Maria Kubara & Kamil Matuszelański

## Short introduction

Flixbus.com is a page of a transport company that offers bus connection between big cities in Poland and Europe. It is known for reliable service and very interesting special pricing offers. Those promotions are extremely time-limited, so if the user wants to buy tickets cheaper, he/she needs to check the website daily. In this case, our project will be a useful tool, that allows to check connection and according prices automatically.

Our project is an advanced web-scraper of flixbus.com site. The purpose is to automatically download the data about trips between cities and at times that the app user is interested in. User will be able to define parameters of the search, and get the results in a form of a display table/xlsx download. This gives the end user ability to analyse the trends visible in the pricing of particular trip. This way user can also semi-automatically check whether the there is a drop in price and tickets are cheaper than usual (using the app user can run the scrapper for a given search parameters at any time). In the future releases the scrapper part of the app will be run periodically (e.g. once a day).

The app consists of 3 main parts:

* Web-scraping module
* SQLite Database
* Bottle Web app

#### Web-scraper part:

* Is triggered by the user in the web app
* Gets parameters of website searches to scrape from the database
* After getting parameters, it scrapes the pages specified, converts them to correct format and commits the scraping results to the database

#### Database:

Stores 4 kinds of information:

* Scraping results - from executed scraper jobs
* Dictionaries for clearer presentation of scraping results to the user
* Authentication management - user logins and passwords
* Logs - from scraping execution, updates of parameters from the users and about users logins

#### App:

Has features (views):

* User authentication
* Adding new parameters for scraping (eg. trip Cracow-Warsaw on 12.01.2020)
* An overlook for a previously defined job with a scrapper trigger to run them
* Fetching the results of previously defined jobs (for each user) with download module
* Admin panel that provides information about current state of an app

Four user stories of the app are:

1. As a user, I want to know the available Flixbus connections from Warsaw to Cracow, that will happen between 01.04.2020 and 3.04.2020 and the return will be between 20.04 and 10.05 with according prices, so that I can get my ticket as cheap as possible[[1]](#footnote-1).
2. As a user, I want to automatically collect data about the prices for connections between Warsaw and Wroclaw for summer months, that are proposed in the foregoing months, so that I can analyse pricing patterns of Flixbus connections
3. As an admin, I want to have access to the data about user and their searches, so that I can manage the site traffic.
4. As an admin, I want to access all the logs provided by the scrawler, so that I can monitor its work on a daily basic.

## Database scheme

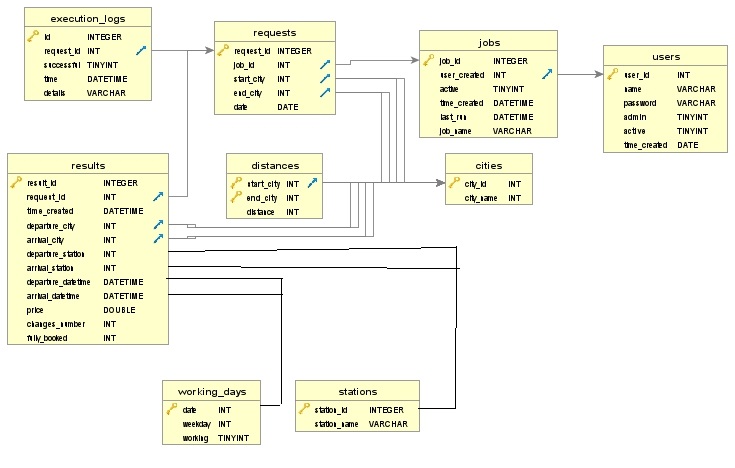


Table **users** holds all information about the users and their passwords. It will also contain information whether or not a user is an admin (and can access more specific sites).

Table **jobs** will contain data about each search description given by a user in an online form. Complementary information will be stored in the **requests** table, that will hold detailed data about each request specification for a given job (for example: job that requires gathering information about travels between the 3rd of April and the 6th of April from Warsaw to Cracow, will actually consist of four separate requests for scrawler – one for each day).

Table **cities** will contain information about destination points – however one needs to notice that not only a city matters, but also a particular stop in a defined city (Warsaw: Warsaw Central, Warsaw East, etc.). This information is stored in **stations** table. Table **distances** holds information about distances between destination cities. This will be used to calculate price per kilometer for a particular trip. Table **working\_days** will gather data regarding the specifics of a day of a given travel – does it happen in a working day or during the weekend (useful tool for future analytics done by users).

Table **results** will contain all data gathered by a scrawler (new records will be added to the existing table after each scrapping task). Table **execution\_logs** holds information about working of the scrawler, and whether set jobs execute correctly.

## SQL queries

Here are queries that will happen in the project:

* 1. For the admin site:
     + select all records from the users table
     + select all records from the jobs table
     + select all records from the execution\_logs table
  2. For the user site:
     + Select user data for a given name (login part and admin authentication)

"SELECT password FROM users WHERE name = '"+loginName+"'"

"SELECT admin FROM users WHERE name = '"+name+"'"

* + - Select available cities from the database

"SELECT city\_id, city\_name FROM cities"

* + - Select jobs for a given user name

"""SELECT j.job\_id, j.job\_name, j.time\_created, j.last\_run

                    FROM jobs j

                    JOIN users u ON j.user\_created = u.user\_id

                    AND u.name = '"""+loginName+"'"

* + - Select job names for a given user name

"""SELECT j.job\_id, j.job\_name

                    FROM jobs j

                    JOIN users u ON j.user\_created = u.user\_id

                    AND u.name = '"""+loginName+"'"

* + - Select jobID for a given job Name

"SELECT job\_id FROM jobs WHERE job\_name='"+jobName+"'"

* + - Queries used for creating a job with given parameters in database

"SELECT user\_id FROM users WHERE name='"+loginName+"'"

"""INSERT INTO jobs (user\_created, time\_created, job\_name)

              VALUES ('"""+str(userId)+"', datetime('"+timeCreated+"'), '"+jobName+"')"

* + - Queries for creating requests liked to a given job

"""INSERT INTO requests (job\_id, start\_city, end\_city, date)

    VALUES (' """+str(jobId)+" ', '"+str(startCity)+"', '"+str(endCity)+"', date('"+timeList[i]+"'))")

* + - select records from the results table, that are assigned to the chosen job name (accessed from jobs table), with detailed information about working/weekend days (from table working\_days) and destination cities (from table cities), including the information about price per one kilometer at this route (calculated from distance table) - this select will require joining tables: jobs, results, working\_days, cities and distances) – triple join.

''' SELECT

  dep.city\_name as departure\_city,

  arr.city\_name as arrival\_city,

   strftime("%d.%m.%Y", res.departure\_datetime) as departure\_date,

  strftime("%H:%M", res.departure\_datetime) as departure\_time,

   strftime("%H:%M", res.arrival\_datetime) as arrival\_time,

   res.price as price,

   round(res.price/dis.distance,4) as price\_per\_km,

   res.fully\_booked as fully\_booked,

   strftime("%d.%m.%Y (%H:%M)", res.time\_created) as date\_obtained

     FROM requests r

     JOIN results res ON r.request\_id = res.request\_id AND r.job\_id = ?

           LEFT JOIN cities arr ON r.start\_city = arr.city\_id

           LEFT JOIN cities dep ON r.end\_city = dep.city\_id

           LEFT JOIN distances dis ON arr.city\_id = dis.end\_city AND dep.city\_id = dis.start\_city

    '''

* 1. From scrawler’s perspective:
     + Selecting requests for a specified job
     + Inserting new records into the results table
     + Inserting new records will update jobs table (column: last\_run)
     + Add logs from scraper execution for specific job using trigger:

CREATE TRIGGER log\_new\_job

AFTER INSERT

ON jobs

BEGIN

INSERT INTO inserts\_logs (

type,

inserted\_id,

time\_added

)

VALUES (

"job",

new.job\_id,

datetime('now')

);

END;

* + - Add logs from scraper execution for specific job using trigger (similar to above query)
    - After job execution, update date of execution to jobs table:

CREATE TRIGGER set\_job\_last\_run\_time

INSERT

ON results

BEGIN

UPDATE jobs

SET last\_run = datetime("now")

WHERE jobs.job\_id = (

SELECT t1.job\_id AS job\_id

FROM jobs t1

LEFT JOIN

requests t2 ON t1.job\_id = t2.job\_id

WHERE t2.request\_id = new.request\_id

);

END;

* + - View containing currently active requests:

CREATE VIEW active\_requests AS

SELECT t2.request\_id,

strftime("%d.%m.%Y", t2.date) AS rideDate,

t2.start\_city AS departureCity,

t2.end\_city AS arrivalCity

FROM jobs AS t1

LEFT JOIN

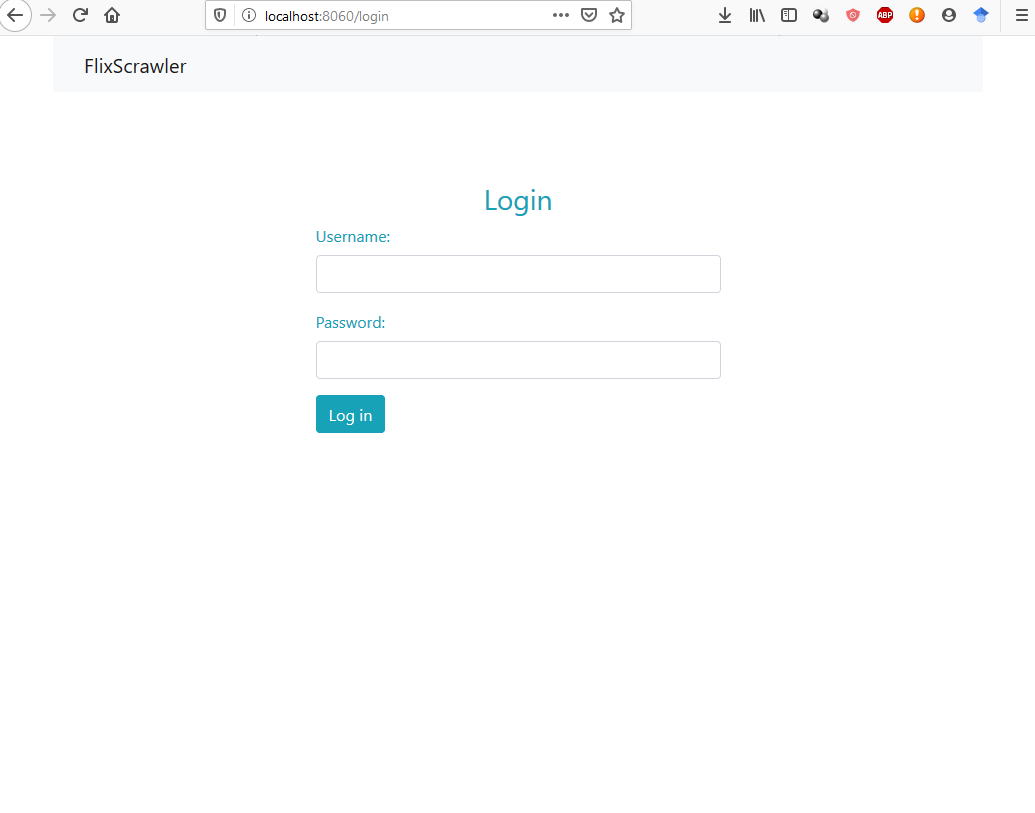
requests AS t2 ON t1.job\_id = t2.job\_id

WHERE t1.active = 1;

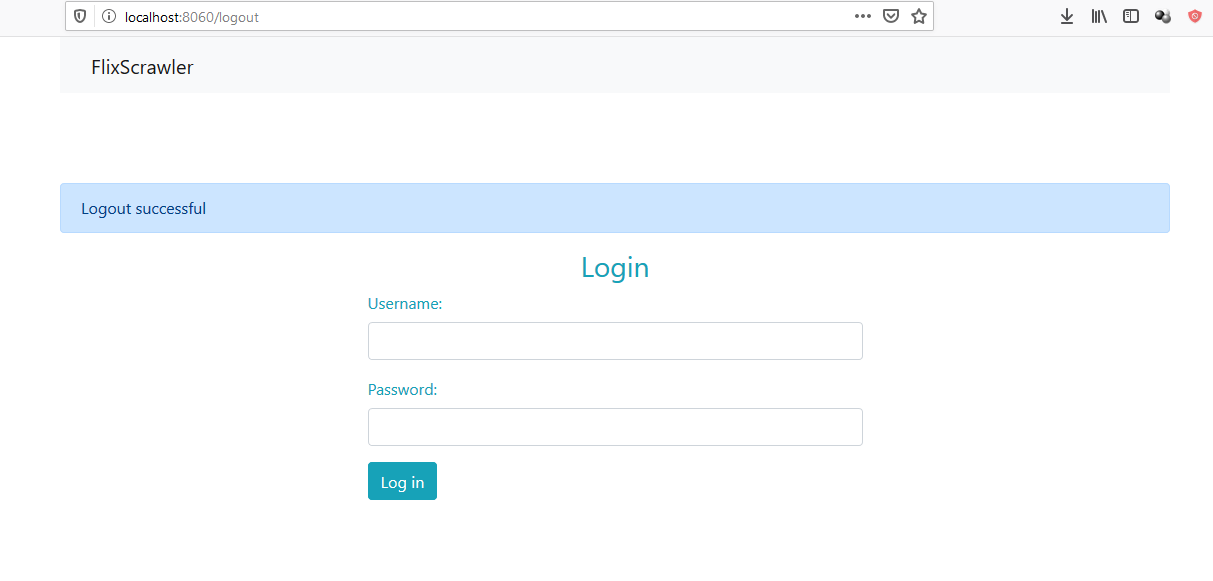
Indexes will be used for increasing the efficiency of search for a given job name (for example in fetching results

## Standard views

Basic view of login page.

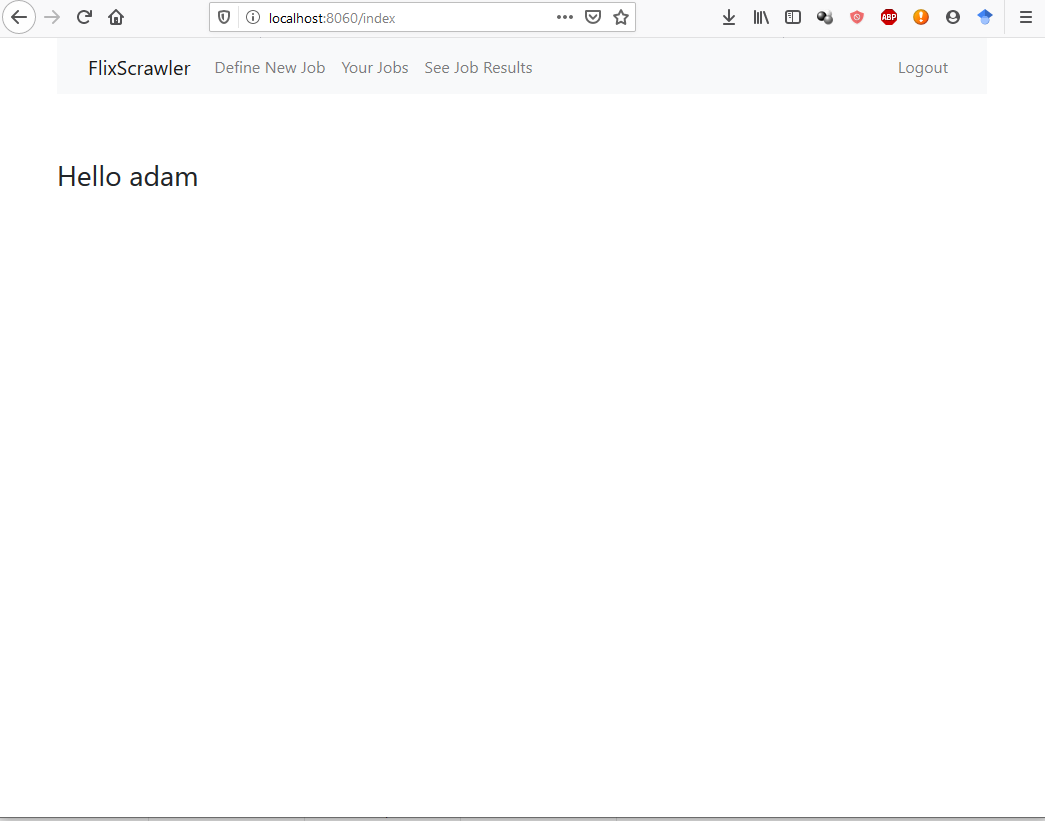


If a logout is conducted (resulting from activating the *logout* button in navbar) the same view is presented with a corresponding message. This action also clears the cookies about logged user.



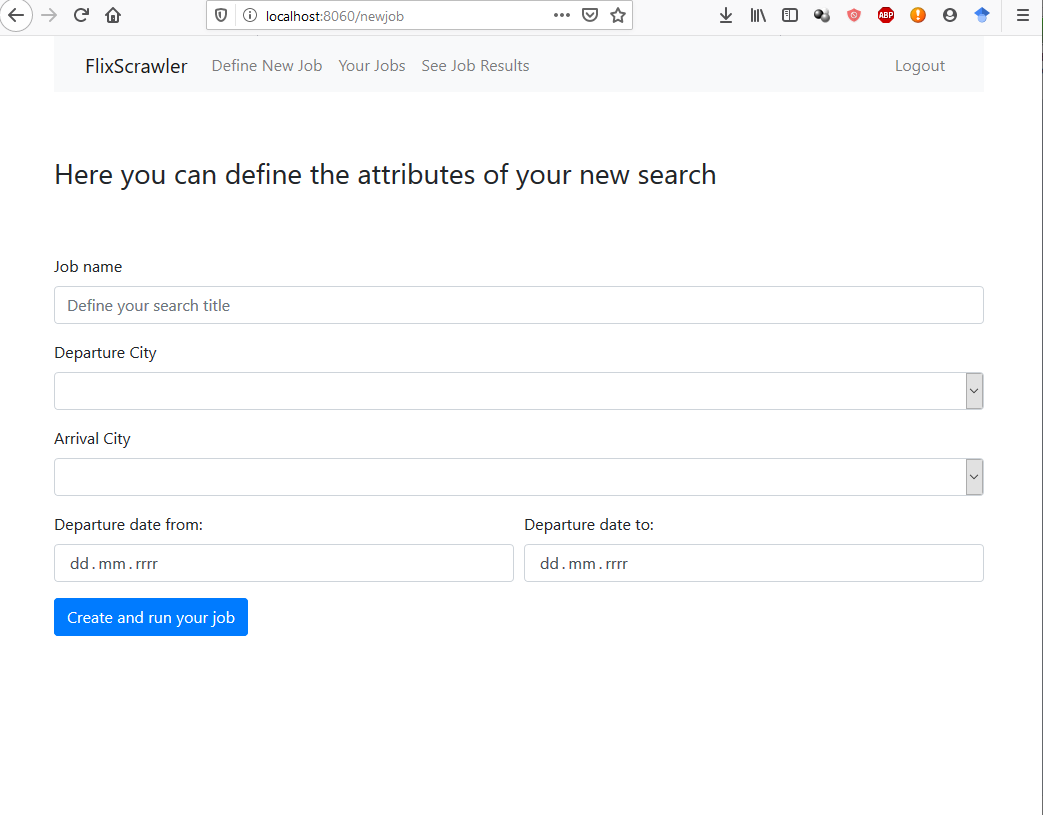
## User views:

After a successful login for a standard user this landing page is shown. Buttons in a navbar allows for navigation in the app.

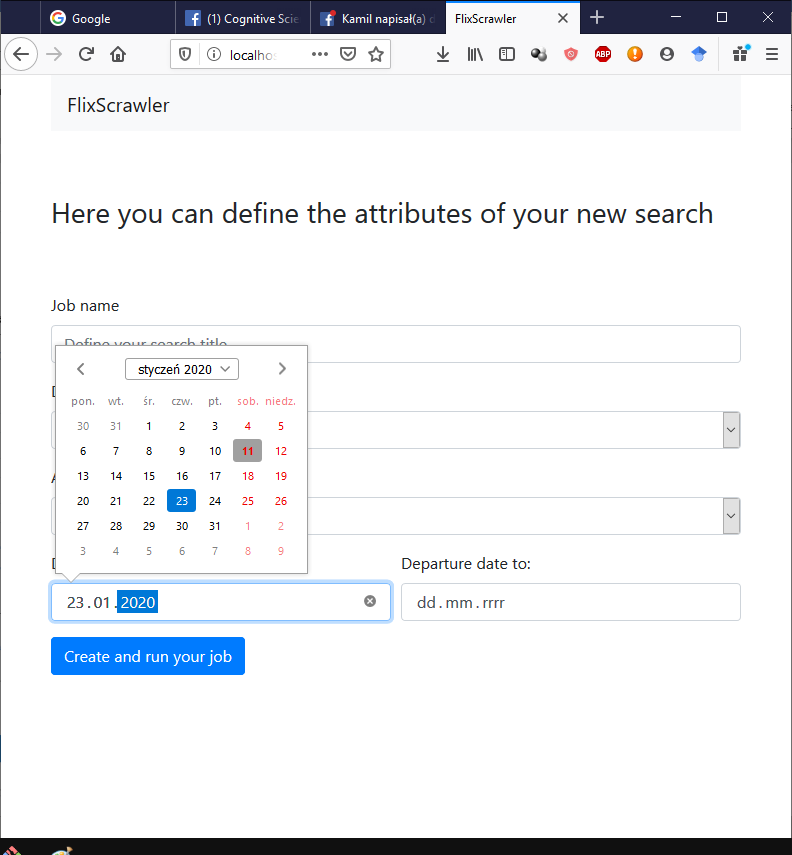


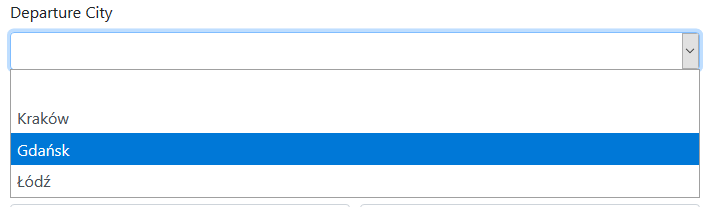
#### Define New Job page

A user can define new search job with its parameters.

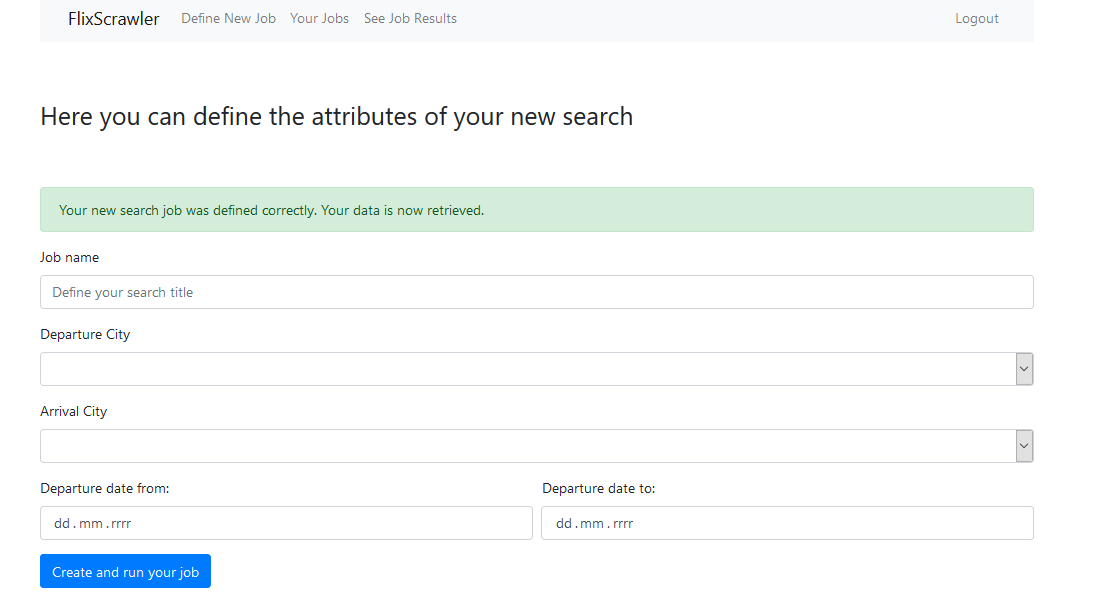


Modifications in this page will be done via drop down lists and date pickers:

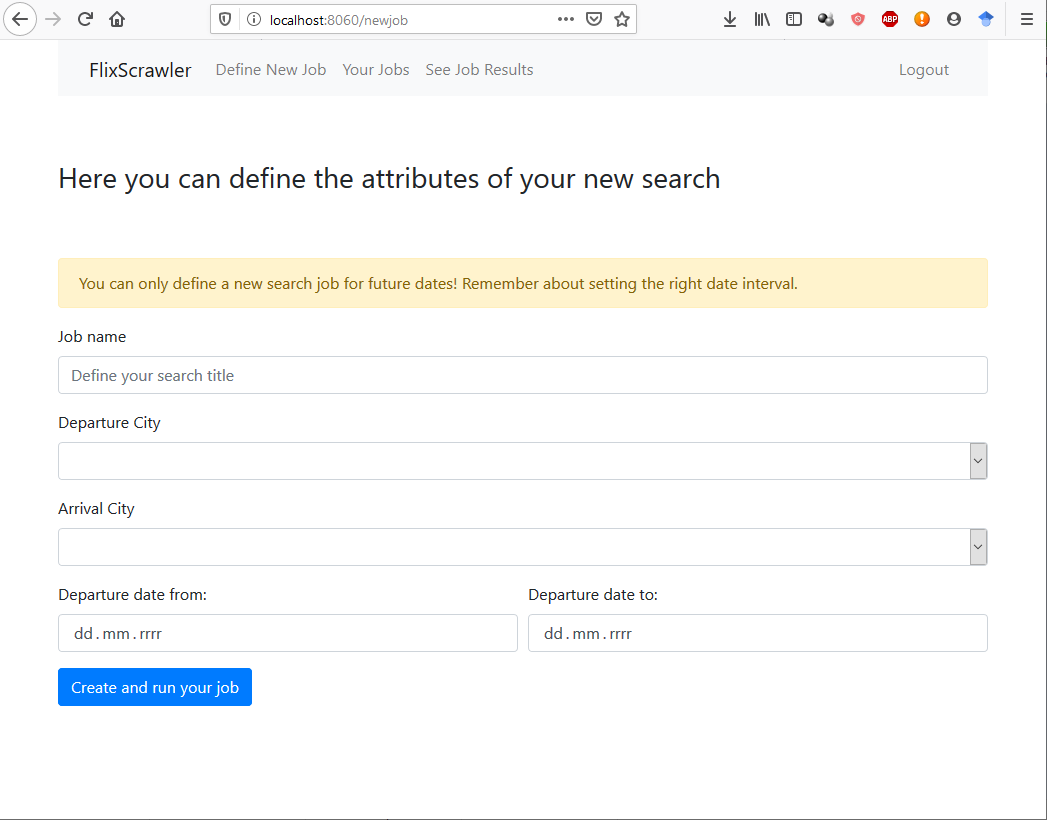


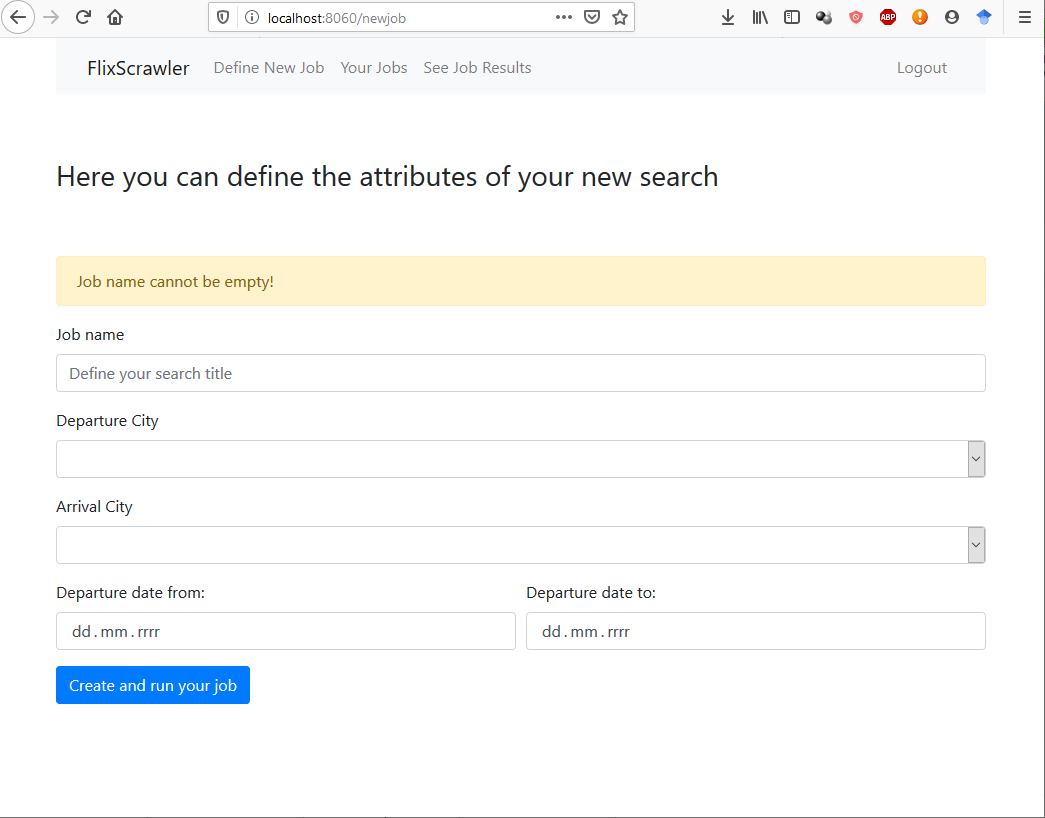
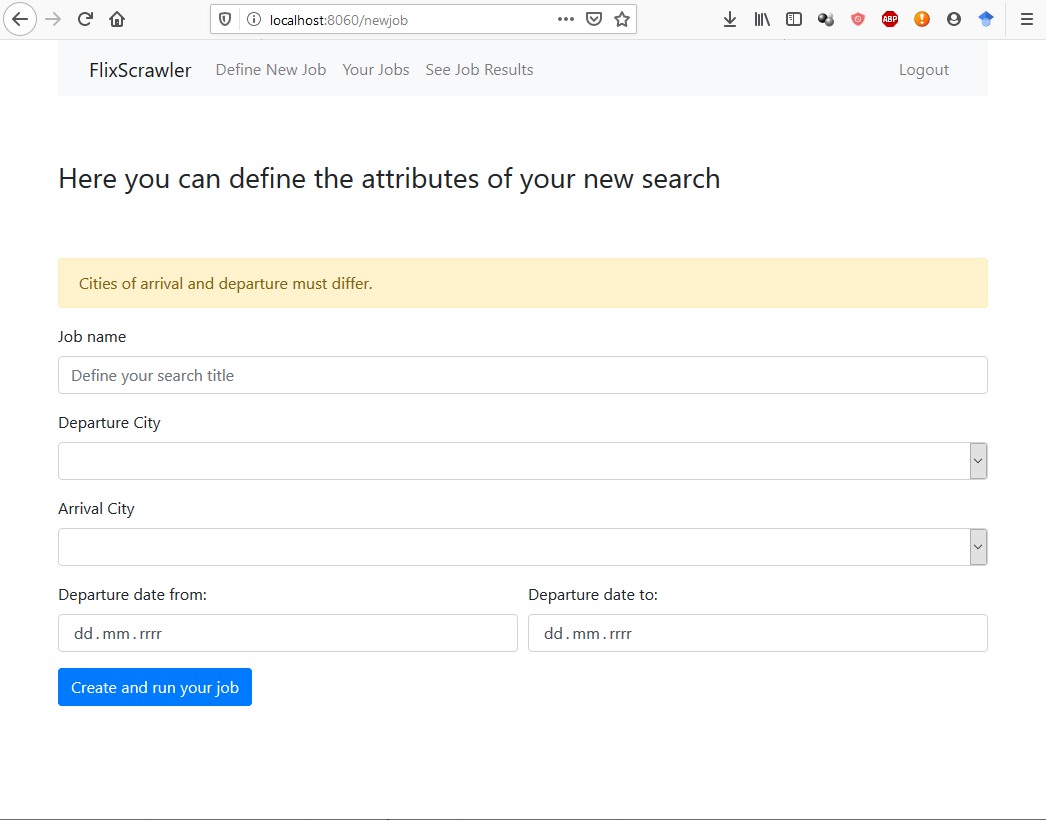
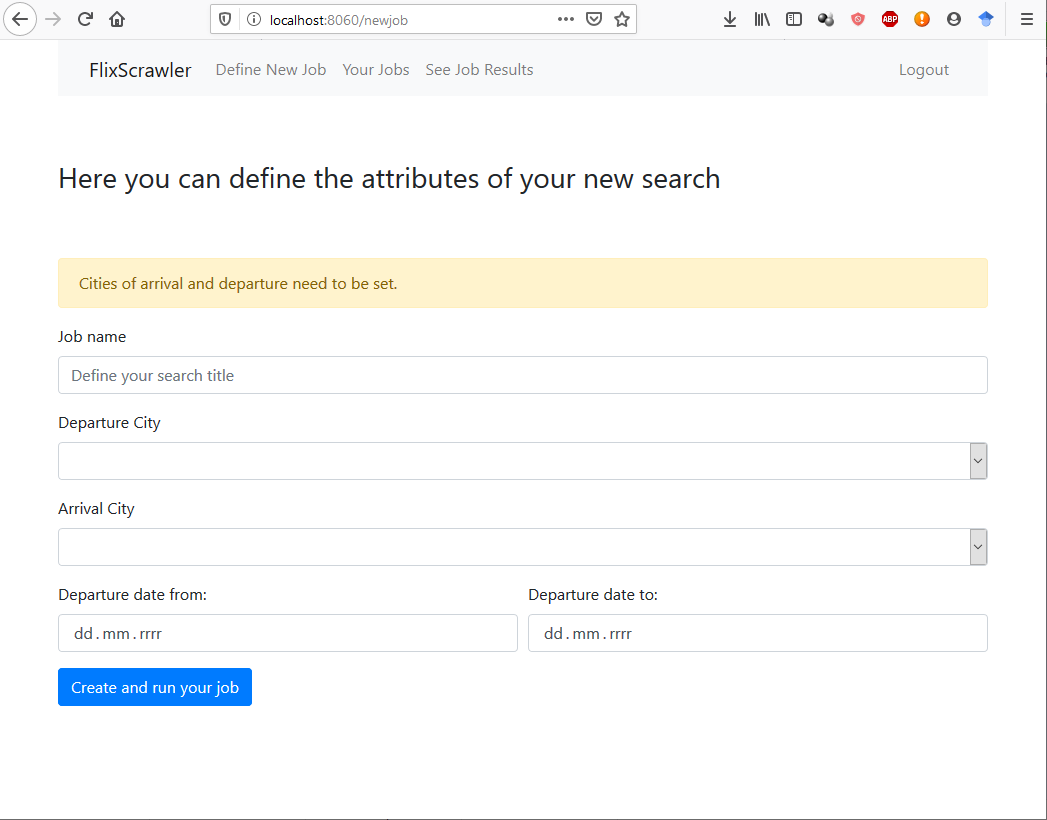
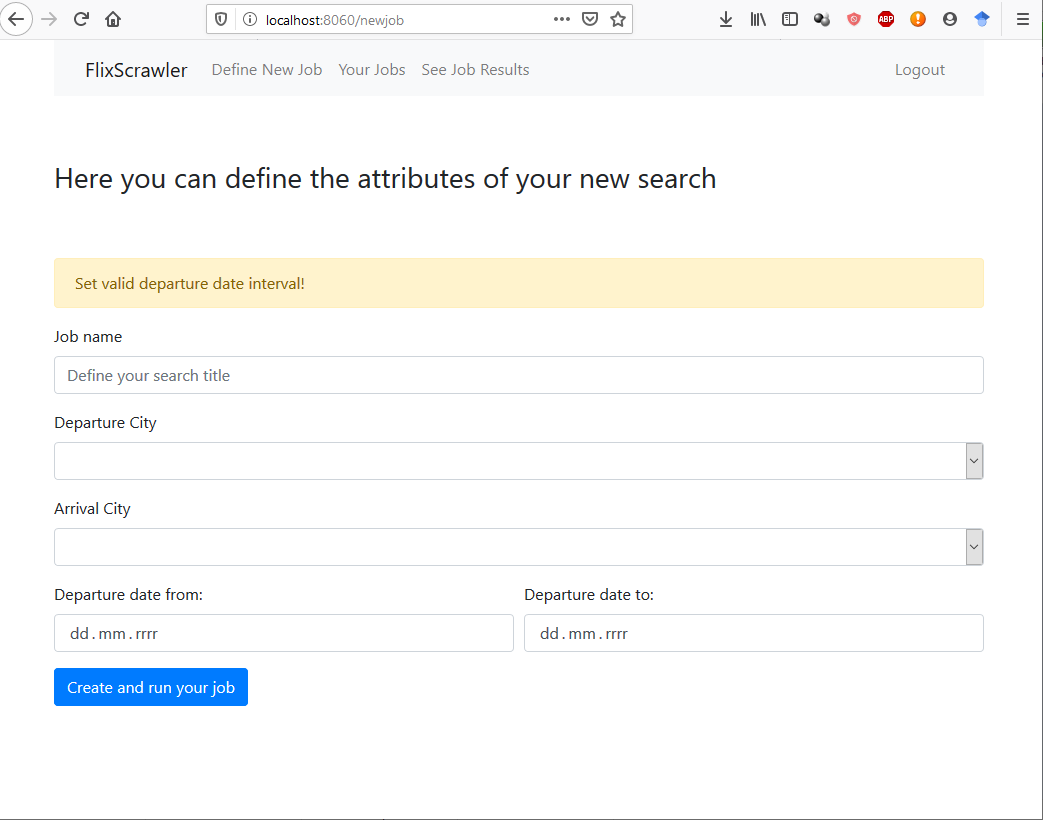


This page shows status messages with regard to the committed changes. If a job is defined correctly, this status is presented. Correctly defined search job is passed to a database and a scrapper is run to retrieve the data.



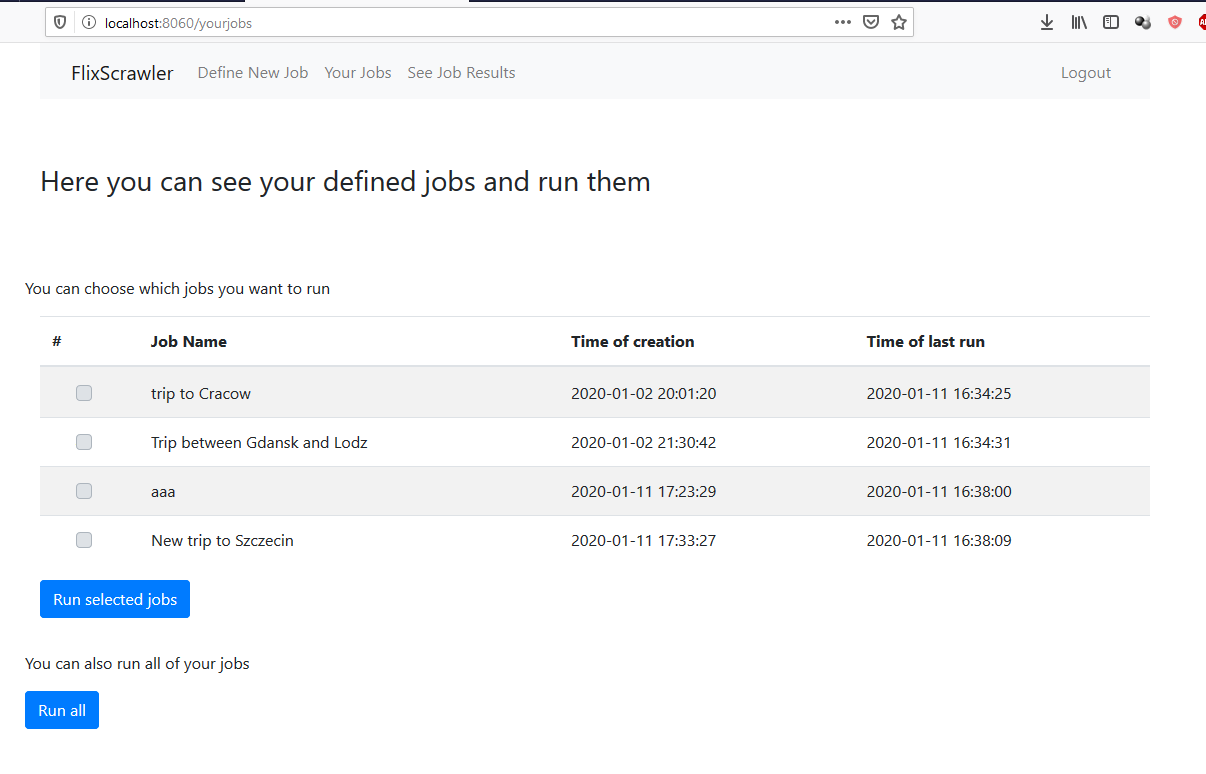
If some mistakes are made – corresponding error message occurs.





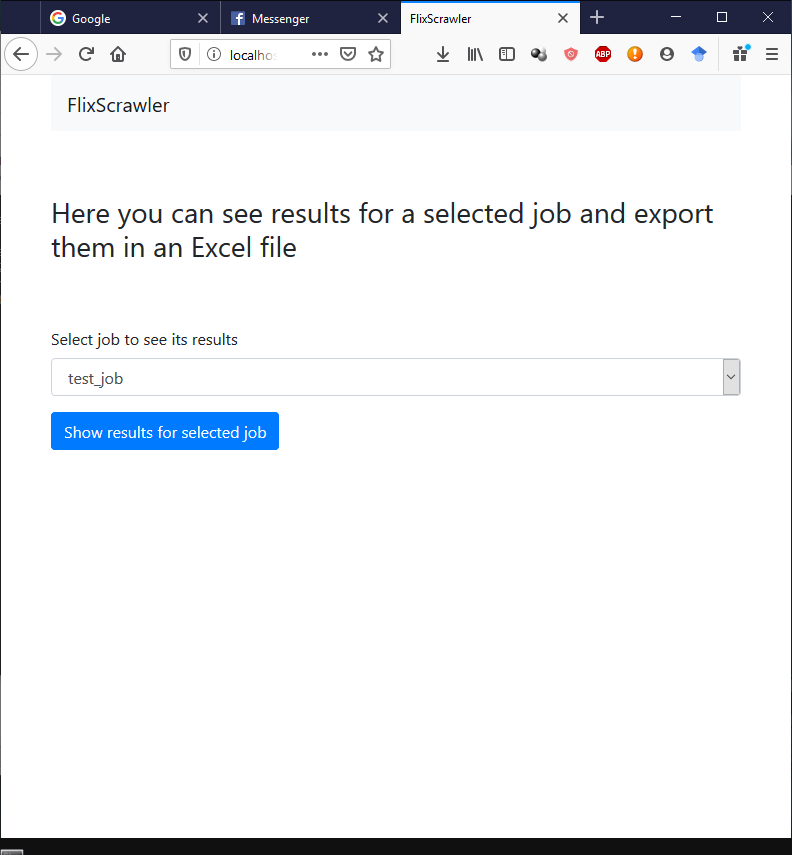
#### Your Jobs page

Here a collection of a user’s jobs will be displayed. In this site the user can force scrawler to make updates in the database. This action is possible for all defined jobs (Run all) or for particular jobs, selected by a user (select job in a table and then button *Run selected jobs*).

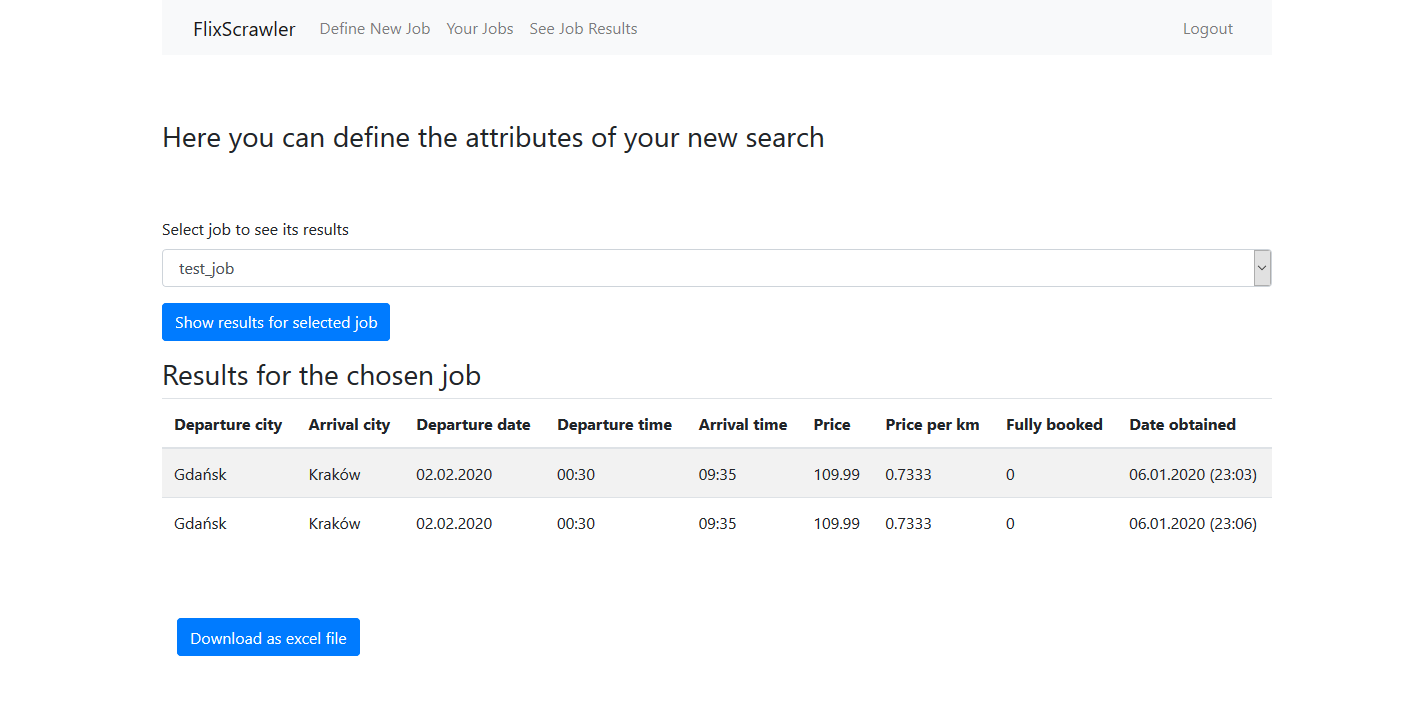


#### See Job Results page

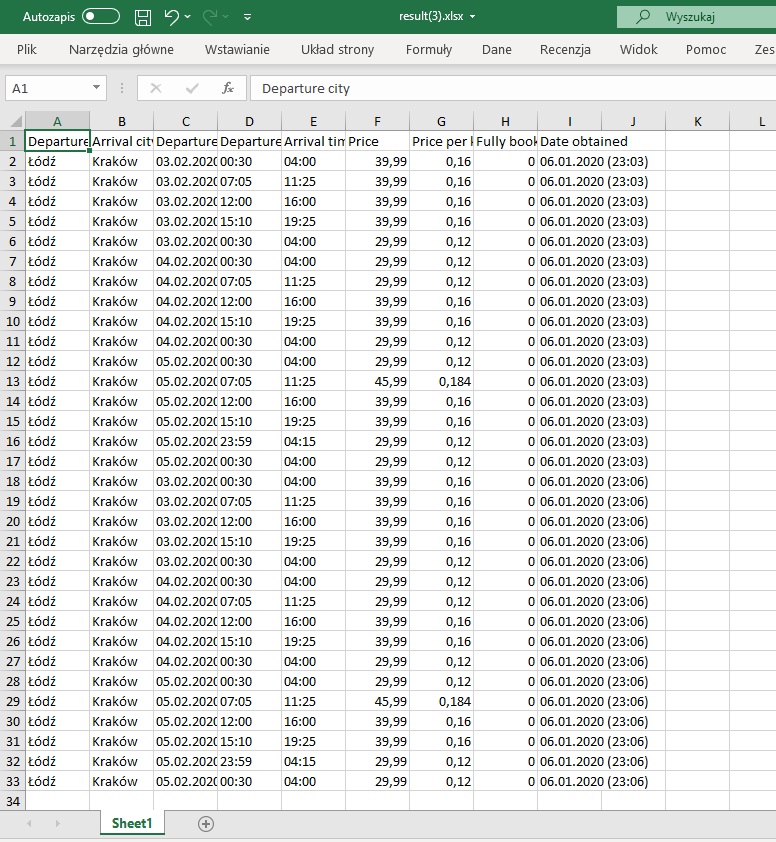
This page allows a user to see results obtained for a specific job. A result table is shown for a specific job (chosen from a drop down list). If a table is presented – user can also export it in a form of an .xlsx file. Basic view looks like this:



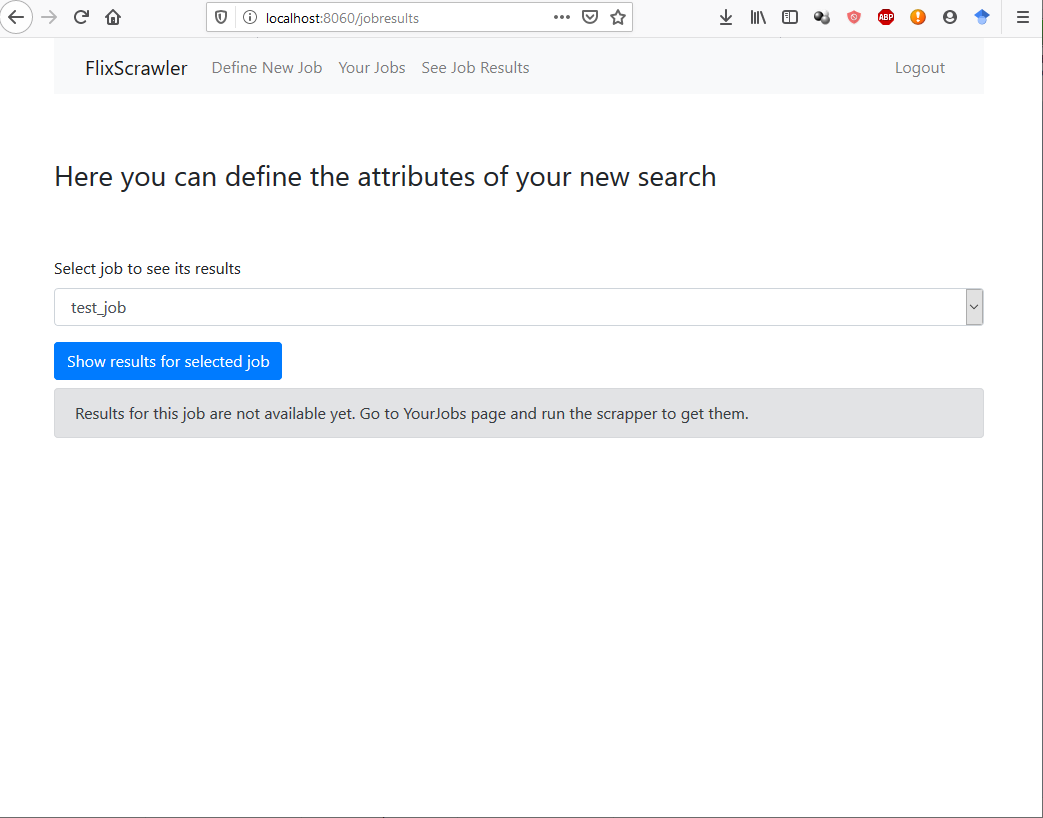
If a particular job is selected, results are shown below in a polished results table.



This table can be exported when clicking on *Download as excel file* button. Then the download in a browser stars. Sample result file looks like this:

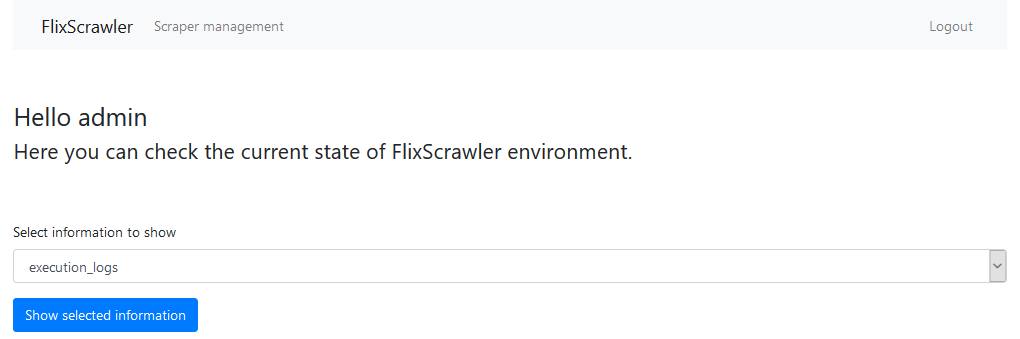


If for a selected job no results can be found in a database – web application informs the user about the error and offers some guidelines for troubleshooting. Instead of a result table a message is shown

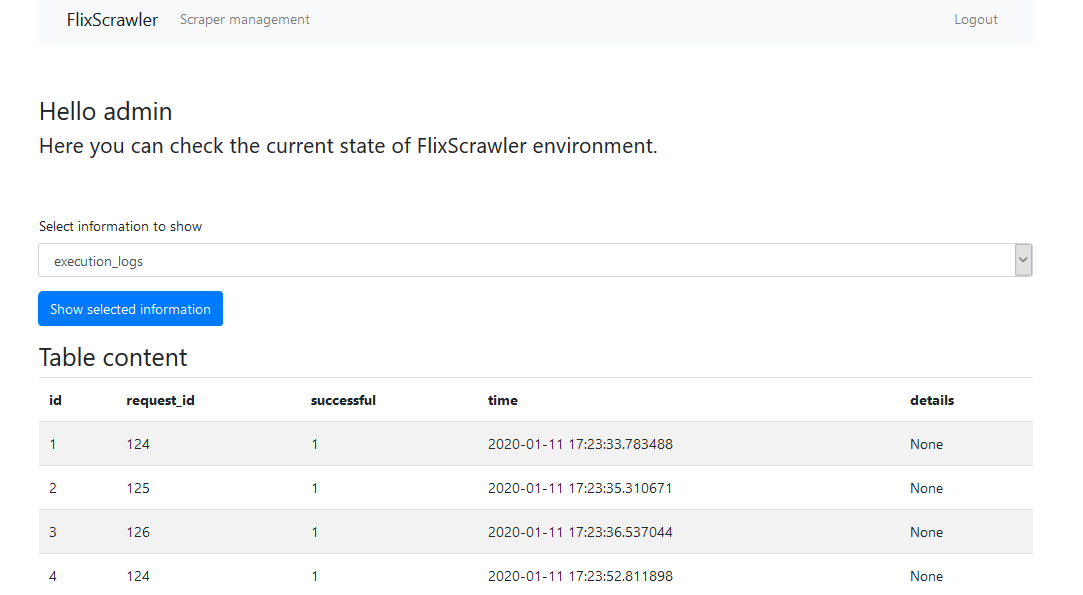


## Admin views:

Main admin view contains a dropdown menu to select information the admin is currently interested in. For example, he can check the logs from scraper execution and see if there are any current problems.



After clicking “show selected information”, the requested table is displayed below.



## Distribution of the work

|  |  |
| --- | --- |
| Maria | Kamil |
| Draft of the written project proposal | Database outline and entity diagram |
| Building the back-end of the webapp in python and preparing the base.tmp view file | Building the scrawler in python |
| Integrating webapp with the database | Integrating scrawler with database |
| Preparing user’s views and according templates | Preparing admin’s views and its bottle templates |
| Preparing the authentication mode (with login and logout page) and xlsx export module | Preparing SQL actions triggered by each scrawler use |
| Building selects and joins used for all the user views | Preparing an SQL view and selects used in admin page |
| Preparing the presentation 50% | Preparing the presentation 50% |
| Explaining the purpose of a project and it’s general description | Explaining the code-specific features, mainly the scrapper work |
| Writing the report 50% | Writing the report 50% |

1. Using standard search box this would require ~120 searches. [↑](#footnote-ref-1)