МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційно-пошукова система резервування та придбання авіаквитків»
здобувача групи ІНз-93-1с Геренко Владислав Ігорович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Геренко В.І
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук, к.ф.-м.н Бадалян А.Ю. _____
(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІНз-93-1с Геренко Владислав Ігорович

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються в Інформаційно-пошукових системах авіаквитків 3) Розробка пошукової системи. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	01.09.22-01.10.22	
2	*Огляд технологій, що використовуються в Інформаційно-пошукових системах резервування та придбання авіаквитків*	02.10.22-01.12.22	
3	*Розробка Інформаційно-пошукової системи резервування та придбання авіаквитків*	02.12.22-01.02.23	
4	*Аналіз отриманих результатів*	02.02.23-03.04.23	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	04.04.23-15.04.23	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 52 стр., 11 рис., 1 додаток, 17 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розробці додатку для пошуку авіаквитків, що є дуже актуальним у наші часи, як для подорожей так і по роботі

Об’єкт дослідження — Інформаційні-пошукові системи резервування та придбання авіаквитків

Мета роботи — розробка Інформаційної-пошукової системи резервування та придбання авіаквитків

Методи дослідження — Існуючі Інформаційні-пошукові системи резервування та придбання авіаквитків

Результати — розроблено Інформаційно-пошукова система резервування та придбання авіаквитків, яка парсить дані з інтернету та виводить їх до вікна додатку. Проведено тестування розробки на реальних запитах та отримано дані

ІНФОРМАЦІЙНА ПОШУКОВА-СИСТЕМА, ПОШУК КВИТКІВ,
АВІАБІЛЕТИ, PYTON, TKINTER, SELENIUM

ЗМІСТ

ВСТУП

У сучасному світі, де межі стають все менш помітними, а зв'язок із світом набуває дедалі більшого значення, подорожі стають невід'ємною частиною життя для багатьох людей. Відкритість культур, зближення народів і розвиток глобальної економіки вимагають швидкого та зручного способу резервування та придбання авіаквитків. Від цього залежить успішне планування подорожей, зустрічі з близькими та колегами, розвиток бізнесу та самореалізація. Тому створення інформаційно-пошукової системи резервування та придбання авіаквитків є актуальною проблемою нашого часу.

Актуальність даної бакалаврської роботи полягає в тому, що існуючі методи пошуку та бронювання авіаквитків нерідко є складними, неефективними та затратними для користувачів. Значна кількість туристичних агентств, авіакомпаній та онлайн-платформ пропонують свої послуги, але відсутність єдиного централізованого ресурсу, який би об'єднував усі можливості, робить процес вибору та придбання авіаквитків заплутаним та часомозатратним.

Тому метою цієї роботи є розробка та реалізація інформаційно-пошукової системи резервування та придбання авіаквитків, яка буде забезпечувати користувачам широкий доступ до актуальної інформації про рейси, ціни та наявність місць на різних авіалініях та платформах. Готова система буде надійним помічником у плануванні подорожей, забезпечуючи зручність, ефективність та економію часу для користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

У сучасному світі, де зростає мобільність та зв'язаність країн, авіаперельоти стають необхідними для багатьох людей. Це призводить до зростання попиту на авіаквитки та потреби в зручному та ефективному процесі резервування та придбання[1-4].

З перших кроків у дослідженні стає очевидним, що зростання обсягу авіаційних перевезень є одним з ключових факторів, що вказують на актуальність цієї проблеми. Останнім десятиліттям спостерігається значне збільшення популярності авіаподорожей, як для бізнес-поїздок, так і для відпочинку. Це призводить до зростання попиту на авіаквитки та потреби в удосконаленні процесу їх резервування та придбання. Крім того, глобалізація та зростання туристичного сектору є ще одним аспектом, що підкреслює актуальність даної проблеми. Все більше людей подорожують міжнародно, що створює потребу в ефективних та зручних способах резервування авіаквитків. Розширення географії авіаперевезень та з'явлення нових авіаліній також вносять свій вклад у актуальність цієї проблеми, оскільки вимагають актуальної та широкої інформації для користувачів. Конкуренція на ринку авіаперевезень є ще одним аспектом, що підкреслює актуальність інформаційно-пошукової системи резервування та придбання авіаквитків. Значна кількість авіакомпаній, туристичних агентств та онлайн-платформ пропонують послуги у цій сфері, що призводить до підвищеної конкуренції. Розробка ефективної системи, що надає користувачам зручний та швидкий доступ до найкращих пропозицій, є ключовим фактором у виграші на ринку[3].

Сучасні подорожуючі вимагають зручності, швидкості та доступності у процесі резервування та придбання авіаквитків. Розвиток технологій та доступ до Інтернету створюють можливості для розробки мобільних додатків та онлайн-платформ, що спрощують процес пошуку та бронювання авіаквитків. Такі системи дозволяють користувачам швидко знайти необхідну інформацію

про рейси, ціни та наявність місць, що робить їх надзвичайно зручними та популярними. Надійність та безпека також дуже важливо. Користувачі очікують, що їхні дані та фінансова інформація будуть захищені під час процесу резервування та придбання квитків. Розробка системи з високими стандартами безпеки є критично важливою для забезпечення довіри користувачів та успішного функціонування системи.

Впровадження електронних квитків стає все поширенішою практикою в авіаційній галузі. Заміна традиційного паперового квитка на електронний формат вимагає розробки інформаційно-пошукової системи, яка забезпечить зручність та безпеку придбання та збереження електронних квитків[1-3].

1.2 Аналіз аналогів та визначення наявних проблем

Розглянемо аналоги, які зараз існують на ринку інформаційно-пошукових систем пошуку квитків. (табл. 1.1)

Таблиця 1.1 – Аналоги інформаційно-пошукових систем пошуку квитків

Аналоги	Головні плюси	Мінуси
Expedia	- Велика база авіаквитків і готелів	- Обмежені можливості фільтрації результатів
Kayak	- Зручний інтерфейс	- Відсутність можливості безпосереднього зв'язку з авіакомпаніями
Skyscanner	- Порівняння цін на різних платформах	- Не завжди точна інформація про наявність місць та розклад рейсів
Google Flights	- Широкий вибір авіаквитків від різних постачальників	- Відсутність можливості здійснювати безпосереднє бронювання
Skiplagged	- Зручний пошук "прихованих" маршрутів	- Обмежена функціональність для
Momondo	- Пошук найкращих пропозицій	- Обмежена підтримка регіональних рейсів
Hopper	- Прогнозування цін на авіаквитки	- Обмежена база авіакомпаній та пунктів призначення

Аналізуючи наведену таблицю, можна зробити кілька висновків щодо аналогів інформаційно-пошукової системи резервування та придбання авіаквитків. По-перше, багато аналогів, таких як Kayak, Skyscanner і Google Flights, мають переваги у вигляді широкого вибору авіаквитків, зручного інтерфейсу та можливості порівняння цін. Однак, у них є свої недоліки, такі як обмеженість функцій безпосереднього зв'язку з авіакомпаніями, не точна інформація про наявність місць та обмежена можливість безпосереднього бронювання. Expedia та Momondo, з іншого боку, пропонують велику базу авіаквитків і готелів, але мають свої обмеження, такі як обмежені можливості фільтрації результатів і обмежена підтримка регіональних рейсів. Серед інших аналогів, Skiplagged пропонує зручний пошук "прихованих" маршрутів, а Hopper зосереджений на прогнозуванні цін на авіаквитки. Однак, вони також мають свої обмеження, зокрема обмежена функціональність для складних перельотів та обмежена база авіакомпаній та пунктів призначення[4-6].

1.3 Постановка задачі

Метою даного проекту є розробка інформаційно-пошукової системи, яка надасть користувачам зручний і швидкий доступ до актуальної інформації про авіаквитки, а також надасть можливість резервування та придбання квитків. Основна функціональність проекту буде заснована на парсингу даних з інших веб-сайтів авіакомпаній для забезпечення користувачам найкращого вибору та найнижчої ціни на квитки. Для досягнення поставленої мети сформульовані наступні задачі:

1) Аналіз вимог та специфікація: Визначити вимоги. Розробити детальну специфікацію системи.

2) Проектування архітектури системи: Розробити архітектуру системи, включаючи компоненти, модулі, інтерфейси та базу даних. Визначити оптимальний алгоритм для парсингу та обробки даних з веб-сайтів авіакомпаній.

3) Реалізація пошукової системи: Розробити програму для пошуку авіаквитків, яка буде використовувати розроблену архітектуру та алгоритми парсингу. Забезпечити функції пошуку, сортування та фільтрації квитків залежно від потреб користувача.

4) Інтеграція з веб-сайтами авіакомпаній: Реалізувати механізми для парсингу та інтеграції з різними веб-сайтами авіакомпаній для отримання актуальної інформації про рейси, наявність місць та ціни на квитки.

5) Реалізація функцій

1.4 Висновок

У результаті дослідження актуальності проблеми інформаційно-пошукової системи резервування та придбання авіаквитків можна підкреслити значну потребу у впровадженні ефективних та інноваційних рішень, які забезпечать зручність, швидкість, доступність, надійність та безпеку для користувачів. Високий рівень конкуренції на ринку та швидкий розвиток технологій створюють фундаментальну основу для розробки імовірних рішень та інструментів у цій сфері.

Загалом, варто враховувати різноманітні фактори при розробці інформаційно-пошукової системи резервування та придбання авіаквитків. Важливо забезпечити широкий вибір авіаквитків, зручний інтерфейс, точну інформацію про наявність місць та розклад рейсів, можливість безпосереднього зв'язку з авіакомпаніями та безпеку та захищеність користувачів. Ці висновки допоможуть визначити ключові пріоритети та напрямки для розробки інформаційно-пошукової системи резервування та придбання авіаквитків, яка буде надавати найкращий досвід користувачам.

2 ОГЛЯД АРХІТЕКТУРИ ТА ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Огляд основних варіантів реалізації

Для успішної реалізації такої системи можна розглянути декілька варіантів, кожен з яких має свої плюси та мінуси. Основні варіанти вирішення проблеми це створити веб додаток, створити мобільний додаток, сворити десктопний додаток.

Веб-додаток з використанням фреймворка:

Один з можливих варіантів реалізації - розробка веб-додатка з використанням популярних фреймворків, таких як Django або Flask. Веб-додаток забезпечує гнучкість та масштабованість системи, що дозволяє легко розширювати функціональність та взаємодіяти з іншими компонентами. Використання фреймворків спрощує розробку, надає готові компоненти для автентифікації, маршрутизації та роботи з базою даних. Однак, розробка веб-додатку може бути часомісткою та потребувати досвіду у роботі зі специфічними фреймворками[7-8].

Мобільний додаток:

Інший варіант - розробка мобільного додатка для платформ Android та iOS. Мобільний додаток надає зручний спосіб доступу до системи резервування та придбання авіаквитків, оскільки користувачі можуть використовувати його на своїх смартфонах чи планшетах. Використання мобільних технологій дозволяє використовувати функціональні можливості пристроїв, такі як GPS для відстеження місцезнаходження або пуш-сповіщення для інформування про оновлення та акції. Однак, розробка мобільних додатків може бути трудомісткою та вимагати розробки окремих версій для кожної платформи.

Десктопний додаток:

Третій варіант - розробка десктопного додатка, що працює на різних операційних системах, таких як Windows, macOS та Linux. Десктопний додаток забезпечує швидку та ефективну роботу з системою резервування та придбання авіаквитків, а також може використовувати локальні ресурси комп'ютера для

обробки та зберігання даних. Однак, розробка десктопних додатків може вимагати великої кількості коду та специфічних знань у роботі зі стандартними бібліотеками певної платформи.

Кросплатформений фреймворк:

Останнім варіантом реалізації є використання кросплатформених фреймворків, таких як React Native або Flutter. Ці фреймворки дозволяють розробляти мобільні додатки, які працюють на різних платформах без необхідності розробки окремих версій для кожної з них. Використання кросплатформених фреймворків забезпечує швидку розробку та зменшує затрати на підтримку додатку. Однак, деякі функціональні можливості платформи можуть бути обмеженими або вимагати додаткових налаштувань[9].

2.2 Вибір мови програмування та бібліотек

Вибір мови програмування та бібліотеки для розробки графічного інтерфейсу є критичним етапом при створенні будь-якого додатку. У даному випадку, мовою програмування обрано Python, а для реалізації графічного інтерфейсу використовується бібліотека Tkinter. Цей підхід має безліч переваг, які дозволяють розробнику створити функціональний та естетичний додаток.

Однією з основних переваг Python є його простота та зручність. Python є однією з найпростіших мов програмування для вивчення та розробки. Він має чистий та зрозумілий синтаксис, що дозволяє швидко створювати функціональні програми з меншими зусиллями. Використання Python спрощує відлагодження коду та полегшує процес розробки. Python також має велику та активну спільноту розробників, що робить доступними різноманітні пакети та бібліотеки для розширення функціональності додатку. Гнучкість та простота мови Python дозволяють розробникам швидше прототипувати та експериментувати з різними рішеннями. Ще однією перевагою Python є його мультиплатформеність. Додатки, розроблені з використанням Python, можуть запускатися на різних операційних системах, таких як Windows, macOS та

Linux, без необхідності внесення великих змін у вихідний код. Це дозволяє забезпечити широкий охоплення аудиторії користувачів та забезпечити зручність використання додатку незалежно від операційної системи[10-12].

Бібліотека Tkinter, яка є стандартною для Python, надає широкі можливості для створення графічного інтерфейсу користувача. Tkinter містить набір вбудованих інструментів та об'єктів, що дозволяють розробникам легко створювати вікна, кнопки, поля введення, списки та інші елементи інтерфейсу. Вона також надає можливість створювати компоненти з різноманітними стилями та налаштуваннями, що дозволяє створити естетичний та зручний інтерфейс для користувача.

Python також має велику кількість розширень та пакетів, що дозволяють легко інтегрувати його з існуючими системами та рішеннями. Це дає можливість розробникам легко використовувати сторонні API, бази даних та інші рішення, що спрощує реалізацію функціональності для резервування та придбання авіаквитків[13].

Наявність великої кількості розширень та бібліотек у Python сприяє розширюваності та підтримці додатку. Розробники можуть скористатися готовими модулями, що спрощує процес розробки та дозволяє зосередитися на основній функціональності системи. Крім того, наявність активної спільноти розробників забезпечує підтримку та відповіді на запитання, що можуть виникнути під час розробки.

Обране поєднання Python та Tkinter має всі необхідні плюси для успішного створення функціонального та зручного інтерфейсу для системи резервування та придбання авіаквитків[14-16].

2.3 Вибір методу парсингу даних

Досліджуючи різні бібліотеки для парсингу даних на Python, такі як Selenium, BeautifulSoup (bs4) та інші, можна виявити їхні переваги та недоліки.

Для кращого розуміння плюсів та мінусів підійде таблиця порівняння (табл. 2.1)

Таблиця 2.1 – Порівняння бібліотек для парсингу даних

Бібліотека	Плюси	Мінуси
Selenium	- Автоматизує браузер і може обробляти динамічний контент - Здатний розпізнавати та заповнювати форми - Може парсити дані, що підвантажуються - Підтримує різні браузери - Дозволяє виконувати скрипти JavaScript - Керування браузером з допомогою API	- Вимагає наявності браузера для виконання - Потребує додаткових драйверів (наприклад, для керування браузерами) - Часом повільніший порівняно з іншими бібліотеками - Має високі вимоги до ресурсів комп'ютера
BeautifulSoup (bs4)	- Простий і зрозумілий API - Підтримка HTML і XML - Зручне розташування та витягування даних - Підтримка CSS - селекторів та XPath - Можливість роботи зі зламаними HTML -документами - Підтримка розширень, таких як lxml	- Не вміє обробляти динамічний контент - Вимагає додаткових бібліотек для завантаження сторінок
Requests	- Простий у використанні і легкий - Підтримує HTTP-запити та відповіді - Зручне керування cookies та сесіями - Підтримка різних методів запиту - Широкі можливості керування заголовками - Добре документований	- Не вміє обробляти динамічний контент - Потребує додаткових бібліотек для парсингу HTML - Вимагає додаткових кроків для здобуття даних, які підвантажуються
Scrapy	- Фреймворк для повноцінного парсингу веб-сайтів - Автоматичне перехід по сторінках - Підтримка розподіленого парсингу - Розширюваність та модульність - Інтеграція з базами даних - Велика спільнота користувачів	- Вимагає вивчення фреймворку - Потребує більше коду для простих задач - Потребує установки додаткових залежностей - Не підходить для малих проектів

Після розгляду плюсів та мінусів цих бібліотек, можна зробити висновок, що Selenium та bs4 є кращими варіантами для багатьох сценаріїв парсингу

даних. Але все ж таки Selenium має більше функції та особливостей, серед яких:

1) Повний контроль над браузером та можливість автоматизації взаємодії: Selenium дозволяє взаємодіяти з веб-сторінками, натомість BeautifulSoup обмежується лише статичним парсингом.

2) Підтримка різних браузерів та платформ: Selenium забезпечує можливість вибору браузера для парсингу та працює на різних платформах, що є важливим для забезпечення сумісності та розширення можливостей.

3) Можливість роботи зі складними веб-елементами та JavaScript: Selenium дозволяє взаємодіяти з різними веб-елементами, такими як кнопки, форми вводу тощо. Крім того, він підтримує виконання JavaScript, що дозволяє обробляти складні динамічні сторінки.

4) Здатність розподілити завдання на різні вікна браузера: Selenium надає можливість виконувати паралельне взаємодію з різними вікнами браузера, що може бути корисним для скрапінгу даних з кількох джерел одночасно.

Хоча BeautifulSoup також має свої переваги, особливо при парсингу статичних сторінок, Selenium надає значно більше можливостей та гнучкість для розв'язання складних задач парсингу даних, особливо коли потрібна автоматизація взаємодії з веб-сторінками та робота з динамічним контентом[5].

2.4 Збір даних IATA для парсингу даних

IATA-коди (International Air Transport Association codes) є трьохлітерними кодами, які використовуються для ідентифікації авіакомпаній, аеропортів, місць призначення та інших важливих елементів в авіаційній індустрії. Ці коди є стандартизованими і використовуються для забезпечення єдиної системи ідентифікації в міжнародному авіаційному спільноті[7-8].

При парсингу авіаквитків IATA-коди є важливими, оскільки вони допомагають ідентифікувати конкретні авіакомпанії, аеропорти та маршрути. Відомість цих кодів дозволяє парсеру правильно ідентифікувати та обробляти

дані, пов'язані з авіакомпаніями і маршрутами. Наприклад, при парсингу авіаквитків IATA-код авіакомпанії може використовуватися для витягування інформації про конкретну компанію, таку як назва, логотип, рейтинг, розклад рейсів тощо. Також, IATA-код аеропорту дозволяє визначити місце вильоту та призначення, часи прильоту і вильоту, дистанцію між пунктами тощо.

Загалом, IATA-коди забезпечують стандартизований спосіб ідентифікації і обробки даних в авіаційній галузі, що допомагає в розробці парсерів та інших програм для обробки авіаційної інформації.

2.5 Архітектура проекту

З візуальної точки зору, архітектура десктоп-додатку по пошуку авіаквитків буде мати таку структуру:

Вікно додатку: Головний контейнер, що відображає основний вміст додатку. Це вікно може бути розміщене на весь екран або мати фіксований розмір, залежно від потреб додатку.

Меню зверху: Розміщено у верхній частині вікна додатку. Містить навігаційні елементи, такі як кнопки або випадаючі списки, для переключення між різними розділами додатку або виконання дій.

Список авіарейсів знизу: Розміщено у нижній частині вікна додатку. Відображає результати пошуку авіаквитків у вигляді списку або таблиці[17].

Кожен елемент списку може включати інформацію про авіакомпанію, час вильоту і прильоту, тривалість польоту, ціну тощо.

Користувач може взаємодіяти з елементами списку, наприклад, обирати певний авіарейс та переходити за посиланням для його бронювання.

2.6 Дизайн додатку

Після проведення аналізу інших інформаційних систем пошуку авіаквитків та зібравши головні особливості дизайну, які притаманні цій темі можна зробити висновок що системи для пошуку авіаквитків, такі як Expedia,

Kayak, Skyscanner та Google Flights, мають спільні риси у своєму дизайні, що допомагають користувачам зручно та ефективно знаходити необхідну інформацію[3-6].

Ці сервіси використовують мінімалістичний дизайн, який зосереджується на основному завданні - пошуку авіаквитків. Їх інтерфейси прості і зрозумілі, забезпечуючи легкість використання для користувачів.

Ключовим елементом є потужний пошуковий модуль, який дозволяє швидко та ефективно знайти авіаквитки. Цей модуль має поля для введення місця відправлення та призначення, дат вильоту та прильоту, кількості пасажирів та інших параметрів. Він ретельно налаштований, щоб забезпечити точні й швидкі результати.

Системи також надають можливість фільтрування та сортування результатів пошуку. Це дозволяє користувачам відбирати найкращі варіанти авіаквитків за різними критеріями, такими як ціна, тривалість польоту, авіакомпанія та інші. Це спрощує процес прийняття рішення користувачем.

Інформаційна ясність також є важливою складовою дизайну систем для пошуку авіаквитків. Вони надають детальну інформацію про кожен варіант авіаквитка, включаючи час вильоту та прильоту, тривалість польоту, проміжні зупинки, авіакомпанію та інші деталі. Це допомагає користувачам зробити обґрунтований вибір.

Не можна забувати і про мобільну доступність. Усі сервіси надають мобільні додатки або оптимізовані версії своїх веб-сайтів, що дозволяє користувачам шукати авіаквитки зі своїх смартфонів та планшетів. Це забезпечує доступність та зручність використання системи в будь-який час та в будь-якому місці.

Отже, система для пошуку авіаквитків повинна мати простий та зрозумілий дизайн, пошуковий модуль, чітку та докладну інформацію про кожен варіант авіаквитка та бути доступною на мобільних пристроях. Ці

спільні риси дизайну допомагають користувачам ефективно та зручно знаходити найкращі авіаквитки за їхніми потребами.

Так як в основному такі системи є веб-додатками не всі властивості підійдуть до десктоп додатку, але головні елементи будуть схожі. Дизайн додатку для пошуку квитків буде виконан у мінімалістичному стилі з основними кольорами у вигляді темно фіолетових та рожевих. Це придасть йому елегантності та сучасності. Такий стиль передає простоту, чистоту та естетичність і може створити приємний візуальний досвід для користувачів (рис. 2.1-2.2). Частини інтерфейсу з якими можна взаємодіяти будуть саме рожевими, що відділить їх від фону, або білими що також буде візуально сповіщати користувача що елемент програми є активним.

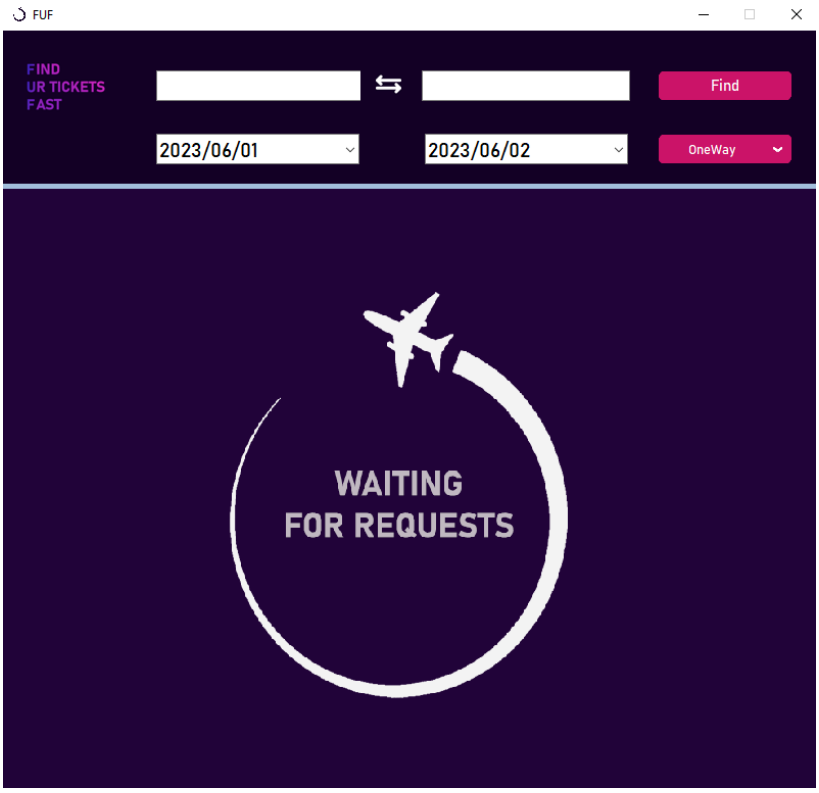

Рисунок 2.1 – Головне вікно

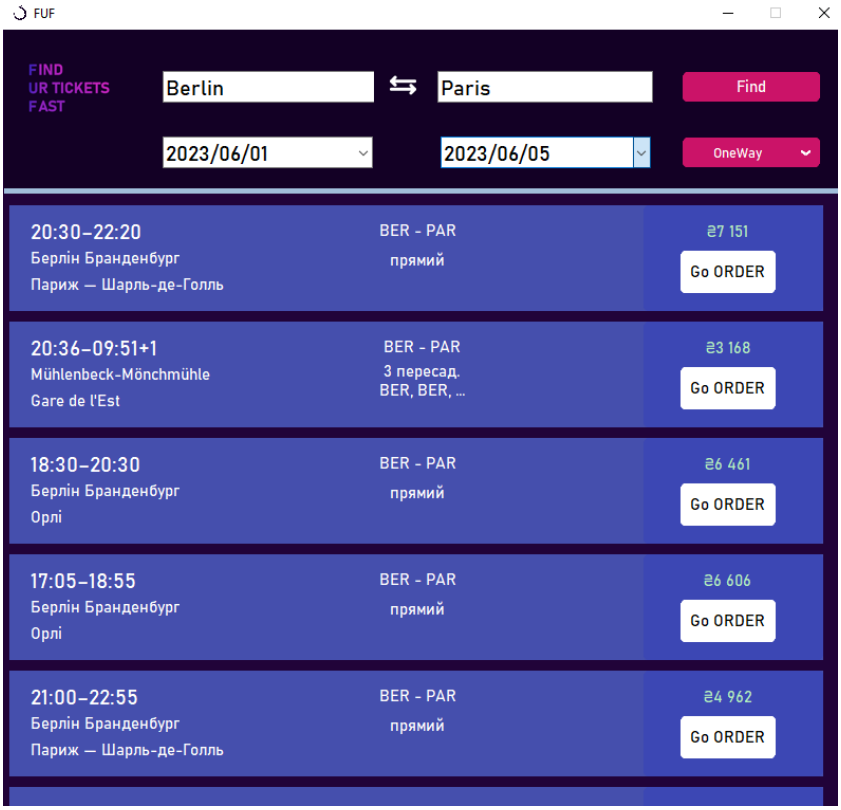

Рисунок 2.2 – Головне вікно заповнене даними

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Схема роботи додатку

Додаток працює за схемою отримання даних, обробкою отриманих даний та парсингом цих даних в інтернеті. Так як приватні API для пошуку авіаквитків коштують доволі дорого, дані будуть отримуватися з іншихї сайтів по продажу авіаквитків. Користувач матиме змогу ввести необхідні йому дані, місто з якого він шукає квиток, місто до якого він шукає квиток, а також дату та тип квитків, будь то в один бік або в обидва. (рис. 3.1)

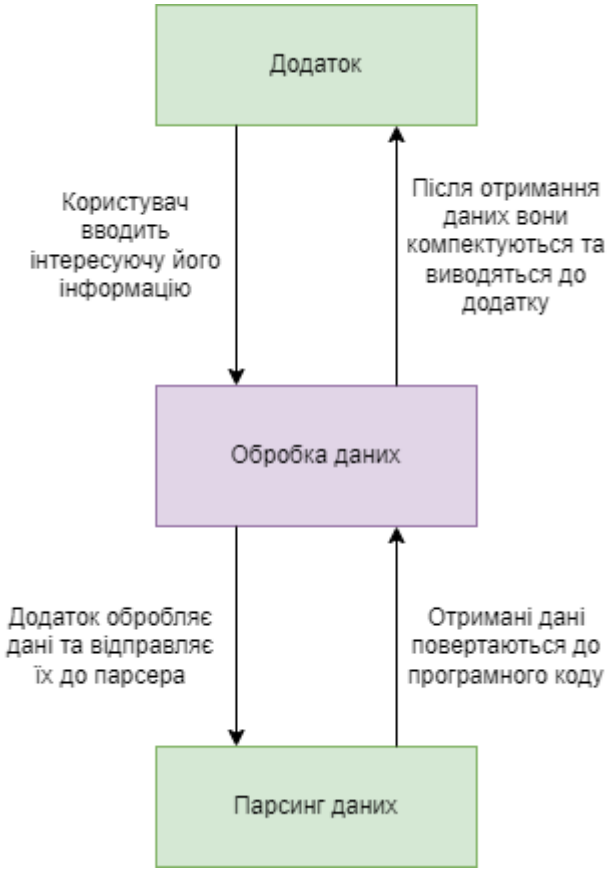

Рисунок 3.1 – Схема роботи програми

Для парсингу даних було обрано один сайт за допомогою аналізу аналогів. Цим сайтом став momondo, в нього немає перевірки на бота, а пошук квитків не розділено на сесії які закінчується. Це допоже додатку працювати безперебійно і вистачить для того щоб показати можливості додатку та його

демонстраційний варіант. У майбутьому з невеликими змінами у коді можна бути дістати доступ до платного API, тоді додаток буде працювати незалежно від інших ресурсів.

3.2 Збір даних IATA та автозаповнення

Для того щоб виконати парсинг даних по датам та містам необхідно отримати IATA коди кожного існуючого аеропорту, ця інформація є у відкритому доступі, так що залишається її лише спарсити. Це необхідно для того щоб користувач міг вводити лише назву міста, а автозаповнення вже підбирало міста з існуючих після чого змінювала назви на IATA коди і по ним шукала доступні квитки. (рис. 3.2-.3.3)

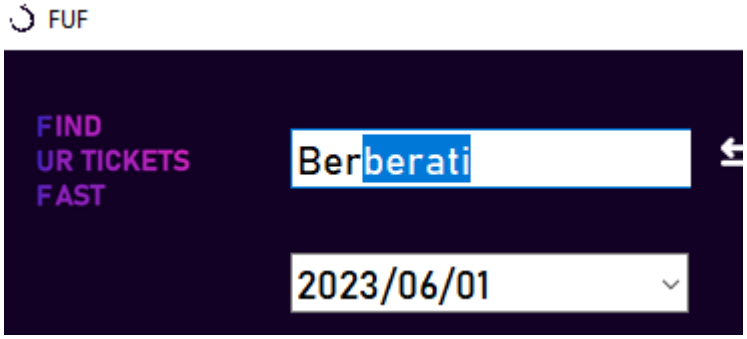

Рисунок 3.2 – Автозаповнення

Рисунок 3.3 – Автозаповнення

Для того щоб зчитати усі назви аеропортів та їх коди була написана функція дял парсингу даних, за допомогою selenium, після чого створено масив з усіма назвами (рис 3.4)

Програмна реалізація:

```
url =
"https://www.nationsonline.org/oneworld/IATA_Codes/airport_code_li
st.htm"
browser = webdriver.Chrome()

browser.get(url)
main_data = browser.find_elements(By.TAG_NAME, "tr")
# time.sleep(3)
for data in main_data:
     smth = []
     data1 = data.find_elements(By.CLASS_NAME, "border1")
     for ha in data1:
          ss = ha.text
          smth.append(ss)
     if (len(smth) > 2):
          print("\""+smth[0]+"\", ")
```

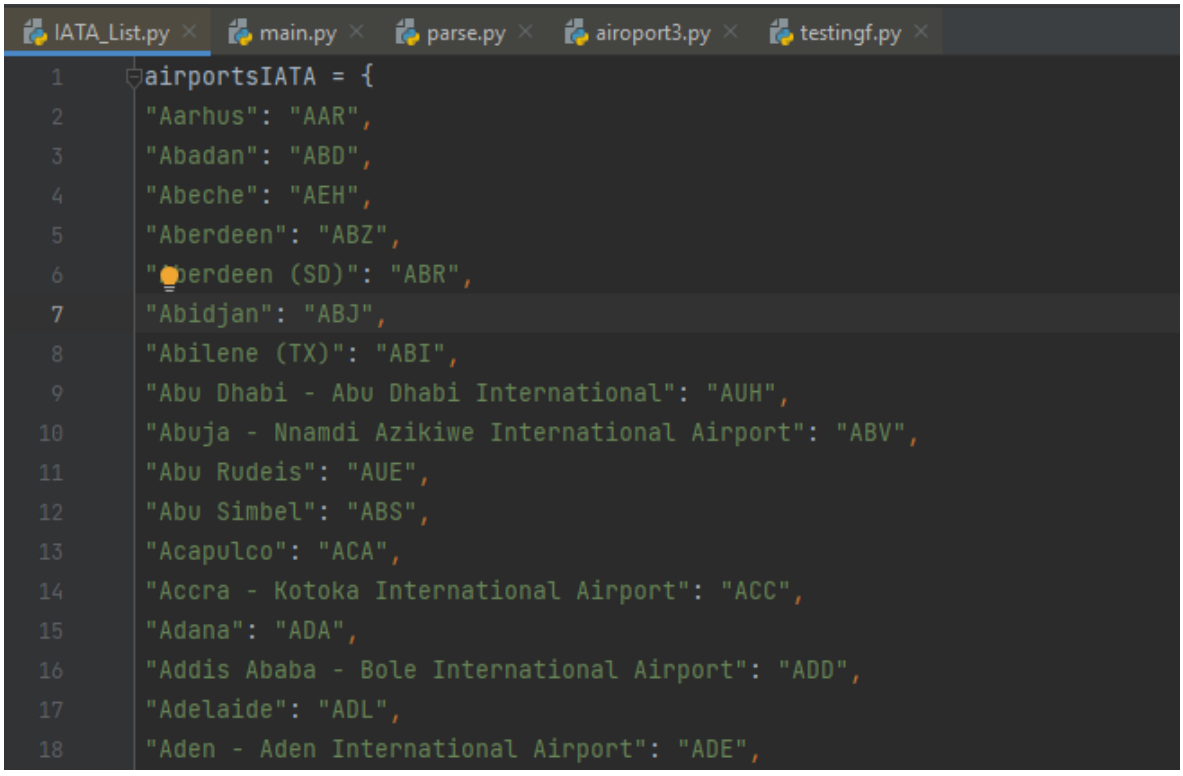

Рисунок 3.4 – Список IATA кодів

Список усіх назв було вставлено у окремний файл коду, з якого потім буде зчитуватися інформація для автозаповнення та зміни назв на IATA коди.

Автозаповнення було реалізовано за допомогою бібліотеки ttkwingets і її функції autocomplete, що є розширенням до основної бібліотеки tkinter, за допомогою якої було створено майже весь інтерфейс програми.

Програмна реалізація:

```
ntryfrom = AutocompleteEntry(
    master=from_frame,
    width=18,
    font=('Bahnschrift', 15),
    completevalues=IATA_List.onluName,
    )
entryfrom.grid(row=1, column=0,padx=(50), pady=(39,0))
```

3.3 Програмна реалізація пошуку квитків

Для пошуку квитків було створено дві основних функція. Одна з функцій повинна шукати квитки лише у один бік, інша ж функція шукає квитки у обидва боки. Перша функція має 3 аргументи які вона отримує а саме frompos, topos, fromdate (табл 3.1).

Таблиця 3.1 – Агрументи функції для парсингу

Назва	Опис	Тип даних
Frompos	Зчитує з першого поля введене користувачем місце виліту	string
Topos	Зчитує з другого поля введене користувачем місце прильоту	string
Fromdate	Зчитує з третього поля введену користувачем дату відльоту	string

Перед тим як передати до функції значення frompos та topos програмний код змінює їх на IATA коди беручи другі зі списку.

Програмна реалізація:

```
from_n=entryfrom.get()
        from_n=IATA_List.airportsIATA[from_n]
        to_n=entryto.get()
        to_n = IATA_List.airportsIATA[to_n]
```

Після того як всі дані отримані, функція підставляє їх до адреси сайту та виконує запит. Після того як запит виконано парсер зчитує дані та зберігає їх у масиві.

Програмна реалізація:

```
frompos = frompos
topos = topos
fromdate = fromdate
url = "https://www.momondo.ua/flight-search/{0}-{1}/{2}?sort=bestflight_a".format(frompos,topos,fromdate)
browser = webdriver.Chrome(chrome_options=options)
browser.get(url)
time.sleep(3)
main_data = browser.find_elements(By.CLASS_NAME, "nrc6")
for data in main_data:
    ticketinfo = data.find_element(By.CLASS_NAME, "c3J0r-container")
    time1 = ticketinfo.find_element(By.CLASS_NAME, "VY2U").text
    time1 = time1.split(" ")
    time1 = time1[0].split("\n")
    time1 = time1[0]
    Placee = ticketinfo.find_element(By.CLASS_NAME, "EFvI").text
    Placee = Placee.split("\n")
    Placee_FF = Placee[0][3:]
    Placee_TT = Placee[2][3:]
```

```
            FlyType      =      ticketinfo.find_element(By.CLASS_NAME,
"JWEO").text
            Cost = data.find_element(By.CLASS_NAME, "f8F1").text
            Link = data.find_element(By.CLASS_NAME, "dOAU")
            Link1           =           Link.find_element(By.TAG_NAME,
"a").get_attribute("href")
            time11.append(time1)
            PlaceeFF1.append(Placee_FF)
            PlaceeTT1.append(Placee_TT)
            FlyType1.append(FlyType)
            Cost1.append(Cost)
            Link11.append(Link1)
      return  time11,PlaceeFF1, PlaceeTT1, FlyType1,Cost1,Link11
```

Після введеня усіх даних, їх обробки та збереження, вони виводяться у головне вікно додатку і вигляді блоків. (рис 3.5)

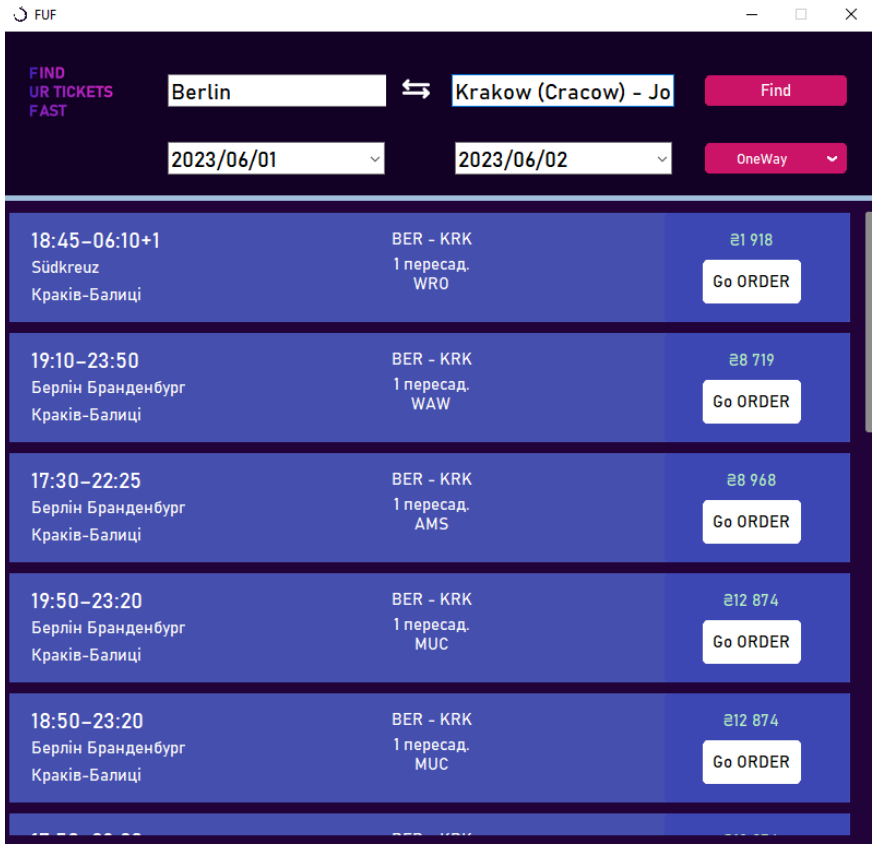

Рисунок 3.5 – Вивід одиночних рейсів

Така ж сама функція написана і для того щоб парсити дані про авіаквитки у обидва боки, але замість 3 аргументів функція отримує зразу 4, серед яких

frompos, topos, fromdate, todate (табл 3.2). Для того щоб шукати квитки у обидва бока користувач повинен вибрати “back and forth” у налаштуваннях

Таблиця 3.2 – Агрументи функції для парсингу

Назва	Опис	Тип даних
frompos	Зчитує з першого поля введене користувачем місце виліту	string
topos	Зчитує з другого поля введене користувачем місце прильоту	string
fromdate	Зчитує з третього поля введену користувачем дату відльоту	string
todate	Зчитує з четвертого поля введену користувачем дату відльоту назад	string

Після отримання даних вони також виводяться до головного вікна, але мають вже більше інформації через те що також присутній і зворотній рейс.(рис 3.6)

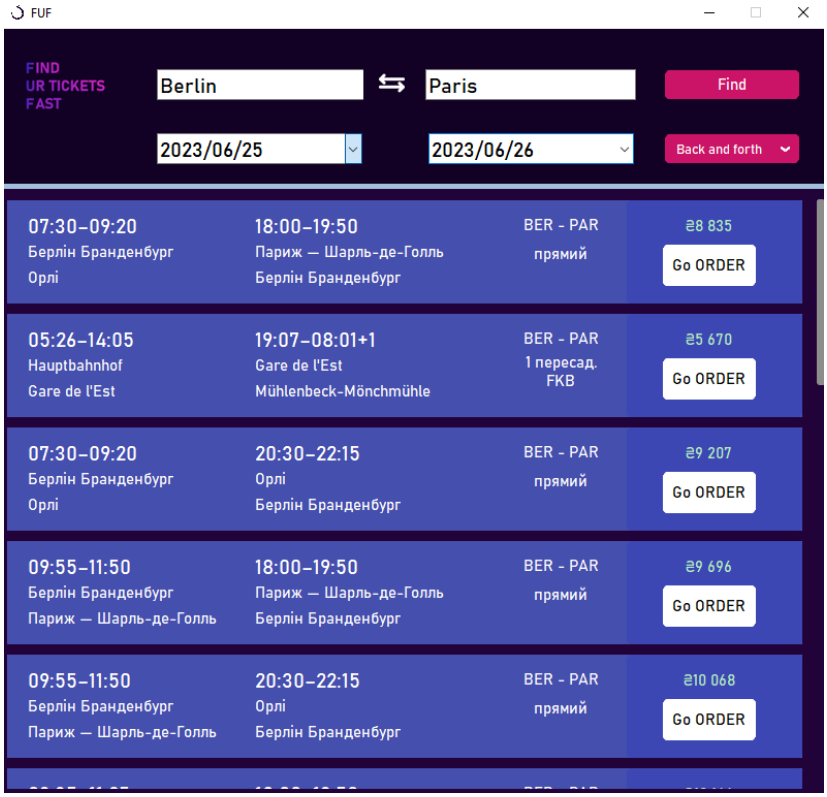

Рисунок 3.6 – Вивід рейсів в обидва боки

До кожного блоку додано кнопку, при написанні на які користувач потрапить на сайт для бронювання білетів, залежно від того яка саме компанія займається бронюванням даного білету, сайти будуть відрізнятися. (рис 3.7)

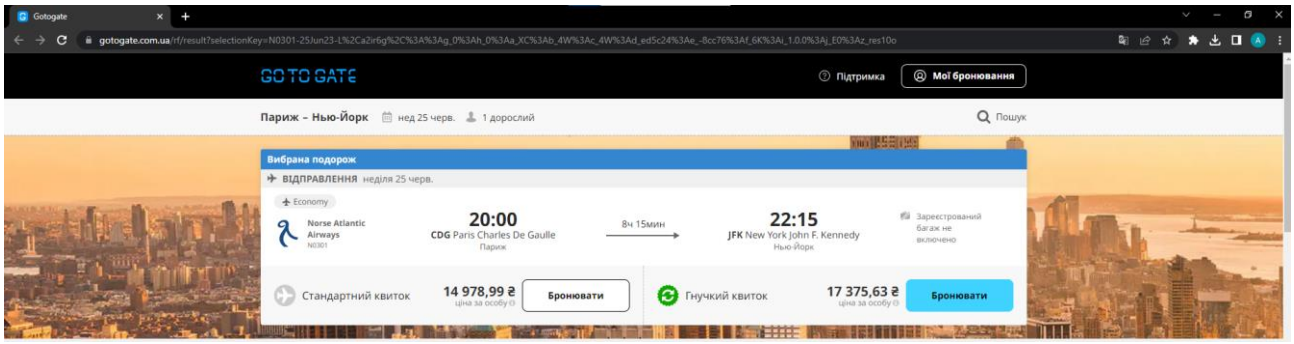

Рисунок 3.7 – приклад сайту бронювання квитків

3.4 Програмна реалізація інтерфейсу

Весь інтерфейс програми вирішено було розробити за допомогою бібліотеки tkinter та інших бібліотек з нею пов’язаних.

Tkinter є стандартною бібліотекою Python для створення графічного інтерфейсу користувача (GUI). Вона надає набір інструментів для створення вікон, об'єктів, віджетів та розміщення їх на екрані. Основними функціями бібліотеки, які також використовувалися під час розробки додатку є створення вікон, рамки, поля, текстові поля, кнопки та інші.

1) Створення вікон (Window) Tkinter дозволяє створювати вікна за допомогою класу Tk. root = Tk(). Можливі налаштування вікна:

title: Задати заголовок вікна.

geometry: Задати розмір вікна у форматі 'ширинахвисота'.

resizable: Встановити можливість зміни розміру вікна.

iconbitmap: Встановити значок вікна.

2) Рамки (Frames): Рамки використовуються для організації та групування віджетів. Вони створюються за допомогою класу Frame. Наприклад: frame = Frame(root) та мають такі основні налаштування:

bg або background: Задати колір фону рамки.

bd або borderwidth: Задати товщину межі рамки.

relief: Задати тип межі рамки (наприклад, "flat", "raised", "sunken", "groove", "ridge").

3) Поля (Entry): Поля використовуються для введення тексту користувачем. Вони створюються за допомогою класу Entry. Наприклад: entry = Entry(root) та мають такі налаштування:

show: Показати спеціальний символ замість введеного тексту (наприклад, для парольних полів).

width: Задати ширину поля.

state: Встановити стан поля ("normal", "disabled" або "readonly").

validate: Встановити тип перевірки введеного тексту ("key", "focusout" або "all").

validatecommand: Задати функцію для валідації введеного тексту.

4) Текст (Text): Віджет Text використовується для відображення та редагування багаторядкового тексту. Він створюється за допомогою класу Text. Наприклад: text = Text(root) і має такі налаштування:

height: Задати висоту віджету у рядках.

width: Задати ширину віджету у символах.

state: Встановити стан тексту ("normal", "disabled" або "readonly").

wrap: Встановити режим перенесення рядків ("none", "char" або "word").

5) Кнопки (Button): Кнопки використовуються для виконання певних дій при натисканні на них. Вони створюються за допомогою класу Button. Наприклад button = Button(root, text='Click me') і мають такі налаштування:

text: Задати текст на кнопці.

command: Задати функцію, яка буде виконуватись при натисканні кнопки.

width: Задати ширину кнопки у знаках.

height: Задати висоту кнопки у знаках.

state: Встановити стан кнопки ("normal", "disabled" або "active").

Це лише декілька основних функцій Tkinter. Бібліотека також надає багато інших віджетів (наприклад, список, комбінований список, чекбокси) та можливостей налаштування, таких як обробка подій, розміщення віджетів на екрані (pack, grid, place) та стилізація віджетів за допомогою CSS-подібних параметрів (за допомогою ttk).

Саме через це і було вирішено застосувати tkinter під час розробки додатку. Тому що за допомогою нього можна створити майже будь-який інтерфейс та стилізувати його.

З початку за допомогою window було створено основне вікно додатку, яке у майбутньому було поділено на 2 фрейми. Фрейм з налаштуванням пошуку та фрейм виводу (рис 3.8)

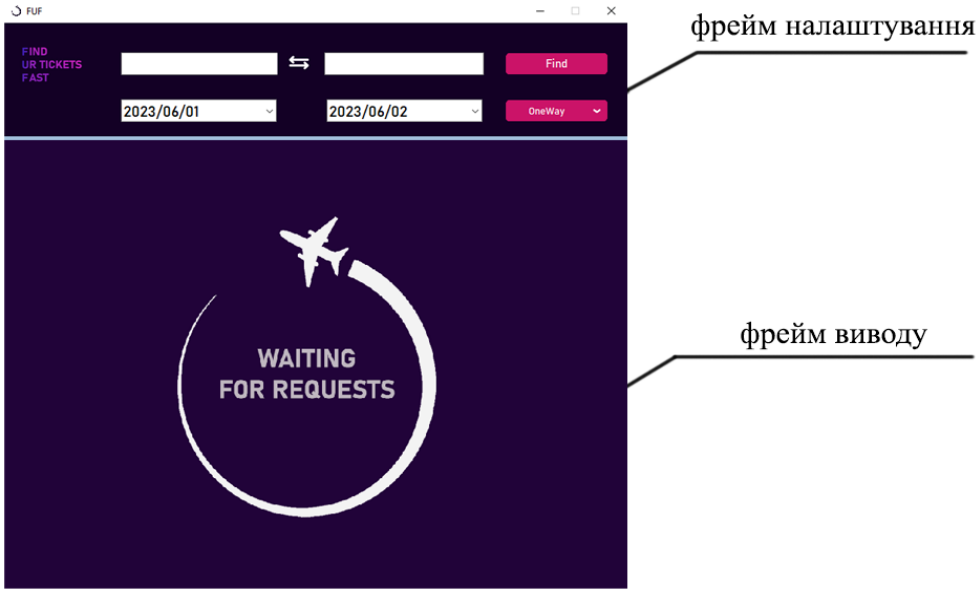

Рисунок 3.8 – 2 основних фрейми

У фреймі налаштування було створено 2 поля автозаповнення для того щоб ввести міста та 2 календраних поля для вибору дати, а також кнопка пошуку і кнопка вибору варіанту перельоту.

Програмна реалізація:

```
    button2 = customtkinter.CTkButton(master=date_frame,
text="Find",width=130, command=Godate, fg_color="#cb1368",

text_color="white",font=('Bahnschrift', 15),
hover_color="#a31456")
    button2.grid(row=1, column=0,pady=(40, 20), padx=(22))
    entryfrom = AutocompleteEntry(
        master=from_frame,
        width=18,
        font=('Bahnschrift', 15),
        completevalues=IATA_List.onluName,
        )
    entryfrom.grid(row=1, column=0,padx=(50), pady=(39,0))



    entryto = AutocompleteEntry(
        master=to_frame,
        width=18,
        font=('Bahnschrift', 15),
        completevalues=IATA_List.onluName
        )
    entryto.grid(row=1, column=0, pady=(39,0))
    cal = DateEntry(master=seti_frame, width=16, year=current_year
,    month=current_month,    day=current_day,background='#130025',
foreground='white',
                    borderwidth=2,          date_pattern='y/mm/dd',
font=('Bahnschrift', 15))
    cal.grid(row=0, column=0,pady=(1, 20), padx=(150,64))
```

Після того як користувач вводить дані то вони відправляються до парсера, у той самий час нижній фрейм з очікуванням результату видаляється,

його місце займає фрейм у який буде виведено блоки з авіаквитками та інформаціює про них знайденою на сайтах по пошуку авіаквитків. (рис 3.9)

Рисунок 3.8 – 2 основних фрейми

Нове вікно вже має слайдер для того щоб можна було переглянути більше інформації при цьому не розтягуючи вікно. На цьому моменті користувач вже може знайти квитки які йому потрібні.

ВИСНОВКИ

За результатами аналізу існуючих інформаційно-пошукових систем резервування та придбання авіаквитків стало очевидним, що існує необхідність у вдосконаленні та розробці нових рішень у цій галузі. Швидкий розвиток авіаційної індустрії та зростання популярності авіаперельотів призвели до збільшення попиту на зручні та ефективні інформаційно-пошукові системи, які б допомагали користувачам швидко знаходити та придбати авіаквитки за оптимальною ціною.

Метою даної бакалаврської роботи була розробка інформаційно-пошукової системи резервування та придбання авіаквитків, яка буде забезпечувати користувачам широкий вибір авіаквитків, швидкий та точний пошук і зручний інтерфейс. Робота була спрямована на розробку програмного продукту, який забезпечить автоматизований процес пошуку та резервування авіаквитків шляхом парсингу інформації з різних джерел та забезпечить зручний і швидкий доступ до актуальних пропозицій від авіакомпаній.

Для досягнення цієї мети були вирішені наступні задачі:

1) Вивчення та аналіз існуючих інформаційно-пошукових систем резервування та придбання авіаквитків для визначення їх переваг і недоліків.

2) Визначення основних вимог та функціональних можливостей розроблюваної системи.

3) Розробка архітектури програмного продукту та вибір необхідних технологій.

4) Реалізація модулів пошуку авіаквитків з використанням методів парсингу даних.

5) Розробка зручного інтерфейсу користувача для швидкого та зручного пошуку квитків.

6) Тестування та налагодження програмного продукту з метою перевірки його функціональності та надійності.

7) Оцінка ефективності та порівняння систем інформаційно-пошуковими системами резервування та придбання авіаквитків.

8) Документування розробленого програмного продукту та підготовка звіту про проведену роботу.

Отже, розробка інформаційно-пошукової системи резервування та придбання авіаквитків є актуальною та важливою задачею в сучасному світі. Результати даної бакалаврської роботи допоможуть покращити процес пошуку та резервування авіаквитків для користувачів, забезпечивши їм зручність, швидкість та точність при виборі та придбанні квитків.

Розроблений додаток задовольняє всім вимогам, поставленим на етапі постановки завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кенеді Берман. Основи Python для Data Science - Print2print, 2023. – с. 272 — ISBN 978-5-4461-2251-6.
2. Python 3.11.3 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/
3. Python GUI Programming With Tkinter [Електронний ресурс] – Режим доступу: https://realpython.com/python-gui-tkinter/
4. Python Tkinter Tutorial [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/python-tkinter-tutorial/
5. The Selenium Browser Automation Project [Електронний ресурс] – Режим доступу: https://www.selenium.dev/documentation/
6. Уес Маккінні. Python та аналіз даних. Третє видання- Print2print, 2023. – с. 536 — ISBN 978-5-93700-174-0.
7. Selenium with Python [Електронний ресурс] – Режим доступу: https://selenium-python.readthedocs.io/
8. Beautiful Soup Documentation [Електронний ресурс] – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
9. IATA 3-Letter Codes of Airports [Електронний ресурс] – Режим доступу: https://www.nationsonline.org/oneworld/IATA_Codes/airport_code_list.htm
10. Створення вікон [Електронний ресурс] – Режим доступу: https://metanit.com/python/tkinter/5.1.php
11. Create a Date Picker Calendar – Tkinter [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/create-a-date-picker-calendar-tkinter/
12. Python - Tkinter Button [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/python/tk_button.htm

13. PyCharm for Productive Python Development (Guide) [Електронний ресурс] – Режим доступу: https://realpython.com/pycharm-guide/
14. How to Find Cheap Flights in 2023 [Електронний ресурс] – Режим доступу: https://www.going.com/guides/how-to-find-cheap-flights
15. Build a Basic Form GUI using CustomTkinter module in Python [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/build-a-basic-form-gui-using-customtkinter-module-in-python/
16. Python. Програми з графічним інтерфейсом [Електронний ресурс] – Режим доступу: https://www.miyklas.com.ua/p/informatica/8-klas/algoritmi-ta-programi-394917/programi-z-grafichnim-interfeisom-395197/re-6038e80b-3201-49cd-a207-7518a6aab2c0
17. Frame [Електронний ресурс] – Режим доступу: https://sites.google.com/comp-sc.if.ua/python-easy/tkinter/%D0%B2%D1%96%D0%B4%D0%B6%D0%B5%D1%82%D0%B8/frame

ДОДАТКИ

Додаток А. Лістинг програмного коду

Main.py

```
import customtkinter
import IATA_List
import parse
from tkinter import *
from tkcalendar import *
from tkinter import ttk
from datetime import datetime
from datetime import date
import webbrowser
from PIL import ImageTk, Image
from ttkwidgets.autocomplete import AutocompleteEntry
customtkinter.set_appearance_mode("light")
customtkinter.set_default_color_theme("blue")



def optionmenu_callback(choice):
    print("optionmenu dropdown clicked:", choice)
def go ():
    print(IATA_List.airportsIATA[entry.get()])

def Godate():
    if (one_or_two.get() == "OneWay"):

        from_n=entryfrom.get()
        from_n=IATA_List.airportsIATA[from_n]
        to_n=entryto.get()
        to_n = IATA_List.airportsIATA[to_n]
        when_n=cal.get()
        when_n = when_n.replace("/","-")
```

```
            second_frame.grid_forget()
            second_frame1                                          =
customtkinter.CTkScrollableFrame(master=main_frame,     height=575,
width=780, fg_color="#210339")
            second_frame1.grid(row=3, column=0)
            s1,s2,s3,s4,s5,s6   =   parse.find_ticket(from_n,   to_n,
when_n)
            i=0

            for smth in s1:


                Find_rez                                           =
customtkinter.CTkFrame(master=second_frame1,            height=100,
width=770, fg_color="#454fad")
                Find_rez.grid(row=i,pady=(5, 5))
                Find_rez.grid_propagate(0)

                Find_rez1                                          =
customtkinter.CTkFrame(master=Find_rez,    height=100,    width=350,
fg_color="#454fad")
                Find_rez1.grid(row=0, column=0)
                Find_rez1.grid_propagate(0)

                label_timef                                        =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=300,
fg_color="#454fad")
                label_timef.grid(row=0,   pady=(10,   0),   padx=(20,
0))
                label_timef.grid_propagate(0)
                Label_time                                         =
customtkinter.CTkLabel(master=label_timef,              text=s1[i],
font=('Bahnschrift', 20),
```

```
text_color="white")
                Label_time.grid(row=0)

                Label_placef                                        =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=300,
fg_color="#454fad")
                Label_placef.grid(row=1, padx=(20, 0))
                Label_placef.grid_propagate(0)
                Label_place                                         =
customtkinter.CTkLabel(master=Label_placef,             text=s2[i],
font=('Bahnschrift', 15),

text_color="white")
                Label_place.grid(row=0)

                Label_placef2                                       =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=300,
fg_color="#454fad")
                Label_placef2.grid(row=2, padx=(20, 0))
                Label_placef2.grid_propagate(0)
                Label_place2                                        =
customtkinter.CTkLabel(master=Label_placef2,            text=s3[i],
font=('Bahnschrift', 15),

text_color="white")
                Label_place2.grid(row=0)



                Find_rez2                                           =
customtkinter.CTkFrame(master=Find_rez,    height=100,    width=250,
fg_color="#454fad")
                Find_rez2.grid(row=0, column=1)
```

```
                Find_rez2.grid_propagate(0)
                Label_City                                          =
customtkinter.CTkLabel(master=Find_rez2,  text=from_n+"  -  "+to_n,
font=('Bahnschrift', 15),

text_color="white")
                Label_City.grid(row=0, pady=(9, 0))
                Label_type                                          =
customtkinter.CTkLabel(master=Find_rez2,                text=s4[i],
font=('Bahnschrift', 15),

text_color="white")
                Label_type.grid(row=1, pady=(0, 0))

                Find_rez3                                           =
customtkinter.CTkFrame(master=Find_rez,   height=100,   width=170,
fg_color="#3c47b4")
                Find_rez3.grid(row=0, column=2)
                Find_rez3.grid_propagate(0)
                Label_cost                                          =
customtkinter.CTkLabel(master=Find_rez3,                text=s5[i],
font=('Bahnschrift', 15),

text_color="#b3edbf")
                Label_cost.grid(row=0, pady=(10, 0), padx=(35, 0))
                Button0                                             =
customtkinter.CTkButton(master=Find_rez3,     text="Go     ORDER",
width=90, height=40, fg_color="#ffffff",

text_color="black",

font=('Bahnschrift',  15),  hover_color="#d6d4d4",  command=  lambda
i=i: webbrowser.open(s6[i]))
                Button0.grid(row=1, pady=(5, 0), padx=(35, 0))
```

```
                i=i+1

        if (one_or_two.get() == "Back and forth"):

            from_n = entryfrom.get()
            from_n = IATA_List.airportsIATA[from_n]
            to_n = entryto.get()
            to_n = IATA_List.airportsIATA[to_n]
            when_n = cal.get()
            when_n = when_n.replace("/", "-")

            when_back = cal1.get()
            when_back = when_back.replace("/", "-")

            second_frame.grid_forget()
            second_frame1                                          =
customtkinter.CTkScrollableFrame(master=main_frame,     height=575,
width=780, fg_color="#210339")
            second_frame1.grid(row=3, column=0)
            s1, s2, s3, s4, s5, s6 = parse.find_ticketback(from_n,
to_n, when_n,when_back)
            i = 0

            for smth in s1:
                try:
                    Find_rez                                       =
customtkinter.CTkFrame(master=second_frame1,            height=100,
width=770, fg_color="#454fad")
                    Find_rez.grid(row=i, pady=(5, 5))
                    Find_rez.grid_propagate(0)
```

```
                    Find_rez1                                       =
customtkinter.CTkFrame(master=Find_rez,    height=100,    width=500,
fg_color="#454fad")
                    Find_rez1.grid(row=0, column=0)
                    Find_rez1.grid_propagate(0)



                    label_timef                                     =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    label_timef.grid(row=0,   column=0,   pady=(10,
0), padx=(20, 0))
                    label_timef.grid_propagate(0)
                    Label_time                                      =
customtkinter.CTkLabel(master=label_timef,              text=s1[i],
font=('Bahnschrift', 20),

text_color="white")
                    Label_time.grid(row=0)

                    Label_placef                                    =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    Label_placef.grid(row=1,   column=0,   padx=(20,
0))
                    Label_placef.grid_propagate(0)
                    Label_place                                     =
customtkinter.CTkLabel(master=Label_placef,             text=s2[i],
font=('Bahnschrift', 15),

text_color="white")
                    Label_place.grid(row=0)
```

```
                    Label_placef2                                   =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    Label_placef2.grid(row=2,  column=0,  padx=(20,
0))
                    Label_placef2.grid_propagate(0)
                    Label_place2                                    =
customtkinter.CTkLabel(master=Label_placef2,            text=s3[i],
font=('Bahnschrift', 15),

text_color="white")
                    Label_place2.grid(row=0)


                    # ----

                    label_timef90                                   =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    label_timef90.grid(row=0,  column=1,  pady=(10,
0), padx=(20, 0))
                    label_timef90.grid_propagate(0)
                    Label_time                                      =
customtkinter.CTkLabel(master=label_timef90,          text=s1[i+1],
font=('Bahnschrift', 20),

text_color="white")
                    Label_time.grid(row=0)

                    Label_placef90                                  =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    Label_placef90.grid(row=1,  column=1,  padx=(20,
0))
```

```
                    Label_placef90.grid_propagate(0)
                    Label_place                                    =
customtkinter.CTkLabel(master=Label_placef90,         text=s2[i+1],
font=('Bahnschrift', 15),

text_color="white")
                    Label_place.grid(row=0)

                    Label_placef290                                =
customtkinter.CTkFrame(master=Find_rez1,    height=25,    width=200,
fg_color="#454fad")
                    Label_placef290.grid(row=2,           column=1,
padx=(20, 0))
                    Label_placef290.grid_propagate(0)
                    Label_place2                                   =
customtkinter.CTkLabel(master=Label_placef290,        text=s3[i+1],
font=('Bahnschrift', 15),

text_color="white")
                    Label_place2.grid(row=0)




                    Find_rez2                                      =
customtkinter.CTkFrame(master=Find_rez,    height=100,    width=100,
fg_color="#454fad")
                    Find_rez2.grid(row=0, column=1)
                    Find_rez2.grid_propagate(0)
                    Label_City                                     =
customtkinter.CTkLabel(master=Find_rez2,  text=from_n  +  "  -  "  +
to_n, font=('Bahnschrift', 15),
```

```
text_color="white")
                    Label_City.grid(row=0, pady=(9, 0))
                    Label_type                                        =
customtkinter.CTkLabel(master=Find_rez2,                text=s4[i],
font=('Bahnschrift', 15),

text_color="white")
                    Label_type.grid(row=1, pady=(0, 0))

                    Find_rez3                                         =
customtkinter.CTkFrame(master=Find_rez,    height=100,    width=170,
fg_color="#3c47b4")
                    Find_rez3.grid(row=0, column=2)
                    Find_rez3.grid_propagate(0)
                    Label_cost                                        =
customtkinter.CTkLabel(master=Find_rez3,                text=s5[i],
font=('Bahnschrift', 15),

text_color="#b3edbf")
                    Label_cost.grid(row=0, pady=(10, 0), padx=(35,
0))
                    Button0                                           =
customtkinter.CTkButton(master=Find_rez3,      text="Go      ORDER",
width=90, height=40,

fg_color="#ffffff",

text_color="black",

font=('Bahnschrift', 15), hover_color="#d6d4d4",

command=lambda i=i: webbrowser.open(s6[i]))
                    Button0.grid(row=1, pady=(5, 0), padx=(35, 0))
```

```
                    i = i + 2
                except:
                    return 0



    window = Tk()
    window.title("FUF")
    p12 = PhotoImage(file = 'iconn2.png')
    window.iconphoto(False, p12)
    window.geometry("800x750")
    window.resizable(False, False)

    main_frame = Frame(master=window, height=700, width=800,
background="#210339")
    main_frame.pack(fill="both", expand=True)
    main_frame.grid_propagate(0)

    top_frame = Frame(master=main_frame, height=100, width=800,
background="#130025")
    top_frame.grid(row=0, column=0)
    top_frame.grid_propagate(0)
    seti_frame = line_frame = Frame(master=main_frame, height=50,
width=800, background="#130025")
    seti_frame.grid(row=1, column=0)
    seti_frame.grid_propagate(0)

    current_year = date.today().year
    current_month = date.today().month
    current_day = date.today().day
```

```
    cal = DateEntry(master=seti_frame, width=16, year=current_year
,    month=current_month,    day=current_day,background='#130025',
foreground='white',
                    borderwidth=2,          date_pattern='y/mm/dd',
font=('Bahnschrift', 15))
    cal.grid(row=0, column=0,pady=(1, 20), padx=(150,64))


    cal1       =       DateEntry(master=seti_frame,       width=16,
year=current_year,                                 month=current_month,
day=current_day+1,background='#130025', foreground='white',
                     borderwidth=2,         date_pattern='y/mm/dd',
font=('Bahnschrift', 15))
    cal1.grid(row=0, column=1,pady=(1, 20))

    one_or_two   =   customtkinter.CTkOptionMenu(master=seti_frame,
width=130, values=["OneWay","Back and forth"],
                                              fg_color="#cb1368",
text_color="white",font=('Bahnschrift', 13),

anchor="center",button_color="#cb1368",         button_hover_color=
"#a31456")
    one_or_two.grid(row=0, column=2,pady=(1, 20),padx=(30,0))




    line_frame  =  Frame(master=main_frame,  height=5,  width=800,
background="#a1bedb")
    line_frame.grid(row=2, column=0)
```

```
    logo_frame = Frame(master=top_frame, height=110, width=100,
background="#130025")
    logo_frame.grid(row=0, column=0)
    logo_frame.grid_propagate(0)

    image2 = ImageTk.PhotoImage(Image.open('Untitled-
23.png').resize(size=(162,60)))
    label_ico1 = Label(master=logo_frame, image=image2,
borderwidth=0,highlightthickness = 0)
    label_ico1.grid(pady=(25, 20), padx=(18))


    from_frame = Frame(master=top_frame, height=110, width=250,
background="#130025")
    from_frame.grid(row=0, column=1)
    from_frame.grid_propagate(0)

    strel_frame = Frame(master=top_frame, height=110, width=60,
background="#130025")
    strel_frame.grid(row=0, column=2)
    strel_frame.grid_propagate(0)

    to_frame = Frame(master=top_frame, height=110, width=210,
background="#130025")
    to_frame.grid(row=0, column=3)
    to_frame.grid_propagate(0)

    date_frame = Frame(master=top_frame, height=110, width=200,
background="#130025")
    date_frame.grid(row=0, column=4)
    date_frame.grid_propagate(0)
```

```
    image1                                                    =
ImageTk.PhotoImage(Image.open('her2.png').resize(size=(25,25)))

    label_ico     =     Label(master=strel_frame,     image=image1,
borderwidth=0,highlightthickness = 0)
    label_ico.grid(pady=(40, 20), padx=(15))


    #  cal  =  DateEntry(master=date_frame,  width=12,  year=2023,
month=6,      day=22,background='#2a064c',      foreground='white',
borderwidth=2, date_pattern='y/mm/dd')
    # cal.grid(row=0, column=0,pady=(20, 20), padx=(50))


    button2      =      customtkinter.CTkButton(master=date_frame,
text="Find",width=130, command=Godate, fg_color="#cb1368",

text_color="white",font=('Bahnschrift',                          15),
hover_color="#a31456")
    button2.grid(row=1, column=0,pady=(40, 20), padx=(22))
    entryfrom = AutocompleteEntry(
        master=from_frame,
        width=18,
        font=('Bahnschrift', 15),
        completevalues=IATA_List.onluName,
        )
    entryfrom.grid(row=1, column=0,padx=(50), pady=(39,0))


    entryto = AutocompleteEntry(
        master=to_frame,
        width=18,
        font=('Bahnschrift', 15),
        completevalues=IATA_List.onluName
```

```
        )
    entryto.grid(row=1, column=0, pady=(39,0))
    global second_frame
    second_frame = Frame(master=main_frame, height=585, width=800,
background="#210339")
    second_frame.grid(row=3, column=0)
    second_frame.grid_propagate(0)



    image3        =        ImageTk.PhotoImage(Image.open('ezgif-3-
c4a779f79e.gif').resize(size=(650,500)))
    label_ico2    =    Label(master=second_frame,    image=image3,
borderwidth=0,highlightthickness = 0)
    label_ico2.grid(pady=(50,0),padx=(70))

    window.mainloop()
```

Parse.py

```
    import time

    from selenium import webdriver
    from selenium.webdriver.common.by import By
    from selenium.webdriver.support.wait import WebDriverWait
    from  selenium.webdriver.support  import  expected_conditions  as
EC

    def find_ticket(frompos, topos, fromdate):
     frompos = frompos
     topos = topos
     fromdate = fromdate
     url        =        "https://www.momondo.ua/flight-search/{0}-
{1}/{2}?sort=bestflight_a".format(frompos,topos,fromdate)
```

```
    options = webdriver.ChromeOptions()
    # options.add_argument("--headless")
    options.add_argument("--window-size=300,300")
    options.add_argument("--window-position=0,2000")
    browser = webdriver.Chrome(chrome_options=options)


    browser.get(url)
    time.sleep(3)
    main_data = browser.find_elements(By.CLASS_NAME, "nrc6")
    time11 = []
    PlaceeFF1 = []
    PlaceeTT1 = []
    FlyType1 = []
    Cost1 = []
    Link11 = []
    for data in main_data:
        ticketinfo = data.find_element(By.CLASS_NAME, "c3J0r-
container")
        time1 = ticketinfo.find_element(By.CLASS_NAME,
"VY2U").text
        time1 = time1.split(" ")
        time1 = time1[0].split("\n")
        time1 = time1[0]
        Placee = ticketinfo.find_element(By.CLASS_NAME,
"EFvI").text
        Placee = Placee.split("\n")
        Placee_FF = Placee[0][3:]
        Placee_TT = Placee[2][3:]
        FlyType = ticketinfo.find_element(By.CLASS_NAME,
"JWEO").text
        Cost = data.find_element(By.CLASS_NAME, "f8F1").text
        Link = data.find_element(By.CLASS_NAME, "dOAU")
```

```
            Link1           =           Link.find_element(By.TAG_NAME,
"a").get_attribute("href")
            # print (time1)
            # print(Placee)
            # print(FlyType)
            # print(Cost)
            # print(Link1)
            # print("-------")
            time11.append(time1)
            PlaceeFF1.append(Placee_FF)
            PlaceeTT1.append(Placee_TT)
            FlyType1.append(FlyType)
            Cost1.append(Cost)
            Link11.append(Link1)
        return  time11,PlaceeFF1, PlaceeTT1, FlyType1,Cost1,Link11


    def find_ticketback(frompos, topos, fromdate, todate):
        frompos = frompos
        topos = topos
        fromdate = fromdate
        todate = todate
        url         =         "https://www.momondo.ua/flight-search/{0}-
{1}/{2}/{3}".format(frompos, topos, fromdate, todate)

        options = webdriver.ChromeOptions()
        # options.add_argument("--headless")
        options.add_argument("--window-size=500,500")
        options.add_argument("--window-position=0,2000")
        browser = webdriver.Chrome(chrome_options=options)

        browser.get(url)
        time.sleep(3)
        main_data = browser.find_elements(By.CLASS_NAME, "nrc6")
```

```
    time11 = []
    PlaceeFF1 = []
    PlaceeTT1 = []
    FlyType1 = []
    Cost1 = []
    Link11 = []
    for data in main_data:
        ticketinfo = data.find_elements(By.CLASS_NAME, "c3J0r-
container")
        for datas in ticketinfo:
            time1 = datas.find_element(By.CLASS_NAME,
"VY2U").text
            time1 = time1.split(" ")
            time1 = time1[0].split("\n")
            time1 = time1[0]
            Placee = datas.find_element(By.CLASS_NAME,
"EFvI").text
            Placee = Placee.split("\n")
            Placee_FF = Placee[0][3:]
            Placee_TT = Placee[2][3:]
            FlyType = datas.find_element(By.CLASS_NAME,
"JWEO").text
            Cost = data.find_element(By.CLASS_NAME,
"f8F1").text
            Link = data.find_element(By.CLASS_NAME, "dOAU")
            Link1 = Link.find_element(By.TAG_NAME,
"a").get_attribute("href")
            # print (time1)
            # print(Placee_FF)
            # print(Placee_TT)
            # print(FlyType)
            # print(Cost)
            # print(Link1)
            # print("-------")
```

```
            time11.append(time1)
            PlaceeFF1.append(Placee_FF)
            PlaceeTT1.append(Placee_TT)
            FlyType1.append(FlyType)
            Cost1.append(Cost)
            Link11.append(Link1)
    return time11, PlaceeFF1, PlaceeTT1, FlyType1, Cost1, Link11
```