FINAL REPORT

Project
WEB scraper for online stores
Team Members
Nikita Ruchkin (n.ruchkin@innopolis.university)
GitHUB
https://github.com/reterman/test_project.git

PART 1: BRIEF DESCRIPTION OF THE PROJECT IDEA

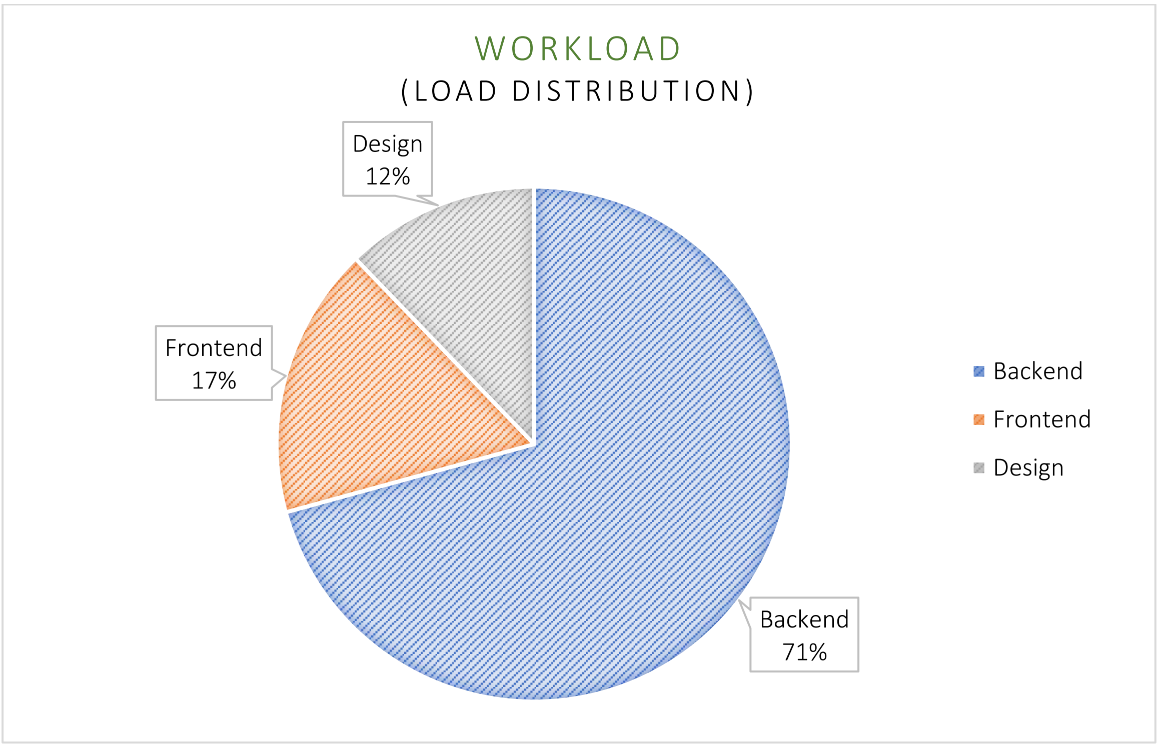
A price comparison tool that uses web scraping to gather pricing information from various online retailers for a specific product, and provides the user with a side-by-side comparison of the prices, allowing them to save money and make informed purchasing decisions.

PART 2: WORKLOAD

2.1. Workload distribution among team members

Since I am the only participant in my team, I performed every part of the project development on my own.

During the four sprints, I spent a total of 20,5 hours.



Result:
Backend: 14,5 hours.
Frontend: ,5 hours.
Design: 2,5 hours.

2.2. Tasks

#	Brief description	Hours spent
#1	Studying and connecting libraries and packages for scraping websites (request, selenium, bs4)	2
#2	Studying and installing libraries and packages to implement a web application (flask, django)	2
#3	Implementation of web pages with minimal functionality to test the functionality	1
#4	Added four popular Russian marketplaces from which the scraper searches for product data (Wildberries.ru, Ozon.ru, KazanExpress.ru, Cdek.shopping)	4
#5	Updated the method of transmitting product data to the site	0,5
#6	Added the ability to run all scrapers at the same time	1
#7	Improved site performance by saving the previous query	0,5
#8	Made buttons for sorting products	1
#9	Changed the product rating systems	0,5
#10	Made a check for the correctness of entering data into the search bar	0,5
#11	Improved page loading time using different methods	1,5
#12	Study CSS framework for quick and quality method for implementing fronted part of the project (Bulma)	2
#13	Applying CSS framework in practice	1,5
#14	Finalized the website design	1,5
#15	Adapted the design to different screen formats	1

2.3. Detailed description of the tasks

#1 Studying and connecting libraries and packages for scraping websites (request, selenium, bs4)

Requests:
Request is used for web scraping to download the HTML content of the web page.

Selenium:
Selenium is used for web scraping when the web page requires user interaction or has dynamic content that is loaded using JavaScript.

BeautifulSoup4 (bs4):
Once you have the HTML content of the web page, you can use BeautifulSoup to extract the relevant information.

#2 Studying and installing libraries and packages to implement a web application (flask, django)

Flask:
Flask is a lightweight web framework for Python that allows you to quickly build web applications.

Django:
Django is a more comprehensive web framework for Python that provides a powerful and scalable architecture for creating web applications.

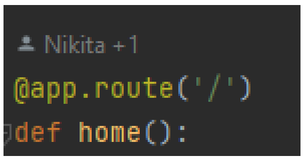
After studying both technologies I settled on flask. However, in the future, if there is a need to use a database, I will have to consider a tool like Django.

#3 Implementation of web pages with minimal functionality to test the functionality

Creating HTML templates:

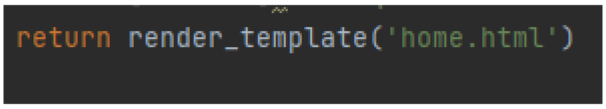
To create web pages using Flask, I need to create HTML templates that define the structure and content of the pages.

Defining routes:
To define routes in Flask, I can use the "@app.route" decorator and specify the URL path for the route. For example, "@app.route('/home')" defines a route for the home page of the web application.
Path: app.py.
Lines: 20, 30.



```
Nikita +1
@app.route('/')
def home():
```

Defining views:
To define views in Flask, I need to create Python functions that return the HTML content for each route. I can use the render_template function to render the HTML templates and pass any necessary data to the templates.
Paths: app.py.
Lines: 23, 58, 60.



```
return render_template('home.html')
```

#4 Added four popular Russian marketplaces from which the scraper searches for product data (Wildberries.ru, Ozon.ru, KazanExpress.ru, Cdek.shopping)

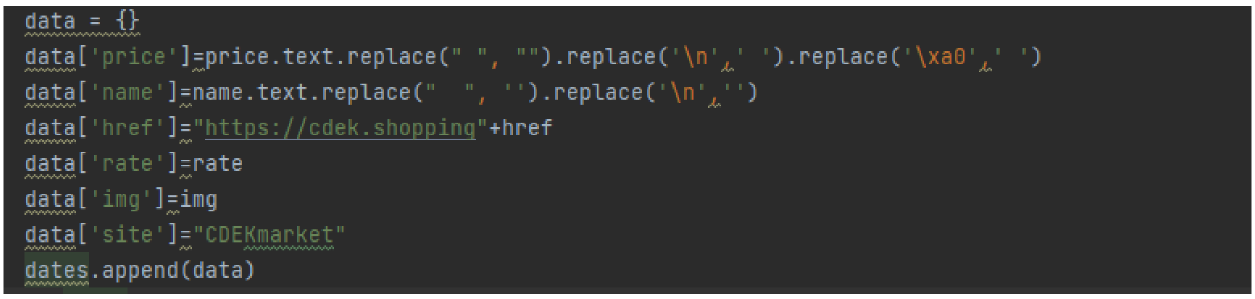
To connect these libraries and packages, I would typically start by using Requests to download the HTML content of the web page. However, most modern web sites use dynamic content, I may need to use Selenium to automate the web browser. Once I have the HTML content, I can use BeautifulSoup to extract the relevant information.

Through the study and analysis of dynamic css-classes, I discovered a constant pattern in their name. Thanks to this vulnerability, I was able to compose an expression by which I managed to get the necessary data about each product with Ozon.ru, KazanExpress.ru, Cdek.shopping, Wildberries.ru.
Files:

- CDEKmarket.py
- WildBerries.py
- KazanExpress.py
- Ozone_Market.py

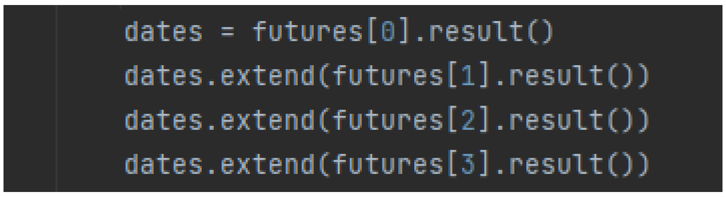
#5 Updated the method of transmitting product data to the site

The early implementation of the data collection system by writing them in a notebook did not justify itself with its complexity. As a result, for a more convenient way of combining data and further manipulations with them, the idea was proposed to use lists.
Example:
Paths: CDEKmarket.py.
Lines: 83-90.



```
data = {}
data['price']=price.text.replace(" ", "").replace('\n',' ').replace('\xa0',' ')
data['name']=name.text.replace("  ", '').replace('\n','')
data['href']="https://cdek.shopping"+href
data['rate']=rate
data['img']=img
data['site']="CDEKmarket"
dates.append(data)
```

Paths: create_data.py
Lines: 35-38.

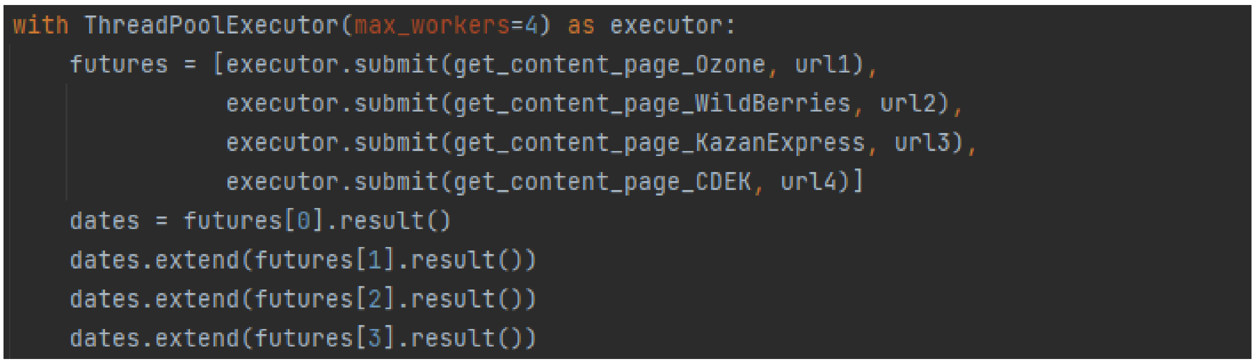


```
    dates = futures[0].result()
    dates.extend(futures[1].result())
    dates.extend(futures[2].result())
    dates.extend(futures[3].result())
```

#6 Added the ability to run all scrapers at the same time

For a more efficient and user-friendly way, it was decided to allocate an additional file in which the ability to allocate multiple threads to run all scrapers simultaneously was implemented. This method has improved performance and will allow you to spend less time on processing.

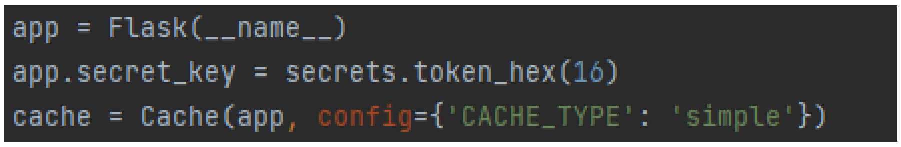
Paths: create_data.py
Lines: 30-38.



```
with ThreadPoolExecutor(max_workers=4) as executor:
    futures = [executor.submit(get_content_page_Ozone, url1),
               executor.submit(get_content_page_WildBerries, url2),
               executor.submit(get_content_page_KazanExpress, url3),
               executor.submit(get_content_page_CDEK, url4)]
    dates = futures[0].result()
    dates.extend(futures[1].result())
    dates.extend(futures[2].result())
    dates.extend(futures[3].result())
```

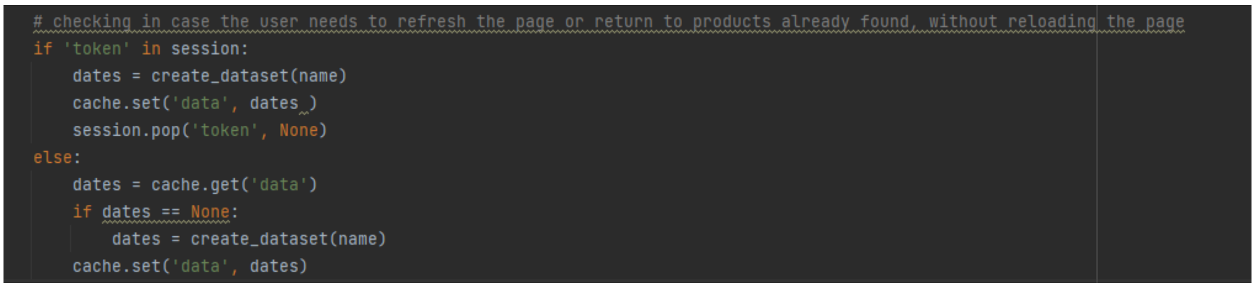
#7 Improved site performance by saving the previous query

In the course of the work, a vulnerability was discovered related to the re-launch of all search functions in the event of a page refresh or a return to the previous one (in cases of the user's desire to continue viewing the list of found products). Using flask-caching and session, I added a check for event data and, if any are detected, instead of running the scraper, I use cached data.
Path: app.py.
Lines: 11-13.
Create token and cache:



```
app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
cache = Cache(app, config={'CACHE_TYPE': 'simple'})
```

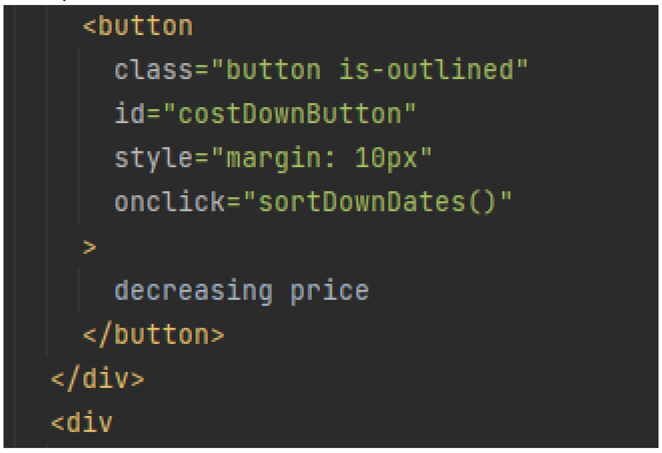
Lines: 47-55
Checking:



```
# checking in case the user needs to refresh the page or return to products already found, without reloading the page
if 'token' in session:
    dates = create_dataset(name)
    cache.set('data', dates )
    session.pop('token', None)
else:
    dates = cache.get('data')
    if dates == None:
        dates = create_dataset(name)
    cache.set('data', dates)
```

#8 Made buttons for sorting products

Buttons were created to sort products by price and rating. When you click on the buttons, JavaScript receives data about the products from the server and sorts them according to the selected parameter. After sorting the items, JavaScript displays the sorted data on the web page according to the selected sort parameter. Thanks to this, the user can easily and quickly find the desired product and compare prices and ratings between different shops.
Paths: templates/scrape.html
Lines: 44-98, 194-307.
Example:



```
    <button
      class="button is-outlined"
      id="costDownButton"
      style="margin: 10px"
      onclick="sortDownDates()"
    >
      decreasing price
    </button>
  </div>
  <div
```

decreasing price
increasing price
decreasing rating
increasing rating

#9 Changed the product rating systems

Some online shops lack an adequate product rating system or do not have one at all. Therefore, in order to ensure uniformity of all product cards on the site, a system of replacing empty values with zero followed by a description was developed.

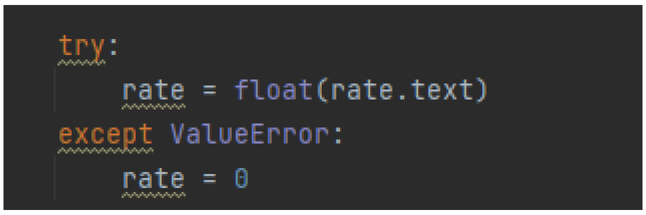
For example, if the online shop does not have a rating for a particular product, then instead of this value will be substituted the number 0 and the description "no rating". This allows to ensure the uniformity of displaying information about products on the site and provide the user with more complete information about products, even if this information is not available on the original sites.

Thus, changing the system of rating of goods on the site allowed to create more complete and uniform information for users who are looking for goods on the site.

Files:

- CDEKmarket.py
- WildBerries.py
- KazanExpress.py
- Ozone_Market.py

Example:
Path: WildBerries.py
Lines: 91-94.



```python
try:
    rate = float(rate.text)
except ValueError:
    rate = 0
```

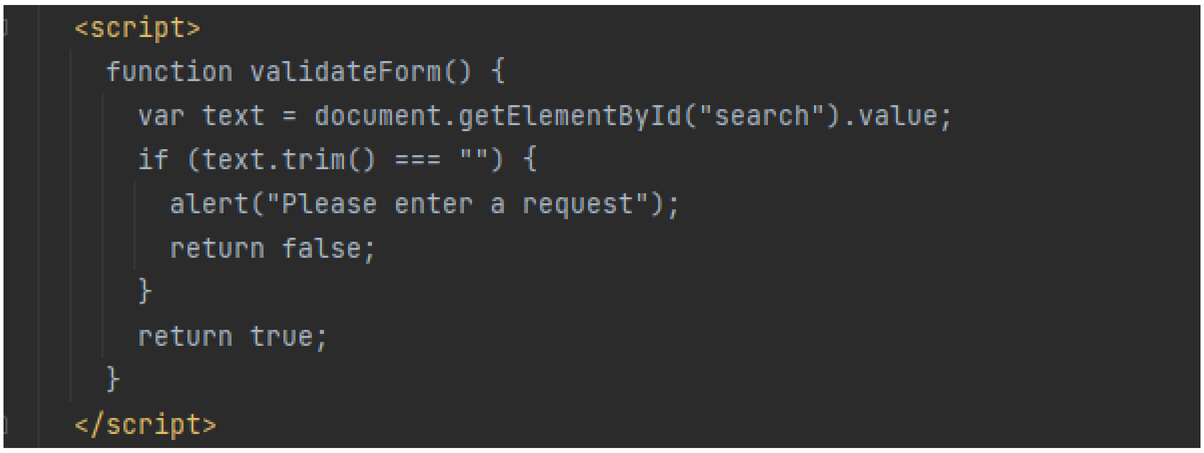
#10 Made a check for the correctness of entering data into the search bar

The JavaScript programming language was used to perform this task. A function was created that checks the search string entered by the user for a query. If the string does not match a normal query, the function displays an error message and prompts the user to try again.

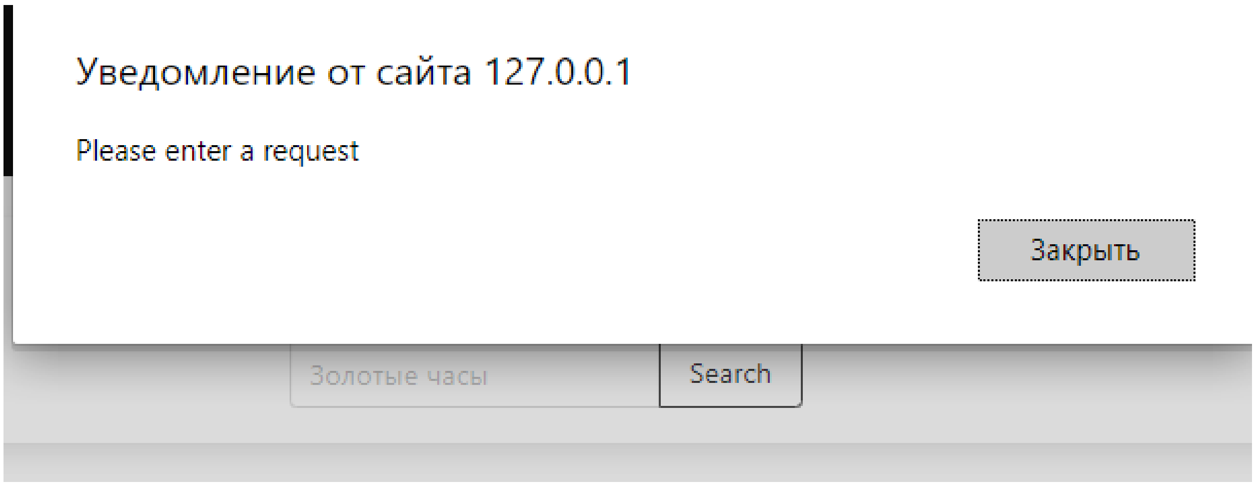
Regular expressions were used to verify that the data entered was correct. Regular expressions are a powerful tool for working with text that allows you to check a string for compliance with a specified pattern.

For example, if the search string contains only spaces. the function displays an error message and does not send the request to the server.

Thus, creating a function to check the correctness of entering data into the search string prevents incorrect requests from being sent to the server and improves the performance of the web application.
Path: templates/home.html
Lines: 19-26, 38-53.



```html
<script>
  function validateForm() {
    var text = document.getElementById("search").value;
    if (text.trim() === "") {
      alert("Please enter a request");
      return false;
    }
    return true;
  }
</script>
```


Уведомление от сайта 127.0.0.1

Please enter a request

#11 Improved page loading time using different methods

first method: analyzed and derived the minimum value of time needed to load all sites and used it to pause the process (time.sleep)

second method: use explicit waiting. The ExpectedConditions method allows you to wait for various events, such as an element loading, an element value changing, an element appearing on the page, etc. Detected and set the observation for each marketplace loading necessary tags and classes to minimize the time spent.

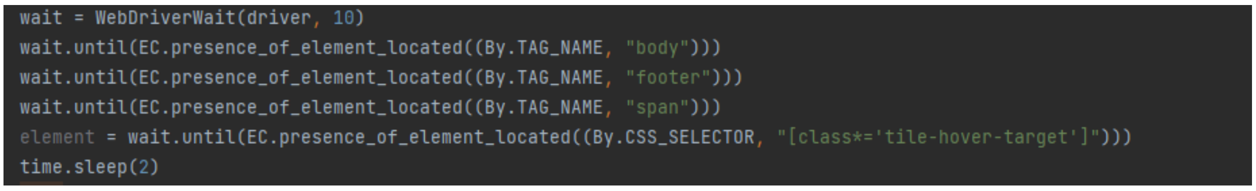
Files:

- CDEKmarket.py
- WildBerries.py
- KazanExpress.py
- Ozone_Market.py

Example:

Path: Ozone_Market.py

Lines: 35-40.



```
wait = WebDriverWait(driver, 10)
wait.until(EC.presence_of_element_located((By.TAG_NAME, "body")))
wait.until(EC.presence_of_element_located((By.TAG_NAME, "footer")))
wait.until(EC.presence_of_element_located((By.TAG_NAME, "span")))
element = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "[class*='tile-hover-target']")))
time.sleep(2)
```

#12 Study CSS framework for quick and quality method for implementing fronted part of the project (Bulma)

Bulma:

Bulma is a modern CSS framework that is based on Flexbox and provides a range of pre-built UI components and styles for creating responsive web pages.

#13 Applying CSS framework in practice

Using ready-made templates and css-classes, quickly managed to implement a nice-looking initial interface.

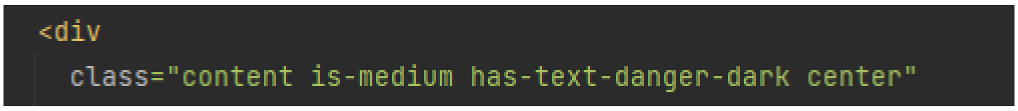
Files:

- templates/home.html
- templates/scrape.html

Example: all css classes

Paths:

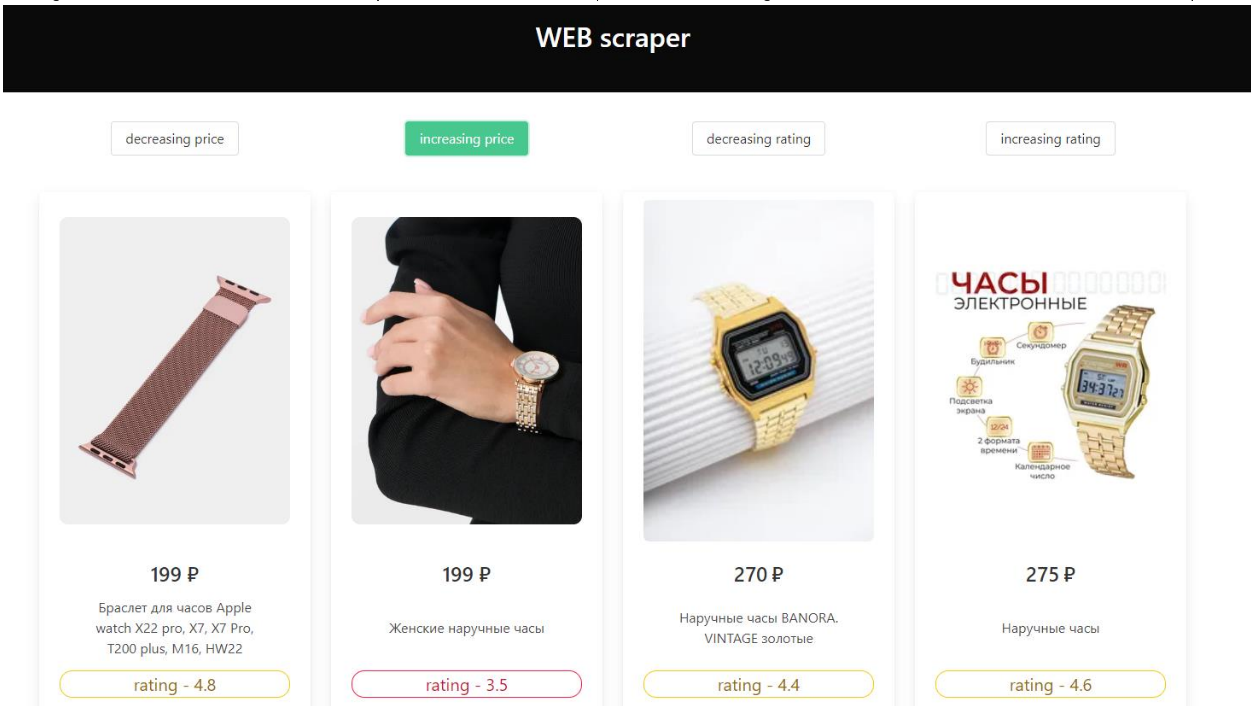
- templates/home.html
- templates/scrape.html



```
<div
  class="content is-medium has-text-danger-dark center"
```

#14 Finalized the website design

Using the Bulma framework and my own css classes, I updated the design of the site, made it more user-friendly.



WEB scraper

decreasing price | increasing price | decreasing rating | increasing rating

199 ₽
Браслет для часов Apple watch X22 pro, X7, X7 Pro, T200 plus, M16, HW22
rating - 4.8

199 ₽
Женские наручные часы
rating - 3.5

270 ₽
Наручные часы BANORA. VINTAGE золотые
rating - 4.4

275 ₽
Наручные часы
rating - 4.6

#15 Adapted the design to different screen formats

With the help of the built-in bulma css classes, I was able to customize the design of my product for different devices: from mobile screens to full hd monitors.

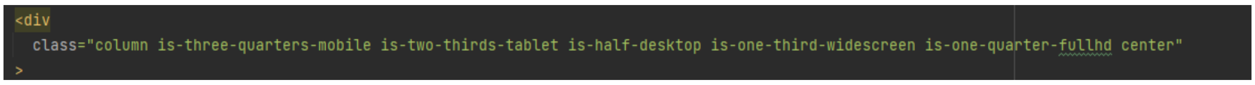
Files:

- templates/home.html
- templates/scrape.html

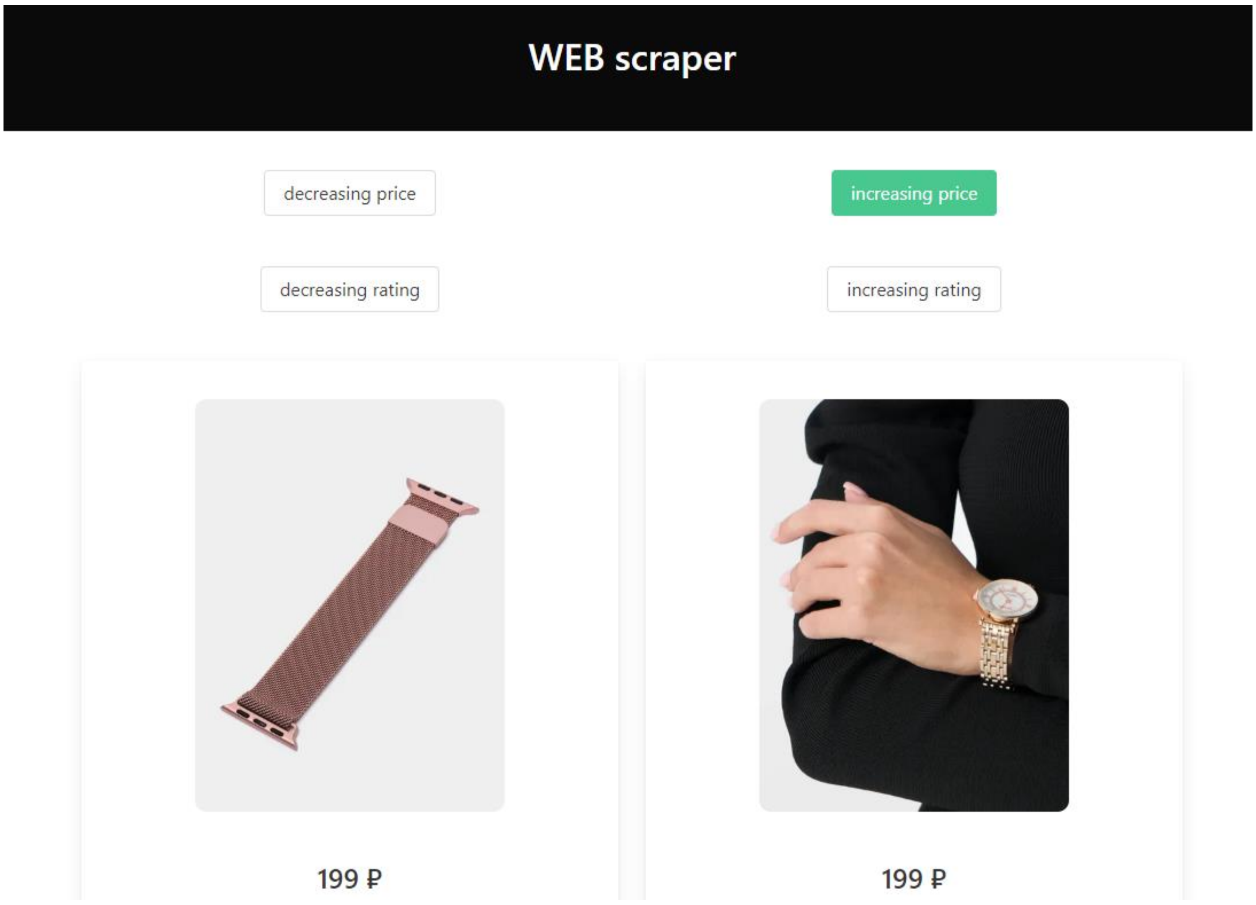
Example: all css classes

Paths:

- templates/home.html
- templates/scrape.html



```
<div
  class="column is-three-quarters-mobile is-two-thirds-tablet is-half-desktop is-one-third-widescreen is-one-quarter-fullhd center"
>
```



PART 3: AN OBSTACLE THAT OCCURRED DURING THE PROJECT

#1 The lack of necessary skills for the implementation of the project, resulting in a waste of resources for their study.

#2 Vulnerability detection pulling information out of html code, as a result of dynamic changes in the naming of css-classes.

#3 Lack of compilation of the final stack of technologies used.

#4 It was discovered that some popular marketplaces use captcha verification, which may become a serious problem in the future.

#5 There are quite a few effective ways to improve the speed of Seleniuma.

#6 Lack of adaptation of the site for different device formats.

#7 Unfortunately, Bulma is still not able to provide an opportunity to completely change the existing system according to the wishes of users. Because of which the design of the system becomes limited.

#8 There are quite a few effective ways to improve the speed of Seleniuma.

PART 4: THE MAIN TASKS FOR THE FOLLOWING SPRINTS

#1 Add the ability for users to track price changes for a specific product.

#2 Add the ability to partially compare products by characteristics.

#3 Refactor the application as a Google extension for faster and more convenient access.

#4 Automatically provide information about searched products when users visit marketplaces.

#5 Add the ability to obtain JSON files or use marketplace APIs to obtain product data, significantly speeding up data processing.

PART 3: THE FINAL LOAD OF THE TEAM :)

Nikita Ruchkin	Nikita Ruchkin	Nikita Ruchkin	Nikita Ruchkin
100%	100%	100%	100%