

## СЪДЪРЖАНИЕ

<b>Увод</b> .....	3
<b>Глава 1</b> Цели и задачи.....	4
<b>Глава 2</b> Анализ на темата .....	5
2.1 Анализ.....	5
2.2 Подобни решения .....	5
2.2.1 Skyscanner .....	5
2.2.2 Esky.....	7
2.2.3 Booking.....	8
2.3 Заключение.....	9
<b>Глава 3</b> Обзор на използваните технологии .....	10
3.1 Python.....	10
3.2 Pycharm .....	11
3.3 Selenium .....	12
3.4 PyQt5 .....	12
3.5 Google Chrome .....	13
3.6 Google Flights .....	14
3.7 Github .....	15
3.8 CSS .....	16
3.9 Библиотеки .....	16
3.9.1 Pytest .....	16
3.9.2 Coverage .....	17
3.9.3 Pylint .....	18
<b>Глава 4</b> Схема на работа на приложението .....	19
4.1 Пълна структура на работния процес .....	19
4.2 Схема за извличане на информация за самолетни полети .....	20
4.3 Схема на визуализация на полети.....	21
<b>Глава 5</b> Софтуерна реализация .....	22
5.1 Настройки на среда за разработка.....	22

5.2	Разработка на интерфейс .....	23
5.2.1	Графичен прозорец с начално меню - MyMenuWindow .....	23
5.2.2	Графичен прозорец с полети - ScannedFlightsWindow .....	24
5.2.3	Информиращ прозорец - popup_info_box .....	24
5.2.4	Стилизиране на проекта .....	25
5.3	Обекти.....	26
5.3.1	Полет - Flight .....	26
5.3.2	Пътуване - Travel.....	27
5.3.3	Функции в конструкторите .....	28
5.4	Автоматизация в Chrome браузър.....	32
5.4.1	Добавяне на всички полети в списък - add_flights.....	32
5.4.2	Въвеждане на данни и търсене на полет - search_flight .....	33
5.4.3	Инструкции за търсене, сортиране и добавяне на полет - adding_set_of_flights .....	34
5.4	Нишки .....	34
5.4.1	Анимация за зареждане – LoadingThread .....	35
5.5	Помощни функции .....	37
5.5.1	Функции с време - /flight_scanner/date_utils.py .....	38
5.5.2	Функции за улеснение - /flight_scanner/intepreters.py .....	44
5.6	Тестване.....	52
5.6.1	Модулни тестове .....	52
5.7	Покритие на тестовите (coverage) .....	54
5.8	Статичен анализ на код (pylint).....	55
<b>Глава 6</b>	<b>Ръководство за работа.....</b>	<b>56</b>
<b>Глава 7</b>	<b>Заклучение и възможности за бъдещо развитие .....</b>	<b>60</b>
<b>Източници</b>	<b>.....</b>	<b>61</b>

## **Увод**

В съвременния свят на бързото развитие на технологиите и динамичната общност, хората търсят бързи и удобни решения за различни сфери от живота си. Една от тях е пътуването, което е все по-често срещано, лесно достъпно и желано. Независимо дали за работа или за забавление, търсенето на оптимални възможности за пътуване може да бъде трудно и отнемащо много време. За хора, които са ограничени от време и бюджет, това може да бъде и доста неприятно. **Целта** на настоящата дипломна работа е да се проектира приложение, което да улесни и забърза процесът на избор на полети в ситуации, когато човек спонтанно решава, че иска да пътува до желана дестинация евтино, знаейки кои дни от седмицата може да отдели за това преживяване, но нямайки конкретни дати. Използвайки технология за управление на уеб браузъри, приложението бързо и лесно автоматизира търсенето из сайт за резервиране на билети и връща като резултат осемте най-евтини полета, които отговарят на желанията на потребителя. Разработката на такова приложение ще бъде от голяма полза за хората, които обичат да пътуват, но са ограничени във времето и парите. То ще предостави възможности, които да оптимизират процеса на планиране на пътуването и да го направят приятелско, бързо и лесно за всеки, който търси най-евтините възможности за пътуване в кратки периоди от време.

За целта:

- В **Глава 1** са разгледани целите и задачи за изпълнение;
- В **Глава 2** е представен Анализ на темата, където са разгледани подобни приложения;
- В **Глава 3** е показан обзор на използваните технологии чрез, които е реализиран проекта;
- В **Глава 4** са представени Структурните схеми на проекта и тяхното обяснение;
- В **Глава 5** е представена Софтуерната реализация на проекта и подробни обяснения на използваните функции;
- В **Глава 6** е указан начина по който се работи с приложението;
- В **Глава 7** е изготвено заключение и направено обобщение на потенциално бъдещо развитие на проекта;

## **Глава 1**

### **Цели и задачи**

Целта на дипломната работа е да се разработи настолно приложение, което да предостави осемте най-евтини полети за последващият месец до дадена дестинация, които стартират и приключват във фиксирани дни от седмицата. Да има графичен потребителски интерфейс, за удобно навигиране, който след въвеждането на желаните данни от потребителя, паралелно да отваря няколко Chrome браузъри и автоматично да намира данни за търсените полети. Като резултат, полетите трябва да се подредят по цена във възходящ ред и да бъдат представени на потребителя отново, чрез потребителски интерфейс.

Други изисквания и задачи към проекта са:

- Да се създаде удобен интерфейс за потребителя чрез PyQt5;
- Да се създаде нишка, която да информира потребителят, че приложението е в процес на извличане и зареждане на информация;
- Да се създадат паралелно по 2 нишки, за всяка възможна дата за отпътуване за последващият месец;
- Да се използва Selenium за автоматизация и контрол на Chrome Browser;
- Да се извлече нужната информация за избраните евтини полети;
- Да се сортират и предоставят удобно за потребителят резултати от полети;
- Да има към всеки полет линк, който да сочи към страницата, откъдето може да бъде закупен самолетният билет;
- Да се изпълнят функционални тестове на приложението, за да се гарантира коректното му функциониране;
- Да се измери code coverage на изпълнението на програмния код, за да се определи каква част от него е изтествана;
- Да се използва Pylint за статичен анализ на кода, за да се провери за наличието на потенциални проблеми и грешки в програмния код.

Крайната цел на дипломната работа е да се предостави на потребителите лесно за използване и ефективно приложение, което ще помогне да се намерят най-евтините полети до желаната дестинация.

## **Глава 2**

### **Анализ на темата**

#### **2.1 Анализ**

Предложеното приложение е насочено към потребителите, които често пътуват и търсят най-евтините полети за определен период от време. Идеята на автоматизирането на процеса за търсене и намиране на най-евтините полети може да бъде много полезна за потребителите, тъй като замества ръчното търсене на информация, което може да бъде трудна и отнемаща много време.

Приложението изисква програмиране на няколко функции за взаимодействие с уеб браузъри и извличане на информация от тях, както и сортиране на резултатите и предоставяне на линк за всеки полет, който води към страницата за закупуване на билета. Използването на Selenium е много добър избор за автоматизация на този процес, тъй като той позволява управление на уеб браузъра и автоматизиране на действията на потребителя, като извличане на данни и кликуване на линкове.

Като цяло, приложението е добре мислено и има потенциал да бъде много полезно за потребителите, които търсят най-евтините полети. Основното предимство е автоматизирането на процеса, което значително намалява времето за търсене на евтини полети.

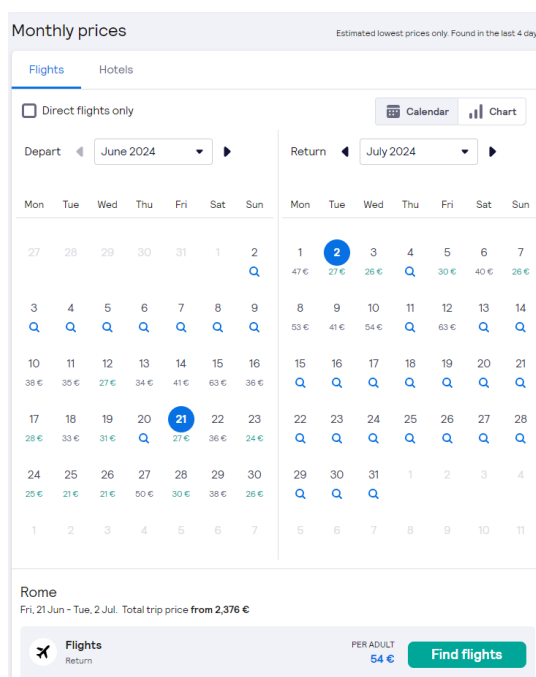
#### **2.2 Подобни решения**

##### **2.2.1 Skyscanner**



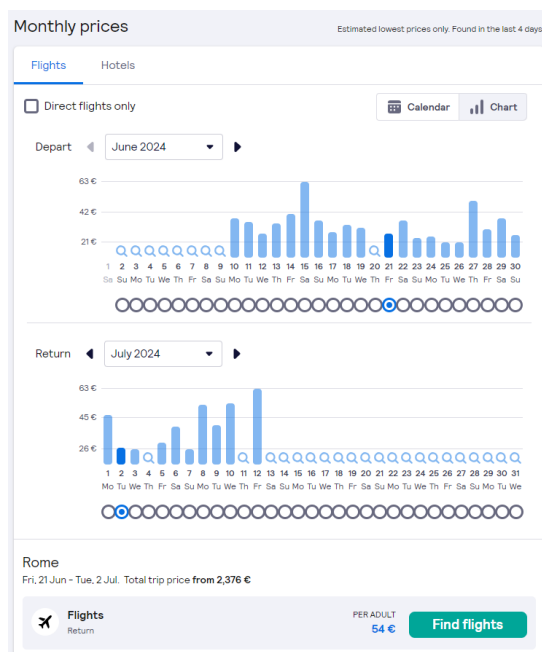
*Фигура 1*

Това е уеб сайт, който търси за най-евтините полети между различни дестинации. Той предлага филтриране на резултатите по различни критерии, като цена, време на отпътуване и продължителност на пътуване. Имат и мобилно приложение.



Фигура 2

Близки функционалности до настоящето дипломно задание са възможността за избор на билети чрез месечен календар. Потребителят може ръчно да погледне прогнозните цени на билетите и да си избере, кога би искал да пътува. Позитивите са, че изглежда естетично, има възможност да проследи цените на билетите за по-голям период от време. Негативите са, че клиентът трябва един по един да проверява полетите, да маркира дни за отпътуване и за връщане, да помни на кои дати, какви са били цените и сам да сравни, кога би било най-икономично да се пътува. Докато разработенето приложение в настоящата дипломна работа прави всичко това автоматично.



Фигура 3

Друг вариант, който Scyscanner предлага е графично представяне, в които са изобразени прогнозните цени за полетите. Отново се предоставя на клиента красив и леснотетим интерфейс с възможност за следене на цените за по-дълъг период. Но за сметка на това, потребителят пак трябва ръчно да маркира различните комбинации и сам да проследи, кои дати излизат най-изгодно.

## 2.2.2 Esky



Фигура 4

Това е туристическа агенция, която предлага услуги за резервации на полети, хотели, наем на автомобили и други туристически услуги. Те предоставят много изгодни оферти и възможности за пътувания по цял свят. Имат и мобилно приложение.



Фигура 5

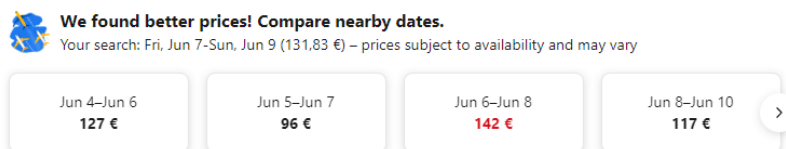
Близко като функционалност, което предлага Esky е календар с изложени цени за различните дни. Също така имат графика с най-евтините полети за различните месеци. Но също както в Skyscanner, клиентът собственооръчно трябва да прегледа цените за избраните дни и да си сравни, кога би му излезнало най-изгодно.

## 2.2.3 Booking



Фигура 6

Това е уебсайт и мобилно приложение, което позволява на потребителите да резервират хотели, вили и други места за настаняване в цял свят. Освен това предлагат и други туристически услуги като наем на автомобили, самолетни билети и трансфери от летищедохотела.



Фигура 7

Сайтът предлага на клиента избор от цени в близките дати, но тук, за да се постигне желаният резултат, трябва да много да търси. Ограничението на датите е до 1 седмица и трябва постоянно да въвежда дати и дестинация. Отнема много време и става неприятно за клиента.



## 2.3 Заключение

В заключение от проведения анализ за подобни продукти бяха установени следните изводи:

- Като основен недостатък за всички проучени уеб сайтове, може да се отбележи фактът, че за всеки един от тях, клиентът собственооръчно трябва да провери, за кои дати са най-евтини полетите и да си направи сравнение. Това води до отнемане на повече време за получаване на желаният резултат
- Като изключим горепосочения недостатък, предимствата за всяко едно от тях също са налице. Функциите като излагането на графики, за предстоящите месеци и гъвкавостта за проверяване на билетите за по-дълъг срок от време, дава възможност за повече избор от дати. Също така имат работещи мобилни приложения за удобство на клиента, както и огромен обем от данни с различни варианти за филтриране и сортиране на информацията.

## Глава 3

### Обзор на използваните технологии

#### 3.1 Python



Python е интерпретативен, обектно-ориентиран език за програмиране от високо ниво с динамична семантика. Вградените в него структури от данни на високо ниво, съчетани с динамичното типизиране (dynamic typing) и динамичното обвързване на данни (data binding), го правят много привлекателен за бързото разработване на приложения, както и за използване като скриптов или „междинен“ език за свързване на съществуващи компоненти заедно.

Езикът разполага с прост и лесен за научаване синтаксис, което прави кода лесно четим и разбираем. Поддържа голям набор от модули и пакети, което допринася за модулярността на програмата и за повторното използване на код. Интерпретаторът на Python и обширната стандартна библиотека се предлагат безплатно за всички основни платформи и могат да се разпространяват свободно. Python осигурява сравнително висока производителност. Тъй като няма стъпка за компилация, цикълът за „редактиране – тест – отстраняване на грешки“ е невероятно бърз.

Езикът има много приложения. Може да се използва, за разработка на уеб приложения, игри, софтуер за изкуствен интелект и наука за данните (Data Science), десктоп графични интерфейси, разработка на сървърни приложения, софтуер за контрол на микроконтролери и много други.

## 3.2 Pycharm



Pycharm е интегрирана среда за разработка на софтуер, която се използва за програмиране на Python. Създадена е от компанията JetBrains и е насочена към начинаещи и професионални програмисти.

Интерфейсът на му е много интуитивен, който прави лесно да се открият функциите и инструментите, които са нужни за програмистите при разработката на софтуер. Pycharm предлага пълен сет от инструменти за разработка, като поддръжка на Git, различни дебъгери, работа с виртуални среди и по-лесна работа с библиотеки и пакети.

Основните функционалности, които предлага Pycharm, включват създаване на проекти, изграждане на програми и отстраняване на грешки. В допълнение, той има интеграция с множество библиотеки и фреймуърки, като Django, Flask, NumPy, SciPy и много други.

Pycharm е платформа, която може да работи на всички операционни системи, включително Windows, Mac OS X и Linux, като предлага много езици за програмиране като Python, JavaScript, CSS, HTML и други.

Истинската мощ на платформата се показва, когато се използва за по-големи проекти с множество файлове и начини на организация на кода. Pycharm е необходим инструмент за всяко съвременно разработчика на софтуер, който търси мощна, лесна за употреба и пълнофункционална интегрирана среда за разработка на софтуер.

### 3.3 Selenium



Selenium е мощен инструмент за автоматизация на уеб браузъри, който позволява на разработчиците да засилят тестовете си върху уеб приложения. Той позволява на потребителите да изпълняват автоматизации, като симулират действията на потребителите в реалният свят. С помощта на Selenium, разработчиците могат да тестват различни уеб приложения и да проверят функционалността на формите, да потвърдят наличието на елементи и да проверят дали сайтът работи както е предвидено. Освен това, Selenium работи като симулира идеалната работа на различни уеб браузъри като Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Apple Safari и други. Това го прави много полезен инструмент за разработчиците и тестерите, когато трябва да проверят бързодействието и стабилността на уеб сайтовете и приложенията.

### 3.4 PyQt5



PyQt5 е популярна библиотека на програмния език Python, която осигурява интеграция с графичен потребителски интерфейс (GUI) и позволява на програмистите да създадат съвременни и привлекателни приложения с помощта на Python. PyQt5 осигурява широк спектър от елементи на потребителския интерфейс, които потребителите могат да използват, като например диалогови прозорци, менюта, инструментални ленти, текстови полета, бутони и други. Освен това, PyQt5 също

поддържа и други визуални елементи като списъци, таблица или графики. Тази библиотека има много документация и обучителни материали, които помагат на начинаещите програмисти да развиват своите умения в програмирането на ГПИ. PyQt5 е мощен инструмент за създаване на по-подробни и интерактивни GUI приложения, като например софтуерни приложения за управление на бази данни, игри и много други.

### 3.5 Google Chrome



Фигура 12

Google Chrome е популярен уеб браузър, разработен от Google и пуснат за първи път през 2008 г. Той е набира голяма популярност заради своята бързина, сигурност и леснота на използване. Google Chrome предлага пълен набор от функции за преглед на уеб страници, включително качане на файлове, синхронизация на данни между различни устройства, поддръжка на HTML5 и много други. Този браузър има уникални функции, като например вградени инструменти за превод, защита от измамници и вредни програми, като анти-вирусен софтуер, и интеграция с още много други Google услуги, като например Gmail, Google Drive и др. Google Chrome запазва позицията си като един от най-популярните уеб браузъри в света, като продължава да се развива и да добавя нови функции, осигурявайки максимално качество и удобство за потребителите си.

### 3.6 Google Flights



Фигура 13

Google Flights е популярна онлайн търсачка за пътувания, която е разработена от Google. Този уеб сайт има за цел да предостави най-изгодните варианти за полети до различни дестинации. Google Flights предоставя на потребителите търсачка на полети и цени, които потребителите могат да филтрират по цена, дата, време на излитане, компании за полети, времетраене, и други параметри. Google Flights предоставя изчерпателна информация за всеки полет, включително информация за багаж, излитане, пристигане, и трансфери от летището.

Освен това, Google Flights предоставя и удобни функционалности за сравнение на различните полети и различни опции за пътуване. Функциите включват картографски изглед, който показва цените на полетите на карта, график с цени, който показва как цените на полетите ще се изменят по разнообразни дни и условия, както и функционалност за сравнение на цени на много различни полети и маршрути. Google Flights предоставя на потребителите голямо количество информация и помага на потребителите да създават пътувания, които отговарят на техните специфични нужди и желания.

### 3.7 Github



GitHub е уеб базиран хостинг за програмен код, който работи върху системата за контрол на версиите Git. Този инструмент позволява на потребителите да съхраняват своите програмни проекти в облака, да сътрудничат с други програмисти в реално време и да управляват целия контрол на версиите на своя код.

GitHub представлява електронна работилница, в която програмистите могат да планират проекти, да работят по тях, да комуникират помежду си и да делят своя код с други потребители. Този уеб сайт позволява на потребителите да следят промените на проекта им, да правят промени в кода си, да коментират, да съобщават за проблеми и да правят корекции, когато е необходимо.

### 3.8 CSS



Фигура 15

CSS (Cascading Style Sheets) е език за стилово оформление на уебсайтове, който се използва за разделяне на дизайна от съдържанието и представяне на уебстраниците. Той позволява уеб дизайнерите да прилагат стилове към елементите на уеб страницата или приложението си, като определят фонове, шрифтове, размери, рамки, подчертавания, разстояния между елементите и други. Езикът е разработен с цел да подобри визуалният дизайн и се използва глобално от милиони уеб дизайнери и разработчици в целия свят. Кодът, използван в CSS, е лесно разбираем, многофункционален за да създава съвременни, интерактивни и атрактивни дизайни за потребителите.

### 3.9 Библиотеки

#### 3.9.1 Pytest



Фигура 16

Pytest е популярна библиотека на програмния език Python, използвана за тестване на софтуерни системи. В основата му е направен удар върху тестването на софтуерни приложения и възможността да се пише директно тестов код.

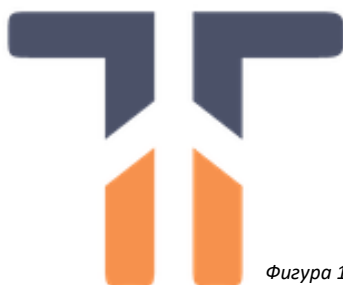
Pytest предлага широк спектър от функционалности, като например конфигурация на тестовете, групиране и управление на тестове. Освен това, той предоставя поддръжка



на fixture-и, които могат да бъдат използвани за подготвяне на тестващата среда във всеки един тест.

Той също е много лесен за използване и прочитаем за програмистите, които търсят ефективен начин за тестване на своите Python приложения. ази библиотека е инструмент за тестване на софтуер, който може да помогне на програмистите да направят значителен напредък в увереността им в софтуерните приложения и да ускорят разработката на качествени софтуерни продукти.

### 3.9.2 Coverage



Фигура 17

Coverage е инструмент за тестване на код, който показва каква част от кода е била използвана при изпълнение на тестови примери. В основата му стои идеята, че ако програмният код е написан за да извършва работа, която никога не се изпълнява, то не се знае дали не съществуват проблеми и грешки в него.

С него потребителите могат да определят какъв процент от кода им е покрит от тестове. Инструментът анализира кода, изпълнявайки тестове и наблюдавайки коя част от кода всъщност е изпълнена и коя не е. Така програмистите могат да идентифицират части от програмния си код, които може да трябва да бъдат променени, опростени или изтрити.

Coverage е полезен инструмент за програмистите, тъй като те могат да осигурят високо качество на своя програмен код. Вземането на решения за подобряване на тестовете на софтуера им помага да се предотвратяват евентуални проблеми и да се увеличи стабилността и готовността на програмните им продукти.

### 3.9.3 Pylint



Pylint е популярен инструмент за анализиране на код, написан на програмния език Python. Той използва статичен анализатор за проверка на кода за наличието на грешки, стила и конвенциите. Целта на библиотеката е да улесни разработката на кода, като осигури типични кодови стилове, конвенции и практики.

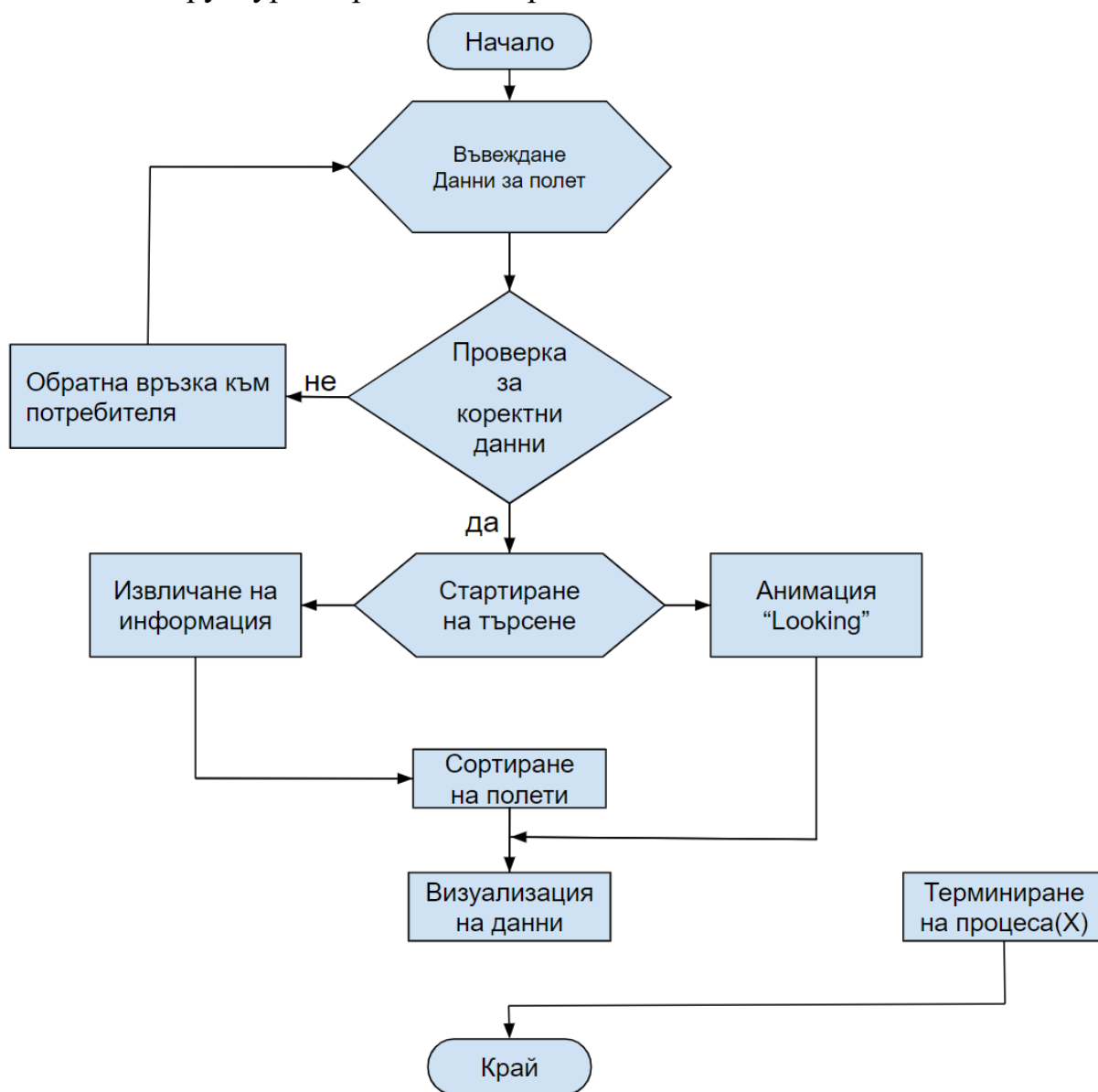
Той може да помогне на програмистите да идентифицират потенциални грешки в приложенията си, като наблюдава за неправилни идентификатори, несъответствия в типове, липса на документация, излишни операции и други. При наличие на такива грешки, инструментът предлага практически съвети за тяхното отстраняване.

Pylint помага на разработчиците да създадат качествен, чист и поддържаем Python код, който следва най-добрите практики. Освен това, той може да бъде интегриран лесно във всеки етап на софтуерния животен цикъл, за да се гарантира, че кодът отговаря на изискванията и задачите си.

## Глава 4

### Схема на работа на приложението

#### 4.1 Пълна структура на работния процес



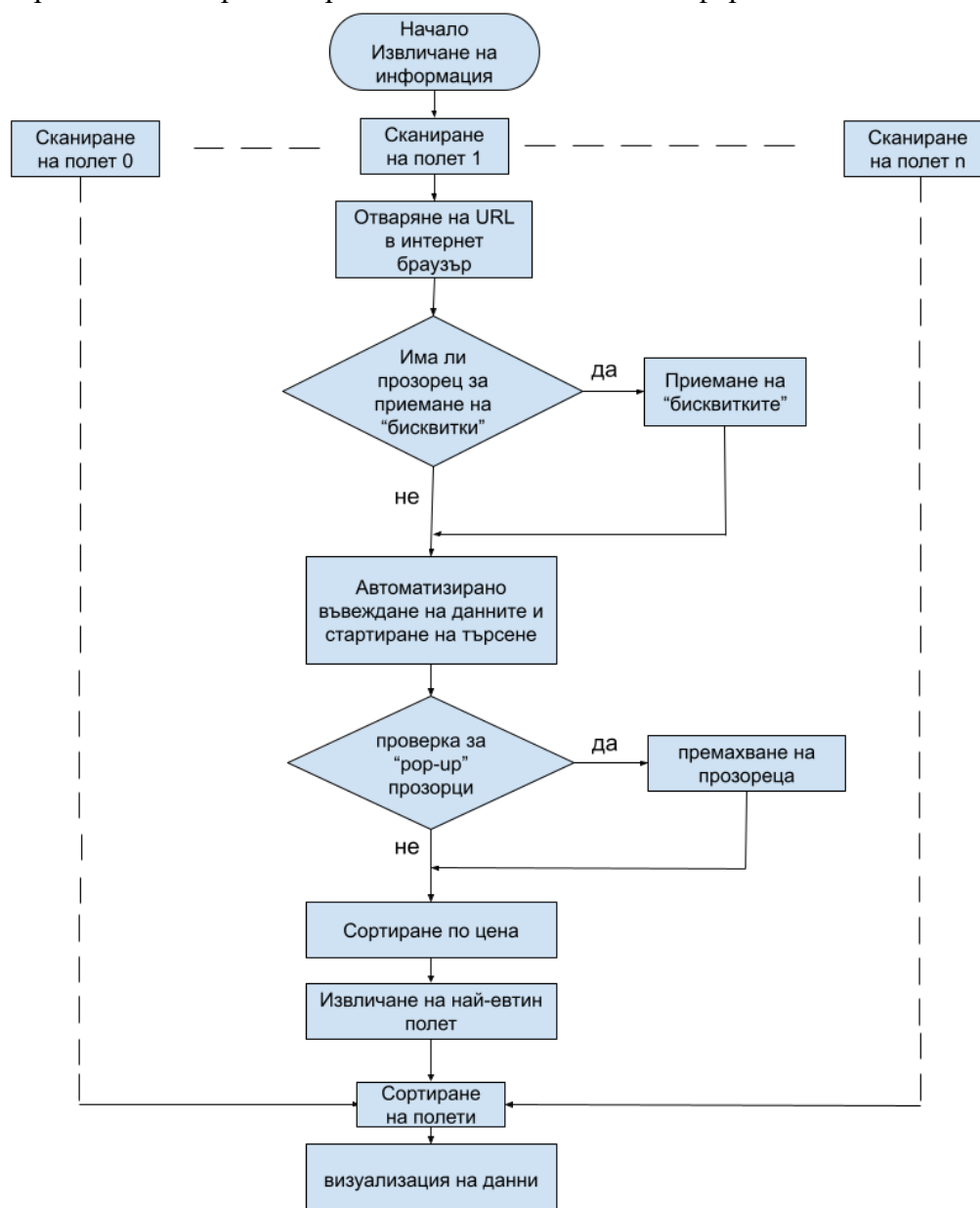
Фигура 19

Тази схема изобразява целият процес на приложението. Клиентът стартира програмата и се отваря прозорец, в който трябва да попълни съответната информация. Прави се проверка дали данните са коректни. Ако не се, се отваря информативен прозорец, уведомяващ потребителят, че данните са некоректни или непълни, връщайки го на първата страница. При успешно изпълнени условия, програмата продължава надолу, стартирайки два основни процеса. Единият е анимация за зареждане, а другият е търсене на полети в интернет. Следа като се намерят всички полети, те биват

сортирани. Тогава и двата процеса приключват своята работа и се почвява нов прозорец, където са визуализирани осемте най-евтини полета. Програмата може да приключи по всяко време от бутон за затваряне.

#### 4.2 Схема за извличане на информация за самолетни полети

Тук е разгледан подробно процесът на извличане на информацията за полетите.

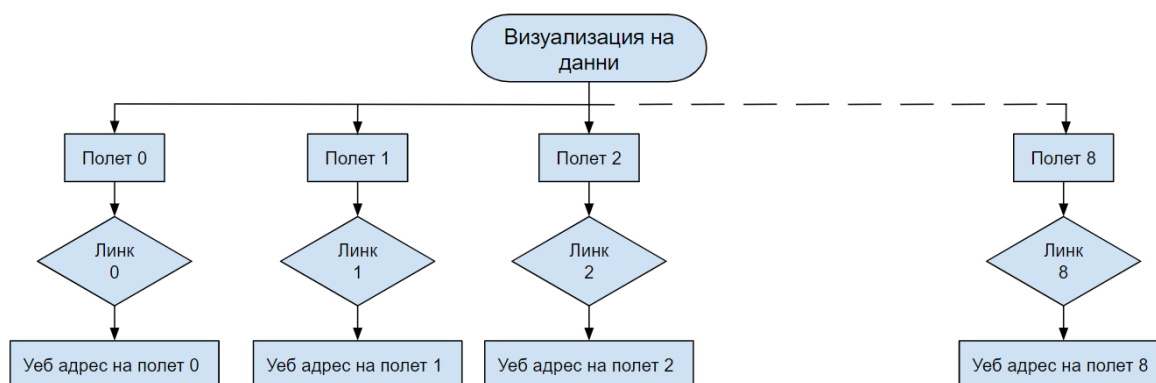


Фигура 20

Както бе описано на предходната фигура, след стартирането на търсенето, започва намирането на информация. Едновременно се стартират няколко процеса (между 8 и 10, в зависимост от датите), като за всеки един от тях процедурата е една и съща. Всеки отваря по един уеб браузър, прави се проверка за бисквитки и започва

въвеждането на данните. Дестинацията е една и съща, но датите се разминават за някои от процесите. Следва да се направи проверка за изскачащи прозорци и да се премахнат, ако има такива. Сортират се полетите за зададените дати и се извлича информацията на най-евтините полети. Всички процеси събират информацията в един общ лист. Нишките приключват работа, полетите се подреждат по цена и накрая се визуализират в нов прозорец.

### 4.3 Схема на визуализация на полети



Фигура 21

Събраната информация за осемте полета се изобразява в последният графичен прозорец, като всеки полет има личен бутон с повече информация, който при кликуване, отваря интернет страница откъдето може да бъде закупен билета за него.

## Глава 5

### Софтуерна реализация

#### 5.1 Настройки на среда за разработка

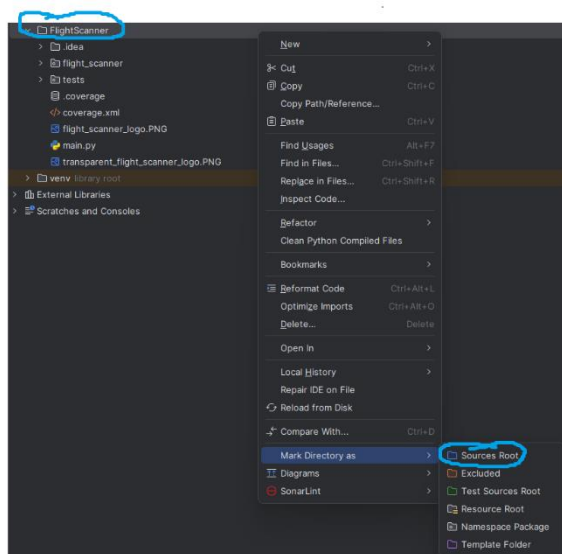
За да се подкара проектът, трябва да се инсталира среда за разработка на софтуер, която да поддържа езикът python. За изработването на приложението е използван Pycharm.

Следваща стъпка, трябва да се настрои интерпретиращата програма за python. Това става по следният начин: File -> Settings -> Project: FlightScanner -> Python interpreter -> Add interpreter -> Add local Interpreter -> Virtualenv environment -> Location: '..\FlightScanner\venv' (къде да бъде инсталирана виртуалната среда) -> Base Interpreter: '..\Python\Python312\python.exe' (пътят към Python, който трябва да бъде предварително инсталиран) -> OK

Следва да се инсталират следните библиотеки:

- PyQt5
- selenium
- python-dateutil
- pytest
- pytest-qt
- freezegun
- pytest-mock

За финал се клика с десен бутон върху FlightScanner папката, която се намира на нивото на .venv, отиваме на Mark Directory as и маркираме папката като Source Root. Така проектът е напълно конфигуриран и готов за ползване.

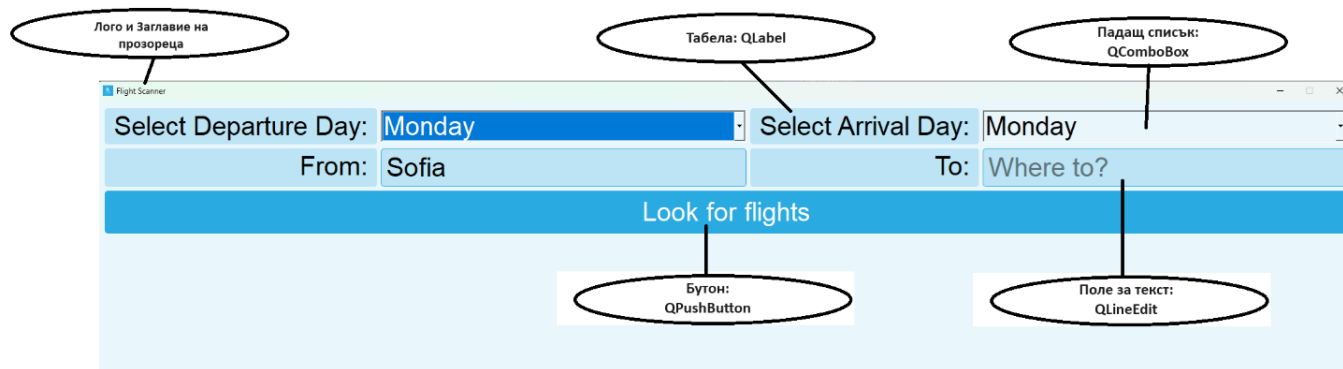


Фигура 22

## 5.2 Разработка на интерфейс

За изработването на интерфейса се използва PyQt5 и включва различни компоненти като текстови полета, бутони, табели и други. Състои се от два прозореца или още наречени класове. Намират се във `/flight_scanner/windows.py`:

### 5.2.1 Графичен прозорец с начално меню - MyMenuWindow



Фигура 23

Кода представя реализацията на графично потребителски интерфейс (GUI) на приложение за търсене на евтини полети. Основните компоненти на графичния интерфейс са QLabel, QComboBox, QLineEdit и QPushButton, които се използват, за да направят приложението достъпно, изобразимо и лесно за управление от потребителя.

Горе в дясно се намират логото и заглавието на прозореца.

*QLabel* – елемент, който се използва, за да се показва текстова информация или изображение в графичен интерфейс. В този прозорец фигурират 4 табели и служат, за да информират потребителят за това, каква информация трябва да попълнят в следващите елементи.

*QComboBox* – елемент, който позволява на потребителя да избере елемент от падащ списък с опции. В горепосоченият прозорец, се съдържат 2 такива падащи менюта, които служат за избор на ден от седмицата за отпътуване и връщане.

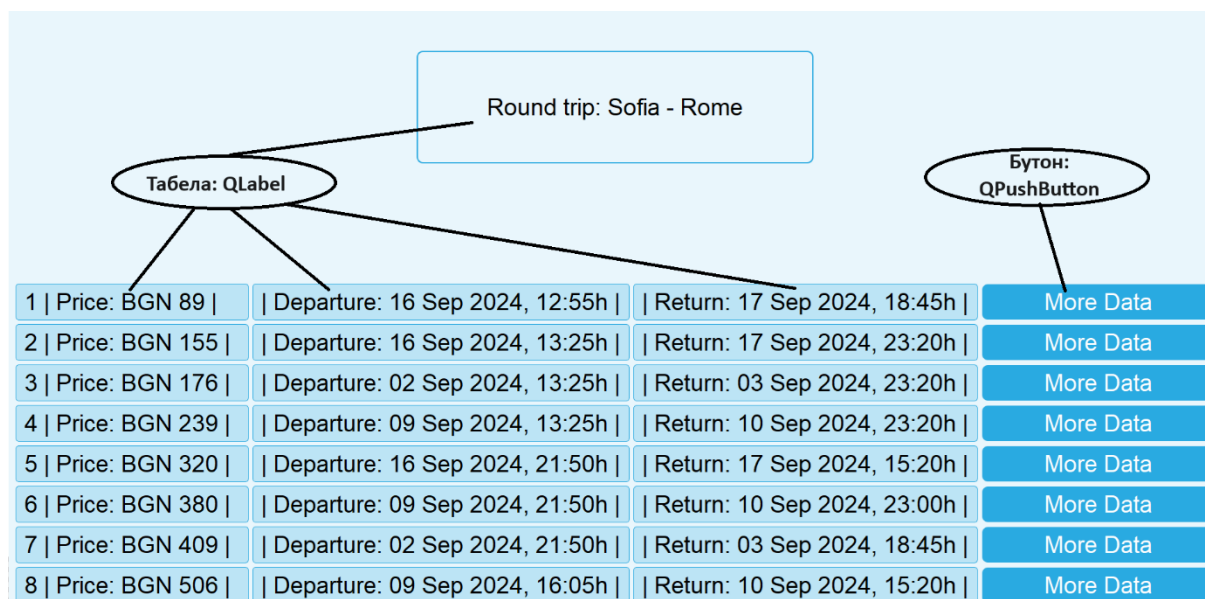
*QLineEdit* – елемент (поле), който позволява на потребителят да въвежда и редактира текст. В горепосоченият прозорец, се съдържат 2 такива полета, които служат за избор на дестинации заминаване и пристигане.

*QPushButton* – елемент (бутон), който може да бъде натиснат от потребителя и да стартира дадена функция. В горепосоченият прозорец има само един бутон, който стартира търсенето на евтини полети и накрая отваря новият прозорец – ScannedFlightsWindow.

Кода също така използва многонишкова обработка, за да оптимизира търсенето на полети и да го направи по-ефективно за потребителя.

MyMenuWindow е интерфейс, който приветства потребителя, позволява му да въведе информацията, необходима за търсене на полетите и да започне визуализация на резултатите.

## 5.2.2 Графичен прозорец с полети - ScannedFlightsWindow



Фигура 24

Класът ScannedFlightsWindow представлява графичен прозорец в приложението, който показва списък от 8те най-евтини намерени полета за желаната дестинация.

Дизайнът на прозореца е насочен към удобството и лесното визуално представяне на полетите. Използвани са QLabel и QPushButton, като елементи за постигане на желаният резултат. Добавени са и hover ефекти за по-добро потребителско изживяване.

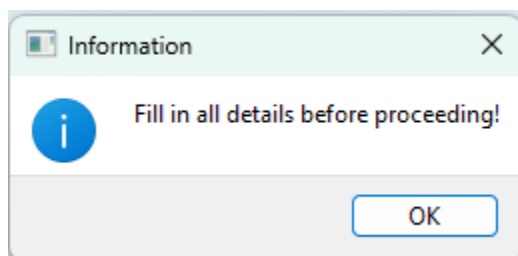
## 5.2.3 Информиращ прозорец - popup\_info\_box

```
def popup_info_box(title, text):
    """
    Displays an information popup box.

    Parameters
    -----
    title : str
        The title of the popup window.
    text : str
        The message text to display in the popup window.
    """
    info_box = QMessageBox()
    info_box.setIcon(QMessageBox.Information)
    info_box.setWindowTitle(title)
    info_box.setText(text)
    info_box.setStandardButtons(QMessageBox.Ok)
    info_box.exec()
```



Тази функция се използва за показване на информационно прозорче в приложението. То съдържа заглавие и текстово съобщение, което може да бъде персонализирано от потребителя в зависимост от нуждите на приложението.



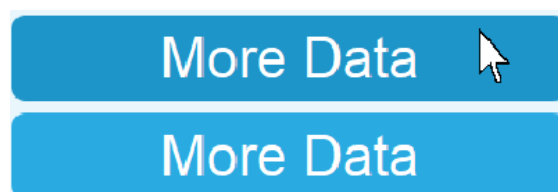
Фигура 25

- `popup_info_box('Information', 'Fill in all details before proceeding!')`
- `popup_info_box('Warning', 'The submitted information is incorrect. Fill in the correct data.')`

В случаят са използвани, за да информират потребителя, че липсва информация или въведените данни са некоректни.

## 5.2.4 Стилизиране на проекта

```
button.setStyleSheet(  
    '''  
    QPushButton {  
        background-color: #29AAE1;  
        color: white;  
        border-style: outset;  
        border-width: 2px;  
        border-radius: 10px;  
        border-color: transparent;  
        padding: 6px;  
        box-shadow: 10px 10px 10px  
grey;  
    }  
    QPushButton:hover {  
        background-color: #1d95c9;  
    }  
    QPushButton:pressed {  
        background-color: #2c3e50;  
    }  
    '''  
)
```



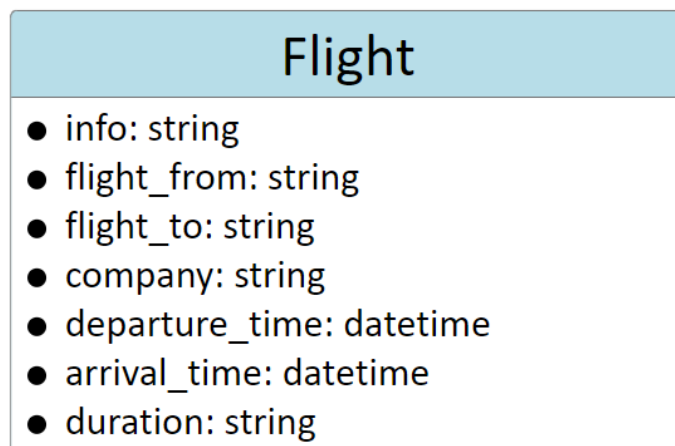
Фигура 26

За стилизиране се използва CSS, като този примерен блок код задава стила на бутоните в ScannedFlightsWindow, използвайки отново CSS синтаксис за определяне на цветове, рамки, сянки и други ефекти. Когато потребителят ховърне или натисне бутона, цветът се изменя. Това прави приложението по-привлекателно за потребителите.

### 5.3 Обекти

В настоящата дипломна работа обектите се намират във файлът `/flight_scanner/flight.py`

#### 5.3.1 Полет - Flight



Фигура 27

Този клас представя полет и съхранява информация за него, като място на заминаване и пристигане, име на компания, продължителност на полета и други. За инициализация на обекта той приема за аргументи информация за полета, извлечена от сайта, входни данни (ден от седмицата за заминаване и връщане, дестинация за отиване и връщане и лист с валидни дати до края на месеца)

Пример:

- Входни данни:

```
flight_info = 'Mon, Sep 2\n1:25 PM\n - \n2:10 PM\nRyanair\nOperated by Ryanair Sun\n1 hr 45 min\nSOF-CIA\nNonstop\n91 kg CO2e\nAvg emissions'
input_data = {'departure_weekday': 'Friday',
              'arrival_weekday': 'Sunday',
              'flight_from': 'Sofia',
              'flight_to': 'Rome',
              'dates_list': [{'End': 'Tue, Sep 3', 'Start': 'Mon, Sep 2'},
                             {'End': 'Tue, Sep 10', 'Start': 'Mon, Sep 9'},
                             {'End': 'Tue, Sep 17', 'Start': 'Mon, Sep 16'}]}
direction = 'straight'
```



```
Travel = Travel.departure_flight == Flight()
        Travel.arrival_flight == Flight()
        Travel.price == 'BGN 176'
        Travel.link == 'https://www.google.com/travel/
                        flights/booking?tfs=CBwQAhpKEgoy'
```

### 5.3.3 Функции в конструкторите

В този файл се намират 5 функции, които се използват в конструкторите на класовете Flight и Travel, за да се инициализират обектите с нужната информация.

#### 5.3.3.1 Място за отлитане - get\_departure

```
def get_departure(flight_info, input_data, direction):
    """
    Extracts and returns the departure airport information from the flight info
    string.

    Parameters
    -----
    flight_info : str
        The string containing details about the flight.
    input_data : dict
        A dictionary containing additional flight search parameters.
    direction : str
        The direction of the flight, either 'straight' or 'back'.

    Returns
    -----
    str
        The departure airport information.
    """
    departure = re.search(r'[A-Z]{3,}-[A-Z]{3,}', flight_info).group()
    if direction == 'straight':
        departure = input_data['flight_from'] + ', ' + departure.split('-')[0]
    else:
        departure = input_data['flight_to'] + ', ' + departure.split('-')[0]
    return departure
```

Тази функция приема три параметъра - flight\_info (низ), input\_data (речник) и direction (низ) и извлича и връща информацията за полета на заминаване. Тя използва регулярни изрази, за да търси шаблон на три или повече поредни главни букви, последвани от тире, последвани от друг набор от три или повече поредни главни букви. Това е стандартният формат за кодовете на летища.

След като извлече летищния код, функцията проверява параметъра direction. В зависимост от това дали посоката е направо или наобратно, функцията връща дестинацията на полета за отлитане под формата на низ.

Пример:

- Входни данни:

```
flight_info = 'Mon, Sep 2\u201125\u2011PM\u201110\u2011PM\u2011\nRyanairOperated by Ryanair Sun\u20111 hr 45 min\u2011SOF\u2011CIA\u2011Nonstop\u201191 kg\nCO2e\u2011Avg emissions'\ninput_data = {'departure_weekday': 'Friday',\n              'arrival_weekday': 'Sunday',\n              'flight_from': 'Sofia',\n              'flight_to': 'Rome',\n              'dates_list': [{'End': 'Tue, Sep 3', 'Start': 'Mon, Sep 2'},\n                             {'End': 'Tue, Sep 10', 'Start': 'Mon, Sep 9'},\n                             {'End': 'Tue, Sep 17', 'Start': 'Mon, Sep 16'}]}\ndirection = 'straight'
```

- Изходни данни:

```
departure = 'Sofia, SOF'
```

### 5.3.3.2 Място за кацане - get\_arrival

Тази функция изпълнява същото действие като предходната, но връща мястото за кацане.

### 5.3.3.3 Продължителност на полета - get\_duration

```
def get_duration(flight_info):  
    """  
    Extracts and returns the duration of the flight from the flight info string.  
  
    Parameters  
    -----  
    flight_info : str  
        The string containing details about the flight.  
  
    Returns  
    -----  
    str  
        The duration of the flight.  
    """  
    duration = re.search(r'(\d+ hr( \d+ min)?)|((\d+ hr )?\d+ min)',  
                        flight_info).group()  
    return duration
```

Тази функция приема един параметър - flight\_info (низ) и връща продължителността на полета. Тя използва регулярен израз, за да търси шаблон от числа и 'min' текст. Това позволява на функцията да е гъвкава и да хване различни формати на времето като:

- "5 hr",
- "1 hr 45 min"
- "45 min"

Пример:

- Входни данни:

```
flight_info = 'Mon, Sep 2\n1:25\u202fPM\n2:10\u202fPM\nRyanair\nOperated by Ryanair\nSun\n1 hr 45 min\nSOF-CIA\nNonstop\n91 kg\nCO2e\nAvg emissions'  
  
direction = 'straight'
```

- Изходни данни:

```
duration = '1hr 45 min'
```

#### 5.3.3.4 Час на отлитане - get\_departure\_time

```
def get_departure_time(flight_info):  
    """  
    Extracts and returns the departure time of the flight from the flight info  
    string.  
  
    Parameters  
    -----  
    flight_info : str  
        The string containing details about the flight.  
  
    Returns  
    -----  
    datetime  
        The departure time of the flight.  
    """  
    departure_time = re.search(r'\d\d?:\d\d(?:PM|AM)', flight_info).group()  
    departure_time = departure_time.replace('\u202f', ' ')  
    flight_date = get_flight_date(flight_info)  
    departure_time = datetime.strptime(departure_time, '%I:%M %p')  
    departure_time = flight_date.replace(hour=departure_time.hour,  
                                         minute=departure_time.minute)  
    return departure_time
```

Тази функция приема един параметър - flight\_info (низ) и връща времето на заминаване на полета от низа за информация за полета. Тя използва регулярен израз, за да търси шаблон от две или три цифри, двоеточие, две цифри, последвани от специални символи и букви 'PM' или 'AM', които означават, че часът е след обяд или преди обяд.

Когато времето на заминаване на полета е намерено в текста, функцията използва метода .replace() за да премахне всички интервали преди и след времето на полета, което са представени чрез специалния символ \u202f.

След като е намерено правилното време на полета, функцията използва друга функция - get\_flight\_date(flight\_info), за да извлече датата на полета. След това, използвайки метода .strptime() от модула datetime, функцията преобразува намереното време във формат на часовник и го добавя към датата на полета, за да получи пълната дата и час на заминаването на полета.

Накрая, функцията връща времето на заминаване като обект от datetime.

Пример:

- Входни данни:

```
flight_info = 'Mon, Sep 2\n1:25\u202fPM\n2:10\u202fPM\nRyanair\nOperated by Ryanair\nSun\n1 hr 45 min\nSOF-CIA\nNonstop\n91 kg\nCO2e\nAvg emissions\ndirection = 'straight'
```

- Изходни данни:

```
departure_time = datetime(2024, 9, 2, 13, 25)
```

#### 5.3.3.5 Час на кацане - get\_arrival\_time

Тази функция е на същият принцип като предходната, но връща времето на полета на кацане.

### 5.4 Автоматизация в Chrome браузър



За да се извлече нужната информация за полетите се използва Selenium за автоматизирането на уеб браузъра. Във файла `/flight_scanner/flight_scanner.py`, се намират нужните инструкции за контролиране на интернет страниците.

#### 5.4.1 Добавяне на всички полети в списък - add\_flights

Тази функция приема 5 параметъра - `flight` (WebElement), `list_flights` (списък), `input_data` (речник), `driver` (WebDriver) и `lock` (threading.Lock) и добавя информация за полета към списъка на полетите. Тя кликва върху елемента `flight`, за да отвори подробна информация за този полет. След това, извършва поредица от операции, за да извлече нужната информацията за него.

Също така, понякога се появяват на екрана съобщение да информира, че има промяна в цената. Тогава функцията търси елементите по зададени други пътища, тъй като в този случай, дефолтните не важат. Ако няма такова съобщение, функцията си продължава инструкциите за извличането на информацията за полета в нормален режим.

Selected flights Track prices ⓘ

	Mon, Jun 17 · 9:50 PM – 10:50 PM Wizz Air	2 hr SOF–FCO	Nonstop	85 kg CO2e -7% emissions ⓘ	▼
	Wed, Jun 19 · 4:20 PM – 7:10 PM Wizz Air	1 hr 50 min FCO–SOF	Nonstop	68 kg CO2e -25% emissions ⓘ	▼

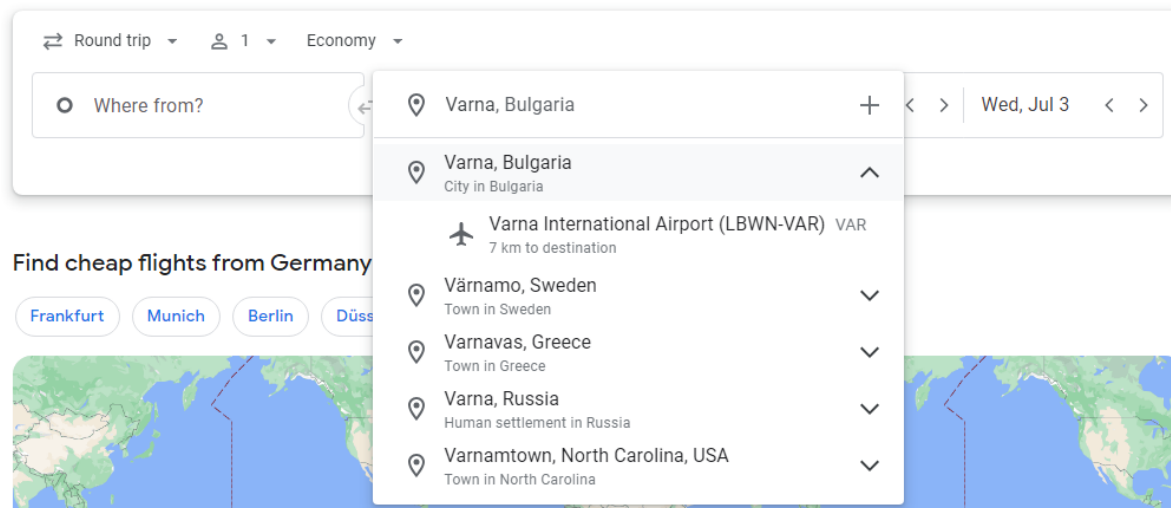
Фигура 29

Данните се записват в речник, от който се образува обект от тип `Travel`. Накрая, функцията използва заключване (`lock`) за да се увери, че списъкът на полетите е в синхронизация и добавя обектите в него.



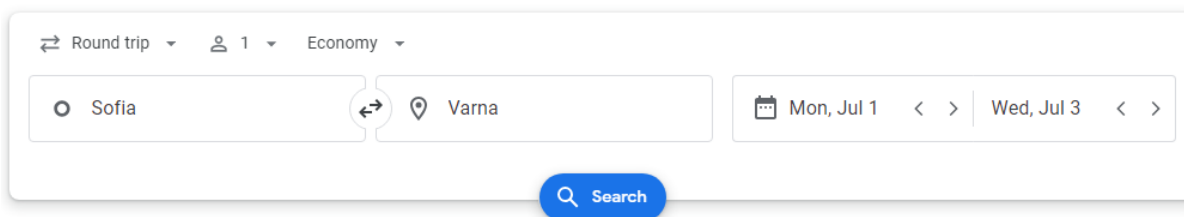
#### 5.4.2 Въвеждане на данни и търсене на полет - search\_flight

Тази функция приема три параметъра - input\_data (речник), set\_num (цяло число) и driver (WebDriver) и търси полети на базата на предоставените данни. Тя първо намира полетата за въвеждане на информация за датите за заминаване и пристигане, както и съответните градове.



Фигура 30

По време на попълването, мениджърът на предложенията отваря лист с въведената дестинация или подобни по-надолу. Тогава програмата е инструктирана да кликне на първото предложение, тъй като то е неправилно в много редки ситуации (От страна на потребителя, при грешно въведени данни). С помощта на метода time.sleep() функцията забавя работата си за 3 секунди, за да се увери, че елементите са напълно заредени. Използва се многократно, тъй като страницата се зарежда по-бавно от четенето на кода, който продължава да върви и е много вероятно да стане разминаване.

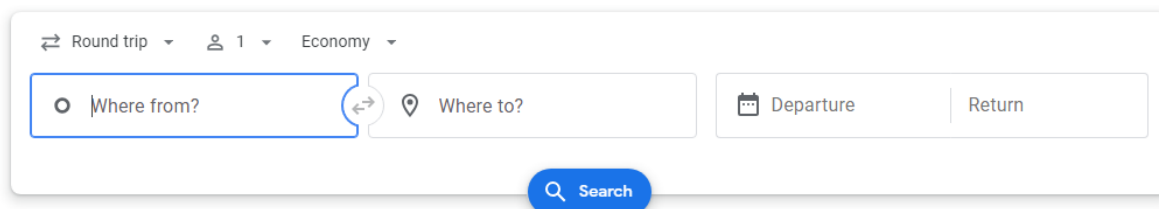


Фигура 31

След това, функцията кликва върху бутона за търсене на полет. За финал, тя връща 0, ако търсенето е успешно, и 1, ако не е.

### 5.4.3 Инструкции за търсене, сортиране и добавяне на полет – flight\_scanning

Тази функция приема пет параметъра - `input_data` (речник), `set_num` (цяло число), `list_flights` (списък), `driver` (`WebDriver`) и `lock` (`threading.Lock`) и добавя група от полети

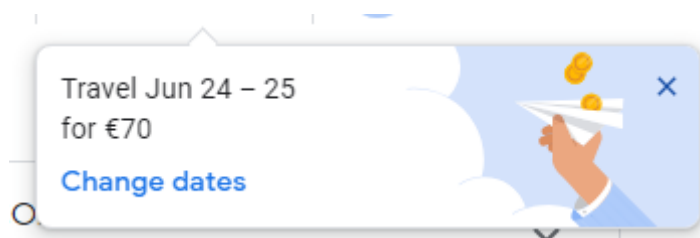


Фигура 32

към списъка с полети.

Тя извиква функцията `search_flight(input_data, set_num, driver)`, която попълва нуцните данни в празните полета, клика върху бутона за търсене и изчаква зареждането на новата страница.

При успех, кодът проверява за евентуално изскочил прозорец (понякога се появява), и го затваря, ако е там. След това използва друга функция - `sort_flights_by_price_driver(driver)`, което подрежда полетите по цена. Следва да се използва метода `get_xpath_for_li(set_num, driver)` за да получи XPATH на елемента,



Фигура 33

съдържащ информацията за въпросния полет. Накрая се извиква `add_flights(find_my_element_by_xpath(driver, flight_xpath), list_flights, input_data, driver, lock)`. Тя има за цел да извлече информацията за полета, да създаде обект от тип `Travel` с нея и да го вкара в списъка с полети.

## 5.4 Нишки

Нишките (`threads`) представляват единици за обработка на едновременни процеси в компютърните системи. Те се използват, когато е необходимо да се изпълнят множество задачи в едно и също време, като всеки един от тези процеси се изпълнява отделно и е независим от другите процеси в системата. Това позволява на целият процес да става по-бързо и ефективно.

В настоящата дипломна работа се намират 2 вида нишки, които могат да бъдат намерени в `/flight_scanner/threads.py`:

### 5.4.1 Анимация за зареждане – LoadingThread

```
class LoadingThread(threading.Thread):
    """
    A thread that updates a loading label in the main window to indicate progress.
    """

    def __init__(self, window, label_loading):
        """
        Initializes the LoadingThread with the given window and label.

        Parameters
        -----
        window : QWidget
            The main window of the application.
        label_loading : QLabel
            The label that displays the loading message.
        """
        super().__init__()
        self.window = window
        self.stopped = False
        self.label_loading = label_loading

    def run(self):
        """
        Runs the thread, updating the loading label with a rotating message.
        """
        while not self.stopped:
            self.label_loading.setText("Loading")
            self.label_loading.repaint()
            time.sleep(1)
            self.label_loading.setText("Loading.")
            self.label_loading.repaint()
            time.sleep(1)
            self.label_loading.setText("Loading..")
            self.label_loading.repaint()
            time.sleep(1)
            self.label_loading.setText("Loading...")
            self.label_loading.repaint()
            time.sleep(2)

    def stop(self):
        """
        Stops the loading thread by setting the stopped flag to True.
        """
        self.stopped = True
```

Това е клас LoadingThread, който наследява threading.Thread и представлява нишка, която се използва за актуализиране на етикета на зареждане на основния прозорец на приложението, създавайки анимация, която да показва напредъка.



Фигура 34

При инициализиране на класа, се предоставят основния прозорец на приложението и етикетът, който се използва за изобразяване на зареждането. Функцията идва с метода `run()`, който се изпълнява, докато флага `stopped` е `False` (предоставено от родителския `threading.Thread` клас). В този метод се актуализира етикета, използвайки ротационно съобщение, което показва зареждането. Функцията променя надписа "Loading" четири пъти, като при всяко промяна добавя точки, за да направи анимацията по-реалистична. Когато методът `run()` завършва, функцията `stop()` се използва за задаване на flag `stopped` на стойност `True`, спирайки нишката на зареждането.

Този клас е създаден за да осигури визуален индикатор за напредъка на приложението, информирайки клиента, че всичко е наред и има леко забавяне.

#### 5.4.2 Нишка за намиране на полет – `ScanningThread`

```
class ScanningThread(threading.Thread):
    """
    A thread that scans for flights and adds them to a shared list.
    """

    def __init__(self, window, index, input_data, list_flights, lock):
        """
        Initializes the ScanningThread with the given parameters.

        Parameters
        -----
        window : QWidget
            The main window of the application.
        index : int
            The index of the current scanning operation.
        input_data : dict
            The input data containing flight search parameters.
        list_flights : list
            The shared list to store the scanned flight data.
        lock : threading.Lock
            A lock to ensure thread-safe access to the shared list.
        """
        super().__init__()
        self.window = window
        self.index = index
        self.input_data = input_data
        self.list_flights = list_flights
        self.lock = lock

    def run(self):
        """
        Runs the thread, scanning for flights and adding them to the shared list.
        """
        driver = driver_setup_headless('https://www.google.com/travel/flights')
        try:
            button_accept_all = find_my_element_by_xpath(
                driver,
                '/html/body/c-wiz/div/div/div/div[2]/div[1]/div[3]/div[1]/div[1]/form[2]/div/div/button'
            )
            button_accept_all.click()
            time.sleep(3)
        finally:
            flight_scanning(self.input_data, self.index, self.list_flights, driver, self.lock)

        driver.quit()
        self.stop()
```

```
def stop(self):  
    """  
    Stops the scanning thread by setting the stopped flag to True.  
    """  
    self.stopped = True
```

Класът ScanningThread наследява threading.Thread и представлява нишка, която сканира за полети и ги добавя в споделен списък.

Конструкторът на класа получава за параметри основния прозорец на приложението, индекса на текущата операция на сканиране, информацията на входа, съдържаща параметрите за търсене на полети, списъка с полетните данни, които ще бъдат сканирани, и ключът за заключване, за да се гарантира безопасен достъп до споделения списък между нишките.

Функцията run() се изпълнява, когато се стартира нова нишка в класа. Започва сканирането за полети, използвайки метода adding\_set\_of\_flights() и добавя резултатите към списъка със споделени данни. Сканирането се извършва с помощта на selenium, като се използва Chrome webdriver. Когато функцията приключи, нишката прекратява работа си чрез метода stop().

ScanningThread е създаден, за да се осигури безопасност в многонишковия отделен процес, който работи едновременно в рамките на приложението, сканирайки за полети и съхранявайки резултатите в споделена памет.

## 5.5 Помощни функции

В програмирането, функциите представляват блокове на код, които могат да бъдат извиквани/изпълнявани няколко пъти, за да изнесат определени задачи и да повишават четимостта и поддръжката на кода. Те могат да приемат аргументи и да връщат резултати, като входните стойности могат да бъдат обработени и използвани за решаване на задачите в тялото на функцията.

Използването им помага да се предотврати дублирането на код, като повтарящите се задачи могат да бъдат извънземени в отделни функции и да бъдат използвани няколко пъти в кода на програмата. Те са от съществено значение за кода на програмата и са един от основните елементи на практичното програмиране.

В това приложение, повечето функции могат да бъдат намерени в двата файла описани по-надолу. Задачите, които са свързани с форматиране и извличане на информация за дати са отделени в свой файл, за по-лесна четимост.

### 5.5.1 Функции с време - /flight\_scanner/date\_utils.py

#### 5.5.1.1 Възможни дати за пътувания - get\_travel\_dates

```
def get_travel_dates(start_day, end_day, num_months=1):
    """
    Generates a list of travel dates within the specified number of months.

    Parameters
    -----
    start_day : str
        The starting day of the week for travel (e.g., 'Monday').
    end_day : str
        The ending day of the week for travel (e.g., 'Friday').
    num_months : int, optional
        The number of months within which to generate travel dates (default is 1).

    Returns
    -----
    travel_dates_list : list
        A list of duplicated dictionaries, each containing 'Start' and 'End' keys with travel
    dates.
    """
    start_day = get_weekday_index(start_day)
    end_day = get_weekday_index(end_day)
    travel_dates_list = []
    today = datetime.now()

    current_day_of_week = today.weekday()
    days_until_start_day = (start_day - current_day_of_week + 7) % 7
    next_weekday = today + timedelta(days=days_until_start_day)

    while next_weekday + timedelta(days=end_day) <= today + relativedelta(
        months=+num_months):
        months+=num_months:
        if start_day >= end_day:
            week_set = {
                'End': (next_weekday + timedelta(days=end_day) -
                    timedelta(start_day - 7)).strftime('%a, %b %d')}
        else:
            week_set = {'End': (next_weekday + timedelta(days=end_day))
                .strftime('%a, %b %d')}
            week_set['Start'] = next_weekday.strftime('%a, %b %d')
            travel_dates_list.append(week_set)
            travel_dates_list.append(week_set)
            next_weekday += timedelta(days=7)
    return travel_dates_list
```

Тази функция генерира списък от дати за пътуване в рамките на определен брой месеци. Тя приема 3 аргумента:

- start\_day: Началният ден от седмицата за пътуване (например 'Понеделник').
- end\_day: Крайният ден от седмицата за пътуване (например 'Петък').

- `num_months`: Броят месеци, в рамките на които да се генерират дните за пътуване (по подразбиране е 1).

На базата на тези входни данни функцията генерира списък от дублирани речници, като всеки речник съдържа ключовете 'Start' и 'End' за датите на пътуване.

Тя първо изчислява броя дни до следващото срещане на указания начален ден от седмицата, който е спрямо текущата дата. След това функцията изчислява датата на следващото срещане на крайния ден от седмицата, въз основа на `start_day` и `end_day` стойностите. Тези стойности се използват за генериране на нов речник с 'Start' и 'End' датите, който се добавя два пъти към списъка `travel_dates_list`. Това се повтаря за всеки един от седмиците в рамките на определения брой месеци.

Кодът използва различни функции на Python `datetime`, като `datetime.now()`, `timedelta()` и `relativedelta()`, за да извърши тези изчисления и да генерира датите за пътуване.

Пример:

- Входни данни:

```
start_day = 'Monday'
end_day = 'Friday'
num_months = 1
```

- Изходни данни:

```
travel_dates_list = [{'End': 'Fri, May 24', 'Start': 'Mon, May 20'},
                     {'End': 'Fri, May 24', 'Start': 'Mon, May 20'},
                     {'End': 'Fri, May 31', 'Start': 'Mon, May 27'},
                     {'End': 'Fri, May 31', 'Start': 'Mon, May 27'},
                     {'End': 'Fri, Jun 07', 'Start': 'Mon, Jun 03'},
                     {'End': 'Fri, Jun 07', 'Start': 'Mon, Jun 03'},
                     {'End': 'Fri, Jun 14', 'Start': 'Mon, Jun 10'},
                     {'End': 'Fri, Jun 14', 'Start': 'Mon, Jun 10'}]
```

#### 5.5.1.2 Индекс на ден от седмицата - `get_weekday_index`

```
def get_weekday_index(weekday):
    """
    Returns the index of the given weekday.

    Parameters
    -----
    weekday : str
        The name of the weekday (e.g., 'Monday').

    Returns
```

```
-----
index : int
    The index of the weekday (0 for Monday, 6 for Sunday).
"""
for index, day in enumerate(['Monday', 'Tuesday', 'Wednesday',
                             'Thursday', 'Friday', 'Saturday', 'Sunday']):
    if day == weekday:
        return index
```

Тази функция връща индекса на дадения ден от седмицата. Тя приема един аргумент:

- `weekday`: Името на деня от седмицата (например 'Понеделник').

В зависимост от името на деня от седмицата, функцията връща съответния индекс: 0 за понеделник, 1 за вторник, 2 за сряда, 3 за четвъртък, 4 за петък, 5 за събота и 6 за неделя.

Функцията използва `enumerate()` функцията на Python, за да достъпи всеки един елемент на списъка с дните от седмицата и за да върне индекса на избрания ден.

Пример:

- Входни данни:

```
weekday = 'Monday'
```

- Изходни данни:

```
index = 0
```

### 5.5.1.3 Форматиране на време - `format_datetime_to_textdate_and_time`

```
def format_datetime_to_textdate_and_time(dt):
    """
    Formats a datetime object to a string in the given format.

    Parameters
    -----
    dt : datetime
        Datetime.

    Returns
    -----
    dt : str
        The formatted date and time string.
    """
    dt = dt.strftime("%d %b %Y, %H:%Mh")
    return dt
```



Тази функция форматира обект от тип `datetime` към текстови низ в зададения формат.

Тя приема един аргумент:

- `dt`: обект от тип `datetime`.

В тялото на функцията, `strftime()` методът се използва, за да зададе формата на текстовия низ. Той форматира обекта `dt` към `"%d %b %Y, %H:%Mh"` вид, който съдържа деня от месеца, месеца в стринг формат, година, час и минута в 24-часов формат, и добавя `"h"` като финален символ.

Накрая, връща текстовия низ, форматиран в желанния вид.

Пример:

- Входни данни:

```
dt = datetime(2024, 5, 25, 15, 30)
```

- Изходни данни:

```
dt = "25 May 2024, 15:30h"
```

#### 5.5.1.4 Форматиране на дата - `convert_date_format`

```
def convert_date_format(flight_date): # "Mon, Sep 2"
    """
    Converts a flight date string to ensure the day part is two digits.

    Parameters
    -----
    flight_date : str
        The flight date string (e.g., 'Mon, Sep 2').

    Returns
    -----
    flight_date : str
        The converted flight date string (e.g., 'Mon, Sep 02').
    """
    flight_date = flight_date.split(' ')
    if len(flight_date[-1]) == 1:
        flight_date[-1] = '0' + flight_date[-1]
    flight_date = ' '.join(flight_date)
    return flight_date
```

Тази функция конвертира формата на дата от полет до изисквания формат. Тя приема един аргумент:

- `flight_date`: низовият формат на датата от полет (например 'Mon, Sep 2').

В тялото на функцията, `split()` методът на Python се използва, за да раздели текстовия низ `flight_date` на две части - на деня от седмицата и на деня от месеца.

След това функцията проверява дължината на последния елемент от разделения низ, който е деня от месеца. Ако дължината на този елемент е 1, функцията добавя 0 преди деня, за да го направи с две цифри. По този начин, датата има формат в стил "Mon, Sep 02", а не "Mon, Sep 2".

Накрая, функцията слива двата разделени елемента чрез интервал и връща резултата в изисквания формат.

Пример:

- Входни данни:

```
dt = 'Mon, Sep 2'
```

- Изходни данни:

```
dt = 'Mon, Sep 02'
```

#### 5.5.1.5 Дата на полет - `get_flight_date`

```
def get_flight_date(flight_info):  
    """  
    Extracts and returns the flight date from the flight info string.  
  
    Parameters  
    -----  
    flight_info : str  
        The string containing details about the flight.  
  
    Returns  
    -----  
    current_date : datetime  
        The flight date as a datetime object.  
    """  
    flight_date_string = re.search(  
        r'(?:(Mon|Tue|Wed|Thu|Fri|Sat|Sun), ' +  
        r'(?:(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) \d+',  
        flight_info).group() # "Mon, Sep 2"  
    flight_date_string = convert_date_format(flight_date_string)  
    flight_date = datetime.strptime(flight_date_string, '%a, %b %d')  
    current_year = datetime.now().year  
    while True:  
        current_date = datetime(current_year, flight_date.month, flight_date.day)  
        if current_date.strftime("%a, %b %d") == flight_date_string:  
            return current_date  
        current_year += 1
```

Тази функция извлича и връща датата на полета от низа с информация за полета.

Тя приема един аргумент:

- `flight_info`: низ, съдържащ информация за полета.

В тялото на функцията се използва регулярен израз, за да се намери низовият формат на датата на полета в низа `flight_info`. Форматът на текстовия низ на датата се проверява и реформатира, използвайки функцията `convert_date_format()`.

След това се използва `strptime()` метода, който конвертира текстовия низ в обект от тип `datetime`.

С помощта на `today()`, функцията взема текущата година и прилагайки `while` цикъл търси датата на полета за текущата година и я връща като обект `datetime`, ако е намерена. Ако не е намерена, годината се увеличава с 1 и се прави ново търсене, докато се намери съвпадение.

Накрая, функцията връща датата на полета на обект от тип `datetime`.

Пример:

- Входни данни:

```
flight_info = 'Mon, Sep 2\u202fPM - \u202fPM\u202fRyanairOperated by  
Ryanair Sun\u202fhr 45 min\u202fSOF-CIA\u202fNonstop\u202fkg CO2e\u202fAvg emissions'
```

- Изходни данни:

```
flight_info = datetime(2024, 9, 2, 0, 0)
```

## 5.5.2 Функции за улеснение - /flight\_scanner/intepreters.py

### 5.5.2.1 Натройка на невидим драйвер - driver\_setup\_headless

```
def driver_setup_headless(url):  
    """  
    Sets up and initializes a Selenium WebDriver for Chrome with specific options.  
  
    Parameters  
    -----  
    url : str  
        The URL to open with the WebDriver.  
  
    Returns  
    -----  
    WebDriver  
        A configured instance of Selenium WebDriver for Chrome.  
    """  
    chrome_options = webdriver.ChromeOptions()  
    chrome_options.add_argument("--window-size=1920,1080")  
    chrome_options.add_argument("--disable-extensions")  
    chrome_options.add_argument("--proxy-server='direct://'")  
    chrome_options.add_argument("--proxy-bypass-list=*")  
    chrome_options.add_argument("--start-maximized")  
    chrome_options.add_argument('--headless')  
    chrome_options.add_argument('--disable-gpu')  
    chrome_options.add_argument('--disable-dev-shm-usage')  
    chrome_options.add_argument('--no-sandbox')  
    chrome_options.add_argument('--ignore-certificate-errors')  
    chrome_options.add_experimental_option("detach", True)  
    driver = webdriver.Chrome(options=chrome_options)  
  
    driver.maximize_window()  
    driver.get(url)  
    return driver
```

Тази функция настройва и инициализира Selenium WebDriver за Chrome с определени опции, при работа в режим без графичен интерфейс (headless mode).

Тя приема един аргумент:

- url: URL адресът, който трябва да бъде зареден с WebDriver.

Функцията първоначално настройва опциите на Chrome, които ще бъдат използвани от WebDriver. Това включва подаване на редица аргументи чрез add\_argument() метода, който променя различни настройки на Chrome:

- "--window-size=1920,1080": задава размерите на прозореца на брауъра.
- "--disable-extensions": деактивира всички инсталирани разширения в брауъра.
- "--proxy-server='direct://'": настройва брауъра да обходи прокси сървърите и да ползва директна връзка.
- "--proxy-bypass-list=\*": определя така, че Chrome да не се свързва към прокси.
- "--start-maximized": зарежда брауъра на максимален размер, когато се стартира.

- "--headless": разрешава изпълнението на Chrome в headless mode.
- "--disable-gpu": изключва използването на GPU от браузъра.
- "--disable-dev-shm-usage": управлява това, как Chrome се свързва с операционната система, като ограничава размера на тим аут.
- "--no-sandbox": деактивира sandboxing-a в Chrome.
- "--ignore-certificate-errors": незабележимите грешки при проверката на SSL сертификат.

След това функцията инициализира WebDriver с настройките и с използването на методите `maximize_window()` и `get()` управлява зареждането на URL адреса `url` в браузъра.

Накрая, функцията връща инстанция на WebDriver за Chrome, който е конфигуриран и готов за употреба в headless режим.

Пример:

- Входни данни:

```
url = "https://www.google.com"
```

- Изходни данни:

```
driver = WebDriver()
```

### 5.5.2.2 Xpath на уеб елемент - find\_my\_element\_by\_xpath

```
def find_my_element_by_xpath(driver, xpath, retries=2):
    """
    Finds an element on a web page by its XPath.

    Parameters
    -----
    driver : WebDriver
        The Selenium WebDriver instance to use for finding the element.
    xpath : str
        The XPath of the element to find.
    retries : int, optional
        The number of retries to attempt if the element is not found. Defaults to 2.

    Returns
    -----
    WebElement
        The found web element.

    Raises
    -----
    Exception
        If the element is not found after the specified number of retries.
    """
    try:
        my_element = WebDriverWait(driver,
10).until(EC.visibility_of_element_located((By.XPATH, xpath)))
    except Exception:
        if retries > 0:
            driver.refresh()
            return find_my_element_by_xpath(driver, xpath, retries=retries - 1)
        raise Exception("Element not found after retries")
    return my_element
```

Тази функция търси елемент на уеб страница по зададен път на търсенето (XPath) с използване на Selenium WebDriver. Тя приема три аргумента:

- driver: екземпляр на WebDriver, който ще се използва за търсене на елемента.
- xpath: XPath заявката за търсене на конкретния елемент.
- retries: Брой опити, които да се направят при неуспешно търсене на елемента, преди да се хвърли изключение. По подразбиране е зададено да бъдат направени 2 опита.

Функцията използва WebDriverWait() метода на Selenium, за да изчака намирането на елемент на страницата. Методът visibility\_of\_element\_located() се използва за проверка и връщане на елемента, ако е видим на страницата. Първоначално, WebDriverWait() функцията изчаква 10 секунди, за да се върне намерения елемент.

Ако той не бъде намерен в указания период от време, функцията извиква себе си (рекурсия), за да направи нов опит за търсене на елемента на страницата. Също така тя

проверява броя на опитите (определени от аргумента `retries`) и намалява стойността му при всяко следващо извикване, докато не стигне до нула.

Ако елементът не може да бъде намерен след максималния брой опити, се хвърля изключението "Element not found after retries".

Накрая, функцията връща намерения елемент на страницата като обект от клас `WebElement`.

Пример:

- Входни данни:

```
driver = WebDriver()
xpath = "//div[@id='test']"
retries = 2
```

- Изходни данни:

```
my_element = WebElement()
```

### 5.5.2.3 Сортиране на полетите в интернет - `sort_flights_by_price_driver`

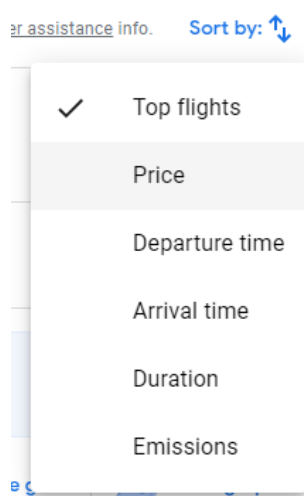
```
def sort_flights_by_price_driver(driver):
    """
    Sorts the flights by price on the web page using the WebDriver.

    Parameters
    -----
    driver : WebDriver
        The Selenium WebDriver instance to use for interacting with the elements.
    """
    try:
        sort_button = WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH, '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-
wiz/div[2]/div[2]/div[3]/div/div/div/div[1]/div/button')
            )
        )
        sort_by_price_xpath = '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-
wiz/div[2]/div[2]/div[3]/div/div/div/div[2]/div/ul/li[2]'
    except Exception:
        sort_button = WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located(
                (By.XPATH, '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-
wiz/div[2]/div[2]/div[2]/div/div[2]/div[1]/div/div/div/div[1]/div/button')
            )
        )
        sort_by_price_xpath = '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-
wiz/div[2]/div[2]/div[2]/div/div/div/div[2]/div/ul/li[2]'
    finally:
        sort_button.click()
        time.sleep(2)
        sort_by_price_button = find_my_element_by_xpath(driver, sort_by_price_xpath)
        sort_by_price_button.click()
        time.sleep(2)
```

Тази функция сортира полетите по цена на уеб страницата, използвайки Selenium WebDriver. Тя приема един аргумент:

- driver: екземпляр на WebDriver, който се използва за взаимодействие с елементите на страницата.

Функцията използва WebDriverWait() метода на Selenium, за да изчака появата на бутон на страницата, който служи за сортиране. Методът visibility\_of\_element\_located() се използва за проверка и връщане на елемента, ако е видим на страницата.



Фигура 35

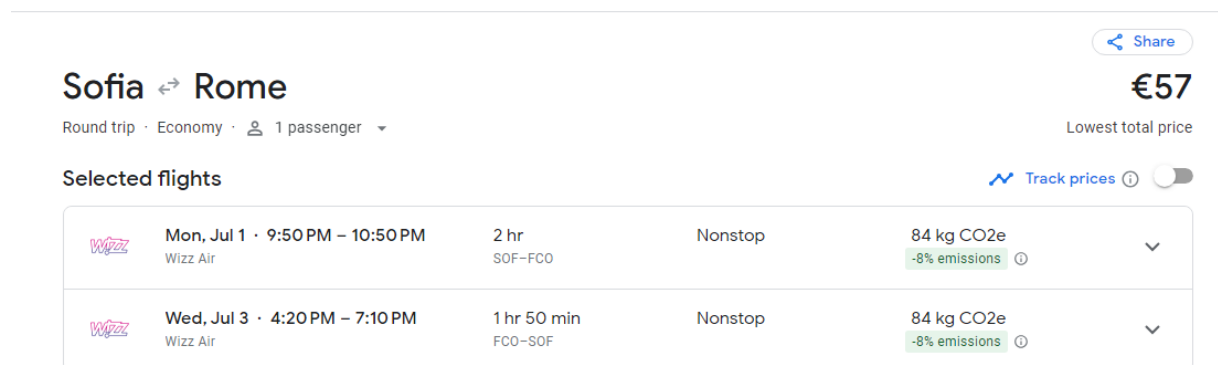
Драйверът клика върху "Sort by" бутона и чака да се зареди менюто с различните варианти за подреждане. Избира се елементът "Price", което води до сортиране на полетите по възходяща цена. Не връща резултат.



#### 5.5.2.4 Отваряне на линк за закупуване на полет - open\_link

```
def open_link(url):  
    """  
    Opens a given URL using the Selenium WebDriver setup.  
  
    Parameters  
    -----  
    url : str  
        The URL to open.  
    """  
    chrome_options = webdriver.ChromeOptions()  
    chrome_options.add_argument("--window-size=1920,1080")  
    chrome_options.add_argument("--disable-extensions")  
    chrome_options.add_argument("--proxy-server='direct://'")  
    chrome_options.add_argument("--proxy-bypass-list=*")  
    chrome_options.add_argument("--start-maximized")  
    chrome_options.add_argument('--disable-gpu')  
    chrome_options.add_argument('--disable-dev-shm-usage')  
    chrome_options.add_argument('--no-sandbox')  
    chrome_options.add_argument('--ignore-certificate-errors')  
    chrome_options.add_experimental_option("detach", True)  
    driver = webdriver.Chrome(options=chrome_options)  
  
    print(f"Opening link: {url}")  
    driver.maximize_window()  
    driver.get(url)  
    return driver
```

Тази функция отваря даден URL адрес, използвайки настройките на Selenium WebDriver за Chrome. Тя е подобна на *driver\_setup\_headless*, но без аргументът '--headless'. По този начин уеб браузърът е видим за потребителя.



Фигура 36

Функцията приема един аргумент:

- url: адресът на URL-то към полета откъдето може да бъде закупен билета.

### 5.5.2.5 Xpath на елемент от лист - get\_xpath\_for\_li

```
def get_xpath_for_li(set_num, driver):  
    """  
    Retrieves the XPath for a specific flight element on a web page.  
  
    Parameters  
    -----  
    set_num : int  
        The set number to determine the XPath.  
    driver : WebDriver  
        The Selenium WebDriver instance to use for finding the elements.  
  
    Returns  
    -----  
    str  
        The XPath expression for the flight element.  
  
    Raises  
    -----  
    Exception  
        If no flights are found.  
    """  
    exit_code = 1  
    list_flights = find_my_element_by_xpath(driver, '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-wiz/div[2]/div[2]/div[4]/ul')  
    count_flights = len(list_flights.find_elements(By.TAG_NAME, 'li'))  
    if count_flights == 0:  
        driver.quit()  
        return exit_code  
    list_index = set_num % 2 + 1  
    xpath = f'/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-wiz/div[2]/div[2]/div[4]/ul/li[{list_index}]'  
    return xpath
```

Тази функция извлича XPath израза за определен елемент на полет от лист уеб страница. Тя приема два аргумента:

- set\_num: Номер на елемента, който искаме да намерим на страницата.
- driver: екземпляр на WebDriver, който ще се използва за взаимодействие с елементите на страницата.

Функцията търси листът ul, който съдържа информацията за всички полети. След това преброява всички елементи в него. Ако листът е празен, се освобождава паметта от WebDriver-a и връща код на изход 1. При пълен лист, се взима първият или вторият li елемент, в зависимост от това дали set\_num е четно или нечетно. Накрая се връща XPath към този елемент.

Пример:

- Входни данни:

```
set_num = 3  
driver = WebDriver()
```

- Изходни данни:

```
my_element = '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-wiz/div[2]/div[2]/div[4]/ul/li[2]'
```

### 5.5.2.6 Лист със сортирани полети - get\_sorted\_list\_flights

```
def get_sorted_list_flights(list_flights):  
    """  
    Sorts the list of flights by price and returns the top 8 flights.  
  
    Parameters  
    -----  
    list_flights : list  
        The list of flights to sort.  
  
    Returns  
    -----  
    list  
        A sorted list of the top 8 flights by price.  
    """  
    sorted_flights = sorted(list_flights, key=lambda flight: int(flight.price.split()[1]))[:8]  
    return sorted_flights
```

Тази функция сортира списък с полети по цена и връща първите 8 полета от сортирания списък. Тя приема един аргумент:

- `list_flights`: Списък с обекти от тип `Travel`. Това са извлечените полети от интернет.

Използва се ламбда функция, която преобразува цената на всеки авиобилет до цяло число (без валутата), за да може сравнението да се извърши според цената. В резултат, функцията връща нов списък с първите 8 най-евтини полета.

Пример:

- Входни данни:

`travel1`, `travel2` и `travel3` са обекти от тип `Travel()`, с различни данни, като съответно и цените на билетите са разнообразни. В посоченият пример са закоментирани цените на полетите.

```
list_flights = [  
    travel1, #price = 'BGN 202'  
    travel2, #price = 'BGN 9'  
    travel3  #price = 'BGN 70'  
]
```

- Изходни данни:

```
list flights = [  
    travel2, #price = 'BGN 9'  
    travel3, #price = 'BGN 70'  
    travel1  #price = 'BGN 202'  
]
```

## 5.6 Тестване

Тестовите в програмирането са метод за проверка на коректността, функционалността и качеството на софтуерни приложения. Те служат за уверяване, че програмните системи извършват очакваните операции и за отстраняване на грешки. Те могат да бъдат написани както на високо, така и на ниско ниво на кодовата база и могат да бъдат прилагани както на по-малки, така и на по-големи проекти. В програмирането се използват много различни видове тестове, като например юнит тестове, функционални тестове, тестове за натоварване и др. Като цяло, те са важен инструмент за увеличаване на качеството на програмния код и за избягване на грешки, които могат да доведат до непредвидимо поведение на софтуерната система при употреба.

### 5.6.1 Модулни тестове

Модулното тестване е метод за тестване на отделните компоненти (функции, класове и др.) на софтуерен продукт, като се използват автоматизирани тестове, за да се провери дали входните данни са обработени правилно и изходните резултати са коректни. Това помага да се гарантира качеството на софтуерните продукти.

В настоящата дипломна работа е изтестван почти целият код, като има няколко вида тестване:

#### 5.6.1.1 Модулно тестване и функционален код

```
def test_get_weekday_index():
    """
    Tests 'get_weekday_index' for all weekdays.
    """
    assert get_weekday_index('Monday') == 0
    assert get_weekday_index('Tuesday') == 1
    assert get_weekday_index('Wednesday') == 2
    assert get_weekday_index('Thursday') == 3
    assert get_weekday_index('Friday') == 4
    assert get_weekday_index('Saturday') == 5
    assert get_weekday_index('Sunday') == 6
```

Това е пример за модулно тестване, като тестовите случаи проверяват отделните стойности на дните от седмицата, за да се уверим, че функцията "get\_weekday\_index" работи правилно и връща правилния индекс на деня от седмицата за всяка от стойностите на дните, като започва от 0 за "Понеделник" до 6 за "Неделя". Също така, този тест е пример за тестване на функционалния код, когато знаем очаквания изход за предоставен вход.

### 5.6.1.2 Модулно тестване с мок обекти

```
@freeze_time("2024-05-20")
def test_get_travel_dates_ending_in_next_week():
    """
    Tests 'get_travel_dates' from Sunday to Wednesday.
    """
    start_day = 'Sunday'
    end_day = 'Wednesday'
    expected_output = [
        {'End': 'Wed, May 29', 'Start': 'Sun, May 26'},
        {'End': 'Wed, May 29', 'Start': 'Sun, May 26'},
        {'End': 'Wed, Jun 05', 'Start': 'Sun, Jun 02'},
        {'End': 'Wed, Jun 05', 'Start': 'Sun, Jun 02'},
        {'End': 'Wed, Jun 12', 'Start': 'Sun, Jun 09'},
        {'End': 'Wed, Jun 12', 'Start': 'Sun, Jun 09'},
        {'End': 'Wed, Jun 19', 'Start': 'Sun, Jun 16'},
        {'End': 'Wed, Jun 19', 'Start': 'Sun, Jun 16'}]
    assert get_travel_dates(start_day, end_day) == expected_output
```

Това е пример за модулно тестване с използване на мок-обекти (mock objects). Тестът използва декоратора `@freeze_time`, който ни позволява да променим текущата дата и час, за да тестваме функцията `"get_travel_dates"` за определен период от време. По този начин се пренебрегва текущата дата и час при изпълнението на функцията. Тестът проверява, дали в рамките на един месец, функцията `"get_travel_dates"` връща правилно всички дати, които започват в понеделник и приключват в петък. Този тест е пример за тестване на функциите, които зависят от външни фактори, като текущата дата и час, чрез “замръзване” на времето.

```
@patch('flight_scanner.interpreters.find_my_element_by_xpath')
def test_get_xpath_for_li_found(mock_find_my_element_by_xpath):
    """
    Test get_xpath_for_li when flights are found.
    """
    set_num = 1
    driver = MagicMock()
    mock_list_flights = MagicMock()
    mock_list_flights.find_elements.return_value = ['flight1', 'flight2']
    mock_find_my_element_by_xpath.return_value = mock_list_flights

    xpath = get_xpath_for_li(set_num, driver)

    expected_xpath = '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-wiz/div[2]/div[2]/div[4]/ul/li[2]'
    assert xpath == expected_xpath
    mock_find_my_element_by_xpath.assert_called_once_with(
        driver, '/html/body/c-wiz[2]/div/div[2]/c-wiz/div[1]/c-wiz/div[2]/div[2]/div[4]/ul'
    )
```

Това е друг пример за модулно тестване с използване на мок-обекти, което обаче тества функции, които използват външни зависимости. Тестът използва декоратора `@patch`, за да мокне функцията `"find_my_element_by_xpath"` в модула `"interpreters"`. Този тест проверява функцията `"get_xpath_for_li"`, която използва мокнатата функция `"find_my_element_by_xpath"` за извличане на елементи от уеб страницата. Тества се

дали тази функция връща правилния XPath път за елементът от лист с полети. Този тест е пример за тестване на различни модули, които работят заедно и използват външна зависимост.

```
@pytest.fixture
def scanned_flights_window(qtbot):
    """Fixture to create the ScannedFlightsWindow instance."""
    test_scanned_result = []
    test_input_data = {'flight_from': 'Sofia', 'flight_to': 'Rome'}
    window = ScannedFlightsWindow(test_scanned_result, test_input_data)
    qtbot.addWidget(window)
    return window

def test_scanned_flights_ui_state(scanned_flights_window):
    """Test the initial state of the UI elements in ScannedFlightsWindow."""
    assert scanned_flights_window.windowTitle() == 'Cheapest flights List'
    assert isinstance(scanned_flights_window.label_round_trip, QLabel)
    assert scanned_flights_window.label_round_trip.text() == "Round trip: Sofia - Rome"
```

Също така, за някои тестове е използван декораторът `@pytest.fixture` – това са функции които стартират преди изпълнението на всеки тест, за който е зададен. В случаят се използва за да тества началното състояние на графичният интерфейс `ScannedFlightWindow`.

## 5.7 Покритие на тестовите (coverage)

Това е метрика, която показва процента от кода на приложението, който е покрит от тестовите. Тя представлява общия процент на кода, който се изпълнява успешно от тестовите. Покритието им предоставя информация за това, какво количество от кода е тествано и колко от него все още не е. То е много важно за качеството на приложението, тъй като тестовите с високо покритие на кода обикновено са по-стабилни и по-малко склонни към проблеми. Тестовите, които покриват по-голяма част от кода, могат да открият недостатъци и грешки в приложението, които иначе може да бъдат пропуснати.

За настоящата дипломна работа е инсталирана библиотеката `coverage.py`. Тя проверява покритието на Python проекти. За генерирането на репорта трябва да се въведе следната команда в терминала:

`“coverage run -m pytest .”`

Това ще инициализира всички тестове с coverage, където инструментът ще проследи изпълнението на кода и ще създаде репорт.

Name	Stmts	Miss	Cover
-----	-----	-----	-----
FlightScanner\flight_scanner\__init__.py	0	0	100%
FlightScanner\flight_scanner\date_utils.py	43	1	98%
FlightScanner\flight_scanner\flight.py	48	0	100%
FlightScanner\flight_scanner\flight_scanner.py	92	35	62%
FlightScanner\flight_scanner\interpreters.py	72	11	85%
FlightScanner\flight_scanner\threads.py	45	8	82%
FlightScanner\flight_scanner\windows.py	152	33	78%
FlightScanner\tests\__init__.py	0	0	100%
FlightScanner\tests\setup.py	4	0	100%
FlightScanner\tests\test_date_utils.py	36	0	100%
FlightScanner\tests\test_flight.py	55	0	100%
FlightScanner\tests\test_flight_scanner.py	96	0	100%
FlightScanner\tests\test_flights_scanner_window.py	10	0	100%
FlightScanner\tests\test_interpreters.py	76	2	97%
FlightScanner\tests\test_threads.py	57	3	95%
FlightScanner\tests\test_windows.py	35	3	91%
-----	-----	-----	-----
TOTAL	821	96	88%

Фигура 37

Отчетът на този проект включва 16 файла, от които 88% са покрити.

Също така в проекта може да се намери и същият репорт записан в coverage.xml.

## 5.8 Статичен анализ на код (pylint)

Pylint е инструмент на Python, който използва статичен анализ на кода за проверка на съответствието на основни правила и стандарти, свързани с кодирането.

Настоящата дипломна работа е тествана и е с максимален резултат.

```
-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Фигура 38

## Глава 6

### Ръководство за работа

Ръководството за потребителя представлява кратко описание и изглед на приложението. То е създадено с цел запознаване на потребителя със системата и упътване за работа с нея.

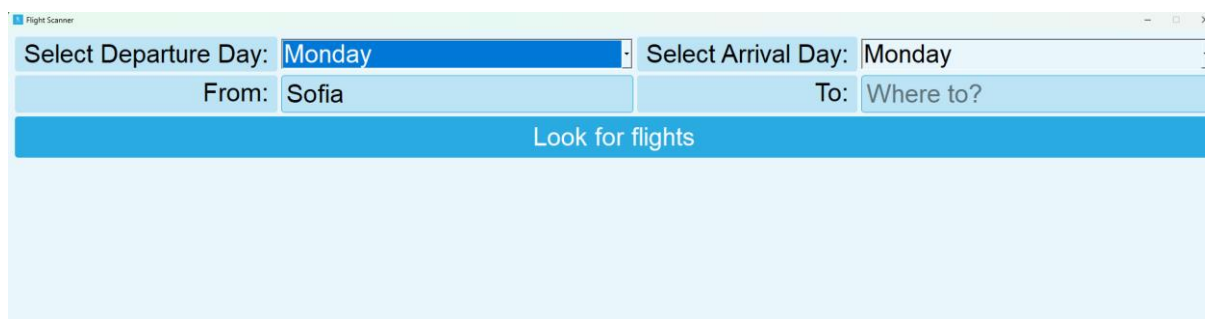
За проектът е използван Github, като система за контрол на версиите. Той може да бъде достъпен на този адрес:

<https://github.com/geri-peikova/FlightScanner>



Фигура 39

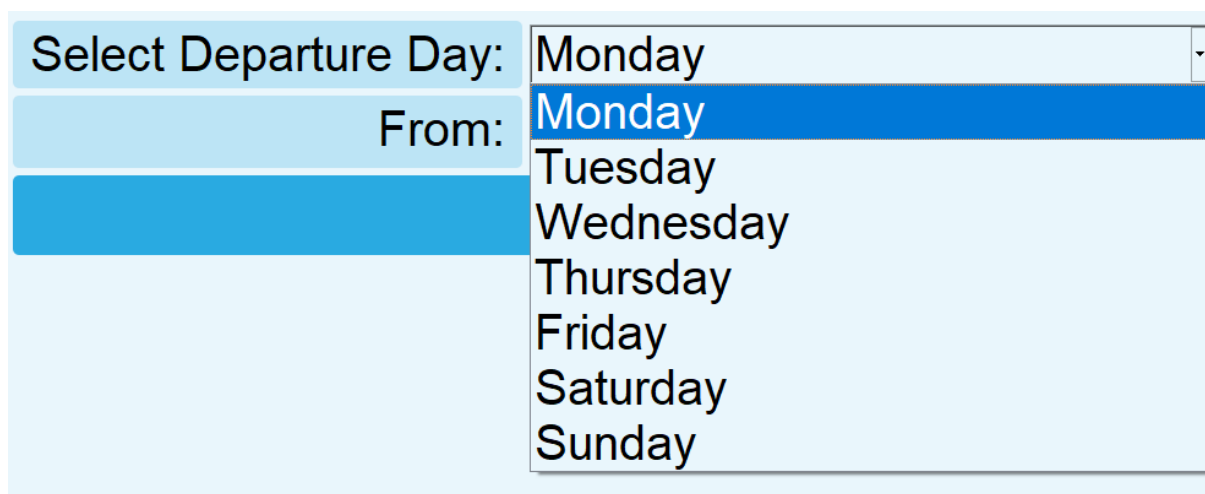
Приложението се нарича FlightScanner, което придава удобство за потребителя, да се ориентира още от името и логото, какво ще се случва в него.

The image shows a screenshot of the Flight Scanner application interface. It features a light blue background with a white border. At the top, there are two dropdown menus: "Select Departure Day:" with "Monday" selected, and "Select Arrival Day:" with "Monday" selected. Below these are two input fields: "From:" with "Sofia" entered, and "To:" with "Where to?" entered. A large blue button with the text "Look for flights" is positioned below the input fields. The entire interface is enclosed in a window frame with a title bar that says "Flight Scanner".

Фигура 40

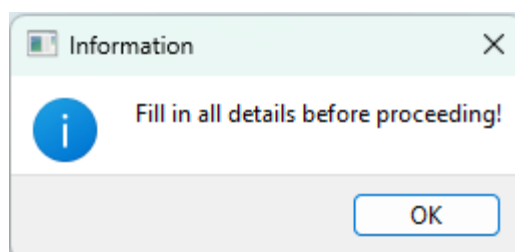


При стартирането на програмата се появява началният прозорец, в който има полета за ръчно въвеждане на информация.

A screenshot of a software interface showing a dropdown menu for selecting a departure day. The label 'Select Departure Day:' is on the left. The dropdown is open, showing a list of days: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday. The 'Monday' option is currently selected and highlighted in blue. Below the dropdown, there is a label 'From:' followed by another dropdown menu that also shows 'Monday' as the selected option.

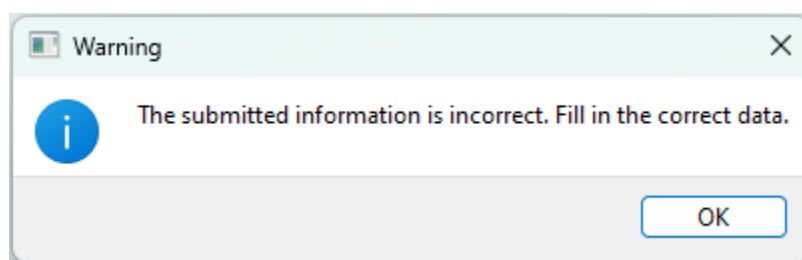
Фигура 41

Потребителят трябва да избере в кой ден от седмицата има възможност да отпътува и съответно, в кой ден да се завърне. Това става, чрез избор на ден от падащото меню. Също така, трябва да се въведат и градовете, от които да замине (има вградена дефолтна стойност ‚Sofia‘) и да пристигне (има помощен текст за допълнително насочване). Задължително трябва да са попълнени всички полета.



Фигура 42

При неизпълнение, и кликане на бутона ‚Look for flights‘, изскача прозорец, който информира, че не приложението не може да продължи нататък, докато не бъде попълнено всичко.



Фигура 43

При грешно въведени данни (несъществуващ град), се появява прозорец с предупреждение, информиращ за неправилната информация и казва на клиентът да попълни наново информацията.



Loading...

Фигура 44

При коректно въведени данни и кликане на бутонът „Look for flights“ се появява анимация, за уведомяване на ползвателят, че приложението изчислява и извършва някакви процеси на заден план.

Round trip: Sofia - Rome			
1   Price: BGN 89	Departure: 16 Sep 2024, 12:55h	Return: 17 Sep 2024, 18:45h	More Data
2   Price: BGN 155	Departure: 16 Sep 2024, 13:25h	Return: 17 Sep 2024, 23:20h	More Data
3   Price: BGN 176	Departure: 02 Sep 2024, 13:25h	Return: 03 Sep 2024, 23:20h	More Data
4   Price: BGN 239	Departure: 09 Sep 2024, 13:25h	Return: 10 Sep 2024, 23:20h	More Data
5   Price: BGN 320	Departure: 16 Sep 2024, 21:50h	Return: 17 Sep 2024, 15:20h	More Data
6   Price: BGN 380	Departure: 09 Sep 2024, 21:50h	Return: 10 Sep 2024, 23:00h	More Data
7   Price: BGN 409	Departure: 02 Sep 2024, 21:50h	Return: 03 Sep 2024, 18:45h	More Data
8   Price: BGN 506	Departure: 09 Sep 2024, 16:05h	Return: 10 Sep 2024, 15:20h	More Data

Фигура 4

Когато се извлече нужната информация за полетите, се визуализира нов прозорец съдържащ информация за начална и крайна точка на полетите, както и списък с 8 полета подредени по цена с информация за дата и час на заминаване и връщане. Също така зад всеки полет седи бутон, който при кликане се отваря уеб страница, откъдето може да бъде закупен съответният билет.

Sofia ↔ Rome

Round trip · Economy · 1 passenger



Share

**BGN 89**  
Lowest total price

Unfortunately, the price you saw on the previous page has changed

Selected flights

Track prices

	Mon, Sep 16 · 12:55 AM - 2:55 PM Wizz Air	2 hr SOF-FCO	Nonstop	84 kg CO2e -8% emissions	▼
	Tue, Sep 17 · 6:45 PM - 8:45 PM Lufthansa	2 hr FCO-SOF	Nonstop	188 kg CO2e +107% emissions	▼

Фигура 46

Приложението може да приключи работа по-всяко време, единствено потребителят трябва да натисне горе в дясно бутонът с „X“.



Фигура 47

## **Глава 7**

### **Заклучение и възможности за бъдещо развитие**

В заключение, разработеното приложение е един удобен и лесен начин за бързо откриване на най-евтините възможности за пътуване. То ще помогне на хората да спестят време и усилия в търсенето на най-добрите предложения за полети и да планират лесно своите пътешествия. Разработката на това приложение е от голяма полза за всички, които искат да пътуват, но имат ограничено време и бюджет, като ги освобождава от стреса на търсене и планиране на пътуванията им.

Приложението има огромни възможности за бъдещо развитие, като подобрене на графичният интерфейс. Това би допринесло с по-добра ефективност, тъй като потребителите, ще имат по-бърз и удобен начин за достъп до полезната информация, както и ще допринесе за тяхното по-добро визуално изживяване, подбуждайки любопитството им за използване на приложението. Друго силно подобрене би било добавянето на опция на повече езици. Това би довело до по-голяма световна популярност и съответно до по-голям брой потребители, тъй като няма да е ограничено, само до хора, които владеят английски език. Също така могат да се добавят неограничено много филтри и опции за търсене и закупуване на билети директно от приложението. Това би направило приложението много търсено, тъй като на едно се намират много възможности.

## Източници

- [1] <https://www.skyscanner.net/> , Skyscanner Ltd 2002 – 2024
- [2] <https://www.esky.bg/> , eSky Ltd
- [3] <https://www.booking.com/> , Booking Holdings Inc.
- [4] <https://www.google.com/travel/flights> , Google
- [5] <https://www.python.org/> , Python Software Foundation 2001 - 2024
- [6] <https://www.jetbrains.com/> , IntelliJ IDEA
- [7] <https://www.selenium.dev/> , Software Freedom Conservancy 2024
- [8] <https://www.youtube.com/playlist?list=PLzMbGfZo4-n40rB1XaJ0ak1bemvlqumQ> , Python Selenium Tutorials, Tech With Tim
- [9] <https://www.tutorialspoint.com/pyqt5/index.htm> , Tutorialspoint
- [10] [https://www.youtube.com/playlist?list=PL3JVwFmb\\_BnRpvOeIh\\_To4YSiebiggyXS](https://www.youtube.com/playlist?list=PL3JVwFmb_BnRpvOeIh_To4YSiebiggyXS) , Python PyQt5 Tutorial Videos, Jie Jenn
- [11] <https://docs.github.com/> , GitHub, Inc. 2024
- [12] <https://en.wikipedia.org/wiki/CSS> , Wikipedia
- [13] <https://www.w3schools.com/css/> , W3School, Refsnes Data 1999-2024
- [14] <https://docs.pytest.org/en/8.2.x/> , holger krekel and pytest-dev team, 2015
- [15] <https://coverage.readthedocs.io/en/7.5.1/> , Ned Batchelder, 2009–2024
- [16] <https://docs.pylint.org/> , Logilab and MohriCorporation, 2013-2014
- [17] <https://stackoverflow.co/> , Stack Exchange Inc, 2024
- [18] <https://realpython.com/intro-to-python-threading/> , Real Python, 2012-2024
- [19] <https://docs.python.org/3/library/threading.html> , Python Software Foundation, 2001-20024

- [20] <https://www.tutorialspoint.com/selenium/index.htm> , Tutorialspoint 2024
- [21] [https://en.wikipedia.org/wiki/Unit\\_testing](https://en.wikipedia.org/wiki/Unit_testing) , Wikipedia Foundation, Inc.
- [22] <https://realpython.com/pytest-python-testing/> , Real Python, 2012-2024
- [23] <https://www.opentext.com/what-is/functional-testing> , Open Text Corporation, 2024
- [24] <https://docs.python.org/3/library/unittest.mock.html> , Python Software Foundation,  
2001-20024
- [25] <https://realpython.com/python-mock-library/> , Real Python, 2012-2024