



BLM-3120

INFORMATION RETRIEVAL & WEB SEARCH  
ENGINES TERM PROJECT

HYBRID SITE FOR TABLET FILTERING WITH WEB CRAWLING

Büşra Medine GÜRAL

20011038

medine.gural@std.yildiz.edu.tr

Mert Tuna Kurnaz

20011031

tuna.kurnaz@std.yildiz.edu.tr

# HYBRID SITE FOR TABLET FILTERING WITH WEB CRAWLING

## 1. Project Details

Nowadays, technology users search intensively to find tablets that fit their budget. In this context, the developed project provides an automatic information gathering function about the tablets sold on various shopping sites in order to make this process easier and more effective for users.

The main goal of the project is to guide users who want to find tablets that fit their budget with the information provided to users through a mobile application. The filtering options within the application provide users with customised options according to characteristics such as brand, price range, size, etc., allowing them to easily find the most suitable tablets for their needs.

In addition to making technology research faster and more effective, this user-friendly application aims to improve the shopping experience by enabling users to make a more informed choice of tablets that suit their needs. In short, the main objective of the project is to facilitate access to technology-related information and help users find the best tablets that fit their budget.

## 2. The Processes Of The Project And The Techniques Used

The methods to be used for collecting tablet information on the e-commerce sites of leading companies such as Vatan, Mediamarkt and Teknosa are quite comprehensive and functional.

### 2.1. Data Collection Process

Python libraries such as Beautiful Soup and Requests provide powerful tools for web scraping. Using these libraries, tablet information was collected from the e-commerce sites of Vatan, Mediamarkt and Teknosa. Thanks to web scraping, the structure of the data from each site is standardised and meaningful information is extracted.

### 2.2. Creating Database And Backend

The obtained data is saved in a database. Flask is a Python-based web framework and is used for database management and backend operations for this project. Flask, with its simple and flexible structure, is an ideal solution for organising data and communicating with the mobile application. At the beginning of the project, Flask is run on the local host and listens to the requests from the mobile application.

### 2.3. Mobile Application Development

Flutter is a popular framework used for cross-platform mobile application development. In this project, a mobile application compatible with both iOS and Android platforms is created using Flutter. The user interface is designed with Flutter's rich widget collection.

### 2.4. Using Firebase

Firebase is used to support the backend of the application and provide various features. Thanks to the http library, Flutter is seamlessly integrated into the Firebase database with the api provided by Flask. Firebase contributes greatly to this project with its various services such as real-time database and server-side operations.

### 2.5. Hybrid User Experience

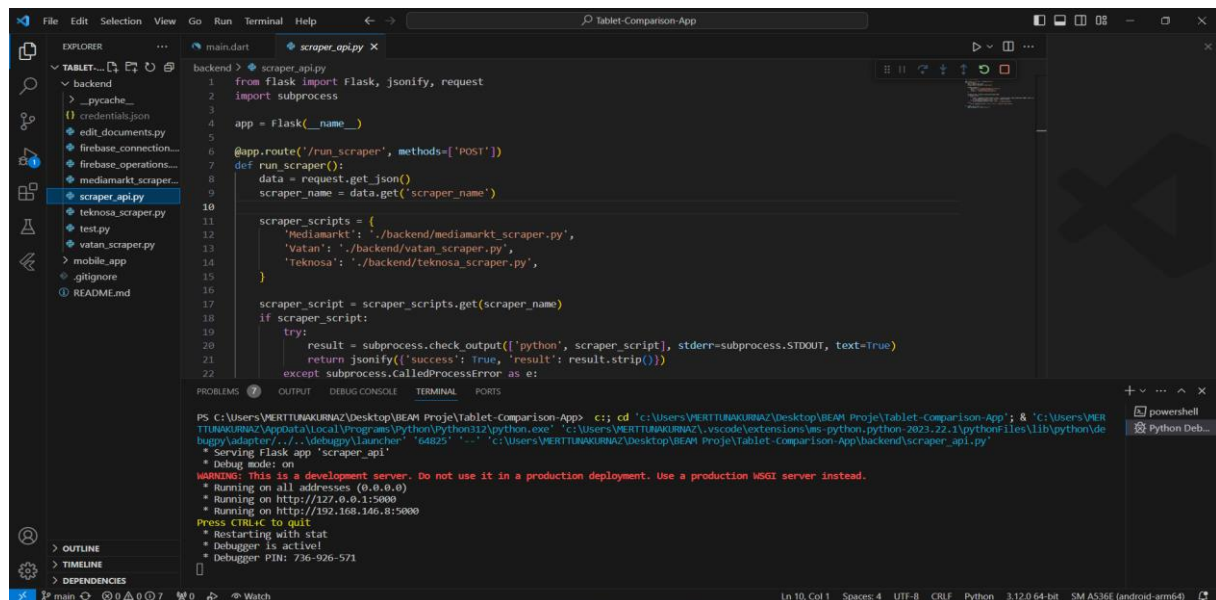
The project offers a hybrid experience to users by combining tablet information from different e-commerce sites in a single application. Users can easily compare tablets through this application and choose the most suitable one for their needs.

### 3. Challenges Encountered in the Project Development Phase

The different structure and layout of each e-commerce site made the use of web scraping tools such as BeautifulSoup and Requests complicated. Writing a separate scraping code for each site, dealing with the dynamic nature of the sites and updates made the development process difficult. To overcome this challenge, it was necessary to analyse dynamic web pages and develop a compatible scraping algorithm. The backend built using Flask was used to manage the data exchange with the mobile application. However, there was a problem that the data could not be properly transmitted to the mobile application after the request was made. To solve this problem, the Flask API and endpoints had to be configured correctly, appropriate data formats had to be used and error management had to be performed effectively. This feature, which was developed to enable users to filter according to criteria such as brand, price range and size, caused various problems. The fact that the data from different e-commerce sites were in different formats made it difficult to perform effective filtering on this data. To overcome this situation, data standardisation and filtering algorithms had to be developed.

### 4. Screenshots

#### 4.1.

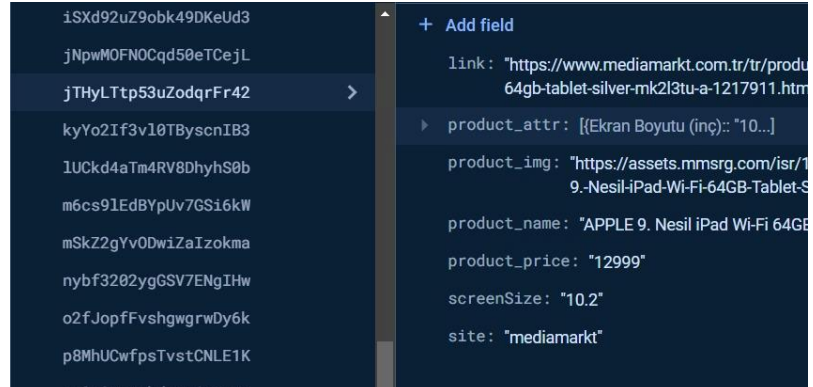
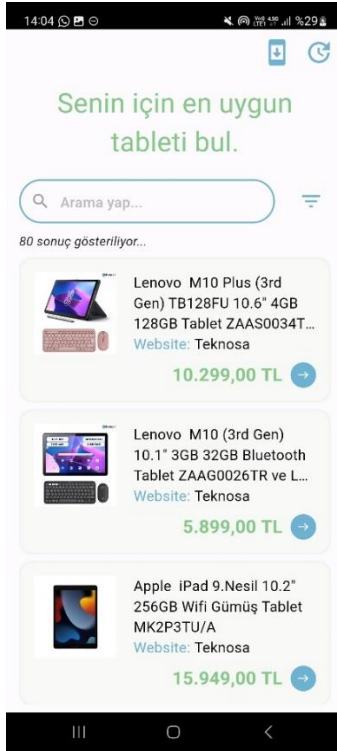


```
1 from flask import Flask, jsonify, request
2 import subprocess
3
4 app = Flask(__name__)
5
6 @app.route('/run_scraper', methods=['POST'])
7 def run_scraper():
8     data = request.get_json()
9     scraper_name = data.get('scraper_name')
10
11     scraper_scripts = {
12         'Mediamarkt': './backend/mediamarkt_scraper.py',
13         'Vatan': './backend/vatan_scraper.py',
14         'Teknosa': './backend/teknosa_scraper.py',
15     }
16
17     scraper_script = scraper_scripts.get(scraper_name)
18     if scraper_script:
19         try:
20             result = subprocess.check_output(['python', scraper_script], stderr=subprocess.STDOUT, text=True)
21             return jsonify({'success': True, 'result': result.strip()})
22         except subprocess.CalledProcessError as e:
```

```
PS C:\Users\VERTUNAKURNAZ\Desktop\BEAM Project\Tablet-Comparison-App> cd 'C:\Users\VERTUNAKURNAZ\Desktop\BEAM Project\Tablet-Comparison-App' & 'C:\Users\VERTUNAKURNAZ\AppData\Local\Programs\Python\Python312\python.exe' 'C:\Users\VERTUNAKURNAZ\vscode\extensions\ms-python.python-2023.22.1\pythonfiles\lib\python\dev\buggy\adapter/.dev\buggy\launcher' '64825' '...' 'C:\Users\VERTUNAKURNAZ\Desktop\BEAM Project\Tablet-Comparison-App\backend\scraper_api.py'
* Serving Flask app 'scraper_api'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.146.8:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 736-926-571
```

First of all, the Flask scraper API that we will connect to in order to run the scrapers on the mobile application must run on localhost. In this example, the API is running on `http://192.168.146.8:5000`. This address contains the ip address of our internet connection, so the person who will run it should replace the ip address in the Flask service file in the mobile application folder of the project with his own. As long as the compiled mobile application is connected to the same internet address, the communication with the API will take place smoothly.

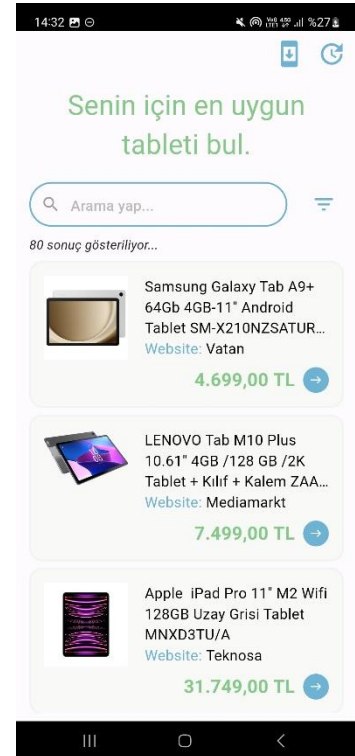
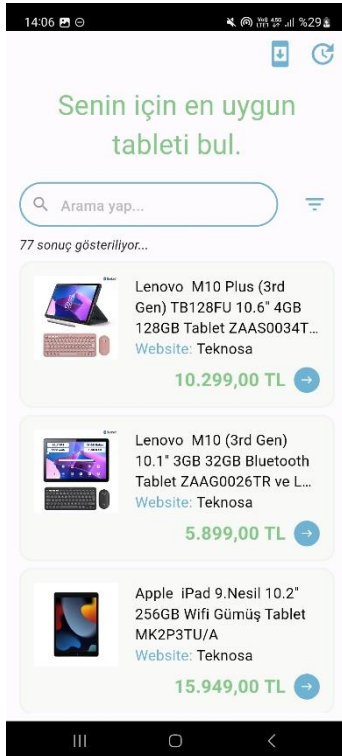
4.2.



The application displays the data previously extracted from the database via scrapers on the home page. The tablet data from the desired site can be updated (button with the Refresh icon) and the data from the database can be refreshed (button with the Download icon) via the buttons in the upper right corner of the home page.

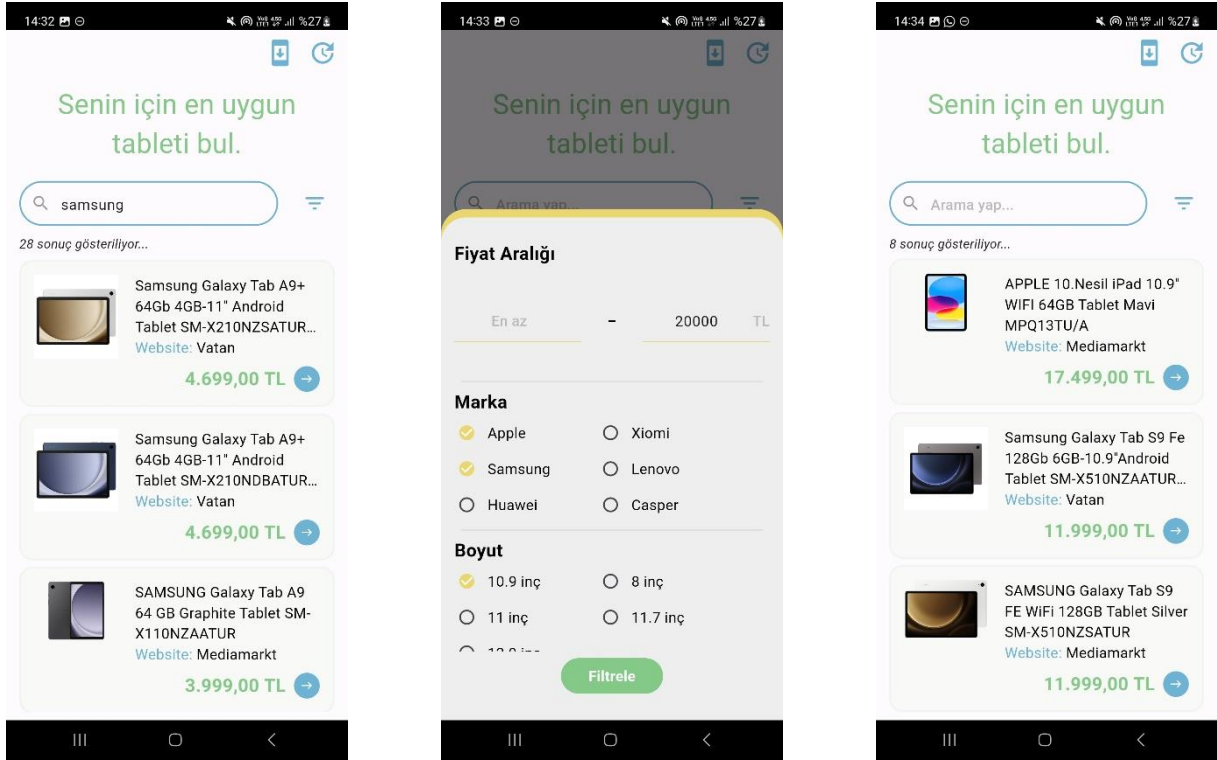
Currently, 80 tablet data from 3 different sites have been actively downloaded. In the Firebase console, the first three tablets from the MediaMarkt site shown above will be deleted.

4.3.



As a result of deleting data from the database, 77 tablets started to be displayed. By pressing the button next to the MediaMarkt site through the panel opened by pressing the refresh button, a request was sent to the API again and the tablet data was updated. The update date was changed to the day of the transaction. The deleted data was obtained again and 80 tablets started to be displayed on the home page again.

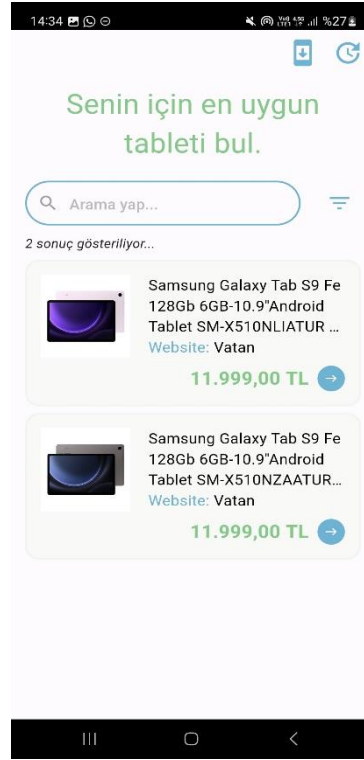
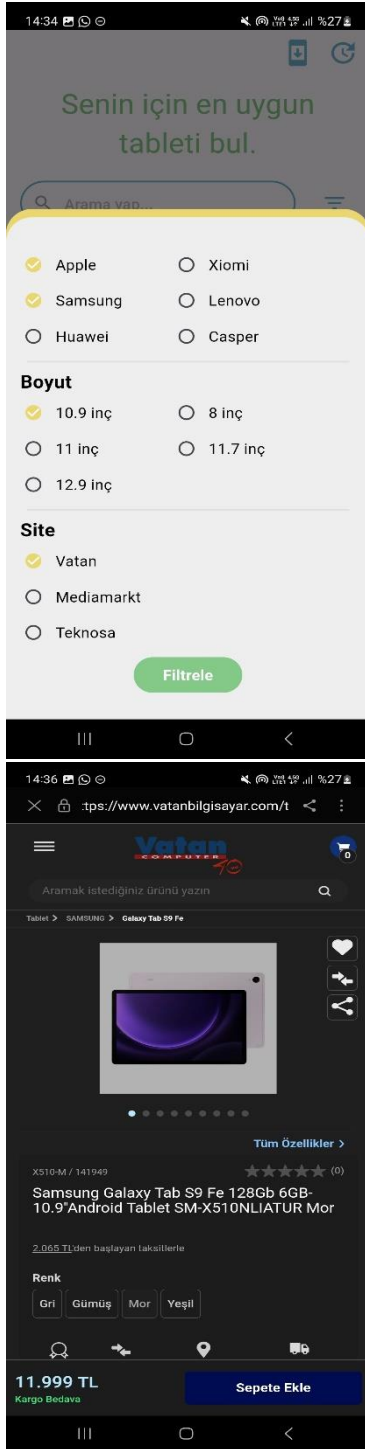
4.4.



In the first screenshot, Samsung tablets are listed via search. Here the names of the tablets are scanned for the occurrence of the word given for the search. As a result of the search, 28 tablets are listed. When Samsung was selected for the brand during filtering, it was seen that 28 tablets were listed again.

In the other two screenshots, both Apple and Samsung were selected for the brand, 10.9 inches for the size and 20000 TL for the maximum price and the filter button was pressed. As a result, as can be seen in the last screenshot, the relevant 8 results were successfully listed. Inverted indexes kept in a table in the database were used in filtering. In this table, tablet ids corresponding to the filtering options are indexed and stored in lists immediately after scraping operations.

4.5.



When the filtering panel is opened again, it appears that the filtering options are stored. In addition, filtering was performed again by selecting the Homeland option in the site section. Related results were listed successfully.

The first one of the results was clicked to go to the detail page. By clicking on the green button with the site and price information, it is directed to the site where the product is located.