Programming assignment 2 - Data extraction

Kim Ana Badovinac, Jakob Petek, Jovan Prodanov
Faculty of Computer and Information Science, University of Ljubljana

I. Introduction

Data extraction from websites is a critical task for businesses and researchers who need to gather valuable information from the internet. One of the most efficient ways to accomplish this task is through the use of automated tools that can extract relevant data from web pages. In this context, regular expressions, XPath, and the RoadRunner algorithm are three popular methods for extracting structured data from web pages which, as per our programming assignment, we had to implement.

II. Implementation

All three extraction methods were implemented in Python. We have extracted data from 6 websites, 4 of which were already specified (rtvslo.si and overstock.com) and 2 optional websites which we decided to be from the Steam store domain. The data collected from Steam store was the name of the video game, its release date, price, rating and amount of reviews as well as any tags associated with the game. The relevant data extracted is shown in figure 1.

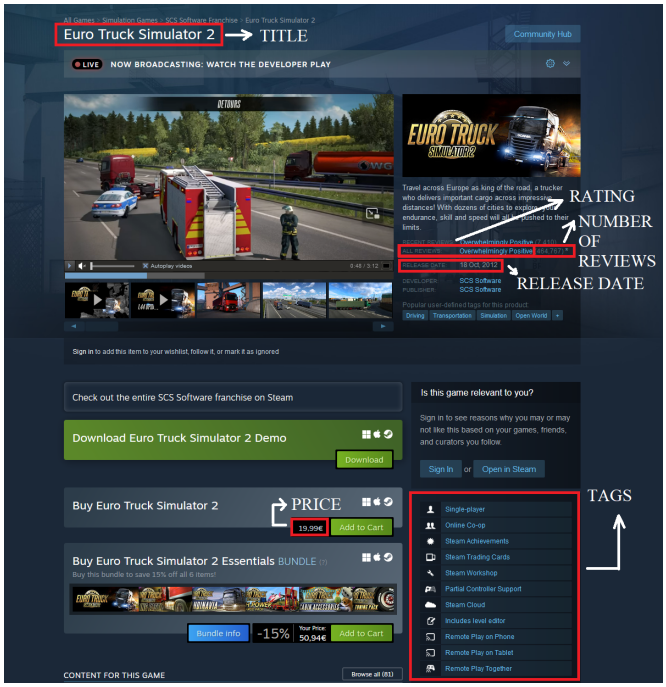

Fig. 1. Data on Steam store

A. Regular expression implementation

As per requirements we have implemented 3 functions (for each type of website) and extracted the required data using only one regular expression. For this we used Python's re module. Each function takes in a html file on which we then perform regular expressions to gather data. Most of the gathered data does not need any additional editing or filtering (except rtvslo article's content). The function then packs the data to a JSON file and prints out its content on the standard output. The statements used for rtvslo, overstock and steam can be seen on figures 2, 3 and 4 respectively.

```
title = re.search(r'<h1>(.+?)</h1>', html_content).group(1)
published_time = re.search(r"<div class=\"publish-meta\">\s+(.+?)<br>", html_content).group(1)
author = re.search(r'<div class=\"author-name\">(.+?)</div>', html_content).group(1)
subtitle = re.search(r'<div class=\"subtitle\">(.+?)</div>', html_content).group(1)
lead = re.search(r'<p class=\"lead\">(.+?)</p>', html_content).group(1)
content = re.search(r'<article class="article">\s*(.+?)\s*</article>', html_content, flags=re.DOTALL).group(1)
```

Fig. 2. Regular expressions on rtvslo

```
titles = re.findall(r"<td\s+valign=\"top\"\s+><b\s+href=\"\S+\">(.+)</b>", html_content)
list_prices = re.findall(r"<td\salign=\"left\"\snowrap=\"nowrap\">(.+)</s>", html_content)
prices = re.findall(r"<span\sclass=\"bigred\"><b>(.+)</b>", html_content)
savings = re.findall(r"<td\salign=\"left\"\snowrap=\"nowrap\"><span\sclass=\"littleorange\">(.+?)\s", html_content)
savings_percents = re.findall(r"<td\salign=\"left\"\snowrap=\"nowrap\"><span\sclass=\"littleorange\">.+?\((\d+)\)</span>", html_content)
contents_data = re.findall(r"<span\sclass=\"normal\">(.+?)<br>", html_content, flags=re.DOTALL)
```

Fig. 3. Regular expressions on overstock

```
game_title = re.search(r"<div\sid=\"appHubAppName\"\sclass=\"apphub_AppName\">(.+)</div>", html_content).group(1)
review = re.search(r"<span\sclass=\"game_review_summary positive\"\sitemprop=\"description\">(.+)</span>", html_content).group(1)
amount_of_reviews = re.search(r"<meta\sitemprop=\"reviewCount\"\scontent=\"(.+)\">", html_content).group(1)
release_date = re.search(r"<div\sclass=\"date\">(.+)</div>", html_content).group(1)
price = re.search(r"<div\sclass=\"discount_final_price\">(.+?)</div>", html_content)
tags = re.findall(r"<div\sclass=\"label\">(.+?)</div>", html_content)
```

Fig. 4. Regular expressions on Steam store

B. XPath implementation

Similarly to the implementation using regular expressions, we have extracted same data using XPath. We have again implemented 3 functions that take in the html document on which XPath expressions were used to gather data. We have again only used one expression per required data, except at certain parts were we had to clean up the output. The XPath statements used for rtvslo, overstock and steam can be seen on figures 5, 6 and 7 respectively.

```
title = tree.xpath("//h1/text()")[0]
published_time = tree.xpath("//*[@class='publish-meta']/text()")[0]
published_time = published_time.lstrip('\t\n\n')
author = tree.xpath("//*[@class='author-name']/text()")[0]
subtitle = tree.xpath("//*[@class='subtitle']/text()")[0]
lead = tree.xpath("//*[@class='lead']/text()")[0]
contents = tree.xpath("//*[@class='article-body']/article/p/text()|//*[@class='article-body']/article/p/strong/text()")
```

Fig. 5. XPath expressions on rtvslo

```
titles = tree.xpath(r"//*[@valign='top']/a[@href]/b/text()")
list_prices = tree.xpath(r"//*[@nowrap='nowrap']/s/text()")
prices = tree.xpath(r"//*[@class='bigred']/b/text()")
savings_data = tree.xpath(r"//*[@class='littleorange']/text()")
contents_data = tree.xpath(r"//*[@class='normal']/text()")
```

Fig. 6. XPath expressions on overstock

```
game_title = tree.xpath(r"//*[@id='appHubAppName'] [@class='apphub_AppName']/text()")
review = tree.xpath("//span[@class='game_review_summary positive'] [@itemprop='description']/text()")
amount_of_reviews = tree.xpath(r"//meta[@itemprop='reviewCount']/@content")
release_date = tree.xpath(r"//div[@class='date']/text()")
price = tree.xpath(r"//div[@class='discount_final_price']/text()")
tags = tree.xpath(r"//div[@class='label']/text()")
```

Fig. 7. XPath expressions on Steam store

C. RoadRunner implementation

Roadrunner is implemented in a way that it generates a common wrapper in the from of HTML code between two web pages. It generates common tags, common attributes of the tags and common content of tags.

Some details are when the content in the tags differ in the two websites (most commonly - a string), when that happens the `#PCDATA` symbol is inserted. Also when the children of tags differ, the `(<tag.*?>.*?</tag>)+` symbol is inserted.

1) Pseudo code:

```
generateWrapper(webpage1, webpage2):
  removeUnwantedTags(webpage1)
  removeUnwantedTags(webpage2)
  regex += generateRegex(head1, head2)
  regex += generateRegex(body1, body2)
  regex = "<html>" + regex + "</html>"
  return regex

generateRegex(tag1, tag2):
  attrs = getCommonAttrs(tag1, tag2)
  regex = "<" + tag1.name + attrs + ">"
  children = zip(tag1.children,
              tag2.children)
  for child1, child2 in children:
    if isinstance(child1, Comment):
        break

    if contentMatches(tag1, tag2):
        regex += tag1.content
    else:
        regex += "#PCDATA"

    if tagsMatch(child1, child2):
        regex +=
            generateRegex(child1, child2)
    else:
        regex += "(<" + child1 +
            ".*?>.*?</" + child1 + ">)+"

  regex += "</" + tag1.name + ">"
  return regex
```

Unwanted tags in our implementation are: `["script", "input", "option", "style", "br", "hr"]` and in the `getCommonAttrs` we avoid `style` and bracket style attributes.

2) Wrappers: There are only parts of the wrappers produced in pictures, however they are currently present in the GitHub repository in `.html` format.

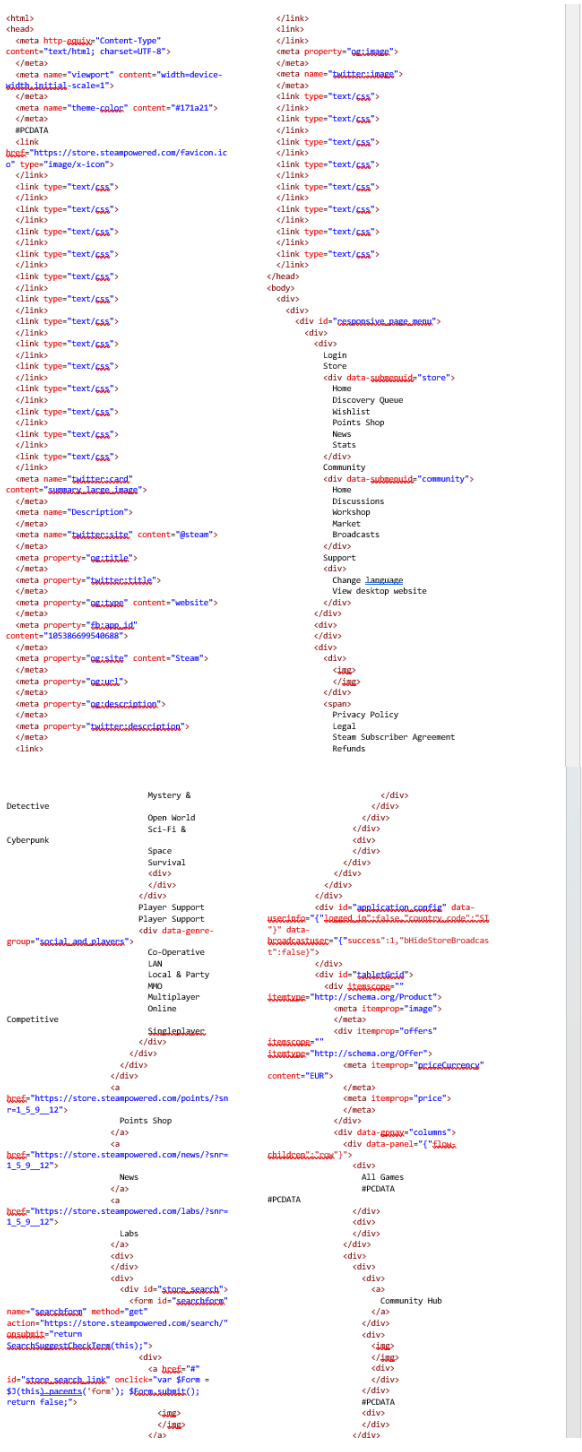

Fig. 8. steam wrapper results

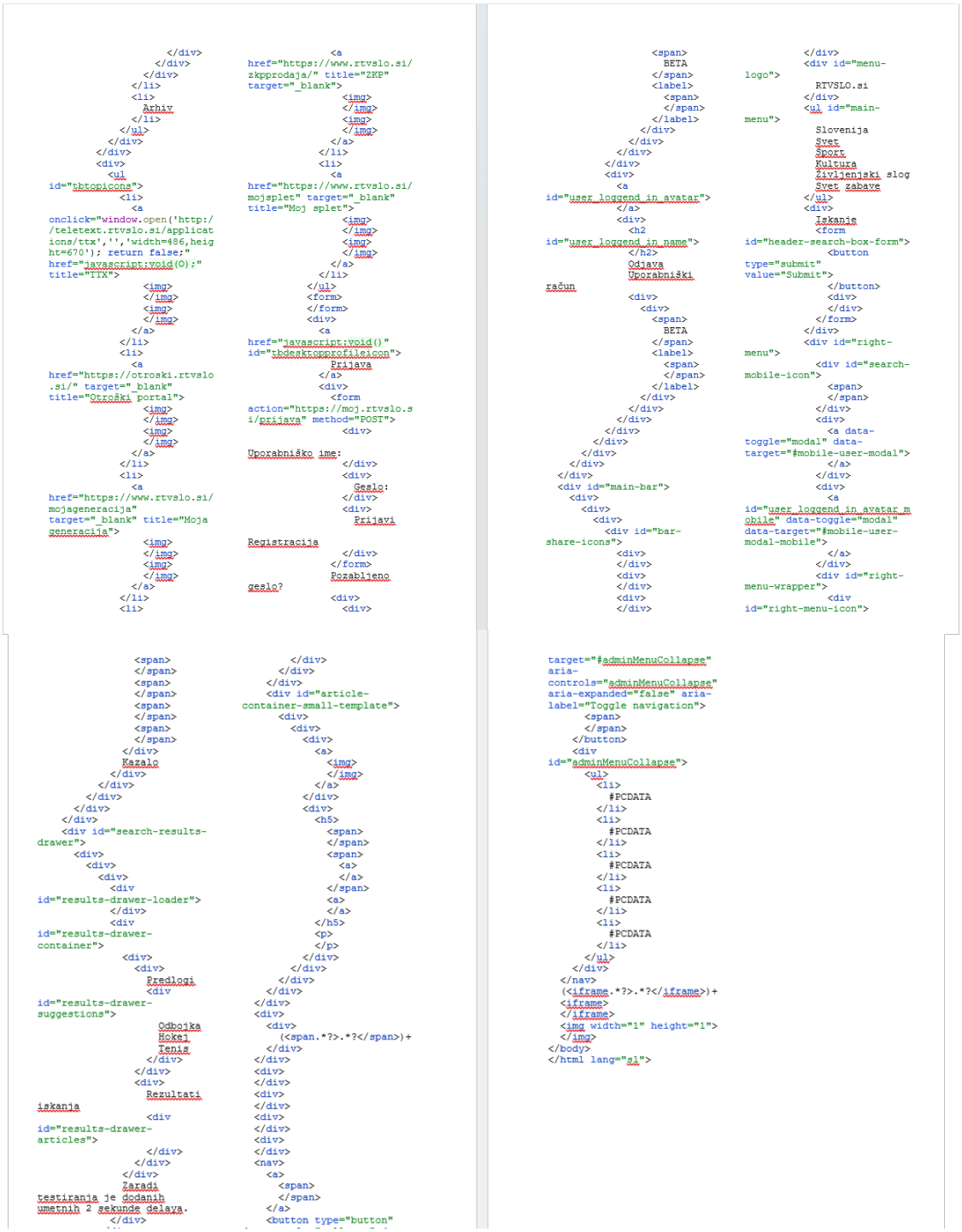

Fig. 9. rtvslo wrapper results

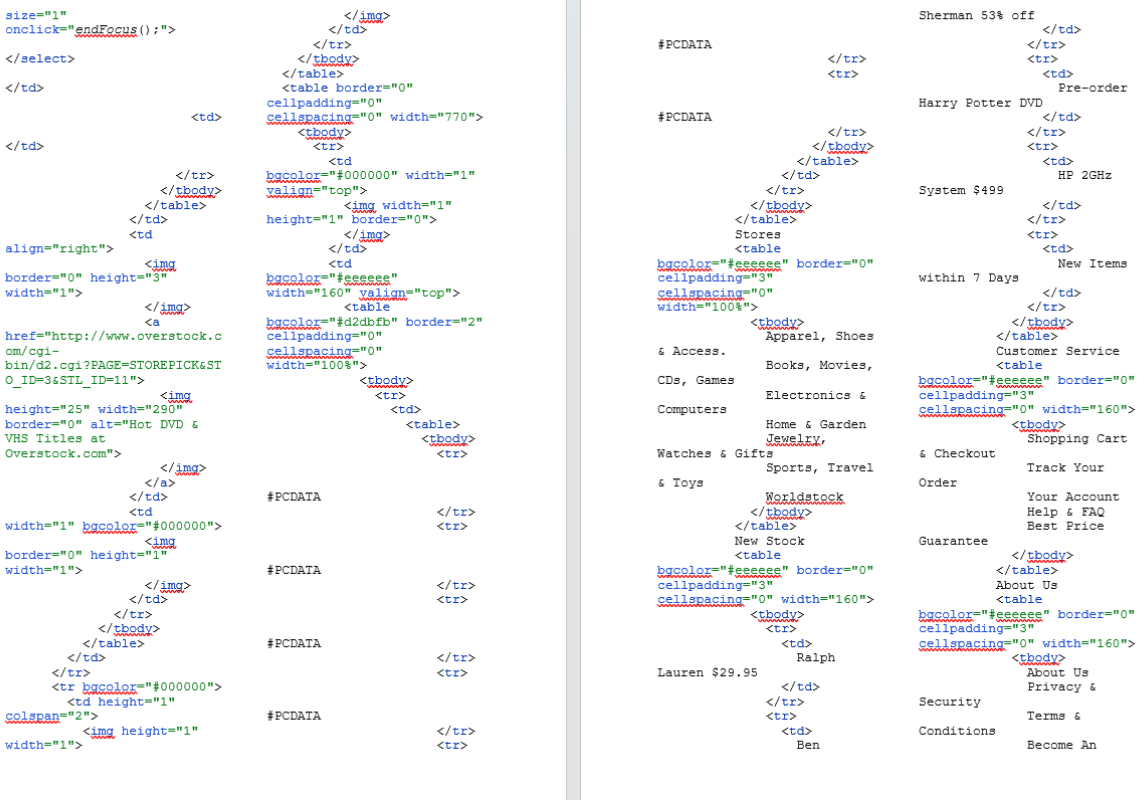

Fig. 10. overstock wrapper results