Haaga-Helia

# Developing an algorithm for odds aggregator

Nicolae Oat

Bachelor's Thesis
Degree Programme in Business
Information Technology

2016

**Author**
Nicolae Oat

**Degree programme**

Business Information Technology

| **Thesis title** | **Number of report pages and appendix pages** |
|---|---|
| Developing an algorithm for odds aggregator | 29+9 |

Generally, people exceedingly underestimate the power of the Internet and modern information technologies. They provide immense opportunities to explore this world of data and use it in our own good. Each valuable application pushes the technological progress forward, makes peoples' lives easier, and helps them in achieving their goals. This research aims at helping the consumers in navigating the world of bookmakers, an extensive and largely profitable business, which is usually unfair towards its users.

The primary goal of this research is to develop an algorithm that aggregates results from different betting websites, analyzes them and suggests to the user the most favorable ones. The outcome represents a functioning application, which presents to the user the best betting coefficients, possible sure bets, and other useful information in the most usable and comprehensive way.

Scraping was chosen as the primary method of data extraction. To this end, appropriate tools for scraping were utilized to achieve the desired results. Another research was performed directly on the betting websites. This research affected the scope of the project. Practical process is the foundation of the thesis. It consists of three stages: design, implementation and testing.

The whole development process was run using the agile methodology Scrum. In this case, the Scrum is employed in a non-traditional manner, when one person combines the roles of Product Owner, Development Team and the Scrum Master. It was decided that this method would increase the efficiency of the project and would allow to beat the deadline.

After the application is finished and all three phases (design, implementation and testing) are successfully implemented, it is possible to say that the major goals of the research were achieved, all tasks completed and the result is a working application without bugs and a decent user interface. This application can be used in real life. However, the work on it can be continued in order to enlarge its functionality and improve its usability.

**Keywords**
scraping, Scrum, odds, Django, web application, Selenium

# Contents

## Terms and Abbreviations

| | |
|---|---|
| DOM structure | Document Object Model |
| MVC architecture | Model-view-controller design pattern |
| ER diagram | Entity-relationship model |
| GUI | Graphical user interface |
| Agile sofware development | A set of principles for efficient software development in teams |
| pip | Python package management system |
| Front-end engineering | The part of software development, responsible for interaction with end-user |
| Back-end engineering | The core of the software development, acts in support of the front-end |
| Product backlog | List of all functionalities, required for the product |
| Sprint backlog | The prioritized list of tasks for the sprint |
| Product Owner | The voice of the customer, responsible for ensuring that that the team delivers the desired product |
| Development Team | Team of developers, working on the project |
| Scrum Master | Manages the Scrum process, a buffer between the development team and all kinds of possible impediments |
| Scrum Sprint | A repeatable work cycle of Scrum, usually one or two-week long |
| Sprint review meeting | Held in the end of each sprint. Team reviews its achievements, often informal |

# 1 Introduction

Most of the people exceedingly underestimate the power of the Internet and other modern information technologies. They provide immense opportunities to explore this world of data and use it for our own good. Big Data is a hot topic, which helps companies to process large quantities of data and extract results. Each valuable application pushes the technological progress forward, makes peoples' lives easier, and helps them in achieving their goals. This research aims at helping the consumers in navigating the world of bookmakers, an extensive and largely profitable business, which is usually unfair towards its users.

## 1.1 Author's personal experience in the domain

The topic is related to something that author has been doing for over twelve years. For the author, sports betting was more than just a simple way to try luck or prove to someone his knowledge in sports industry. The author looked at it as a mathematician, as a scientist, starting betting while being a rookie in the university, studying the Applied Mathematics. At that time, being young and self-confident, he was sure that it is possible to apply (as the name of the degree programme is suggesting) author’s studies and make from hobby a profitable business. Unfortunately, this idea failed, and there are multiple reasons for that. The most important difference is that at that time, author did not have the power of the Internet on his side, this activity was solely offline. Nowadays, this is not a problem anymore, because author has all the technologies available.

## 1.2 The goals of the development project

The primary goal of this research is to develop an algorithm, that will aggregate results from different sports betting internet pages, analyze them and suggest to the user the best ones. Naturally, the amount of data in this particular area is immense, so this project analyzes only a part of it. If successful, the algorithm can be later used to find the best offers and the best value for the end-users.

In order to achieve the above stated goal, it is required to investigate a range of sports betting websites. From this range a few are picked, that allow for scraping, possess a structured and clear HTML code, and last, but not least, present reliable web applications.

The outcome of this research represents a functioning application, which presents to the user the best betting coefficients, possible sure bets, and other useful information in the most usable and comprehensive way. The complexity of this algorithm is determined during

the research itself. For example, this application will compare different betting options, and obviously, the more options there are the better is the application. The problem is that every bookmaker company offers hundreds of possible bets on every sports event, which are almost impossible to track, compare, and analyze. This research includes the most significant betting options, as following: 1 (the win of the home team), X (the draw), and 2 (the win of the visiting team). Also for the search of the possible arbitrage bets are needed the following betting options: 1X (the home team will win or draw), X2 (the visiting will win or the draw) and 12 (there will be no draw). All of the other options are out of scope.

Another challenging task in this research consists in the comparison and analysis of the live bets. Live betting represents the action of placing the bets at the moment of the event happening. This is a challenging assignment because of the high volatility of the live bets. Often (every 3-4 minutes) the access to the live bets is partially or totally closed. This usually happens if some important or even crucial changes are going on (e.g., dangerous free kick, penalty kick or red card in football). Among other specific features of the live betting is that the probability of finding a sure bet is much higher than in before-the-game bets, so it gives possibilities to track down and find no-lose bets. The practical drawback of this strategy is that in real life it is almost impossible to use this advantage, or better said, extremely difficult. Live bets have very short lives (sometimes 20-30 seconds), so within this short time bettor has to track down the sure bet, place two bets on at least two different betting sites, and get both of these bets accepted. The bookmaker companies usually protect themselves from sure-bet-hunters by giving a 5-seconds delay for every live bet placed, which is not necessarily a guarantee of safety, but creates serious problems and risks for those who are in search of no-lose bets.

Besides the importance of a functional application, the developer ensures that the product is user-friendly, and is simple to use. The application presents only the necessary amount of information, leaving out the unneeded details. The most relevant information for the user is highlighted, so it is easily visible when accessing the website. In case an arbitrage is found, an alarming message appears which is meant to draw bettor’s attention.

No matter how complex, due to the posed requirements, the final application is, it has to deliver useful features for people who bet on sports. In other words, it has to optimize and summarize the process of finding the best value coefficients, saving in this way these people time and energy.

### 1.2.1 Research questions

One of the most important questions of the research is whether it is possible to extract online data from the web pages, compare them, analyze, and graphically reflect the results within very short period of time (split second). This becomes even more challenging when analyzing the live bets, which are dynamic in their nature. In order to analyze the live bets, it is imperative that the information is updated promptly, otherwise it becomes useless. Before the game has started, the bets (called fixed bets) are considerably less dynamic, almost static, so this part is not as challenging.

The complexity of the goals becomes evident during the research development. Before the project is started, the challenges are still unclear until the actual work begins. There exist several levels of complexity that may be taken when developing the application. Obviously, the lowest level requires less effort and time, while the highest needs more planning, implementation and testing. This level is adjusted during the evolution of the project based on the time left, challenges that appear during the advancement of the project, and other variables.

Among the keywords for this thesis is the so-called arbitrage. Briefly, it means a no-lose bet. In theory, if an arbitrage bet is found, the chances to lose the money is 0%. Just to make things clear, those kind of cases are an extremely rare phenomenon. Basically, sure bets are a Philosopher's stone in the bettors' world, except that it is perfectly real and possible to detect. Among the additional goals for this research, is the discovery of arbitrage.

## 1.3 Scope of the thesis

The scope of this thesis represents a working application, which reflects the fixed odds provided by two or more bookmakers in one type of sports from one league. Additionally, the odds that are analyzed are limited to six types, which facilitate the search of arbitrage cases. The application has to search the possibility of fixed arbitrage bets. Usable design and simplicity is desired.

## 1.4 Out of the scope of the thesis

Out of scope of this thesis are live odds and live arbitrage bets as well as multiple sports and leagues, sophisticated design.

# 2 Theoretical framework

Lead

## 2.1 History of bookmaking

Bookmaking has been a considerably big business for hundreds of years all around the world. From old Chinese games like keno and medieval European games like baccarat or craps to modern online betting, gambling industry has evolved into a huge corporation with profits of billions of dollars. A lot of things have changed during these centuries: the technologies evolved, especially in the past fifty years, the agriculture-based economy was followed by industrial era and finally nowadays we live in a post-industrial world, dominated by the almighty Internet. With all these tremendous changes and crucial inventions some things remained unchanged: the will of people to play for money, or in other words gamble, and their ability to get addicted to this kind of entertainment, complemented by the lack of common sense and logical thinking. This makes them an easy victim of the smart, logical and scientifically-grounded people, which we call bookmakers. This is an unequal struggle. Science always beats ignorance and ice-cold thinking is stronger than emotions. This is how the bookmaking business has become a sure-win lottery-ticket. In other words, this situation reminds a duel – where one of the parts has a machine gun, the other has a kitchen knife. This metaphor highlights the advantage of mathematical knowledge in everything what is related to betting. Bookmaking and mathematics are inseparable. Every bookmaker has to be scientist in order to succeed. The detailed explanation of the relations between bookmaking, mathematics and programming will be later in this chapter. It is important that every clever gambler in order to achieve better results and have higher probability of winning has to use some important compartments of Mathematics. When talking about the choices of a bettor all of these choices have right answers, in case if the player looks to maximize his return or minimize his loss. They all can be at least partially solved through the use of mathematical theory. The intelligent player must have a basic understanding of the mathematics behind the game or games he plays if he wants to survive financially or actually profit (Thorpe, 1984, 3-4).

Almost every mathematical problem can be applied and solved with the help of programming, this tool optimizes and brings to perfection the processes and their results.

## 2.2 Mathematics of bookmaking

Betting is directly related to such crucial branches of Mathematics as Probability theory, Game Theory, Boolean Algebra and so on. Mathematically speaking gambling is just

guessing different types of random events, probability of which is possible to calculate with the help of one of the above mentioned sciences.

Professor Jürgen Maasz from Institute for Mathematics Education (University of Linz) even proposed to introduce this particular topic (Mathematics of Bookmaking) in mathematics courses for adults (Maasz, 2016). It is relatively easy to understand these calculations with basic mathematical knowledge – the four fundamental operations of mathematics are used. If order to understand the mathematical background of daily work of a bookmaker better, you have to get into a lot of statistical models. Students should be able to use their mathematical knowledge for analyzing situations, finding rational solutions for problems and to see structures and the influence of different factors. As a result of this studies students will know more about the calculations of bets and odds" and will be able to understand that gamblers are the ones who give bookmakers money, which would be a good lesson (Maasz, 2016).

The goal of the bookmaker is to offer and accept bets from the player in the right proportions, in order to gain profit regardless of the result of the event. The goal of the player is directly opposite, he has to find the loopholes in the odds, created by the bookmaker, and take advantage of them. The odds for most of the events can change, because of the different factors, that can affect the possible result before the game event started (some important player disqualified, injured etc.)

A true odd is a perfect or clean coefficient without the fee from the bookmaker. For example, in a fifty-fifty event when two perfectly equal teams play, the true odds are 2 for each. In reality, the bookmaker always ensures his profit by lowering the odds. By doing these, he earns an expected profit up to 20-25% of the bet. The less events (even with a higher coefficient) are in the same stake, the higher is the probability to win the bet. This has a very trivial mathematical explanation: in case of multiple bets the logical operation & (AND) is applied, which means the probabilities (as well as the odds) in this combination are also multiplied. It is known that probability is counted in 0 to 1, so it is always less than 1 (alas, there are no sure events available), so all the fractions smaller than 1 after multiplication, obviously, become smaller and smaller, considering also that the true odd was lowered by the bookmaker. That's why the bookmaker is always interested in the bettors playing multiple bets, and actively promotes this kind of combinations. The expected profit for the bookmakers from a double bet compared to the singular bet can be twice higher, which makes betting companies offer all kinds of bonuses to the players. The most popular type of betting options in Finland is "pitkäveto" with more than 10 events, extremely profitable for

the betting companies and exciting for most of the bettors, who unfortunately did not do their Mathematics homework.

Mathematically, this situation can be analyzed without using any complex formulas. Like in previous paragraph, let's consider two perfectly equal tennis players A and B. Their chances to win are 50%. True odds would be 2, but in real life bookmakers will give about 1,85 for each. The expected profit for the bookmaker will be calculated using this formula: 100*(1/(1/1,85 + 1/1,85)) which is about 8,1% profit. Now in case of the combination of two games, where two more tennis players C and D are added, there are four possible outcomes and all of them are equally probable – AC, AD, BC or BD. Probability for each of these combinations is 0,25. In the perfect case, the odd would be 4. Thus, in case of betting 100 euros for any of the winning combinations the bettor would get 400 euros. The actual win of the bettor is calculated using the above mentioned formula: 100*1,85*1,85 and the result is only 342-euro win. Obviously, the difference is significant. The percentage of expected profit for the bookmaker is calculated as (100/3,42) *4 and it is about 16,2%. Here is see the difference between the expected bookmaker's win for a single bet (8%) and for the double bet (16%). Generally, the compound expected profit is calculated by the multiplication of the expected wins using this formula – (A1*A2*...*An) *100-100.

### 2.3 The odds comparison and analysis

The importance of odds comparison and analysis is difficult to overestimate, as it increases considerably the probability of the bettor’s profit. Obviously, the odds for the same event vary from a betting company to another. The difference can be shocking, especially when it comes to live bets, where risks are higher, but also possibilities to find a loophole are significantly bigger. When betting online, the comparison of different bookmakers is utilized in order to find the best value coefficients.

The bettors, who develop their own betting system, have a tremendous need in an online odds aggregator, which shows, compares and analyzes odds of different betting companies. This ensures that the player will find the best available odds and will maximize his success rate.

Another important advantage of the odds aggregator is that provides an automatic search of arbitrage situations.

## 2.4 Arbitrage betting

Arbitrage betting, also called sure betting or just “arbing” is one of the most well-known techniques in sports betting, and definitely the most controversial. On one hand, it is the only possible methodology to have a consistent profit by betting online, on the other hand, it has its own risks, which cannot be noticed by those who don't have enough experience in the domain. The technique is considered to be a 100% profitable, but there is nothing sure in this world.

Arbitrage betting existed from the very beginning of the bookmaking era. Fortunately, the technological progress has considerably facilitated the applying of this particular technique, making it more efficient. Online betting is so simple that people around the world figured out different ways to find and efficiently take advantage of bookmakers' loopholes. However, betting companies are not happy with this recent trend and do whatever is possible to block this phenomenon.

Arbing is the process whereby a gambler takes advantage of the variation in odds offered by different bookmakers, in order to make a profit regardless of the outcome of an event.

> Also known as matched betting, miracle bets, or sure wins, arbing involves simultaneously betting on every outcome of an event, whilst making a calculation that the combined bets will lead to a guaranteed profit.
> The key to arbing is to find two different bookmakers offering significantly different odds on an event. This could be an event with two possible outcomes (win or lose), such as tennis or basketball, or an event with three possible outcomes (win, lose, or draw), such as football.
> Arbing has been dubbed "the only way to make a constant profit" in sports betting, and it has certainly proved profitable for a lot of followers of the method. That said, it requires large stakes and a great deal of patience, given that the typical return on investment is between two and five per cent, depending on the event and a number of other factors.
> (Betfair, 2016)

The above presented explanation, authored by Betfair, provides a clear picture what sure betting is. The arbing process, according to the same source, is not complicated, because the formula is clear and simple (Betfair, 2016). The more experienced bettors can even calculate in their mind whether a set of odds from two different bookmakers forms a sure bet.

When looking for an arbitrage, one needs to search for different odds for the same event in different betting companies. To note, it is impossible to find a sure bet within one betting company. Naturally, all bookmakers ensure their profit, as it was mentioned in previous

chapter, by lowering the true odd by a percent which is usually around 7-12%, and it represents the bookmaker's expected profit.

Thus, it is possible to find a sure bet by monitoring different expectations of different betting companies on the same event. The key for a successful sure bet strategy is diversity. When two or more betting companies diverge in their opinion on the outcome of the same event. When this diversion is big enough, an arbitrage is possible.

Below is presented a practical example. In the example, two bookmakers set different odds for the same event: The Presidential elections in the USA.

Table 1. An example of odds

| | Bookmaker 1 | Bookmaker 2 |
|---|---|---|
| Hillary Clinton | 1,18 | 1,4 |
| Donald Trump | 5 | 2,9 |

The criteria for the existence of an arbitrage bet is calculated using a simple formula. If $c1^{-1} + c2^{-1} < 1$, then this becomes a sure bet. In this case c1 and c2 are two complementary odds for the same event, which means p(c1) + p(c2) =1 (sum of the probabilities of these odds is 1).

As mentioned earlier, for the same betting bookmaker the result of the formula will always be over 1. The higher the result, the bigger the expected profit for the bookmaker, and the worse the deal it is for the bettor.

$1{,}18^{-1} + 5^{-1} = 1{,}047$, so the bookmaker's expected profit will be:

1-(1,18*5)/(1,18+5)=0,046, so 4,6%

By picking the highest possible odds from these two bookmakers, it is possible to find a sure bet:

$1{,}4^{-1} + 5^{-1}=0{,}91<1$, so here is an example of a sure bet.

In order to ensure the profit and to maximize the win, it is enough to bet X amount of money on first outcome and X*c1/c2 on the second outcome. In this case, the bet is 100 euros on Hillary Clinton and 100*1,4/5 (28) euros on Donald Trump. It is easy to see that in case Hillary Clinton wins the elections, the win will be 140 – 28 = 112 euros, and in case Donald Trump wins, it will be 140-100=40 euros. This represents an example of an arbitrage with a rather high profit.

### 2.5 Scraping method

Scraping represents the primary method for the “Odds Aggregator” application to aggregate the content from the bookmakers' web pages. Web scraping is an automated process of gathering the information from some other websites.

> Some call it theft, others call it legitimately gathering business intelligence - and everyone is doing it. Screen scraping might sound like something you do to the car windows on a frosty morning, but on the internet it means copying all the data on a target website.
> Every corporation does it, and if they tell you they're not they're lying," says Francis Irving, head of Scraper Wiki, which makes tools that help many different organizations grab and organize data.
> To copy a document on a computer, you highlight the text using a mouse or keyboard command such as Control A, Control C. Copying a website is a bit trickier because of the way the information is formatted and stored.
> Typically, copying that information is a computationally intensive task that means visiting a website repeatedly to get every last character and digit.
> If the information on that site changes rapidly, then scrapers will need to visit more often to ensure nothing is missed.
> (Mark Ward, BBC News)

The above quote from BBC explains what is scraping and what it is used for. The legality of this process is, however, arguable. Scraping is among the most efficient ways to collect the content from the web for the case of arbitrage search. However, it is also challenging from the legal point of view. Search engines perform scraping when they index internet pages. Moreover, they parse the data gathered from these pages and transform it into a different format. In order to not cross legal boundaries and avoid the risks, it is suggested to follow the below requirements:

- not to scrap personal data or other sensitive information;
- the content should not be protected by copyright;

- in case of a big company, it is better to consult lawyer before scraping;

- follow the Terms of Use of the scrapped website.

The most reliable way to get odds is by scraping them directly from the betting sites' pages. The other possible methods of getting the odds from the bookmakers, such as XML feeds, provided by the betting companies or the third party odds providers, are less trustworthy. due to the fact that they are not always up to date.

### 2.5.1 The advantages and challenges of scraping method

Scraping, just as any other method of data aggregation from the web pages, has its advantages and its drawbacks. However, it is definitely the most suitable method for data fetching from the bookmakers. The next quote reflects the advantages of this method and the challenges that are experienced when applying it.

> Because the system takes the odds directly off the website and reports them asap, the odds are reliably what a user will find when they visit the website themselves. The creator of the html scraping program can also control what pages, and therefore, which sports and odds are scraped – so there are no restrictions to what odds can be collected. It is also free – ignoring the time/effort requirement to develop an adequate scraping program, and then maintain it with respect to the bookmaker website i.e.: If the bookmaker changes their website layout, the scraper needs to be updated to match the new layout.
> For all of the benefits of HTML scraping though, it has the significant problem of requiring constant HTML access to the bookmaker website for re-scraping. In order to keep your scraped odds up to date, you need to scrape the whole website regularly – for arbitrage purposes, ideally you would scrape it every minute – but no website owner wants a bot hitting their website and loading every page on it every minute. That wastes their bandwidth, slows down the website for all of their users and costs them money. So any ip found repeatedly hitting a page at that frequency will usually be ip-blocked by the bookmaker. So you need to be much more clever with how you scrape the odds from bookmakers.
> (Aegist, 2011)
>
> What are some of the challenges of acquiring data through scraping?
> - Scale — It's obvious that terabytes of data will cause problems, but so (on most file systems) will having tens of millions of files in the same directory tree.
> - Metadata — It's a chicken-and-egg problem. Since few programs can draw on rich metadata, it's not much use annotating it. But since so few datasets are annotated, it's not worth writing support into your applications. We have an internal data-description language that we plan to open source as it matures.
> - Historical complications — Statisticians like SPSS files. Semantic web advocates like RDF/XML. Wall Street quants like Mathematical exports. There is no One True Format. Lifting each out of its source domain is time consuming.
>
> But the biggest non-obvious problem we see is source domain complexity. This is what we call the "uber" problem. A developer wants the answer to a reasonable

> question, such as “What was the air temperature in Austin at noon on August 6, 1998?” The obvious answer — “damn hot” — isn’t acceptable. Neither is:
> Well, it’s complicated. See, there are multiple weather stations, all reporting temperatures — each with its own error estimate — at different times. So you simply have to take the spatial- and time-average of their reported values across the region. And by the way, did you mean Austin’s city boundary, or its metropolitan area, or its downtown region?
> There are more than a dozen incompatible yet fundamentally correct ways to measure time: Earth-centered? Leap seconds? Calendrical? Does the length of a day change as the earth’s rotational speed does?
> Data at “everything” scale is sourced by domain experts, who necessarily live at the “it’s complicated” level. To make it useful to the rest of the world requires domain knowledge, and often a transformation that is simply nonsensical within the source domain.
> (Watters, 2011)

## 2.6 Methods and tools

Scrum method is an agile software development framework for developing complicated projects. Scrum is a very efficient methodology, because it insists on delivering complete increments of business value helping us to learn rapidly and completely. Scrum forces us to rest and integrate our experiments and encourages us to release them to production, so that we have a complete learning cycle. Scrum focuses on delivering the highest priority business value, as defined by the Product Owner. Another reason Scrum works is that it unleashes the brainpower of many minds on a problem (Schwaber, 2004)

According to its official website, Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design (Django, 2015). Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source and it is a ridiculously fast, pretty secure and very scalable. (Django, 2015)

Another helpful tool for scraping is the Python library called Beautiful Soup.

> Three features make it powerful:
> 1.Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
> 2.Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
> 3.Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

> Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class `externalLink`", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."
> Valuable data that was once locked up in poorly-designed websites is now within your reach. Projects that would have taken hours take only minutes with Beautiful Soup." (Richardson, 2016)

Additionally, other useful tools for scraping are Selenium and PhantomJS.

Selenium is a software testing framework for web applications. It can be also used for parsing and getting the access to the HTML code.

PhantomJS is a scripted web browser, which does not have a graphical user interface, used for automating web page interaction. It is an open source tool, providing a JavaScript API and can be used for various manipulations with the web pages (e.g. opening web pages, execute user actions and so on).

As a text editor for code, Sublime Text was used. According to the official site of the editor (Sublime Text, 2016), there are plenty of things users love about it. Here are some of them:

- Goto Anything command allows to open files and quickly search anything within a file;
- multiple selections. It is possible to make changes very efficiently, e.g. changing multiple lines at once;
- it has a mighty Python-based Application Programming Interface;
- high level of customization;
- it is possible to split the screen in order to edit multiple files;
- Command Palette makes it easy to navigate and increases the usability;
- there is a possibility to use Distraction free mode when working with Sublime Text. It allows to concentrate exclusively on works (Sublime Text, 2016).

# 3 Research and development plan

This thesis carries a practical significance. Its major part represents the web application implementation. This app will be fetching the data from a set of bookmaker websites, then will process and analyze it. Meanwhile, the theoretical part of the thesis will be including the descriptive and research components.

Preliminary to the practical phase of this project, a detailed research, related to suitable software and tools, will be performed. As mentioned earlier, scraping will be selected as the primary method of data extraction. Therefore, a rather detailed research will be performed on the range of existing tools for effective scraping. Another phase will be focused on the research of the betting sites websites. The results of this research will affect the scope of the thesis.

The descriptive part of the thesis will reflect the process and the results of the practical development.

As stated before, the practical component of the project will represent the foundation for the thesis. It consists of three particular stages: design, implementation and testing.

The design phase would be definitely the most important step in the whole development of the product. A mistake committed in this phase can lead to severe consequences later in the implementation process. That's why this part will be handled with particular responsibility. To this end, an ER diagram will be constructed, based on which would be eventually developed database tables and Django models.

Implementation stage will represent another important phase of this project. In fact, it would possibly be the most time-consuming part. The deliverable of the previous phase will be taken into consideration when proceeding with this stage. Shortly, this process is about working on the view and controller from the MVC architecture. The implementation will be started from the server side, after which will be followed by the front-end development, i.e. the design of the graphical user interface (GUI). Because this project implies interaction with the third-party services, a significant and demanding part represents the integration of the aggregator app with the bookmaker websites. Additionally, the deployment of the final app to a cloud-based server is planned as a requirement for the project.

Testing is the final phase of practical component. The main goal is to ensure the functionality and liability of the application deployed in the cloud.

The following research questions will be defined for this thesis:

- Is it possible to extract and process the data from the bookmaker pages? The question arises due to the commonly used nowadays policies, which protect the websites from the scrapers and crawlers.

- How to fetch the data for ongoing events (live odds) and how up-to-date it is?

- Is it possible to find the fixed odds which correspond to an arbitrage? Is the same possible for the live odds?

- How critical is the update-time factor for processing the fixed and live odds?

The whole development process will be run using the agile methodology Scrum. In this case, the Scrum will be followed in untraditional manner, where one person combines the roles of Product Owner, Development Team and Scrum Master. This method was chosen to increase the efficiency and to complete the project within the deadline limits. This method is especially efficient in projects of this type (Schwaber, 2004). The application is planned to be completed in three sprints of six days each. In addition, a Sprint review will be held at the end of each sprint.

# 4 Empirical part - The Actual Research/Development

During the research it was decided to take an agile approach for the project. This allows to organize the work schedule and achieve the targeted results in the most rational and efficient way and in fixed terms. The most preferable methodology in this particular situation is considered to be Scrum (Schwaber, 2004). However, this methodology must be adjusted to suit the existing conditions, requirements and deadlines of the project. To note, Scrum is generally a team-oriented technique, yet in this case only one person is performing the whole set of development tasks. Therefore, the necessary changes were introduced to the traditional agile planning.

According to Scrum rules, the Product Backlog has to be worked out before the sprints and actual beginning of actual development. Since the author of the thesis plays the role of the Product Owner, Development Team and Scrum Master, it was set to include all these preliminary planning activities to the first day of first sprint (J.Highsmith & A.Cockburn, 2002).

Considering the tight deadlines, it was decided to implement the technical part of the project in three sprints. In order to succeed in obtaining the final results, each sprint consists of 6 working days. Author is fully concentrated on the realization of tasks from Product Backlog. In case of an off-schedule situation, one week was reserved for an extra fourth sprint.

## 4.1 Sprint 1 – Preliminary stage

Day 1. The skeleton of every IT project, implemented using an agile method, is the Product Backlog. It comprises the business requirements of the final program, which are presented below:

- the final application should extract betting data from at least three different bookmaker websites;
- the fetched data must be processed and transformed to a suitable and uniform format (decimal), and then summarized in a table;
- the betting information should be up-to-date and synchronized with the bookmaker websites;

- all the collected data must be presented to the user in an understandable and structured way.

- application must contain a mechanism that highlights the most favorable odds and must inform the user about the possible arbitrage cases.

The above presented points represent the Sprint Backlog for the week 1. For the same week it was scheduled to complete the planning and design phases, and to implement the model component of the final MVC architecture. Additionally, the list of tasks for Sprint 1 was prepared, including the approximate time for each task completion. Also has been started the research of betting companies related to their correspondence with Product Backlog.

Day 2. Second day started with a research about the betting websites. The focus of the work was set on the following tasks:

- the websites must have an accessible HTML source code, in order to be able to scrape them. An example of incompatibly for scraping is Flash application (e.g., Bet365.com);

- the web page should not be developed using any front-end framework, especially ReactJS, because it's not scrapeable (e.g., betsbc.com);

- the web page should contain a clear DOM structure, so that the scraping process would be logical;

- scrapers are not welcome on many bookmaker websites, probably to avoid extra traffic and activities against the terms of service. To protect their data, betting websites sometimes hide their betting numbers inside JavaScript.

During this day, about 30 betting websites were researched. Only six portals out of this number complied with all of the above mentioned requirements. These were bookmarked into a special folder.

The most recent research showed one minor problem that was not noted neither in initial plan, nor in Product Backlog. It concerns uniformity: the same teams are differently named in various bookmaker sites (West Bromwich Albion, West Brom, West Bromwich Albion FC, West Bromwich). Finding the solution for this issue was planned for the next sprint.

The last task of the day was sketching the first draft of the database prototype. Database consists of at least two tables: Event and Odds in relation one to many.

Day 3. Few hours were allocated for test scraping. For this purpose, a test virtual environment for Django project was initiated. Based on the preliminary list of suitable websites, the process of scraping was performed. As a direct tool for extracting data was used Python library, BeautifulSoup. This brought to light particular difficulties, related to pulling the needed information out of HTML code. After investigation of thematic sources, it became clear that Selenium would be a better option for scraping. The other relevant tool in this case would be PhantomJS. The use of these tools led to a more successful extraction of data from three out of six betting websites, the other three turned out to be inaccessible for scraping for various reasons. This activity was performed on the stage of planning and design only as a proof of website selection accuracy.

The remaining part of this day was entirely dedicated to search of suitable and scrapeable web pages. As a result of reviewing almost 20 addresses, one more suitable betting agency was found.

Day 4. Having the practical results of the previously mentioned research, the actual design of the application models started on this day. Since the program is implemented using the powerful back-end framework, there was no need in real database development. Skeleton of the application are models. To the initial model's sketch, designed on Day 2, were added all the missing fields. Also, in addition to the existing models Odds and Event, the new Bookmaker model was planned.

On Day 4 the actual start of product real implementation took start. Using virtual environment and pip package manager, there was locally installed Python 2.7.9 and Django 1.10.3. Although by default Django 1.10 uses the third version of Python 3.4, it was considered more reasonable to use Python of version 2. Later, there were installed Selenium (under virtual environment) and PhantomJS (globally). Some problems occurred while installing PhantomJS under virtual environment, that's why it was installed globally. Inside this Django instance the application named main was created using startapp command.

After the creation of the whole folder structure, the models sketched on the paper were transferred to models.py. The first lines of the code were added to views.py, urls.py and settings.py. At this stage was also created the first and brief template index.html. There

were executed commands, needed for models and database initiation. Running the application on the local server (127.0.0.1:8000) turned out to be successful.

Day 5. This day was completely dedicated to models structure finalization, testing and making sure that the internal skeleton of the application is fully functional and solid. In the beginning, testing of the models operationability was performed in terminal. According to Django Tutorial N2 from official Django documentation the database API was tried out in Python shell. For this purpose, was used both embedded environment (invoked by command" python manage.py shell") and ipython (much more versatile shell version, installed previously in application's virtual environment using pip). Among the other operations the following important operations, meant to check model functionality and sustainability, were performed:

- all three objects imported from the models;
- a few new instances of each object were created and populated with some relevant data, then deleted;
- as an experiment some of the instances were inserted into database using save() method;
- while trying to directly access Odds object fields some omissions in model structure were revealed;
- database storage ability was tested using objects.all(), objects.get(), objects.filter();
- foreign key relation among the tables/objects was tried out using choice_set.all(), choice_set.create() and choice_ set.filter() methods.

In order to extend the testability of the database/models, superuser was created. So-called superuser is a user having administrator rights. Then to the admin.py file under main application folder were added strings of code to ensure the admin interface for Event, Odds and Bookmaker objects. Experiments with creating, editing and deleting objects and fields followed.

Day 6. This day started with adding minimal using interface for displaying objects according to existing models. Experiments with displaying objects in real application (index.html) continued. So far objects were created manually and populated with some test data using

admin interface. Minor adjustments were applied to the models. Thus, all the issues, identified during the testing in console, were eliminated.

The day was concluded with the Sprint review. Overall, almost everything from the Sprint Backlog was completed, except few smaller tasks. These were rescheduled for the next sprint.

### 4.2 Sprint 2 – Practical implementation

Day 1. Second Sprint Backlog in general terms could be reduced to just one point: fully functional back-end logic. For the second sprint it is planned to extend front-end side for comprehensive displaying of extracted and processed data. In this time span it is necessary to complete all code related to parsing.

It was decided to start the actual parsing activities with seemingly the least troublesome website betwin.com. Using method find_element_by_class_name(), it was possible to extract from the DOM all the elements that contain teams and odds in two different lists. On the next step, two empty arrays are declared and then populated with exclusively text data from the above mentioned lists using text() property. Thus, each element of these two newly created lists are of String type. So far, even this numeric data from Odds list is in String format.

On the scraped page there are three different types of results: win of the first team (1), draw (X) and win of the second team (2). Important feature of this site DOM is that along with the team names the X character, that indicates draw, is parsed as well in the same array. In order to avoid these useless elements, it can be ignored using the loop counter. As the result, both odds and teams arrays contain the same number of elements: number of available events multiplied by three.

So, running the new loop up to array's length divided by 3, it is possible to arrange the extracted data. Each [i*3] and [i*3+2] elements are the names of teams, participating in the same event. On every iteration the new instance of Event type is created using these strings as arguments. In the same iteration a new Odds object is instantiated, populated and then appended to the list of odds of the current (on this step of the loop) event. In the final line of code in this loop the resulting event instance is appended to the common array events.

Overall, the parsing process went smoothly for above mentioned website, the only difficulty was related to accessing the date and time of the event and logically connect it to the event itself. This task was postponed.

Day 2. Another website, which fully corresponds the requirements for non-problematic scraping, is betway.com. It's also quite easily scrapeable, because it contains only three possible results: 1, X and 2. Its HTML code is very well logically structured and, as in bwin case, all the needed data can be fetched by filtering the DOM by class name. The algorithm of arranging the information and populating the resulting events is almost the same as in the previous case. The only difference consists in the number of elements of teams lists. If on the first day of this sprint the final number of elements in teams array was narrowed by one third, in this particular case it was multiplied by two. When parsing the text component from betway page, it is possible to get only the string of following type" team1 - team2".

To divide the string in two logical parts, the split ("-") command was used. Because this split method sometimes causes value error (" Need more than 1 value to unpack"), it was necessary to add a preliminary if condition to check whether the number of split parts is equal to two.

On this stage, when the data is fetched from a few sources, it becomes necessary to fit it together. Consequently, the next step consists in comparison of event instances. Matching of events is possible only by comparing both home and guest teams. However, as it was noticed earlier, same teams are differently named. Additionally, in some cases the resulting string can contain extra spaces after the parsing or splitting. The completion of this task was postponed to the next day.

Day 3. First of all, a function responsible for matching team names was written. Using logical operand OR and IN operator, this function identifies the team using only key part of its name. As a result, this function returns the actual name of the team.

Secondly, inside the event model was embedded Python function __eq__(), which returns Boolean value and takes as an argument another Event object. It is responsible for matching two strings (guest and home teams) in order to find out whether the events are the same. Based on this work it became possible to put together the data from several sources into one object. To this end, a separate function add_odds was created.

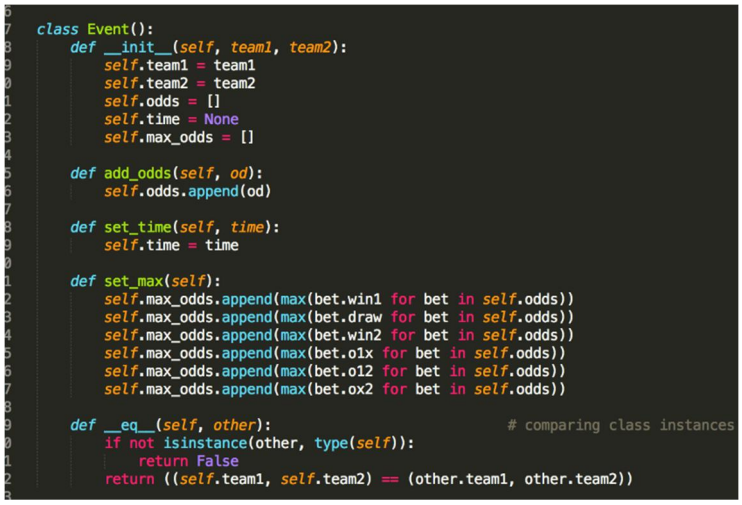
On this day was identified a fundamental issue, related to the business logic. Since the application works with the new dynamic data: on every start or refresh it gets new info and

processes it without storage. According to our requirements (Product Backlog), this application is not supposed to save the old data. That's why there is no real need in neither permanent nor temporary storing of this information in database. Therefore, after some reflections, it was decided to prefer common classes over models and database. Object-oriented approach to data operations was kept, classes were placed directly into views.py.

Day 4. The third bookmaker page in the list of scrapable websites is marathonbet.com. The advantage of this page is that it allows to extract three more result types (1X, X2,12). On Odds object declaring only win1, draw and win2 are set in constructor. To set the values of these three extra fields, a new function set_other_odds() was added.

Another challenge concerning this website, is that by default the odds are displayed in fractional format, not decimal (13/25, which is 1,52 in decimal), while Selenium parser uses default version. To solve this issue another function was added to Odds class – transform_to_decimal(). It uses already mentioned split() method and it converts to decimal using common formula. The same function is also applied for converting of the initial odds directly in the constructor.

In addition, this webpage is lucrative because it's easier to scrape. The corresponding function set_time() was added to class Event. The below screenshots represent the structure of the classes Event and Odds.



```
class Event():
    def __init__(self, team1, team2):
        self.team1 = team1
        self.team2 = team2
        self.odds = []
        self.time = None
        self.max_odds = []

    def add_odds(self, od):
        self.odds.append(od)

    def set_time(self, time):
        self.time = time

    def set_max(self):
        self.max_odds.append(max(bet.win1 for bet in self.odds))
        self.max_odds.append(max(bet.draw for bet in self.odds))
        self.max_odds.append(max(bet.win2 for bet in self.odds))
        self.max_odds.append(max(bet.o1x for bet in self.odds))
        self.max_odds.append(max(bet.o12 for bet in self.odds))
        self.max_odds.append(max(bet.ox2 for bet in self.odds))

    def __eq__(self, other):                    # comparing class instances
        if not isinstance(other, type(self)):
            return False
        return ((self.team1, self.team2) == (other.team1, other.team2))
```

Figure 1. Event class (screen capture)

```
class Odds():
    def __init__(self, win1, win2, draw, bookmaker):
        if "Marathonbet" in bookmaker:
            self.win1 = self.transform_to_decimal(win1)
            self.draw = self.transform_to_decimal(draw)
            self.win2 = self.transform_to_decimal(win2)
        else:
            self.win1 = Decimal(win1).quantize(TWO_PLACES)
            self.win2 = Decimal(win2).quantize(TWO_PLACES)
            self.draw = Decimal(draw).quantize(TWO_PLACES)
        self.o1x = None
        self.o12 = None
        self.ox2 = None
        self.bookmaker = bookmaker

    def set_other_odds(self, o1x, o12, ox2):
        if "Marathonbet" in self.bookmaker:
            self.o1x = self.transform_to_decimal(o1x)
            self.o12 = self.transform_to_decimal(o12)
            self.ox2 = self.transform_to_decimal(ox2)
        else:
            self.o1x = Decimal(o1x).quantize(TWO_PLACES)
            self.o12 = Decimal(o12).quantize(TWO_PLACES)
            self.ox2 = Decimal(ox2).quantize(TWO_PLACES)

    def transform_to_decimal(self, var):
        data = var.split("/")
        if len(data) == 2:
            one, two = var.split("/", 2)
            return (Decimal(one)/Decimal(two) + 1).quantize(TWO_PLACES)
```

Figure 2. Odds class (screen capture)

Day 5. In order to extend the functionality of the program, the template check_html was created, views.py and urls.py were modified correspondingly. At this stage, the index page is meant to display the list of available events. Each event is clickable and redirects to the check page. On the check page the pivot table is presented, containing all the existing odds for the particular event. Horizontally the bookmaker names are displayed and vertically the betting options are located. To layout correspondingly this content, was utilized Table markup.

The issue at this stage is related to difficulties in passing an object from one view to another through index.html template. Since in this project the information is not stored in database, there is no way to detect in the check view what particular event was clicked in index template. Normally, when database is available, object id is passed to view through GET request.

It was decided to replace database with sessions. Session management was introduced to handle variable passing between the views (pages). Therefore, inside the url tag in index template there are introduced two arguments (team1 and team2) to identify the event. The spaces and other special characters were url encoded, e.g. %20 in case of a space character.

Day 6. The day started with unsuccessful trials to fetch the data from the fourth betting agency parimatch.com. While testing the scraping in the shell environment, the data was fetched successfully. On the other hand, when working with the same line of code in real application, extraction of the data was blocked for an unknown reason. As the result, to meet the initial requirements, it was decided to continue searching of the fourth suitable bookmaker page with the condition of scrapable extra odds (1X, X2, 12). Because all major betting pages were already reviewed on the earlier stages, the searching process continued on the smaller local pages. Unfortunately, by the end of the day none of the suitable webpages were found.

Sprint review indicated that except of completing the scraping part (at least 4 pages), all the other tasks were successfully fulfilled. The search of the fourth webpage was transferred to Sprint 3.

### 4.3 Sprint 3 – GUI and Testing

Day 1. The first day of third agile segment started from writing Sprint Backlog. Generally, it contains all undone tasks from Product Backlog. These are: finding the fourth betting agency, providing extra odds suitable for scraping; creating algorithms for detecting the highest odds and possible arbitrage cases; developing graphic user interface for the application and performing necessary testing tasks.

The search of fourth bookmaker webpage took almost half a day and as a result a smaller betting company from Kazakhstan Olimp was found. It contains all the required odds, it is easy to parse and a clear class structure. The only issue is related with the fact that the content of this page's default version is in Cyrillic encoding. This assumes the necessity to update the function which is responsible for team names uniformity. Everything related to this agency except for this task, was completed during the Day 1.

Day 2. Adding Cyrillic letters to the above mentioned function caused encoding error. Although the utf8 encoding is set in Python by default, placing non-ASCII characters directly in views.py for some reason leads to application failure. After consulting the internet sources, it was found out that this is a typical Django error and it can be fixed by adding to the mentioned file the following peace of code (import sys; reload (sys); sys.setdefaultencoding('utf8')).

Next step was the completion and finalization of the mechanism for defining the most advantageous odds and arbitrage situations. Set_max() function inside the Event class is

responsible for the first part of this task while arbitrage algorithm was placed inside the function of the same name.

Day 3-4. At this point, all the server side of the application was ready and the following two days of this sprint were completely dedicated to front-end tasks, visual design and usability. On the planning phase there was idea about using bootstrap, the most popular html framework for developing responsive design. A suitable scheme among the available bootstrap templates was not found and it was decided to use the pure HTML5 and CSS3. The design is based on center alignment of the main content.

Day 5. This day was spent completely on real time testing. It involves both functionality and usability testing. The application was tested in different browsers and on various devices. The application was also tried out by three different persons apart from the author. Only one of these three people have never used betting websites. The received feedback was positive: the application functionality and operationability is comprehensive and understandable, its design is user-friendly. The only issue, mentioned by two out of three respondents, was related to long response time after reloading the page.

Functionality testing revealed one problem that occurs occasionally - Stale Element Reference Exception. This exception is thrown by Selenium in case if one or more elements are no longer attached to the DOM. This issue was not completely avoided, although the frequency of the its appearance was reduced using WebDriverWait mechanism.

Day 6. On the last day of the Scrum process, minor changes were inserted to the design and comments added to the code. A bit of efforts was applied to solve this issue. By the end of the day, the product features were matched with the tasks from the Sprint and Product Backlog and it was acknowledged that the application met its requirements completely.

# 5 Evaluation and Conclusions

## 5.1 Final result versus expectations

The result of the three-week agile development and the preparatory theoretical research is a working application, which efficiently and accurately extracts data from four different external sources. After the extraction of data, the application arranges logically this data, analyzes it and displays the final output in a comprehensive and accessible manner. Application has a clearly defined scope both in the data selection phase, as well as in the processing criteria. The final product offers a user-centered solution by providing a smooth and easy interaction with the service. The back-end, in turn, handles effectively the data gathering, its processing and analyzing.

The crosschecking of the final result against Product Backlog is presented in the following table.

Table 2. The comparison of the goals and final results

| Product Backlog | Final product |
|---|---|
| To extract betting data from at least three bookmaker websites. | Fetches data from four betting agencies. The agencies from which the data is extracted are of different sizes as well as from different countries. |
| To fetch data for processing. The data is then to be transformed into a suitable and uniform format (decimal), after which to be summarized into a table. | The data is successfully arranged into chronological list of events with the possibility to display additional information by clicking on a particular event. The final output is in decimal format and in Latin alphabet (utf8 encoding). Additional information is represented in a 6X4 pivot table. |
| The betting information must be up-to-date and synchronized with the bookmaker websites. | All of the data is fully synchronized and up-to-date with the data from the sources. The application is functional even for the live bets, although the waiting time when reloading the app is around 20-30 seconds. |
| The collected data to be accessible for the user in a clear and concise manner. | The GUI is user-friendly and comprehensive. The Bootstrap framework was not used in the implementation, even though this was planned previously. |

| | |
|---|---|
| To include a mechanism that highlights the most favorable odds and inform about any possible arbitrage cases. | The server side of the program comprises fully functional algorithms that execute the corresponding sorting mechanism and detect arbitrage situations. |

To note, the Odds Aggregator does not involve the use of databases. The models still exist, but the data exchange between the pages is performed through sessions.

### 5.2 Validity of results

The validity of the practical results can be checked manually by a bare eye, and by performing simple arithmetical calculations. The tester can compare the source bookmaker websites with the Odds Aggregator to ensure the validity of the presented information. As for the theoretical results, their validity can be confirmed by verifying the algorithms utilized in the application.

The achieved results for this project meet the originally set scope and even exceed for particular goals the initial expectations. The final scope of the implemented application consists of one sport discipline, one league, six types of outcome and four external sources.

### 5.3 Relevance of sources

All of the information sources used in the theoretical part were also used in the actual development process. Particularly useful sources proved to be those concerning the scraping methods. For the practical part, a set of forums, such as stackoverflow.com, were consulted, but not mentioned in the reference list.

### 5.4 Tools and methods assessment

The stage involving the selection of tools for developing the application and data gathering was a decisive moment for the rest of the application development. Initially, the following frameworks and scraping tools were considered: Django/ BeautifulSoup, PHP / SimpleHtmlDom, Node.js / parse package. The final choice was made after a thorough research. However, later the BeautifulSoup package was replaced with the Selenium framework. The achieved results confirm that this combination was suitable and correct. During the back-end development of the product as well as the data extraction no particular

serious challenges were faced, other than the Stale Element Reference Exception that was encountered at one stage of the development.

The project was implemented using the agile method Scrum. This allowed to efficiently and consistently develop the project within the initially determined deadlines. Scrum methodology provided the ideal project management solutions. The workload was structured into clearly specified subtasks and even limit surpassing was later fixed by Scrum's clever agile rules.

### 5.5 Knowledge and experience acquired

During the theoretical and practical work upon this thesis, the following learning skills were acquired:

- practical project management from the agile development perspective;
- writing a full-scale report, based on the web development project;
- working with the external services;
- familiarizing with the specifics of the scraping process and converting diverse sets of data to a uniform format;
- object-oriented programming in Python environment and working with sessions;
- learning the web development using the Django framework;
- arranging the table layouts with the css.

### 5.6 Significance

The Odds Aggregator has a clear practical significance. It aggregates the data from four betting websites, and can serve both the sports bettors and sports analysts. There exist just a few applications with the similar functionality but with a wider scope. Besides, a number of these services require membership payment. The Odds Aggregator, on the other hand, is completely free of charge and, additionally, open-sourced.

### 5.7 Further development

A further development is planned for the application: the extension of its features and the widening of the scope. Further plans comprise the deployment of this product to the Heroku cloud server. The planned additions to the existing application are: expansion in the number of the scrapeable betting agencies; increase in the types of the sport disciplines and leagues; increase in the types of the outcome; the extension of the time limits. Thanks to these additions, the Odds Aggregator would become a competitive product among the other existing applications in the respective segment.

# 6 References


Aegist. 21 April 2011. Visitor Question about Data Feeds. URL: http://sportsarbitrageguide.com/blog/author/admin/. Accessed: 6 October 2016.

Betfair. 2016. What is Arbing? URL: https://betting.betfair.com/what-is-arbing-arb-betting-explained.html. Accessed: 8 October 2016.

Djangoproject. 2015. Meet Django. Accessed: URL: https://www.djangoproject.com/. 11 October 2016.

Highsmith, J. & Cockburn, A. 2002. Agile software development: the business of innovation. IEEE. Accessed: 23 November 2016.

Maasz, J. Bet and lose: learning mathematics or losing money. URL: http://www.alm-online.net/images/ALM/proceedings /alm16/Articles /16maasz.pdf. Accessed: 9 October 2016.

Richardson, L. 3 August 2016. Beautiful Soup. URL: https://www.crummy.com/software /BeautifulSoup/. Accessed: 9 October 2016.

Schwaber, K. 2004. Agile Project Management with Scrum (Second p.). Microsoft Press. Accessed: 24 November 2016.

Sublime Text. 2016. URL: https://www.sublimetext.com/. Retrieved: 21 October 2016.

Thorpe, E. 1984. Mathematics of Gambling (First ed.). Gambling Times. Retrieved: 3 October 2016.

Ward, M. BBC News. 2013. Screen scraping: How to profit from your rival's data. London. URL: http://www.bbc.com/news/technology-23988890. Accessed: 8 October 2016.

Watters, A. 11. June 2011. Scrpaing, cleaning, and selling big data. URL: http://radar.oreilly.com/2011/05/data-scraping-infochimps.html. Accessed: 16 October 2016.

# Appendices

### Appendix 1. Source code in views.py (back-end logic)

```
import sys

reload(sys)

sys.setdefaultencoding('utf8')

from django.shortcuts import render

#from main import models

from selenium import webdriver

from contextlib import contextmanager

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support.expected_conditions import staleness_of

from selenium.common.exceptions import StaleElementReferenceException

import urllib

from decimal import Decimal

TWO_PLACES = Decimal("0.01")

@contextmanager

def wait_for_page_load(self, timeout=30):

    old_page = self.driver.find_element_by_tag_name('html')

    yield

    WebDriverWait(self.driver, timeout).until(staleness_of(old_page))

def find(driver):

    element = driver.find_elements_by_class_name("mb-option-button__option-odds")

    if element:

        return element

    else:

        return False
```

```
class Event():
    def __init__(self, team1, team2):
        self.team1 = team1
        self.team2 = team2
        self.odds = []
        self.time = None
        self.arbitrage = "No arbitrage"
        self.max_odds = []
  def add_odds(self, od):
        self.odds.append(od)
    def set_time(self, time):
        self.time = time
    def set_max(self):
        self.max_odds.append(max(bet.win1 for bet in self.odds))
        self.max_odds.append(max(bet.draw for bet in self.odds))
        self.max_odds.append(max(bet.win2 for bet in self.odds))
        self.max_odds.append(max(bet.o1x for bet in self.odds))
        self.max_odds.append(max(bet.o12 for bet in self.odds))
        self.max_odds.append(max(bet.ox2 for bet in self.odds))
    def find_arbitrage(self):
        for bet in self.odds:
            for other_bet in self.odds:
                if other_bet.o12 == None:
                    continue
                if (1/bet.draw + 1/other_bet.o12 < 1) or (1/bet.win1 + 1/other_bet.ox2 < 1) or (1/bet.win2 +
1/other_bet.o1x < 1):
                    self.arbitrage = "Arbitrage found!"
        #return False
```

```
    def __eq__(self, other):                    # comparing class instances
        if not isinstance(other, type(self)):
            return False
        return ((self.team1, self.team2) == (other.team1, other.team2))
class Odds():
    def __init__(self, win1, win2, draw, bookmaker):
        if "Marathonbet" in bookmaker:
            self.win1 = self.transform_to_decimal(win1)
            self.draw = self.transform_to_decimal(draw)
            self.win2 = self.transform_to_decimal(win2)
        else:
            self.win1 = Decimal(win1).quantize(TWO_PLACES)
            self.win2 = Decimal(win2).quantize(TWO_PLACES)
            self.draw = Decimal(draw).quantize(TWO_PLACES)
        self.o1x = None
        self.o12 = None
        self.ox2 = None
        self.bookmaker = bookmaker
    def set_other_odds(self, o1x, o12, ox2):
        if "Marathonbet" in self.bookmaker:
            self.o1x = self.transform_to_decimal(o1x)
            self.o12 = self.transform_to_decimal(o12)
            self.ox2 = self.transform_to_decimal(ox2)
  else:
            self.o1x = Decimal(o1x).quantize(TWO_PLACES)
            self.o12 = Decimal(o12).quantize(TWO_PLACES)
            self.ox2 = Decimal(ox2).quantize(TWO_PLACES)
```

```
def transform_to_decimal(self, var):
    data = var.split("/")
    if len(data) == 2:
        one, two = var.split("/", 2)
        return (Decimal(one)/Decimal(two) + 1).quantize(TWO_PLACES)
    def t(team):
if "Arsenal" in team or "Арсенал" in team:
    return "Arsenal"
if "Bournemouth" in team or "Борнм" in team:
    return "Bournemouth"
if "Burnley" in team or "рнли" in team:
    return "Burnley"
if "Chelsea" in team or "Челси" in team:
    return "Chelsea"
if "Crystal Palace" in team or "Кристал" in team:
    return "Crystal Palace"
if "Everton" in team or "Эвертон" in team:
    return "Everton"
if "Hull City" in team or "Халл" in team:
    return "Hull City"
if "Leicester" in team or "Лестер" in team:
    return "Leicester City"
if "Liverpool" in team or "Ливерпуль" in team:
    return "Liverpool"
if "Manchester City" in team or "Man City" in team or "Манчестер Сити" in team:
    return "Manchester City"
if "Manchester United" in team or "Manchester Utd" in team or "Манчестер Ю" in team:
```

```
        return "Manchester United"
    if "Middlesbrough" in team or "длсбро" in team:
        return "Middlesbrough"
    if "Southampton" in team or "Саутг" in team:
        return "Southampton"
    if "Stoke" in team or "Сток" in team:
        return "Stoke City"
    if "Sunderland" in team or "Сандерл" in team:
        return "Sunderland"
    if "Swansea" in team or "Суонси" in team:
        return "Swansea City"
    if "Tottenham" in team or "Тоттен" in team:
        return "Tottenham"
    if "Watford" in team or "Уотфорд" in team:
        return "Watford"
    if "West Brom" in team or "Вест Бром" in team:
        return "West Bromwich Albion"
    if "West Ham" in team or "Вест Хэм" in team:
        return "West Ham United"
    return team


def index(request):
    driver = webdriver.PhantomJS()
    driver.get("https://sports.betway.com/#/soccer/england/premier-league")          # BetWay
    elem_teams = driver.find_elements_by_class_name("event_name")
    elem_odds = driver.find_elements_by_class_name("outcome_button")
    odds = []
```

```
    teams = []
    events = []
   for team in elem_teams:
        data = team.text.split(" - ")
        if len(data) == 2:
            team1, team2 = team.text.split(" - ", 2)
           teams.append(t(team1))
        teams.append(t(team2))
    for el in elem_odds:
        odds.append(el.text)
for i in range(10):
        event = Event(teams[i*2],teams[i*2+1])
        od = Odds(odds[i*3], odds[i*3+2], odds[i*3+1], "BetWay")
        event.add_odds(od)
        events.append(event)
driver.get("https://sports.bwin.com/en/sports#leagueIds=46&sportId=4")                # BWin
    elem_teams = driver.find_elements_by_class_name("mb-option-button__option-name")
    elem_odds = driver.find_elements_by_class_name("mb-option-button__option-odds")
    odds = []
    teams = []

    for team in elem_teams:
        teams.append(t(team.text))
    for el in elem_odds:
        odds.append(el.text)
for i in range(len(elem_teams)/3):
        event = Event(teams[i*3], teams[i*3+2])
```

```
od = Odds(odds[i*3], odds[i*3+2], odds[i*3+1], "BWin")
        for ev in events:
            if ev == event:
                ev.add_odds(od)
        event.add_odds(od)
driver.get("https://www.marathonbet.com/en/betting/Football/England/Premier+League/?menu=215
20") # Marathonbet
    elem_teams = driver.find_elements_by_class_name('member-name')
    elem_odds = driver.find_elements_by_class_name("price")
    elem_times = driver.find_elements_by_class_name("date")
    odds = []
    teams = []
    times = []
for team in elem_teams:
        teams.append(t(team.text))
    for el in elem_odds:
        odds.append(el.text)
    for time in elem_times:
        times.append(time.text)

    for i in range(len(elem_teams)/2):
        event = Event(teams[i*2], teams[i*2+1])
        od = Odds(odds[i*10], odds[i*10+2], odds[i*10+1], "Marathonbet")
        od.set_other_odds(odds[i*10+3], odds[i*10+4], odds[i*10+5])
        for ev in events:
            if ev == event:
                ev.add_odds(od)
                ev.set_time(times[i])
```

```
event.add_odds(od)
driver.get("http://olimp.com/betting/index.php?page=line&action=2&sel[]=11664") # Olimp
    elem_teams = driver.find_elements_by_class_name("m")
    elem_odds = driver.find_elements_by_class_name("bet_sel")
    odds = []
    teams = []
for team in elem_teams:
        data = team.text.split(" - ")
        if len(data) == 2:
            team1, team2 = team.text.split(" - ", 2)
teams.append(t(team1))
        teams.append(t(team2))
    for el in elem_odds:
        odds.append(el.text)
for i in range(12):
        event = Event(teams[i*2], teams[i*2+1])
        od = Odds(odds[i*10], odds[i*10+2], odds[i*10+1], "Olimp")
        od.set_other_odds(odds[i*10+3], odds[i*10+4], odds[i*10+5])
        for ev in events:
            if ev == event:
                ev.add_odds(od)
                ev.set_max()
                ev.set_time(times[i])
            request.session[ev.team1 + '-' + ev.team2] = ev
event.add_odds(od)
context = {
        'site_title': 'Odds aggregator',
```

```
        'odds': odds,
        'teams': teams,
        'events': events,
        'times': times,
        }
    return render(request, "main/index.html", context,)
def check(request, team1, team2):
    event = request.session[team1 + '-' + team2]#ev.team1 + '-' + ev.team2]
    context = {
        'site_title': 'Odds aggregator',
        'event': event,
        }
    return render(request, "main/check.html", context,)
```