопила Patriot PT554 pro" (11 390 P), "Цепная пила бензиновая Ударник УБП 170" (4 450 P), and "Бензопила Ударник УБП 210" (3 950 P). The sixth item is a "Мягкая игрушка Beanie Babies - Слоненок Sahara" (377 P). Each item has a small image, a price tag, and a link to the source website.

Здесь пользователь может перемещаться по своим коллекциям, а также пользоваться сортировкой, которая фильтрует карточки по названиям и отображает общее количество найденных карточек и их количество в каждой из категорий, которые также можно отфильтровать.




9 950 ₽

www.xcom-shop.ru

Наушники Apple Beats Solo2 On-Ear Headphones Blue (MHB12ZE/A)

[Переместить](#) [Поделиться](#)

Реализована возможность удаления карточек по клику на  и перемещения между коллекциями по клику на “переместить”.

При клике на “поделиться”, открывается модальное окно с возможностью поделиться ссылкой на товар в Яндекс.Маркете в соцсетях или скопировать ее в буфер обмена.



9 950 ₽

Наушники Apple Beats Solo2 On-Ear Headphones Blue...

Отправить пиньон



При выборе определенной коллекции, справа сверху над карточками появляются кнопки “переименовать” и “удалить”, выполняющие соответствующие действия.

Сервер

Архитектура

Серверная часть проекта написана на платформе **Node JS** с использованием **Typescript**. Как серверный фреймворк был выбран **Express**. В качестве базы данных задействована **MongoDB** и **Mongoose ODM**. В роли временного хранилища результатов поиска по изображениям выступает **Redis**. Изображения, сконвертированные и сохраненные на диск для возможности предоставить на них статическую ссылку, удаляются сразу после получения результата поиска через API Яндекса. С помощью **Winston** осуществлено логирование ошибок. Для авторизации используется **passport.js** со стратегией Яндекс-паспорта.

API

Загрузка картинки

/api/image/upload

POST

Request:

```
imgUrl: string; (base64 изображение)
```

Response:

```
{
  searchId: string; // id поисковой сессии
  cards: [
    {
      image: string; // ссылка на изображение
      link: string; // ссылка на товар в маркете
      price: string; // цена
      title: string; // название
      modelId: string; // id модели
      domain: string; // сайт продавца
    }
  ]; // 5 наиболее релевантных запросов
  cardsCount: number; // количество найденных товаров
  serviceLink: string; // линк для перехода на страницу выдачи
}
```

Получение результатов поиска

/api/image/\${searchId}

GET

Response:

```
{
  searchId: string; // id поисковой сессии
  cards: [
    {
      image: string; // ссылка на изображение
      link: string; // ссылка на товар в маркете
      price: string; // цена
      title: string; // название
      modelId: string; // id модели
      domain: string; // сайт продавца
    }
  ]
}
```

Добавление товара в коллекцию

/api/wishlist/add

POST

Request:

```
searchId: string; // id поисковой сессии
modelId: string; // id модели
collectionId: ?string; // id коллекции
```

Response:

```
status: 'OK';
card: {
  _id: string; // id из монги
  image: string; // ссылка на изображение
  link: string; // ссылка на товар в маркете
  price: string; // цена
  title: string; // название
  modelId: string; // id модели
  domain: string; // сайт продавца
  collectionId: ?string; // id коллекции
}
```



```
// В случае неудачи отправляется только поле status:

// товар уже в вишлисте, код 400
status: 'ALREADY EXISTS';

// неверный запрос (несуществующий searchId и т.д.), код 400
status: 'BAD REQUEST';

// нужна авторизация, код 401
status: 'NEED AUTH';

// ошибка сервера, код 500
status: 'SERVER ERROR';
```

Изменить коллекцию карточек

/api/wishlist/edit

POST

Request:

```
id: string; // _id карточки
collectionId: ?string; // id коллекции
```

Response:

```
// Совпадает с ответом ручки wishlist/add за тем исключением,
// что нет статуса "ALREADY EXISTS"
```

Удалить карточку из коллекции

/api/wishlist/delete

POST

Request:

```
id: string; // _id карточки
```

Response:

```
// Совпадает с ответом ручки wishlist/edit
```

Добавление коллекции

/api/collections/add

POST

Request:

```
title: string; // название коллекции
```

Response:

```
status: 'OK';
collection: {
  _id: string; // id из монги
  title: string; // название коллекции
}

// В случае неудачи отправляется только поле status:

// коллекция уже существует, код 400
status: 'ALREADY EXISTS';

// отсутствует title, код 400
status: 'BAD REQUEST';

// нужна авторизация, код 401
status: 'NEED AUTH';

// ошибка сервера, код 500
status: 'SERVER ERROR';
```

Изменить название коллекции

/api/collections/edit

POST

Request:

```
id: string; // _id коллекции
title: string; // новое название
```

Response:

```
// Совпадает с ответом ручки collection/add за тем исключением,  
// что нет статуса "ALREADY EXISTS" и статус "BAD REQUEST"  
// отправляется в случае неверного id
```

Удалить коллекцию

/api/collections/remove

POST

Request:

```
id: string; // _id коллекции
```

Response:

```
// Совпадает с ответом ручки collections/edit
```

Получить коллекции и сопутствующие данные

/api/wishlist/

GET

Response:

```
{  
  user: {  
    name: string; // имя пользователя  
    photo: string; // ссылка на фото  
  },  
  cards: [  
    {  
      _id: string; // id из монги  
      image: string; // ссылка на изображение  
      link: string; // ссылка на товар в маркете  
      price: string; // цена  
      title: string; // название  
      modelId: string; // id модели  
      domain: string; // сайт продавца  
      collectionId: ?string; // id коллекции  
    }  
  ]  
  collections: [  
    {  
      _id: string; // id из монги  
      title: // название коллекции  
    }  
  ]  
}
```

```
// если user не авторизован:  
{  
  user: null;  
}
```

```
// если ошибка сервера:  
{  
  error: 'SERVER ERROR'  
}
```

Инфраструктура

Сборка, линтеры

Для поддержания единого стиля кода в команде были использованы линтеры [TypeScript линтер \(TSLint\)](#), [JavaScript линтеры \(ESLint\)](#), [CSS линтер \(stylelint\)](#) и [линтер коммитов \(commitlint\)](#). Для унификации настроек IDE использован [\(editorconfig\)](#). Для автоматического форматирования кода [Prettier](#) и [CSScomb](#). Запуск всех линтеров происходит автоматически с помощью [husky](#). [Commitlint запускается при написании текста комита](#), а [TSLint, ESLint, stylelint перед самим комитом](#), коммит проходит успешно только если весь код и текст коммита соответствует правилам установленным в команде.

Сборка проекта разделена на сборку [браузерного расширения Pinion](#) и сборку [сервиса Pinion](#). При сборке [браузерного расширения Pinion](#) использован [webpack 4](#) с модулями для компиляции TypeScript и SCSS, а сама сборка разделена на сборку под браузеры Chrome/Yandex и Firefox. Для сборки сервиса использован [react-app-rewired](#) в котором была подключена поддержка SCSS и Inline SVG.

CSS codestyle

Именования

Именования селекторов осуществляется по [методологии БЭМ](#), а именно в стиле [Two Dashes](#): `block-name__element--modifier`

Синтаксис

- Длина отступа равна 4 пробелам. Для правильного форматирования в редакторе / IDE в проекте существует файл `.editorconfig`.
- После любой запятой ставится один пробел.
- В коде не идёт подряд больше одной пустой строки.
- После двоеточия в правилах ставится один пробел. А перед двоеточием пробел не нужен. `top: 10px;`
- Каждое объявление в правиле пишется на новой строке.

- Перед открывающейся фигурной скобкой ставится один пробел. После скобки запись идёт с новой строки.
- Закрывающая фигурная скобка пишется на новой строке и без отступа. Следующее после этого правило отделяется пустой строкой.
- Единицы измерения не пишутся, там где в них нет необходимости. Например: `border: 0`
- Рекомендуется не опускать лишний ноль.
 - *Неправильно:* `margin: .5em;`
 - *Правильно:* `margin: 0.5em;`
- Псевдо-элементы (такие, как `::before`, и `::after`) должны начинаться с двух двоеточий (`::`).
- Псевдо-классы (такие, как `:hover` и `:first-child`) должны начинаться с одного двоеточия (`:`).
- Математические операторы (такие, как `+`, `-`, `*` и т.д.) должны быть отделены с обеих сторон одним пробелом: `margin: 5px + 5px;`
- Текст помещается внутрь одинарных скобок: `font-family: "YS Text";`

Правила CSS

- Для стилизации не используются *`#id`*, только классы.
- Запрещено использование *`!important`*.
- Цвета следует задавать в hex long lowercase (`#f7f7f7`), а не ключевыми словами вроде `green`.
- Не должно быть пустых блоков свойств (конструкция вида `.selector { }`)
- Следует использовать короткую форму записи свойств:
 - *Неправильно:* `margin: 20px 15px 20px 15px;`
 - *Правильно:* `margin: 20px 15px;`

- Одна строка — один селектор:
- Не нужно самостоятельно писать вендорные префиксы, поскольку в сборку включен autoprefixer.

Правила SASS

- Мы используем версию SCSS (со скобками "{", "}" и ";").
- Максимальный уровень вложенности - 3.
- Вместо явного указания медиа-запросов следует использовать установленные в проекте миксины, например `@include mobile`
- **import** стилевых файлов не должен содержать нижнее подчеркивание и расширение файла:
 - *Неправильно:* `@import "foo/_bar.scss";`
 - *Правильно:* `@import "foo/bar";`

Порядок свойств

Мы придерживаемся [идеи разбиения свойств на логические блоки](#).

Объявления логически связанных свойств группируются в следующем порядке(согласно конфигу .csscomb.json):

1. Позиционирование
2. Блочная модель
3. Анимация
4. Оформление
5. Типографика
6. Разное

CI

В качестве CI платформы использовался TeamCity, работающий на площадке Яндекса. Интеграция была настроена для тестирования [браузерного расширения Pinion](#). Запуск CI происходит при создании pull requests в Bitbucket репозитории [браузерного расширения Pinion](#), CI проверяет соответствие кода pull requests стилю установленному в команде, запуская [TypeScript линтер \(TSLint\)](#) и [CSS линтер \(stylelint\)](#). После успешной проверки линтерами, запускаются модульные тесты, после успешного прохождения которых, pull request попадает из ветки feature в ветку develop. Не удалось настроить CI для сервера и сервиса, т.к была возможность сделать его только для одного репозитория.

Тестирование

Для тестирования **Pinion** были написаны модульные тесты и интеграционные тесты.

WEB-приложение было протестировано с помощью интеграционных тестов. Для написания интеграционных тестов была использована [Hermione](#) и assert, при написании тестов возникли сложности.

Первая проблема была в том что выдача товаров найденных Pinion хранится в течении часа в Redis поэтому при старте тестов нужно было постоянно получать новую выдачу. Данная проблема была решена при помощи [beforeEach](#) в котором был fetch запрос на сервер за актуальным url страницы на которой запускались тесты

Вторая проблема была в том что для добавления товаров в коллекцию необходимо авторизоваться через Яндекс-Паспорт. Было решено запускать сервер в с подменой авторизации через passport-dummy и прогонять тесты с фейковыми данными пользователя. Данный функционал не был реализован из-за нехватки времени.

Браузерное расширение было протестировано с помощью unit-тестов. Для запуска тестов был использован test-runner - [Karma](#), фреймворк - [Mocha](#), библиотека - [chai](#), для заглушек использован [Sinon](#).

Была попытка тестирования работы с DOM в расширении с помощью [mocha-jsdom](#), но возникли проблемы с типами в typescript. Также возникли сложности

с интеграционным тестированием расширения. Были предприняты разные попытки заставить глобальный объект chrome. В результате это получилось сделать с помощью [sinon-chrome](#). Таким образом, у нас появилась возможность протестировать обработчики событий, но данный функционал не был реализован из-за нехватки времени.

Документация

Для разработки проекта были использованы следующие ресурсы:

- <https://github.com/>
- <https://reactjs.org/>
- <https://redux.js.org/>
- <https://developer.chrome.com/extensions/devguide>
- Документация задействованных в проекте библиотек
- При возникновении сложностей в процессе разработки или поиске информации, очень помогала проработка вопросов с кураторами

Командная работа

Для работы над проектом была использована гибкая методология разработки **Agile** и фреймворк **Scrum**. Почти для всех нас это был первый опыт работы в команде и изначально мы опирались в основном на документацию и информацию из открытых источников, но, после тренинга по Scrum, значительно улучшили свои навыки, что положительно сказалось на процессе разработки.

В нашем случае дизайнер выступал также в роли менеджера. В процессе разработки мы постоянно находились на связи и принимали решения сообща, постепенно внося необходимые изменения и нововведения в проект.

В начале каждого спринта в порядке очередности должность scrum-мастера занимал новый член команды. Это было сделано для того, чтобы каждый из нас мог почувствовать свою долю ответственности и получить первый опыт в этой сфере.

Процесс разработки был спланирован следующим образом:

- Составление бэклога продукта
- Спринты длительностью в неделю
- Планирование спринта в начале недели
- Ежедневный стэндап
- Дополнительная встреча с кураторами в течение недели
- Дополнительные синки в течение дня по необходимости
- Общий чат в telegram
- Ретроспектива на следующий день после ШРИкатона

Для планирования и документации были использованы Яндекс-Вики и Яндекс-Трекер. Был составлен подробный Flow проекта, где были описаны соглашения по именованиям, код-стайл и подробно расписан [git-flow](#). Все задачи спринта прописывались в трекере, а их номера использовались при именовании веток и коммитов. Для каждой задачи назначался исполнитель и наблюдатели.

Мы постарались в полной мере реализовать возможности командной работы, поэтому в течение спринта нагрузка была распределена следующим образом: по два человека на один из подпроектов для повышения эффективности с помощью

парного программирования, и один проект закреплен за оставшимся участником. Также со сменой спринта происходила ротация участников между подпроектами для того, чтобы каждый член команды мог принимать участие в разработке каждого компонента проекта и получить новые знания в процессе разработки. Перед каждым вливанием веток заданий проводился код-ревью проделанной работы.

Личные результаты и достижения

Мария Вершинина:

Получила опыт:

- разработки и тестирования браузерных расширений;
- работы с TypeScript;
- работы с сервером;
- парного программирования и работы в команде;
- командной работы с git;
- работы по Scrum и git-flow;

Саян Жакупбеков:

- научился писать браузерные расширения для chrome и firefox;
- прикоснулся к React, Redux;
- попробовал себя в TypeScript;
- потренировал парное программирование и опробовал git flow;
- настройка vps сервера;

Илья Борисов:

- Навык работы в команде
- Опыт парного программирования
- Опыт код-ревью
- Опыт взаимодействия с менеджером/дизайнером продукта
- Знакомство с гибкой методологией разработки
- Опыт разработки расширений для браузеров

- Опыт работы по git-flow
- Опыт взаимодействия с командой и дизайнером при разработке каждой части продукта (Очень ценно то, что ты можешь внести свои поправки в любые мелочи, даже в UX продукта. Круто не только кодить, но и влиять на продукт в целом)

Гордеев Валерий:

- изучил API браузеров
- научился разрабатывать браузерные расширения
- глубже узнал react и познакомился с redux
- изучил TypeScript
- получил знания о том как тестировать браузерные расширения
- улучшил навыки написания интеграционных тестов с помощью Hermione
- познакомился с Agile и фреймворком Scrum
- попробовал себя в роли Scrum мастера
- получил опыт парного программирования
- получил опыт проведения код ревью
- получил опыт работы и взаимодействия с дизайнером продукта
- изучил wiki разметку и попробовал [писать статьи в Яндекс wiki](#)
- освоил работу в Яндекс трекере.

Иван Воронин:

- получил опыт разработки браузерных расширений
- попрактиковал написание на TypeScript
- освоил работу с нереляционными БД (mongo)
- попробовал парное программирование
- познакомился с redux в бою (до этого юзал только flow)
- получил опыт работы по настоящему git-flow
- улучшил коммуникативный навык при взаимодействии с командой

Итоги

Подводя финальную черту, самым главным можно отметить, что каждый из нас получил незабываемый и очень ценный опыт работы в команде под руководством опытных разработчиков. Нам удалось на 100% выполнить изначальный MVP и даже перевыполнить его в некоторых местах. Конечно проект далек от завершения, но это полноценный продукт с готовым функционалом, которым может воспользоваться любой желающий.

В дальнейшем, мы могли бы осуществить поддержку абсолютно всех браузеров и реализовать desktop-приложение с помощью Proton-Native/Electron.

Мы очень надеемся, что наш прототип, претерпев те или иные изменения, когда-нибудь будет запущен как полноценный сервис Яндекса, а кто-нибудь из нас будет иметь непосредственное отношение к его дальнейшей разработке.