MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

FINANCIAL EFFICIENCY BOOSTING SYSTEM FOR CONSUMER-GRADE PRODUCTS

LICENSE THESIS

Graduate: Peter Tibor ZAVACZKI
Supervisor: Assoc. Prof. Dr. Eng. Delia Alexandrina MITREA

2019

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Peter Tibor ZAVACZKI**

FINANCIAL EFFICIENCY BOOSTING SYSTEM FOR CONSUMER-GRADE PRODUCTS

1. **Project proposal:** *A financial efficiency boosting system for consumer-grade products based on an extension for the Google Chrome browser, webcrawling, a web platform and a RESTful web service.*

2. **Project contents:** *The presented thesis contains eight chapters followed by the Bibliography. The chapters are as follows: Project Context, Project Objectives and Specifications, Bibliographic Research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's Manual and Conclusions. Each chapter is divided, respectively, into a series of sections and subsections for a clearer description.*

3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants:** Assoc. Prof. Dr. Eng. Delia Alexandrina MITREA

5. **Date of issue of the proposal:** November 1, 2016

6. **Date of delivery:** July 8, 2019

Graduate: _____

Supervisor: _____

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) **PETER TIBOR ZAVACZKI** legitimat(ă) cu **C.I.** seria **MM** nr. **971552** CNP **1970131245031**, autorul lucrării **FINANCIAL EFFICIENCY BOOSTING SYSTEM FOR CONSUMER-GRADE PRODUCTS** elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea **CALCULATOARE, ENGLEZA** din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea **IULIE** a anului universitar **2018-2019**, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

_____

Nume, Prenume

Peter Tibor ZAVACZKI

Semnătura

Contents

Chapter 1

Project Context

The information age has greatly revolutionalized the lives of people all around the globe, bringing along changes that made lives easier, but also more difficult at the same time. The Internet has transformed modern society into homebodies, people who do anything from the comfort of their homes rather than stepping outdoors to complete tasks. People can do it all online : shopping, chatting, paying bills, working, learning, entertaining themselves, even ordering food. Even though life is much simpler than 50 years ago, an average person probably doesn't even know of 50% of the wonders that the World Wide Web offers them, let alone take advantage of it. That said, an increasing number of individuals use the internet every day to purchase what they need, focusing on the average Joe, who open their browser wishing to buy an item, which, in today's world can be an electronic, articles of clothing, or even groceries.

According to a statistic on the number of e-customers published in 2018 [1], there were 1.66 billion individuals engaging in Business-to-Consumer e-commerce (the graph can be seen on figure 1.1), which, considering an approximate 7.55 billion population count of 2017, this would mean that 21.98% of the global population bought something at least once in 2017 . A statistic on the frequency of purchases made by online shoppers [2] states that, 20% of e-customers did online shopping once a week, 25% once every 2 weeks, 31% once a month, 15% 3 to 4 times every 3 months and 10% once every 3 months . All these 1.66 billion people, making an average of 2 purchases every month generated 2.3 trillion US dollars of revenue in 2017 and the total revenue generated by e-commerce transactions is predicted to hit 4.9 trillion US dollars by 2021, based on a statistic regarding ecommerce revenue [3]. This graph can be seen on figure 1.2

Humans as consumers, on the other side remained unchanged in two aspects which are relevant to our situation: one being that they will always try to seek the easiest way of doing a task and the other being the nature of constantly trying to spend less on the product they wish to buy. Unfortunately, from these perspectives the modern webshops

[1]The numbers for the 2018-2021 interval are estimates based on the actual data

[2]Image from [1]

[3]Image from [4]

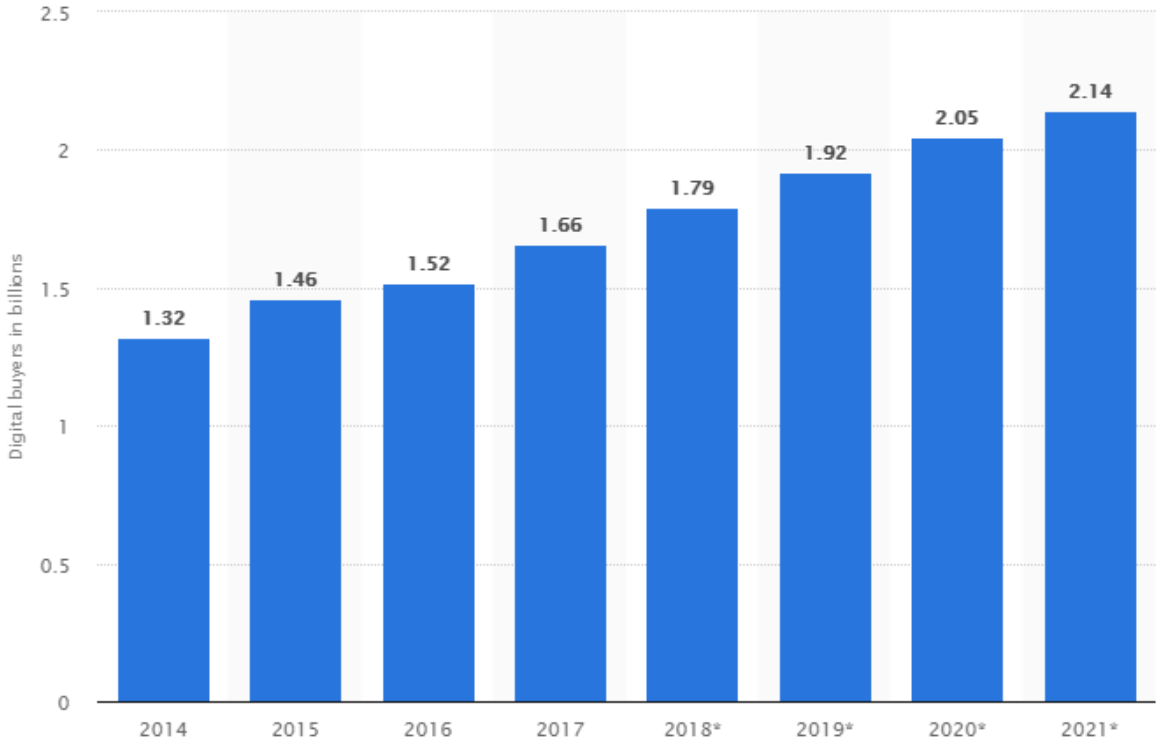

Figure 1.1: The total number of digital buyers by year, in the 2014-2021 interval [1] [2]

can be a blessing as much as a burden. They allow you to purchase whatever you need and have it delivered to your doorstep with the minimal amound of physical work necessary. On the other hand they eliminate the option of good old bargaining, but given the vast number of retailers, prices are much more diverse also. And because of the fact that the customer can just as easily click a few times to buy a product from one retailer as from the other, the greatest way for a consumer to save money on their purchase is to find the retailer with the lowest price. As such, people who intend to save some money on their online purchases end up spending valuable amounts of time looking for a product in the webshops they know, which is a very limited number from the ocean of options they would have. These people usually end up just accepting the price they find in the first few webshops to reduce the time they spend shopping. What's more, some actually trustworthy webshops might not even have a very user-friendly interface, often confusing customers when they wish to purchase a product. This means that the probability of finding the lowest price is minimal and the process is extremely time consuming, inefficient and often tiring.

Online retailers, very similarly to their offline counterparts, have a major target in their activities too, that of reaching as many consumers as possible, and creating customers out of them. This, of course can be done via advertisements, but reaching a great amount of people with an advertisement placed on a high traffic website, such as a social network, can

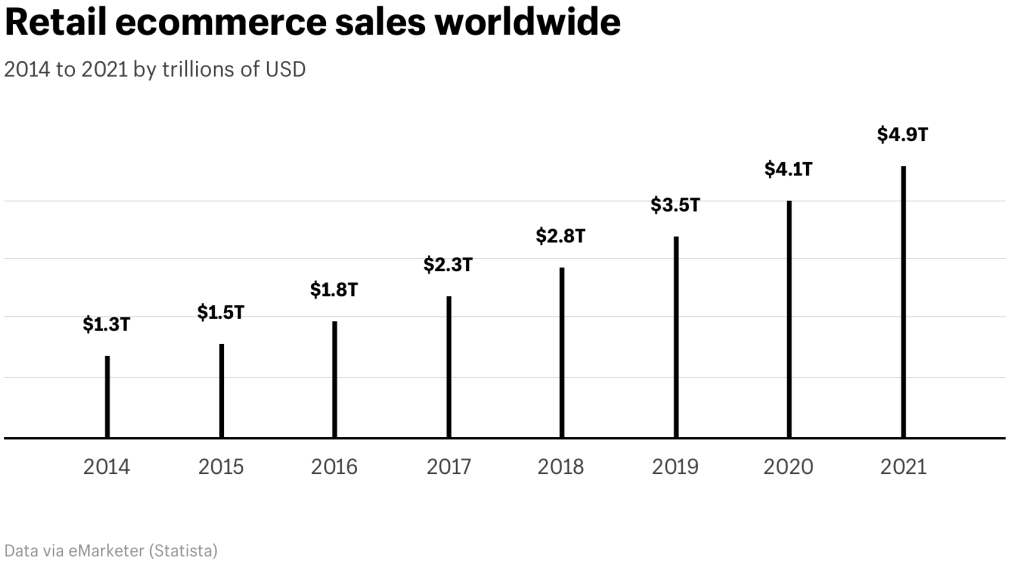

Figure 1.2: The total revenue generated by ecommerce sales by year, in the 2014-2021 interval [3]

be extremely expensive nowadays. Taking into consideration another factor, such as the possible bad reputation the website on which the advertisement appears can also negatively impact the web retailer's or the product's image in the consumer's subconscious. The final, and probably greatest threat to the advertisement based attempts to reach customers is the usage of ad-blocking software, which modify accessed websites' DOM, completely eliminating elements containing advertisements. According to some studies regarding ad-blocking software usage [5], 236 million desktop users actively used ad-blockig software in December 2016 and 2.3 million mobile users browsing the web from a browser that blocks advertisements by default, majorly due to the fact that 'too many ads are annoying or irrelevant'. This means that many users wont even get to see the advertisement the retailer is paying for, and even if they do, it's highly probable that it is of such poor quality or so invasive that the user will just get annoyed by it and he will develop a kind of hate for it.

Chapter 2

Project Objectives and Specifications

2.1 Introduction

The purpose of this chapter is to collect, analyze, and define high-level needs and features of this license thesis. It focuses on the capabilities needed by the stakeholders and the target users, and why these needs exist.

2.2 Positioning

2.2.1 Problem Statement

Everyday more and more people are using the internet to buy what they need, from car tires to shoes, from a mobile phone to groceries. Due to the fact that the number of people shopping online increased thus increasing the demand for these opportunities, a naturally companies increased the supply by each creating one or more webshops offering the same product at different prices. This leads to the clients being lost among the options that have, and in a world where you would want to save money at every spending, a tool is necessary for the everyday webshopper to find the best price for their preferred product.

The problem of	looking for the cheapest offer for a product a client wishes to buy
affects	everyday people doing their shopping online
the impact of which is	excessive time spent looking for an offer, ending in unsatisfactory results
a successful solution would be	easy to use easily accessible able to track the prices of a set of products across multiple webshops easy to extend to cover other webshops

2.2.2 Product Position Statement

The Financial Efficiency Boosting System (FEBS) comes as a solution to the problem presented in the previous section by the use of webcrawling, a web platform and a Google Chrome extension.

For	customers of webshops
who	need a tool to simplify their search for a good price
The FEBS	is a system which tracks a set of products on online marketplaces
that	stores current data about products and tells the user about the lowest price of the product they are looking at
unlike	compari.ro
The FEBS	will be more accessible

2.3 Stakeholder and User Descriptions

2.3.1 Stakeholder Summary

Name	Description	Responsibilities
Webshop customer	Person who engages in online shopping	Find the product they wish to buy and/or track
Online merchant	Retailers of products who have their products tracked in the FEBS	Have the data about their products as clear and accessible as possible
FEBS developer	Person who creates and maintains the FEBS	Create, improve and offer technical support for the FEBS

2.3.2 User Summary

Name	Description	Responsibilities	Stakeholder
Unregistered customer	Person who rarely engages in online shopping	Access the application to find a good offer	Webshop customer
Registered customer	Person who frequently engages in online shopping	Access the application to find a good offer and add products to their favourites list	Webshop customer

2.3.3 User Environment

2.3.3.1 Users

The application is public and the number of users may fluctuate based on the time of the day/week/month/year. The total number of supported users depends on the server.

2.3.3.2 Time Limits

The application should be available all the time, except for maintenance downtimes or unpredictable/uncontrollable downtimes such as power outages. A user can use the application anywhere from a minute if he finds an ideal price to a large time period if they wish to add a product to their favourites list to have it in an easily accessible location.

2.3.3.3 Collaboration

The application is used by a single person, anything a user performs with the system should not influence other users' experience.

2.3.3.4 Infrastructure

The application will be accessible from web browsers. For full functionality a desktop based Google Chrome will be necessary, as this is the only browser to which an extension will be developed for further ease of use.

2.3.4 Summary of Key Stakeholder or User Needs

Need	Priority	Concerns	Current solution	Proposed solution
Support a multitude of domains	0	Customers, Merchants	Crawler for each domain	Analyze and implement a crawler for each new domain
Easy to use interface	1	Customers	Web platform	Google Chrome extension which recognizes its environment
Have an up to date database of items	1	Customers, Merchants	Crawlers for each domain	Run the crawling task at regular intervals

2.3.5 Alternatives and Competition

There are similar tools currently available. One of those is compari.ro, which also lists the different webshops and prices for an item, but the fact that a client would have to actually access the website technically makes it harder to use than accessing a Google Chrome extension that is always present and provides data at one click distance.

2.4 Product Overview

The Financial Efficiency Boosting System should provide an easy to use way to users to see the best offer for their desired product. This concept can be seen in figure 2.1.

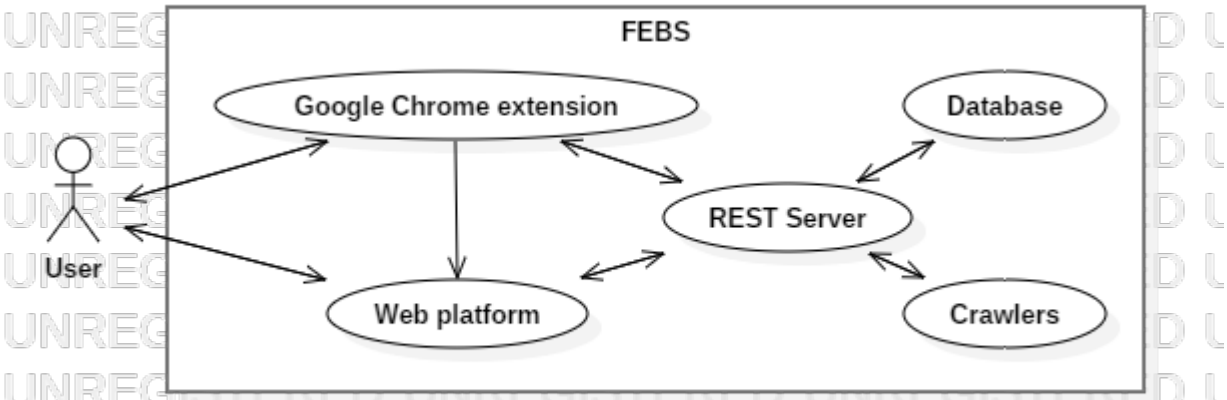

Figure 2.1: The FEBS's system diagram

2.4.1 Product Perspective

Compared to some competitors, such as compari.ro or PriceMon, FEBS aims to further ease the interaction of the user with the system by offering a font-end through a Google Chrome extension. FEBS is a standalone application thus it is not part of a larger system.

2.4.2 Assumptions and Dependencies

A client is assumed to have a constant internet connection while using the application.

For the server side:

Database	**MySQL**
Programming language support	**Java 8**, **Python 3**, **Angular 7**
Frameworks, packages	for Java: **Spring framework 2.1.3** or larger, **MySQL connector 8.0.13**, **Commons codec 1.12** or larger for Python: **Scrapy 1.6.0** or larger, **requests 2.22.0** or larger for Angular: **Angular CLI 7.2.4**
Miscellaneous	**Node.js**, **Node Package Manager (NPM)**

For the client side:

Web browser	**Google Chrome 75.0.3770.100**

2.5 Product Features

1. **Simplistic web platform**

   The web platform should be as simple as possible so that the user can use and navigate it with maximum ease, while still offering the necessary features to accomplish the necessary.

2. **Context recognition**

   The Google Chrome extension should automatically check if the user opens a link to a supported domain, in which case it automatically activates in case the user accessed the URL of a supported product it can shows different things if the webshop sells the product at the cheapest price or not: a message stating that the user is looking at the best offer or a link to the product page of the webshop with the cheapest price.

3. **Data monitoring**

The system tracks changes about the supported products and refreshes the already available information and each distinct running of the respective crawlers.

2.6 Other Product Requirements

1. **Visually minimalist interface**

   The front-end of the application should be as clutter-free as possible, thus improving intuitiveness and ease of use.

2. **Usability**

   The application should require close to no input from the user, as it should recognize its working environment to automatically help the user find the cheapest offer. In case user input is necessary, it should be as intuitive as possible to do this, such as interacting with a search bar on the web platform.

3. **Performance**

   The system's performance is measured in the client side response time, which should be at most 5 seconds for most operations.

4. **Availability**

   Besides maintenance downtimes, which should take at most 2 hours each week, preferably in a time period when there is as little user activity as possible, there shouldn't be any availability issue.

5. **Scalability**

   The application should be able to serve many concurrent users without introducing too much stress on the system.

6. **Maintainability**

   The system should be easily maintained, as most problems could be solved by either rebooting one of the services, or fixing an outdated webcrawler.

7. **Extensibility**

   The application should be easily extensible by adding supported products, adding URLs from supported domains to the existing products to gather data about them, or create further crawlers to support more domains.

Chapter 3

Bibliographic Research

3.1 RESTful Web Services

The concept of Representational State Transfer (REST) as an architectural style for distributed systems was invented and presented in 2000 by Roy Thomas Fielding in the 5th chapter of his doctoral dissertation [6]. Fielding described it as follows: "REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems".

Fielding build the REST style as a hybrid style, including constraints from other, already defined architectural styles. REST was conceived from the ground up, starting with the client-server architecture, adding several constraints: statelessness, caching, a requirement for uniform interface, a layered system, and code-on-demand. The client-server architecture was used as a first step in bulding the architectural style. This was necessary in order to divide the user interface from the data storage, thus portability and scalability was improved. Adding statelessness made it mandatory that all the information necessary for the server to fulfill the request to be included in it. This means that all the information regarding the session has to be stored on the client. Statelessness is a compromise between reliability, scalability and visibility, at the downside of a potential hit in network performance due to more and more repetitive data being sent in a sequence of requests. Fortunately networking technology advanced from the, back then state of the art twisted-pair copper telephone wire solution using ADSL to provide speeds of 10 Mbit/s with 56 kbit/s download speeds being more common, while today's optic fiber solution is using the G.fast protocol to reach 1 Gbit/s speeds [7], so the presented disadvantage of statelessness shouldn't be an issue nowadays. The option of caching response data was added to improve network efficiency. This means that whenever a response is sent, data should be labeled as cacheable or non-cacheable. In case a response is cacheable, a client may store and reuse the data for other equivalent requests. This allows some interactions to be eliminated, scalability and efficiency to be improved. The downside introduced with caching is that

cached memory can reduce reliability due to stale data being reused. The main introduced aspect was an emphasis on making the interface between components uniform. Adding the concept of generality to the component interface simplifies the system architecture, improves the visibility of interactions and dissolves coupling between implementation and provided service, which in turn allows independent evolution of the two. The trade-off lies in decreased efficiency due to the data being passed in a standard form, rather than one tailored to the application's needs. Interface uniformization is achieved through the application of four constraints: resource identification in requests, representation based manipulation of resources, the usage of hypermedia as a means of keeping the application state and self-descriptive messages. The concept of a layered system was introduced to protect and encapsulate services, creating a downside by adding some latency to the processing of data. Finally, code-on-demand was introduced to decrease the complexity of the client.

Web services that take advantage of the REST architectural style are called RESTful web services. These kind of web services grant compatibitlity between software systems. By using RESTful web services, client systems are capable of accessing and managing web resources by using a uniform and pre-defined collection of stateless operations. Web resources are defined as documents or files identified by their Uniform Resource Identifiers (URI), and accessed using their Uniform Resource Locators (URL). RESTful web services nowadays access web resources which are exposed an a URL form. The most commonly used way to communicate with a web service is via the Hypertext Transfer Protocol (HTTP). Systems can use the URLs to send HTTP requests to the web service, which then return an HTTP response with an error message, in case something went wrong, or the data requested. The data can be the contents of a file or the result of a method. This whole process is quite similar to a RPC based web service, but it's a less rigid solution than, for example, a SOAP based solution, which responds to specific method names [8]. Thus, RESTful web services profit of the code-on-demand feature of the architecture while also keeping coupling to a minimum. The most common formats for representing data in an HTTP request or response is HTML, JSON or XML.

3.1.1 The Spring Framework

Spring is an application framework developed by Pivotal Software for Java, first released in 2002. It has several modules, providing a great number of services, the most important ones for me being the data access module, the inversion of control feature and the model-view-controller module.

Inversion of control (IoC) is a software engineering principle, falling into the architectual design patterns category, regarding the flow of control of the code. This principle inverts traditional flow, giving full control of execution to the used libraries, decoupling elements, increasing modularity and enabling extensibility. The concept of IoC is tightly related to the priciple of Dependency Injection. The Spring framework takes advantage of this pattern by integrating them. Dependency Injection is the standard way of linking

components in the framework.

The data access module connects relational database management systems with the application using the spring framework. This is done using the Java Database Connectivity API (JDBC) and object-relational mapping tools, such as Hibernate ORM. Spring uses Hibernate ORM to map plain old java objects (POJO) to data which is supported by databases such as the MySQL relational database management system. Hibernate then further uses JDBC to connect with the database.

The most important feature of spring for my situation is the model-view-controller module, which, being a servlet and HTTP based framework, makes the creation of RESTful web services easier. The Spring MVC module is based on the architectural pattern with the same name, thus introducing the decoupling of the model, view and controller from eachother. The model part of an application is the one that deals with the object modelling of the application's domain. This term encapsulates data management and logic. As part of the model-view-controller interaction cycle, it is responsible for receiving user input from the controller, executing it and sending updates to the view if necessary. The view part of the application is an independent user interface through which information from the model is exposed to the user. The controller of the application is the part which accepts the user input, validates it if necessary and passes it to the model as commands to be executed.

The Spring framework, uniting all its modules becomes an easy to use framework that greatly helps a software developer in building web applications with the Java EE (Enterprise Edition) platform. The lowest element of a Spring MVC based application are the entities. These are the classes which model the domain of the application. The Spring framework heavily relies on a set of annotations to mark certain properties or automatically generate beans from classes which can then be injected in other classes. Entity classes are no different, as have to be marked with the '@Entity' annotation before each class definition to be discovered by the framework. The attributes of these classes have to be annotated according to the position they will fulfill in the database. Based on these annotations and the attribute names Spring can automagically generate the complete database in the defined database management system, in such a way that the created database will be a prefect representation of the application model domain, supporting the data to be manipulated. The basic communication with the database is encapsulated in interfaces called 'Repository' and annotated with the '@Service' annotation so that they can be injected in the appropriate component in the business layer. These use the objects defined as 'Entities' and execute the SQL code that Hibernate generated based on the command given to it. After executing the SQL code, JDBC gets a value or set of values from the database and returns them further into the application, to the Hibernate session, which in turn converts it to the appropriate Entity object using ORM and returns it to the calling Repository. Classes encapsulating the business logic of the application also have to be annotated with the '@Service' annotation, so that they in turn can be injected into the 'Controller' classes. The direct interfacing with the rest of the application is done via the layer of 'Controller' classes, which are annotated with the '@RestController' annotation to mark their nature, and '@RequestMapping' annotation marking the path for finding the

resources that are encapsulated in it and have further annotations to aid in locating them.

Using these mechanics and architecture, Spring becomes an ideal framework to use in the creation of a RESTful web service, which would then form the back-end of a full stack application.

3.2 Web scraping

A web crawler is a bot used to automatically explore web pages on the World Wide Web as a method of web indexing. A web scraper is a specific kind of web crawler, with the difference of it being used to extract data from the visited websites. The extracted data is usually persisted in a database for later analysis and/or manipulation. Web scraping is the act of using web scrapers in a system to extract some data from websites and then persist it in a database.

A web scraper operates on a set of simple steps. First, it receives a set of URLs usually called seeds to start scraping. The scraper sends an HTTP request to the URL to fetch the page, the response usually being in the form of HTML but in case the scraper has direct access to the application API, application data can be an option, which can be encoded in JSON or other formats. The received data then has to be parsed and preferably put into objects so that it gains a uniform structure. As a final step, the processed data should be stored in a database for further use. Scrapers may have a multitude of uses, such as data mining, online price monitoring and price comparison, contact scraping, weather monitoring and website change detection.

Web scraping is a very valuable tool in the modern world, seeing as the internet is filled with so much available and useful information publicly available. The development of the internet and the amount of content it holds has been much more rapid than that of web crawling solutions. As such, the domain of web crawling is still a field with major potential and space for major breakthroughs in fields such as text processing or artificial intelligence. Web scraping solution in use nowadays include manual copying, regex matching, HTTP request based scraping, HTML parsing, DOM parsing or computer vision based web scraping. The technique of manual copying involves a human agent manually opening a website on a browser, locating the necessary data on the webpage, copying it and then pasting it into a database. This method is extremely time and money consuming, as a bot does not need to perform physical actions and does not ask for a monthly salary. Unfortunately there are some cases where automated scrapers aren't able to do the job due to scraping blockers or lack of accuracy, therefore human labor becomes necessary. The regex matching technique involves taking the whole webpage in textual form and applying a regular expression based search on it to find the data necessary. This, again is quite inefficient and might even result in inaccuracies, because website htmls can be quite lengthy, finding the correct regular expression might take a very long time, and even be so long that it's would not be worth doing. HTTP request based scraping involves directly sending HTTP requests to a public API of the website we wish to crawl, thus having its

resources exposed. This is by far the most efficient scraping method, as there is not too much hassle for the development team, as they don't have to bother with parsing the website in any way. Unfortunately there aren't too many websites freely exposing their API. HTML and DOM parsing are quite similar to eachother, because in the case of HTML parsing, the code is first converted into a DOM, and from there the technique is the same: the adequate elements are selected from it to narrow down the amount of text that has to be processed by other means. HTML parsing is the most commonly used attempt to web scraping. Computer vision based web scraping is a method which appeared as a response to the attempts to block other types of scraping bots on certain sites. Computer vision brings a more modern approach to scraping by combining machine learning with image processing to locate and extract data from a website. This web scraping method is much more computationally intensive than any other one before, therefore it cannot be used as a main approach to scraping.

There are several tools that could help in web scraping or pre-build web scrapers that already solve the problem. Wachete is a browser based tool for website change monitoring. Users have to include the URL of the page they wish to monitor with an option to then monitor its subpages up to three levels deep too. The step after giving a starting URL is selecting an area on the downloaded and presented page so that the tool can create a selector for it, setting the format of the data to be checked, and the frequency of the monitoring process. This means that the user is basically given a user friendly interface to create a scraper, which has every element necessary for a simple scraper: a starting URL, the crawling depth, the crawling frequency, and a single entry of data. cURL is a command line tool that can send requests to URLs, supporting a multitude of HTTP methods, as well as the inclusion of headers or cookies in the request. This makes it a decent tool to test the parameters of requests before implementing them in an actual scraper. Diffbot is a company which develops algorithms and APIs to extract infromation from a website based on machine learning and image processing. In the situation when a developer wishes to develop their own scrapers they would need extensive knowledge about HTTP requests, HTML, CSS, JavaScript and security elements such as session data. Fortunately there are frameworks specially designed for web scraping.

3.2.1 Scrapy

Scrapinghub, the developer and maintainer of Scrapy defines it as "an application framework for crawling web sites and extracting structured data which can be used for a wide range of useful applications, like data mining, information processing or historical archival." [9]. It was first released in 2008 and is still in constant development, having reached the last major update, version 1.6 in January 2019. Scrapy presents itself as a quite complete and well documented web scraping solution. It includes several elements which are critical for web scraping: scrapers (called spiders) for data and link extraction, a powerful request making mechanism, both an XPath and a CSS based HTML selecting solution, a complete logging system, command line integration for generating projects

and spiders, opening webpages for the developer to see how the framework sees them, or starting crawling tasks, amongst others.

3.2.1.1 Scrapy Architecture

The Scrapy architecture is constructed from a few main elements. Understanding these components and the way they connect is vital for using Scrapy to its full potential An overview of the architecture and the data flow between its components can be seen in figure 3.1.

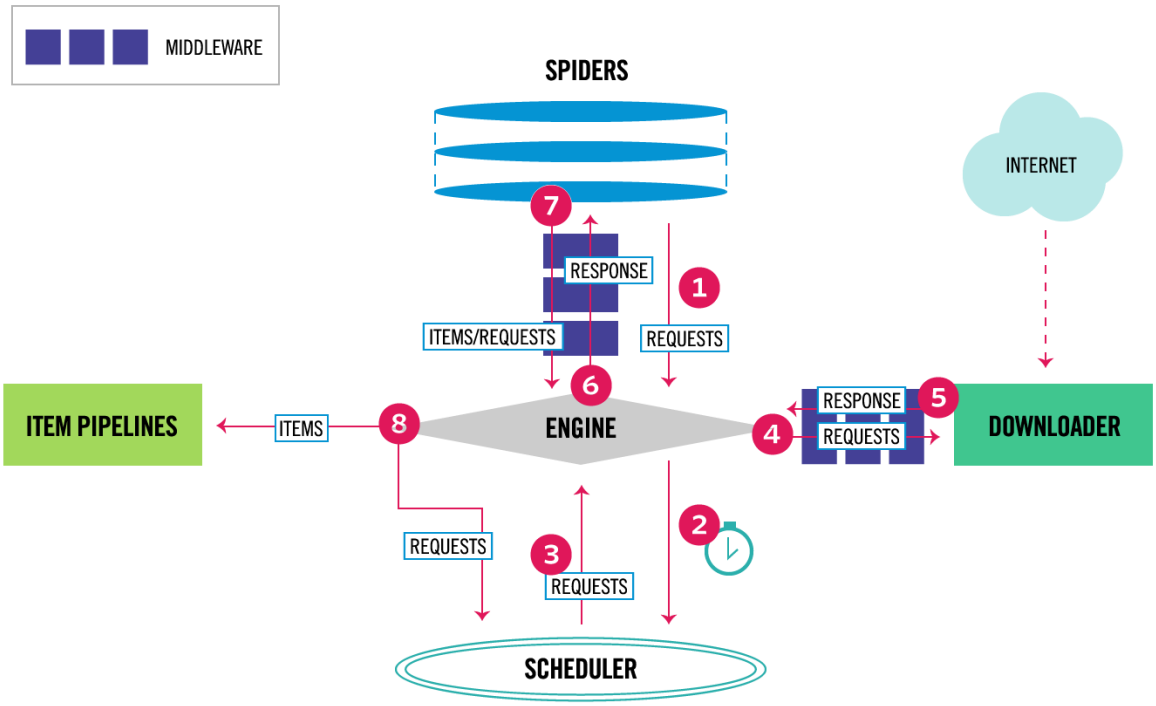

Figure 3.1: Overview of the Scrapy architecture and data flow between its components [4]

At the center of the architecture seems to be the Scrapy Engine, which controls the data flow between its elements, but actually the whole system revolves around the spiders. It is worth noting though that the Scrapy Engine is built on the Twisted event-driven framework. This means that the whole cycle is implemented using non-blocking code, granting concurrency to the applications written using Scrapy. Therefore writing blocking code while using Scrapy is contradictory to the whole idea behind this architectural decision [11]. The whole scraping process starts with when a spider is invoked, either via the Scrapy

[4]Image from [10]

command line interface or from a script. The spider then sends the details of the requests built based on the given start URLs, headers and cookies to the Engine. The Engine then passes these requests to the Scheduler adding a delay between the requests which are sent to the same website. The value of the delay is set by taking values set in the settings file of the project, or directly from the spider, if it has a download_delay attribute defined. If neither are defined, the delay takes a default value, which is 0. Setting a download delay might help with avoiding some scraping detection and blocking mechanisms which detect the interval of time and the regularity of the requests received from a given IP address. The Scheduler keeps returning requests to the Engine every time a previously sent request is completed, thus controlling the number of concurrent requests. The Engine forwards the recieved request from the Scheduler to the Downloader, which then fetches a response based on the given request and returns it to the Engine. This interaction between the Engine and the Downloader is done through a framework of hooks into the request/response handling. This part of the architecture is composed of elements called 'Downloader Middleware'. When activated, Downloader Middlewares alter the requests and responses passing through the Engine-Downloader section. Upon receiving them, the Engine in turn routes the responses to the spiders. The spiders then process the information contained in the request. One way of using Scrapy is extracting data using the built-in XPath and CSS selectors and inserting it into Items. Alternatively, additional requests can be created by using the spiders to extract URLs from the response similarly to how Item filling data is extracted. This second part of the iteraction between the Engine and the Spiders happens through elements called 'Spider Middleware'. They, similarly to the Downloader Middlewares, are a framework of hooks into the spider processing process. Spider Middlewares might seem to be useless at first, since the spiders should compeltely process everything. This might be true in theory, but there may be situations in practice where start requests, returned Items or extracted requests need post-processing, or some exceptions returned by the spider need to be handled. The extracted Items and generated requests are sent to the Scrapy Engine to be accordingly distributed: the requests generated from the URLs are sent to the Scheduler and the Items are sent to the Item Pipeline. The Item Pipleline is composed by a sequence of scripts that manipulate the returned item. They can be used to clean, validate and/or persist the data in the Items. The process repeats itself from the point where the Scheduler returns requests to the Engine until there are no requests left to process.

3.2.1.2 Spiders

A Web scraper which is generated by the Scrapy framework and run in the Scrapy cycle is called a 'Spider' according to scrapy. According to the regular definition of a web scraper, spiders take a list of starting URLs and at the end returns extracted data. Of course, in the background, the way this happens is more complex. Spiders form the core or the Scrapy cycle, as they create the requests to be sent and they process the response of those requests. Essentially, scraping couldn't exist without these spiders.

To be able to generate spiders, you should be inside the directory of a Scrapy project. Scrapy projects can be created using the scrapy startproject <project_name> [project_dir] command line command, which gets the project name as an argument, as well as the preferred directory for the project to be generated in, as an optional argument. Spiders can be generated using the Scrapy command line tool using the scrapy genspider [−t template] <name> <domain> command line command, which gets the spider's name and the web domain on which it should operate as arguments and as an optional argument, template, which is used to so that Scrapy can generate the spider from a list of predefined spider templates. If the 'template' optional argument isn't defined, it creates a blank spider in the 'spiders' directory of the currently open project. In case you are not in a project directory, the spider will be generated in the current directory, but this is not an ideal situation, since the spider can't run alone without a Scrapy environment and it will need to be moved to an adequate directory.

A generated spider implicitly includes some functionality and it automatically inherits the settings defined in the project unless explicitly defined otherwise. Implicit functionality may always be redefinied explicitly, thus overruling the minimal implicit definitions. One of the implicit methods is start_requests(), which takes the start_urls and generates an iterable which includes the initial requests to be scheduled and performed. This iterable is then forwarded to the Scrapy Engine according to the description in subsubsection 3.2.1.1. When the spider receives the response to a request, one of two options can be executed: the callback method passed to the request is executed, or, in case no callback method was passed, the execution defaults to the parse() method, but essentially the all same the same purpose: parsing the response received to extract. Since the response is usually in the HTML format, parsing it most commonly consists in parsing the returned HTML. This can be done in two ways: with CSS selectors or with XPath expressions, by using the response object's .css() or .xpath() methods. XPath is a very powerful language which is used to select nodes in an XML documents, and it can be used in HTML aswell. It is worth noting that selecting elements using the CSS solution converts the selector to an XPath selector in the background, and applies this on the HTML to get the requested node. As thus, almost everything about the two object methods is equal after the CSS selector is transformed into an XPath selector. Both of them are implemented using 'parsel', a Python library created to extract data from HTML and XML using XPath and CSS selectors. Both of them return a SelectorList which can be used to further apply selectors to it, or data can be extracted via the .getall(), .get(), .re() or .(re_first). The .getall() method returns a list of extracted results, while .get() returns the first of that list regardless if it there are more or not. The regular expression based methods are similar to the previously presented ones, except for the fact that they apply the regular expressions passed as parameters onto the selected result(s) and return either a list of unicode strings, in case of .re() or a single unicode string, which is the result of applying the regular expression to the first element in the SelectorList, in case of .re_first(). The resulting extracted data should be returned either as part of an Item or as the URL part of a Request element of an iterable containing the same type of elements. From this point onward it is either the Scheduler's job to

continue with adding the returned Requests to the current list of incomplete requests, or some part of the Item Pipeline's job to manipulate the returned Item(s) in some way.

3.2.1.3 Items

Scrapy defines Items as "simple containers used to collect the scraped data." as part of their official documentation [9]. Even though Scrapy spiders can return the extracted data in the format of Python dicts, they lack structure an introduce the risk of making an error in the field name or introducing inconsistent data. The purpose of Scrapy Items is to provide a uniform structure to data outputted by spiders without taking away the benefits offered by dicts, as they have a dictionary-like API, since its a replication of the standard dict API, including the constructor. The only deviation from the standard dict API is the addition of the fields attribute, which returns every field defined in the Item, not only those which are filled. Declaring fields in an Item is extremely convenient due to the usage of the Field objects defined by Scrapy. Additionally, some Scrapy components take advantage of different features of Items for tracking memory leaks, serialization or data exporting.

In Scrapy, custom Items are defined as classes which extend the scrapy.Item class. Fields are then defined by assigning scrapy.Field() objects to the custom Item's class attributes. The Field objects' purpose is to define a field's metadata in one place. There is no restriction on what that metadata can or should contain, therefore a developer choose to omit everything or define any new metadata which is necessary for their project. Items can also be extended by simply declaring a subclass of the original custom Item. Subclassing Items may change or add metadata to the original Item's fields, or add additional fields to it.

The two main components beside spiders which use Items are Item Loaders and the the Item Pipeline. Item Loaders are a way of conveniently populating the fields of items with extracted data. Even though Items can very easily be filled by using the dict-like API, Scrapy states that "Item Loaders provide a much more convenient API for populating them from a scraping process, by automating some common tasks like parsing the raw extracted data before assigning it." [9]. The Item Pipeline is one of the main components of the Scrapy architecture. After a spider has finished extracting data from the response and filling an Item either via the dict-like API or an Item Loader, the Item will be sent to the Item Pipeline. The Item Pipeline processes the returned Item further by sending it sequentially through several components. Activating an Item Pipeline component is done in the settings of the project, by adding the component's class to the ITEM_PIPELINES setting and assigning them an integer value in the 0-1000 interval. The assigned value dictates the order of the components that the Items returned by the spiders have to pass through. Item Piplelines must implement the process_item() method, which takes two parameters. The first parameter is called item and represents the item scraped, while the second parameter, called spider and it stores the spider that scraped the item. Typical uses of Item Pipeline components includes filtering duplicate items, field validation and

dropping of items which have errors in them, storing items in a file with JSON being a common choice, or persisting the items in a database.

By the aggregation of the presented basic concepts and many more, Scrapy proves to be an ideal framework for anyone wishing to create a web scraper, be it lightweight or poweful af, either as a standalone application, or a module in a larger project.

3.3 Google Chrome Extensions

Google Chrome is a web browser developed by Google. It first got into public hands in September 2008, being released for Microsoft Windows. Since its first release, the browser has been ported to several platforms, such as macOS, iOS, Linux, and Android

Prateek Mehta defines Google Chrome extensions in his book [12] as:

> (...) browser extensions for the Google Chrome web browser. Browser extensions are programs that run within the context (security sandbox) of a web browser. They help to provide new functionality(ies) by combining existing features of the web browser and make it possible for users to do many things at once!

3.4 Web development

3.4.1 Angular 7

Chapter 4

Analysis and Theoretical Foundation

4.1 Conceptual Architecture

Chapter 5

Detailed Design and Implementation

Chapter 6

Testing and Validation

Chapter 7

User's Manual

Chapter 8

Conclusions

Bibliography

[1] J. Clement, "Number of digital buyers worldwide from 2014 to 2021 (in billions)," Nov 29, 2018. [Online]. Available: https://www.statista.com/statistics/251666/number-of-digital-buyers-worldwide/

[2] ——, "Online shopping frequency according to online shoppers worldwide as of october 2018." [Online]. Available: https://www.statista.com/statistics/664770/online-shopping-frequency-worldwide/

[3] A. Orendorff, "Global ecommerce statistics and trends to launch your business beyond borders." [Online]. Available: https://www.shopify.com/enterprise/global-ecommerce-statistics

[4] "Retail ecommerce sales worldwide." [Online]. Available: https://cdn.shopify.com/s/files/1/0898/4708/files/Retail_ecommerce_sales_worldwide.png

[5] A. Guttmann, "Statistics and facts about ad blocking." [Online]. Available: https://www.statista.com/topics/3201/ad-blocking/

[6] R. T. Fielding, "Architectural styles and the design of network-based software architectures," 2000. [Online]. Available: https://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm

[7] [Online]. Available: https://en.wikipedia.org/wiki/Bit_rate

[8] L. Richardson and S. Ruby, *RESTful Web Services*, 1st ed. O'Reilly Media, 2007.

[9] "Scrapy 1.6 documentation." [Online]. Available: https://docs.scrapy.org/en/latest/intro/overview.html

[10] "Overview of the scrapy architecture and data flow." [Online]. Available: https://docs.scrapy.org/en/latest/_images/scrapy_architecture_02.png

[11] D. Kouzis-Loukas, *Learning Scrapy*, 1st ed. Packt Publishing, 2016.

[12] P. Mehta, *Creating Google Chrome Extensions*, 1st ed. Apress, 2016.

Appendix A

Relevant code

```
 /** Maps are easy to use in Scala. */
object Maps {
  val colors = Map("red" -> 0xFF0000,
                   "turquoise" -> 0x00FFFF,
                   "black" -> 0x000000,
                   "orange" -> 0xFF8040,
                   "brown" -> 0x804000)
  def main(args: Array[String]) {
    for (name <- args) println(
      colors.get(name) match {
        case Some(code) =>
          name + " has code: " + code
        case None =>
          "Unknown color: " + name
      }
    )
  }
}
```

Appendix B

Other relevant information (demonstrations, etc.)

Appendix C

Published papers