Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного засобу моніторингу авіарейсів»

Виконав: студент 4 курсу групи 1ПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гоменюк Н.В.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ Ліщинська Л.Б.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЛОТ Кожем'яко А. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____Романюк О.Н.
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гоменюку Назару Володимировичу

1. Тема роботи – «Розробка програмного засобу моніторингу авіарейсів»

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. №66.

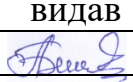
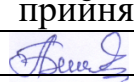
2. Строк подання студентом роботи 13 травня 2022 р.

3. Вихідні дані до роботи: Середовище розробки – IntelliJ IDEA, Мова розробки – Java, Операційна система – Windows 7/8/8.1/10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка архітектури та алгоритмів програмного додатку; розробка програмного додатку; тестування додатку, висновки; список використаних джерел, додатки, графічна частина.

5. Перелік графічного матеріалу: демонстрація роботи аналогів, блок-схема функціонування програмного продукту, блок-схема взаємодії модулів програмного продукту, загальний алгоритм роботи додатку та тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л. Б., д.т.н., професор кафедри ПЗ		
		28.03.22	10.06.22

7. Дата видачі завдання 25 березня 2022р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26. 03.2022 – 30.03.2022	Вик.
2	Розробка архітектури та алгоритмів програмного додатку	01.04.2022 – 10.04.2022	Вик.
3	Вибір середовища та мови розробки	11.04.2022 – 20.04.2022	Вик.
4	Розробка програмного продукту	22.04.2022 – 30.05.2022	Вик.
5	Тестування роботи системи	01.06.2022 – 03.06.2022	Вик.
6	Оформлення матеріалів до захисту БДР	04.06.2022 – 10.06.2022	Вик.

Студент _____ **Гоменюк Н. В.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Ліщинська Л. Б.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 62 сторінок формату А4, на яких є 44 рисунка, 3 таблиці, список використаних джерел містить 20 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз стану програмного забезпечення, моніторингу авіарейсів. Описано об’єкт, предмет, завдання та методи дослідження. Сформульовано мету досліджень – підвищення продуктивності роботи користувача з засобом моніторингу авіарейсів за рахунок автоматизації та покращення алгоритму обробки анімації літаків. Для реалізації мети розроблено алгоритми роботи, обробки анімації, графічний інтерфейс, та програмну реалізацію програмного засобу моніторингу авіарейсів.

Розроблено алгоритми програми та обробки анімації літаків , які враховують переміщення літаків у повітряному просторі та дозволяють збільшити ефективність отримання й обробки даних для кінцевого користувача.

Реалізований програмний засіб моніторингу авіарейсів за допомогою мови програмування Java, бібліотеки Swing, та Swing GUI-builder для швидкого створення графічного інтерфейсу програми У результаті виконання бакалаврської дипломної роботи розроблено програмний засіб, працездатність і коректність роботи якого перевірено, створена інструкція користувача.

Отримані в бакалаврській дипломній роботі результати можна використати для розширення функціоналу існуючих програмних реалізацій моніторингу авіарейсів.

Ключові слова: засіб моніторингу, авіарейс, літак, повітряний простір.

ABSTRACT

The bachelor's thesis consists of 62 A4 pages, which have 44 figures, 3 tables, the list of sources used contains 20 titles.

In the bachelor's thesis a detailed analysis of the state of software, flight monitoring. The object, subject, tasks and research methods are described. The purpose of the research is formulated - to increase the productivity of the user with the means of flight monitoring by automating and improving the algorithm for processing aircraft animation. To achieve this goal, algorithms have been developed, animation processing, graphical interface, and software implementation of flight monitoring software.

Algorithms for the program and processing of aircraft animation have been developed, which take into account the movement of aircraft in the airspace and increase the efficiency of data acquisition and processing for the end user.

Implemented software for flight monitoring using Java programming language, Swing library, and Swing GUI-builder for quick creation of the graphical interface of the program.

The results obtained in the bachelor's thesis can be used to expand the functionality of existing software implementations of flight monitoring.

Key words: monitoring tool, flight, airplane, airspace.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ, вже не такий як був сто років назад, з розвитком сучасних комп'ютерних технологій, світ змінився та змінюється далі з кожним днем. Інформаційні технології зайняли найвищу людську нішу, у нашому житті, вони спрямовані на організовану сукупність інформаційних потоків, з залученням засобів обчислювальної техніки, що забезпечує значну швидкість обробки даних, швидкий пошук інформації, незалежно від місця розташування. З еволюціонуванням технологій, сучасний світ змінюється, автоматизується, покращується та розвивається великими кроками, за для того, щоб зробити наше людське життя комфортнішим і безпечнішим. У сьогоднішній час сучасні інформаційні технології впроваджуються на багатьох підприємствах, установах, фірмах, тощо. Оскільки інформація та спосіб її подання суттєво відрізняються між галузями людського життя, приймати, оброблювати, зберігати, та видавати результат потрібно відповідно до технічних і наукових можливостей. Тому, не винятком розвитку інформаційних технологій стала і сфера авіації[1].

У сьогоднішній час найшвидші перевезення людей, товарів, транспорту, воєнної інфраструктури здійснюються завдяки авіації. Найширшого та найпопулярнішого використання авіації набула сфера перевезення людей, адже саме за допомогою реактивних двигунів, літак може дістатися до іншої точки планети за лічені години, що було би дуже довго, накладно, та незручно, а місцями й нереально зробити це за допомогою наземного транспорту. Щоб забезпечити безпечне переміщення літаків у повітряному просторі використовуються різні системи відслідковування літаків.

Головним завданням авіації є забезпечення безпеки пасажирів. На безпеку польотів впливає кілька факторів: надійність літальних апаратів і наземної техніки, якість підготовки льотного складу, клімат, якість системи управління. Аналізуючи авіакатастрофи, науковці стверджують, що основним чинником є помилки пілотів або диспетчерів.

Для того щоб працівник, який здійснює керування повітряним рухом, міг швидко прийняти рішення в стресових умовах, викликаних недоліком часу або інформації, він повинен мати відповідні практичні навики.

Практика показує, що після тривалих перерв у роботі, викликаних різними обставинами (хвороба, відпустка), оператор втрачає свої навички. Саме тому, актуальною задачею є розробка тренажеру відновлення та тренування навичок для диспетчерів керування повітряним рухом.

Тому актуальними є питання підвищення продуктивності роботи користувача з засобом моніторингу авіарейсів за рахунок автоматизації та покращення алгоритму обробки анімації літаків, оскільки існуючі засоби відслідковування авіапольотів не повністю задовольняють потреби сфери. Це передбачає покращення алгоритмів обробки анімації літаків і розробку засобів моніторингу авіарейсів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою роботи є підвищення оперативності роботи користувача з засобом моніторингу авіарейсів за рахунок автоматизації та покращення алгоритму обробки анімації літаків.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів моніторингу авіарейсів для відслідковування повітряних засобів у просторі;
- покращити алгоритм обробки анімації літаків;
- підвищити продуктивність роботи користувача з засобом керування повітряного простору;
- збільшити безпеку пасажирів під час польоту;
- розробити програмні компоненти та систему візуалізації моніторингу авіарейсів на основі запропонованих методів;
- провести тестування програмного продукту.

Об’єкт дослідження – процес відслідковування положення літаків у повітряному просторі.

Предмет дослідження – методи та засоби відслідковування повітряних засобів.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів для розробки моделей та методів обробки анімації літаків, комп’ютерне моделювання для перевірки розроблених методів і засобів.

Наукова новизна отриманих результатів.

Подальшого розвитку отримав метод обробки анімації літаків, на відміну від існуючих, використовує методи обробки у реальному часі, що дозволило підвищити продуктивність моніторингу авіарейсів та їх безпеку.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби моніторингу авіарейсів для відслідковування повітряних суден.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві [2], автору належать такі результати: розробка програмного засобу моніторингу авіарейсів.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на конференціях: LI Науково – технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2022).

Публікації. Основні результати досліджень опубліковано в 1 науковій праці у матеріалах конференцій.

1 АНАЛІЗ ПРОБЛЕМИ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз стану моніторингу авіарейсів

Авіаційний диспетчер [3] є важливою професією у роботі авіаційної галузі. Вона з'явилась в період, коли повітряні судна почали масово літати і тому важливо було керувати ними з поверхні землі, при цьому повинна бути присутня людина, яка забезпечувала б виконання певних правил повітряного руху та організації повітряного простору.

Так з'явилася система обслуговування повітряного руху, в котрій диспетчер здійснює функцію безпосереднього впливу на роботу екіпажів літаків та інших повітряних суден, безпеку і надійність. Крім того, в обов'язки диспетчерів управління повітряним рухом входить напрямок руху літака упродовж всього маршруту: з моменту підняття літака в одному аеропорту, до моменту випуску шасі в кінцевому.

Вони, перебуваючи в різних країнах, дуже уважно спостерігають за рухом повітряних суден в своїй зоні, а потім передають спеціалістам з відповідних областей.

Для успішного виконання своєї цілі диспетчер управління повітряним рухом застосовує знання нормативних правових документів, що надають порядок виконання польотів, льотно-технічних характеристик повітряних суден, повітряної навігації, авіаційної метеорології, англійської мови, радіотехнічних засобів забезпечення польотів повітряних суден.

Авіадиспетчер не має права на помилку, бо відповідає за людські життя. Диспетчер управління повітряним рухом надає екіпажам повітряних суден три види послуг – управління, допомога та інформування.

Для зручного, надійного та безпечного відслідковування польотів, й були розроблені засоби моніторингу авіапольотів, котрими авіадиспетчери вдало користуються.

«Моніторинг» [4] - це слово, яке досить часто використовується в авіації, і це природньо, що його значення могло б зазнати плутанини. По простому, звичайною мовою: Моніторинг – це спостереження, ведення, відстеження або перехресна перевірка.

Нижче наведено деякі під-навички або дії, необхідні для фактичного виконання завдання з моніторингу:

- Управління увагою процедури та методи для спрямування уваги пілота на певне місце і конкретний час;
- Перехресна перевірка - порівняння окремих незалежних джерел інформації для підтвердження чи спростування розуміння, похідного від вихідного джерела, простіше - "пошук іншої думки".

Оскільки «моніторинг» є дуже широким терміном, існує багато завдань екіпажу, котрі передбачають моніторинг.

Між іншим, пілоти зобов’язані контролювати стан літальних систем, конфігурацію літака, шлях польоту та дії іншого пілоту на майданчику. Слід часто проводити моніторинг одночасно з іншими завданнями, такими як, управління повітряними суднами, внесення даних та спілкування з УПР.

На рисунку 1.1 показано всю фізичну систему керуванням літака.

Рисунок 1.1 – Система керування літаком між двома пілотами

Робоча група одразу зрозуміла, що навички, пов’язані з найширшим поняттям моніторингу, охоплюють майже весь набір навичок TEM.

Пілоти під час кожного польоту складають вичерпний звіт про моніторинг довгого та важкого завдання.

Керування маршрутом польоту літака, у тому числі енергетичний рівень літака - це основна відповідальність пілота. На жаль, багато пілотів пов’язують управління польотом, просто керуючи траєкторією польоту, або через ручні керуючі входи (включаючи введення тяги), або маніпулювання різними рівнями автоматизованої системи.

1.2 Аналіз методу моніторингу авіарейсів

Новий метод стеження, який поступово впроваджується в усьому світі, забезпечує набагато більшу частоту, точність і охоплення при менших витратах. Вона називається ADS-B [5], що означає "Автоматичне залежне спостереження-мовлення".

За допомогою ADS-B бортове обладнання визначає місцеположення повітряного судна за допомогою супутникової навігації, і кожні півсекунди передає цю GPS-інформацію разом з іншими даними, включаючи висоту, швидкість і курс, а також ідентифікаційний код. Це називається «ADS-B Out». Зліт або посадка можуть бути визначені на підставі швидкості, висоти і розташування.

Революційний елемент ADS-B полягає в тому, що сигнали можуть бути отримані за допомогою обладнання, вартість якого складає всього 100 доларів США (набагато дешевше, ніж установка радара). Незашифровані сигнали, що передаються з частотою 1090 МГц, можна приймати в радіусі близько 200 миль. В даний час існують десятки тисяч таких приймачів, в основному експлуатуються авіалюбителями, які пересилають отримані дані в комерційні та некомерційні служби спостереження, іноді за помірну винагороду.

Використання власного обладнання може добре працювати локально, як це описав Джон Кіф, який хотів знати, що вертольоти робили над ним в Нью-Йорку.

Шляхом зіставлення цих локальних точок з іншими даними можна створити всеосяжну картину польотів.

Отримана картина не завжди повна. Є мертві зони, де взагалі немає приймачів, такі як пустелі, океани, полярні крижані шапки і мало розвинені країни. Супутникові приймачі ADS-B допоможуть з часом вирішити цю проблему. Крім того, зростає кількість наземних приймачів ADS-B. FlightAware, одний з найбільших сайтів відстеження, використовує до 20 000 приймачів.

Принцип роботи методу ADS-B зображено на рисунку 1.2.

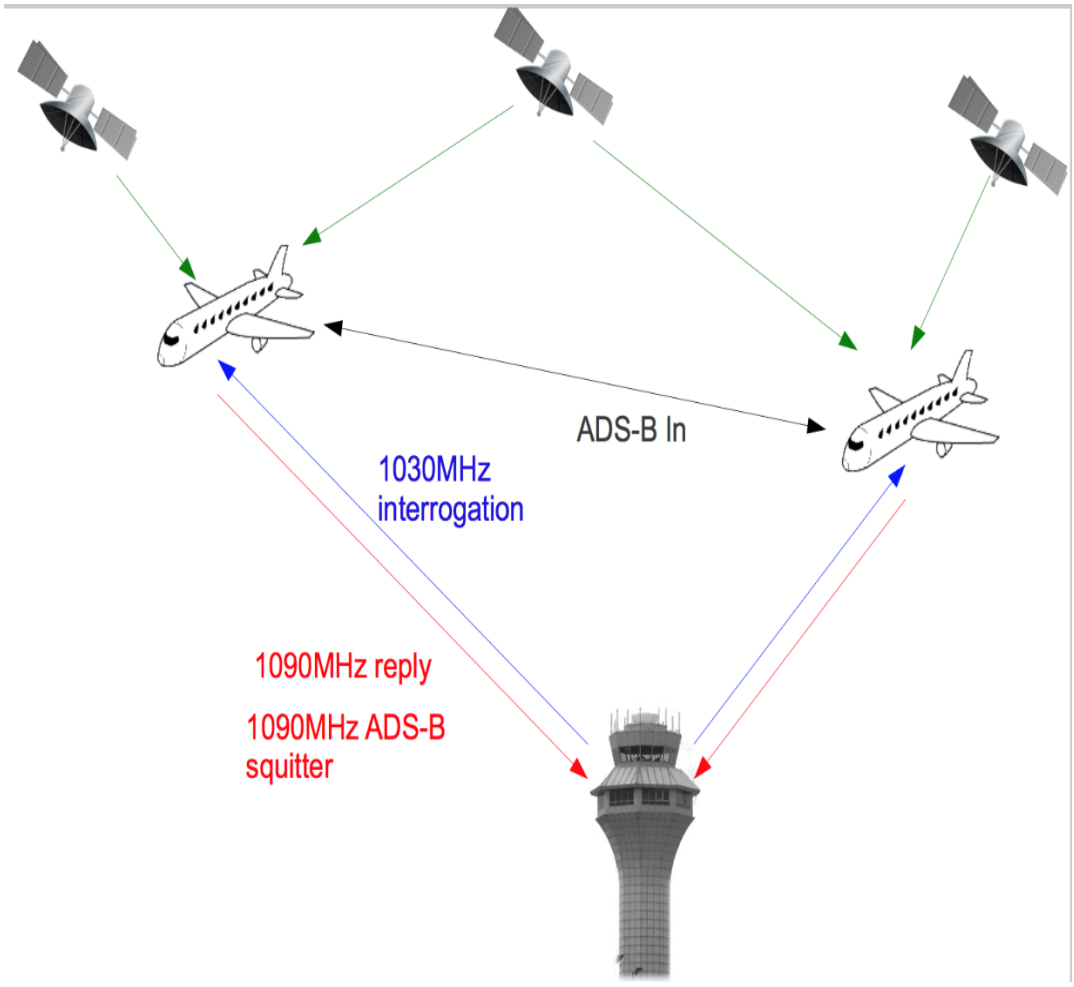


Рисунок 1.2 – Принцип роботи методу ADS-B

Сервіси авіарадарів працюють з використанням методу ADS-B:

- повітряне судно визначає своє місце розташування за допомогою GPS-навігатора;

- борт, обладнаний транспондером, передає в ефір повідомлення, в яких містяться всі дані про нього;
- наземний приймач приймає всі дані і передає їх на сервер онлайн-радара.

З ростом популярності радарів, пов’язаної з тотальною комп’ютеризацією і можливістю користуватися онлайн послугами навіть з мобільного пристрою, їх кількість з кожним роком збільшується. Питанням, як відстежити літак, тепер мало кого здивуєш, оскільки процес цей доступний будь-якій людині, зацікавленій в отриманні актуальної інформації про пересування того чи іншого повітряного судна.

1.3 Аналіз існуючих засобів моніторингу авіарейсів

На сьогоднішній день, існує великий перелік як вільно доступних так і платних засобів моніторингу авіарейсів. Серед таких систем можна виділити найпопулярніші онлайн-радари:

- Flightradar24;
- FlightAware;
- Plane Finder.

Flightradar24 - інтернет-ресурс, створений 2006 року, шведською фірмою Travel Network, який дає можливість в режимі реального часу спостерігати за літаками на зручній карті світу. При натисканні на повітряне судно виводиться його детальна інформація: тип літака, бортовий номер, авіакомпанія, місце відправлення та посадки тощо. В додаток є пошук літака за номером рейсу. Кожні кілька секунд вся інформація про всі літаки оновлюється та змінюється геолокація повітряного засобу на карті. Доступний запис історії польотів за минулі дні, тому є функція перегляду будь-якого літака: куди він летів, з якою швидкістю, та на якій висоті.

Розглянемо принцип роботи та інтерфейс інтернет-ресурсу Fightradar24 на рисунку 1.3.

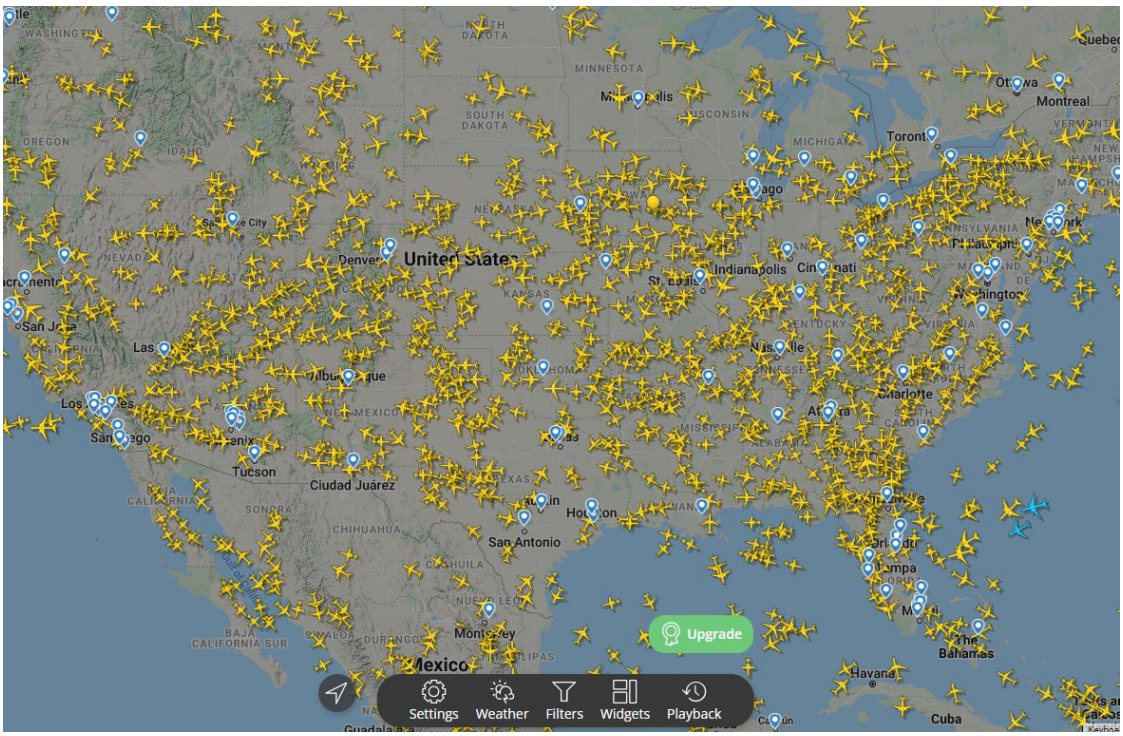


Рисунок 1.3 – Інтерфейс роботи додатку Fightradar24

До основних переваг можна віднести:

- Доступність;
- Простота у використанні;
- Надійність;
- Багатофункціональність;
- Популярність.

FlightAware [6] - американська багатонаціональна технологічна компанія, яка надає дані та продукти відстеження польотів у реальному часі, історичні й прогнозовані дані. Наразі це найбільша у світі платформа відстеження польотів із мережею з понад 32 000 наземних станцій ADS-B у 200 країнах. FlightAware також надає авіакомпаніям, операторам аеропортів і розробникам програмного забезпечення авіаційні дані та прогнозовані приблизні терміни прибуття. Компанія є приватною, зі штаб-квартирою в Х'юстоні та офісами продажу в Нью-Йорку, Остіні, Сінгапурі та Лондоні. Крім того, сайт містить послуги для пілотів, такі як планування польотів, авіаційні новини, фотографії та авіаційний дискусійний форум. Користувачі можуть безкоштовно зареєструватися на сайті, що додай

функцій, а також можливість брати участь у спільнотах. Реєстрація дозволяє користувачам завантажувати фотографії, надсилати новини, пов'язані з авіацією брати участь у дискусійних форумах та налаштовувати сповіщення про літаки. FlightAware повідомляє про понад десять мільйонів зареєстрованих користувачів.

Розглянемо принцип роботи та інтерфейс додатку FlightAware на рисунку 1.4.

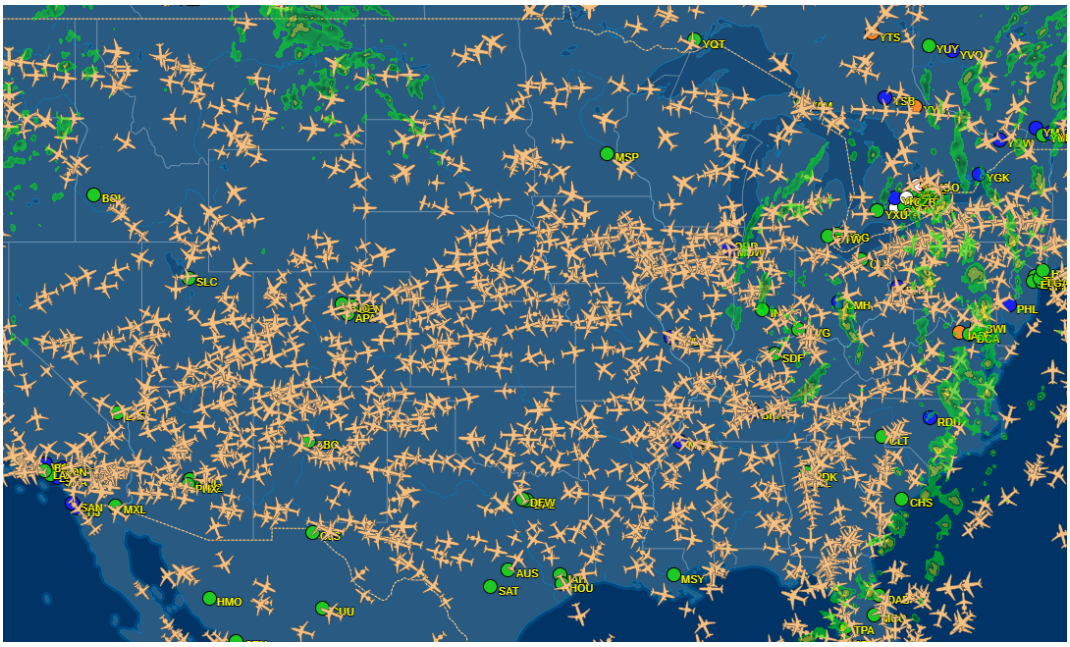


Рисунок 1.4 – Інтерфейс роботи додатку FlightAware

До основних переваг можна віднести:

- Безкоштовне використання;
- Можливість зареєструватися та отримати більше функцій;
- Широкий перелік вибору мов.

Plane Finder — це служба відстеження польотів у реальному часі у Великобританії, запущена у 2009 році, яка може відображати дані польотів у всьому світі. Доступні дані включають номери рейсів, швидкість руху літака, його висоту і пункт призначення. Є кілька варіантів послуг, які доступні у вигляді мобільних додатків, включаючи безкоштовні, преміум-версії 3D та доповненої реальності. Доступ до карти відстеження польотів і бази даних можна отримати за допомогою веб-браузерів. Plane Finder дозволяє зареєстрованим користувачам

ділитися своїми даними ADS-B і MLAT через клієнт, доступний для macOS, Windows і Linux. Підтримує діаграми VFR від NATS і був першим великим додатком для відстеження польотів, який представив функцію відтворення, що дозволяє користувачам відтворювати польоти, датовані 2011 роком.

Розглянемо принцип роботи та інтерфейс веб-сайту Plane Finder на рисунку 1.5.

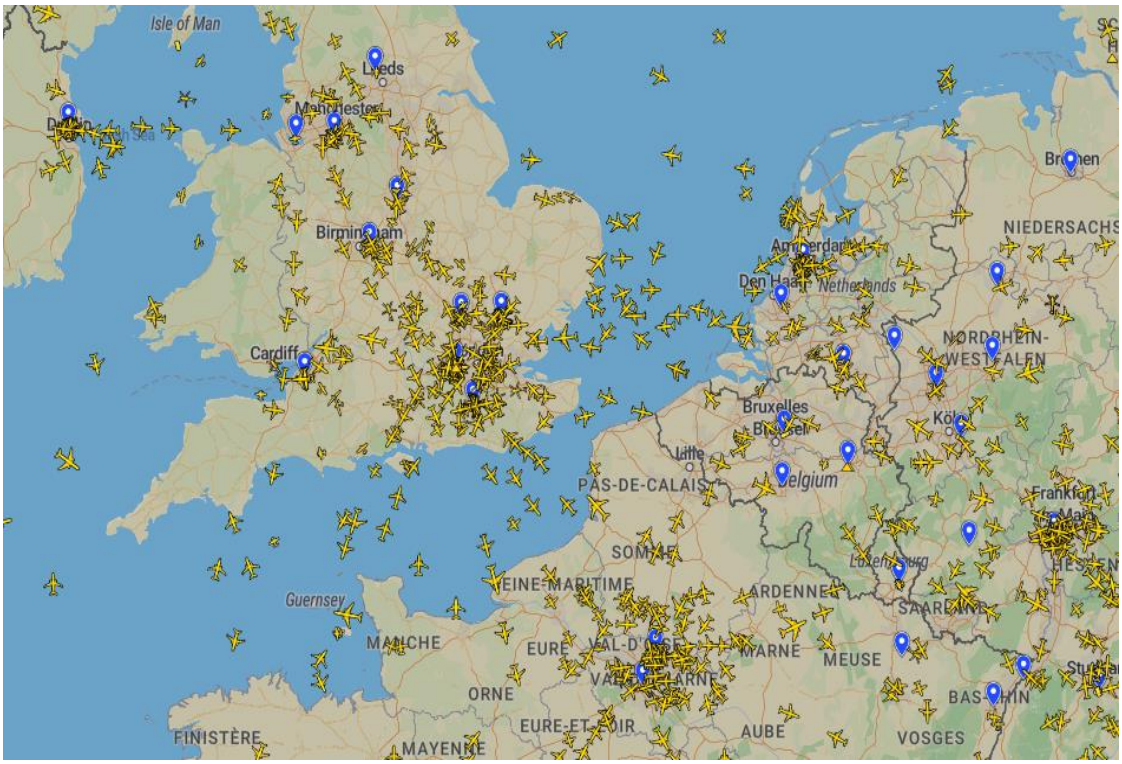


Рисунок 1.5 – Інтерфейс роботи веб-сайту Plane Finder

До основних переваг можна віднести:

- Безкоштовна демо-версія;
- Кроссплатформеність;
- Зручний сайт;
- Наявність декількох мов.

Розглянувши дані аналоги було створено таблицю (Таблиця 1.1) для порівняння їхнього функціоналу з програмним засобом моніторингу авіарейсів «Flight Monitoring».

Таблиця 1.1 – Порівняльні характеристики програмних додатків

	Fightradar24	FlightAware	Plane Finder	Flight Monitoring
Простота використання	1	1	1	1
Кількість мов	0	0	1	1
Кроссплатформеність	1	1	0	1
Безкоштовне використання	1	1	1	1
Багатофункціональність	1	1	1	1
Сумарний коефіцієнт	3	3	3	5

Аналізуючи вищенаведені дані, Flight Monitoring має вищий сумарний коефіцієнт відносно аналогів: Fightradar24, FlightAware, Plane Finder на 40% (100%-3/5*100% = 40%).

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує кращу, у порівнянні з безкоштовними аналогами, ефективність.

1.4 Постановка задачі для розробки програмного засобу моніторингу авіарейсів

Проаналізувавши питання розробки програмного засобу моніторингу авіарейсів, було визначено завдання, які необхідно виконати:

- провести аналіз існуючих методів і засобів моніторингу авіарейсів для відслідковування повітряних засобів у просторі;
- покращити алгоритм обробки анімації літаків;
- підвищити продуктивність роботи користувача з засобом керування повітряного простору;
- збільшити безпеку пасажирів під час польоту;

- розробити програмні компоненти та систему візуалізації моніторингу авіарейсів на основі запропонованих методів;
- провести тестування програмного продукту.

Розроблений засіб моніторингу авіарейсів, повинен задовольняти наступні функціональні вимоги:

- Користувач перед користуванням програми, повинен потрапити на екран запуску програмного засобу;
- Даними для вводу у пусті поля, мають бути лише літери латиниці та арабські цифри;
- При пустому полі, користувач має бачити, який саме тип данних потрібно заповнити;
- При неправильно введених данних, користувач має бачити, попередження;
- Користувач після екрану запуску, повинен потрапити у головне меню програмного засобу;
- Після входу в систему, користувач має бачити справа-зверху кнопку виходу з системи;
- Користувач повинен мати змогу додати новий авіарейс;
- Користувач повинен мати змогу додати нового пасажира на борт літака;
- Користувач повинен бачити таблицю поточних та майбутніх авіарейсів;
- Користувач в таблицях повинен бачити унікальний номер літака; аеропорт відльоту та прильоту, час відльоту та прильоту;
- Користувач має бачити мапу всіх перельотів;
- Користувач повинен мати змогу вийти з системи.

Також засіб моніторингу авіарейсів, повинен задовольняти наступні нефункціональні вимоги:

- Унікальний логотип;
- Повнота дизайну;
- Адаптивність дизайну;

- Прозора для користувача навігація та цільова орієнтація в програмі;
- Зрозумілість і чіткість користувачем текстів, іконок, та кнопок;
- Швидкість навчання при роботі з засобом, для чого необхідно використовувати переважно стандартні елементи взаємодії, їх традиційне або загальноприйняте їх розташування;
- Відповіді на запити користувача повинні бути максимально швидкими, 1-2 секунди;
- Система має бути легко переносною на інші платформи.

1.5 Висновки

У першому розділі було розглянуто стан питання моніторингу авіарейсів на сьогоднішній день.

Проаналізовано існуючі аналоги та проведено їх порівняння з розроблюваним програмним продуктом. У результаті порівняння було доведено доцільність розробки власного програмного додатку.

Проведено аналіз існуючих підходів до вирішення поставленої задачі. У результаті аналізу було обрано алгоритм обробки анімації літаків для вирішення задачі.

Сформульовано основні завдання, функціональні та нефункціональні вимоги, які необхідно виконати для розробки програмного додатку.

2 РОЗРОБКА МЕТОДІВ, СТРУКТУРИ ТА АЛГОРИТМІВ МОНІТОРИНГУ АВІАРЕЙСІВ

2.1 Обґрунтування вибору інтерфейсу

До основних видів інтерфейсу відносять:

- графічні інтерфейси користувача;
- інтерфейси командного рядка;
- веб-інтерфейси.

Графічний інтерфейс користувача – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації. Робота даного виду інтерфейсу заснована на абстракціях події і обробці потоку подій. Події можна поділити на три класи:

- події, що генеруються при зміні стану фізичних пристроїв користувацького введення;
- події, що генеруються при зміні стану відображуваних вікон;
- події підтримки взаємодії між програмами - клієнтами.

На рисунку 2.1 зображено приклад графічного інтерфейсу користувача.

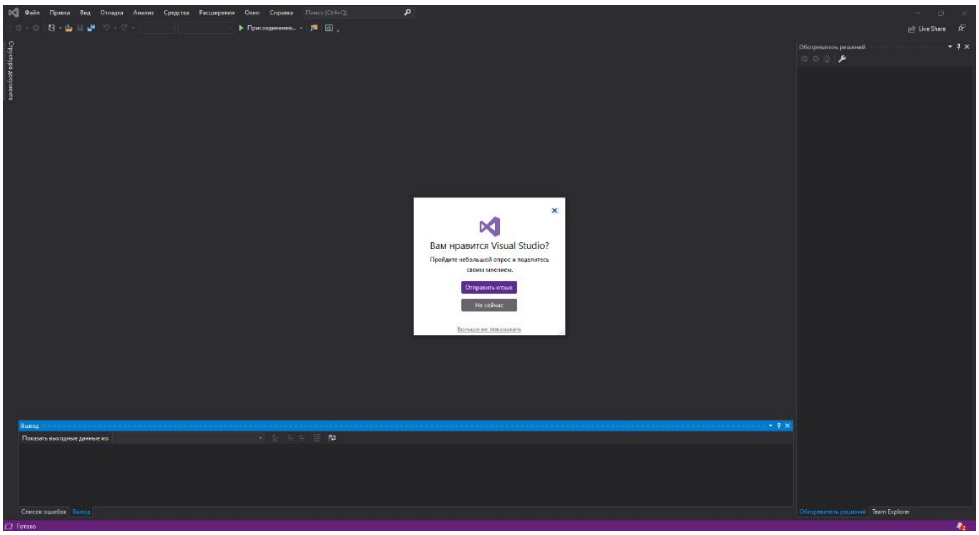


Рисунок 2.1 – Приклад графічного інтерфейсу користувача

Інтерфейс командного рядка [7] – це різновид текстового інтерфейсу користувача й комп’ютера, в якому інструкції комп’ютеру можна дати тільки введенням із клавіатури текстових рядків (команд). Також відомий під назвою консоль. Інтерфейс командного рядка може бути протиставлений системам управління програмою на основі меню чи різних реалізацій графічного інтерфейсу. Формат виводу інформації в інтерфейсі командного рядка не регламентується, звичайно це простий текстовий вивід, але може бути й графічним, звуковим виводом тощо.

На рисунку 2.2 зображено приклад інтерфейсу командного рядка.

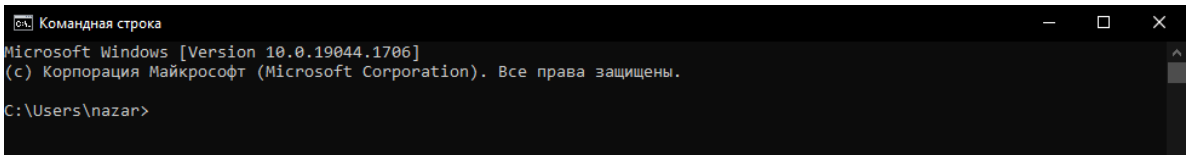


Рисунок 2.2 – Приклад інтерфейсу командного рядка

Веб-інтерфейс [8] – це тип інтерфейсу, в якому навігація виконується в рамках одного або декількох додатків з використанням текстових або візуальних гіперпосилань. До найбільш поширених компонентів веб-інтерфейсу відносять: заголовки, навігаційні панелі і візуальні або текстові гіперпосилання, впорядковані різними способами.

Графічний інтерфейс користувача надає можливість користувачу легко та зрозуміло адаптуватися до роботи з програмою через наявність графічних об'єктів, таких як вікна, значки, меню, кнопки та інше. Даний інтерфейс легкий для сприйняття новими користувачами але у свою чергу він потребує значно більших затрат пам'яті ніж текстовий інтерфейс.

Інтерфейс командного рядка дозволяє керувати програмами, які не мають графічного типу відображення інтерфейсу, а внаслідок такий інтерфейс має зручну систему трасування яка полягає у тому що, користувач завжди може переглянути попередньо введені команди. Інтерфейс командного рядка також має значні

недоліки, до яких можна віднести складність розуміння новими користувачами та обов’язкове використання заданого синтаксису роботи з програмою.

Веб-інтерфейс зручний тим, що дає можливість вести спільну роботу співробітникам, які працюють на різних локаціях. Також даний тип інтерфейсу дозволяє користувачу працювати не навантажуючи локальний комп’ютер. Веб-інтерфейс дає можливість універсального віддаленого доступу до служб та пристроїв, у цьому технології практично нема альтернатив. Але водночас, оскільки такий інтерфейс доступний усім, постають серйозні питання забезпечення безпеки, зокрема автентифікація та авторизація користувачів, шифрування переданих даних від сторонніх очей, модерація вмісту тощо.

На рисунку 2.3 зображено приклад веб-інтерфейсу користувача.

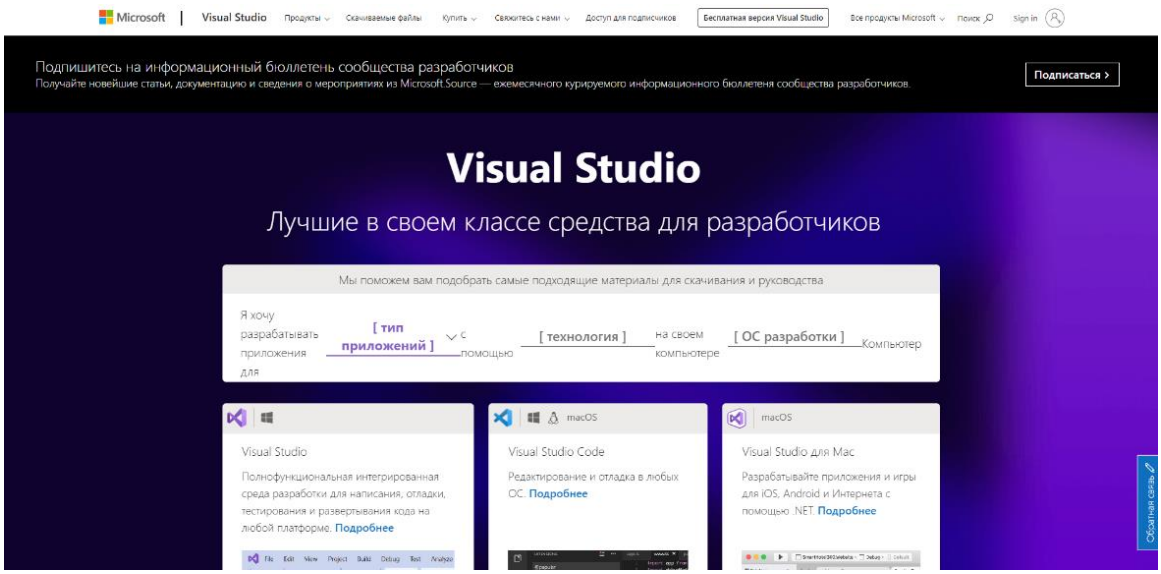


Рисунок 2.3 – Приклад веб-інтерфейсу користувача

Отже, програмний додаток повинен бути зручним у використанні для нових користувачів тому необхідно обрати інтерфейс, який підтримує роботу з графічними елементами, тому інтерфейс командного рядка не може бути використаний у розробці. Також даний програмний додаток не передбачає роботи кількох користувачів у мережі, тому веб-інтерфейс ми теж виключаєм зі списку. Тому було вирішено обрати графічний інтерфейс для подальшої розробки.

2.2 Розробка структури інтерфейсу

Інтерфейс користувача [9] складається з вікна запуску програми, одного головного вікна, меню керування літаками, меню списків літаків, діалогового вікна додавання нового рейсу та діалогового вікна додавання нового пасажиру на борт літака. Головне вікно є масштабоване і призначене для відображення інформації про літаки.

На рисунку 2.4 зображено приклад графічної схеми головного вікна.

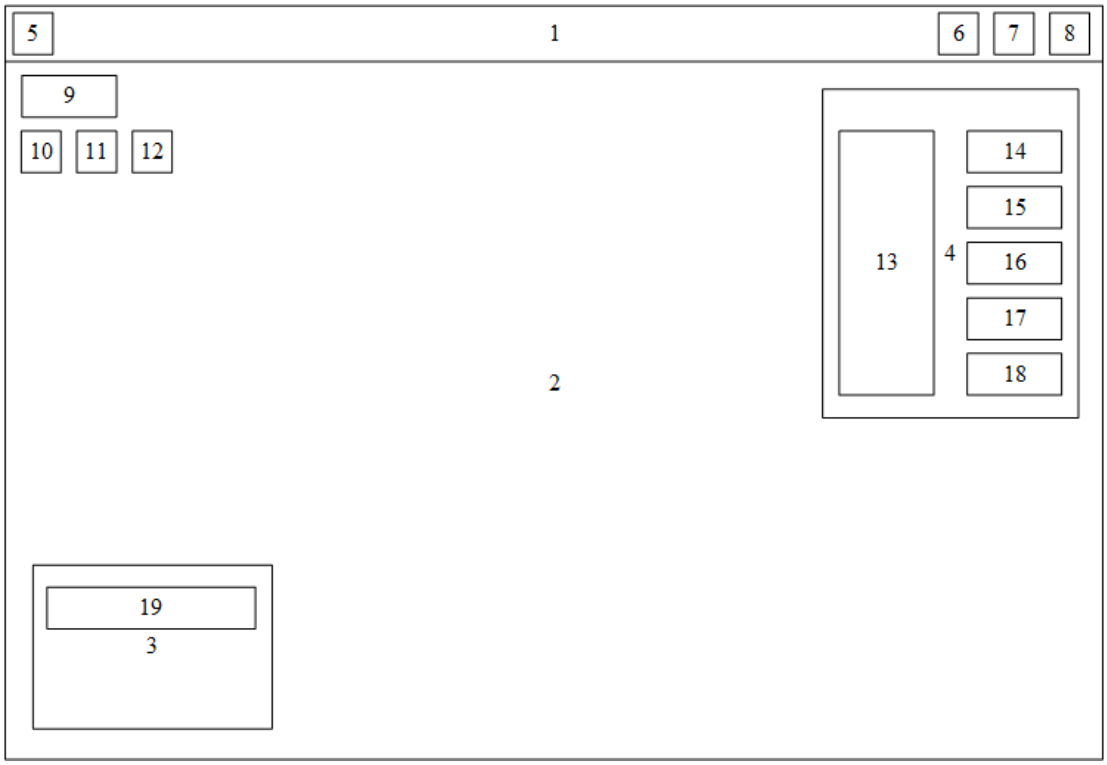


Рисунок 2.4 – Графічна схема головного вікна

Елементи головного вікна:

1. Стрічка заголовку;
2. Робоча область;
3. Область керування літаками;
4. Область інформації про літаки;
5. Іконка програми;
6. Системна кнопка згорнути;

7. Системна кнопка розгорнути;
8. Системна кнопка закрити;
9. Годинник;
10. Пауза;
11. Пришвидшити;
12. Пришвидшити втричі;
13. Список унікальних номерів літаків;
14. Інформація про перший літак;
15. Інформація про другий літак;
16. Інформація про третій літак;
17. Інформація про четвертий літак;
18. Інформація про п’ятий літак;
19. Кнопка керування літаками.

Меню керування літаками містить інформацію про поточні, майбутні рейси та можливість додавання нового літака та пасажира. На рисунку 2.5 зображено графічну схему меню керування літаками.

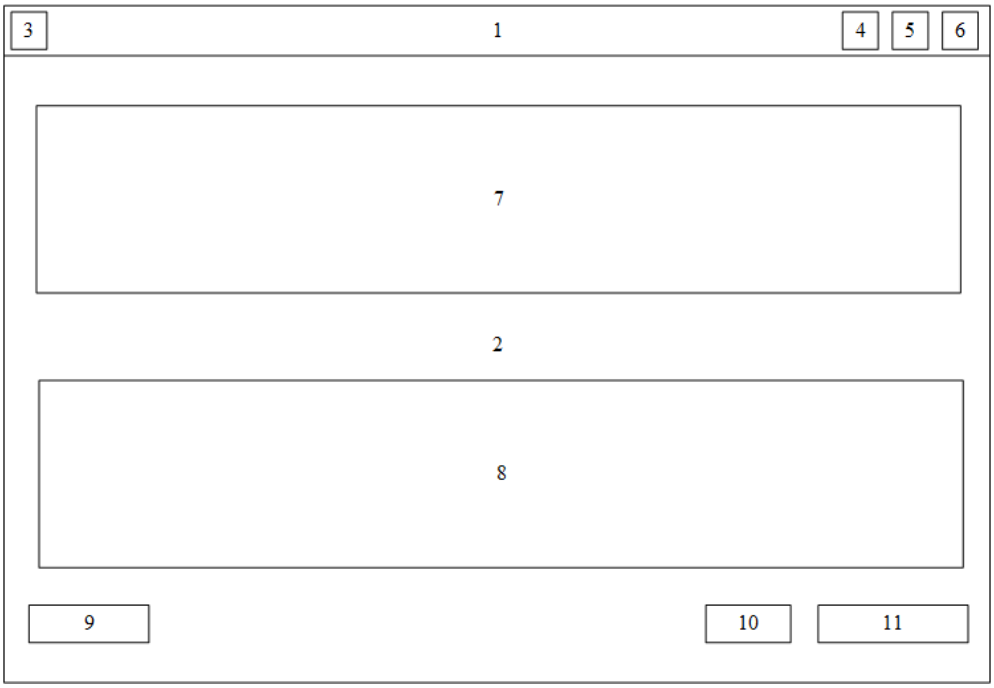


Рисунок 2.5 – Графічна схема меню керування літаками

Елементи меню керування літаками:

1. Стрічка заголовку;
2. Робоча область;
3. Іконка програми;
4. Системна кнопка згорнути;
5. Системна кнопка розгорнути;
6. Системна кнопка закрити;
7. Область інформації поточних рейсів;
8. Область інформації майбутніх рейсів;
9. Кнопка повернення до радару;
10. Кнопка додавання нового рейсу;
11. Кнопка додавання нового пасажира на борт літака.

Меню інформації про літаки містить інформацію про пасажирів, ім’я, вік та країну проживання. На рисунку 2.6 зображено графічну схему меню інформації про літак.

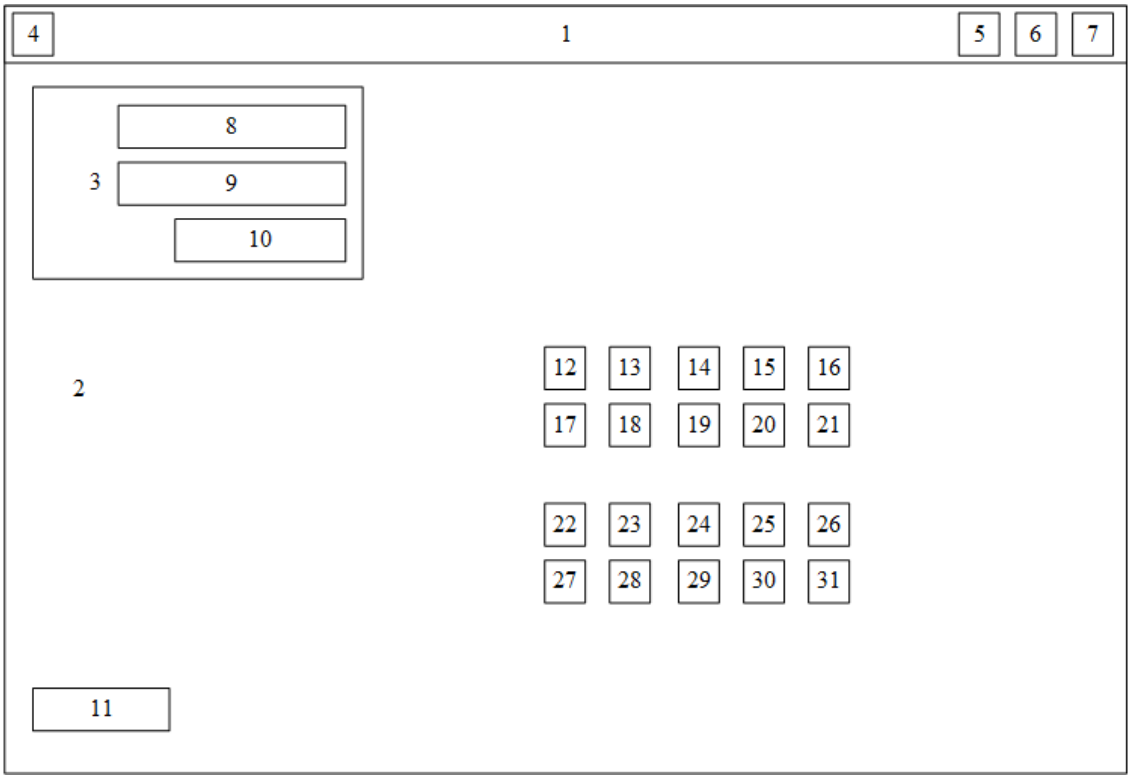


Рисунок 2.6 – Графічна схема інформації про літак

Елементи меню інформації про літаки:

1. Стрічка заголовку;
2. Робоча область;
3. Область інформації про пасажира;
4. Іконка програми;
5. Системна кнопка згорнути;
6. Системна кнопка розгорнути;
7. Системна кнопка закрити;
8. Поле відображення ім'я пасажира;
9. Поле відображення віку пасажира;
10. Поле відображення країни проживання пасажиру;
11. Кнопка повернення до радару;
12. Кнопка з інформацією про місце номер 1;
13. Кнопка з інформацією про місце номер 2;
14. Кнопка з інформацією про місце номер 3;
15. Кнопка з інформацією про місце номер 4;
16. Кнопка з інформацією про місце номер 5;
17. Кнопка з інформацією про місце номер 6;
18. Кнопка з інформацією про місце номер 7;
19. Кнопка з інформацією про місце номер 8;
20. Кнопка з інформацією про місце номер 9;
21. Кнопка з інформацією про місце номер 10;
22. Кнопка з інформацією про місце номер 11;
23. Кнопка з інформацією про місце номер 12;
24. Кнопка з інформацією про місце номер 13;
25. Кнопка з інформацією про місце номер 14;
26. Кнопка з інформацією про місце номер 15;
27. Кнопка з інформацією про місце номер 16;
28. Кнопка з інформацією про місце номер 17;

29.Кнопка з інформацією про місце номер 18;

30.Кнопка з інформацією про місце номер 19;

31.Кнопка з інформацією про місце номер 20.

Діалогове вікно додавання нового рейсу складається з: унікального номеру літака, випадаючого списку міста відправлення та прибуття. На рисунку 2.7 зображено графічну схему діалогового вікна додавання нового рейсу.

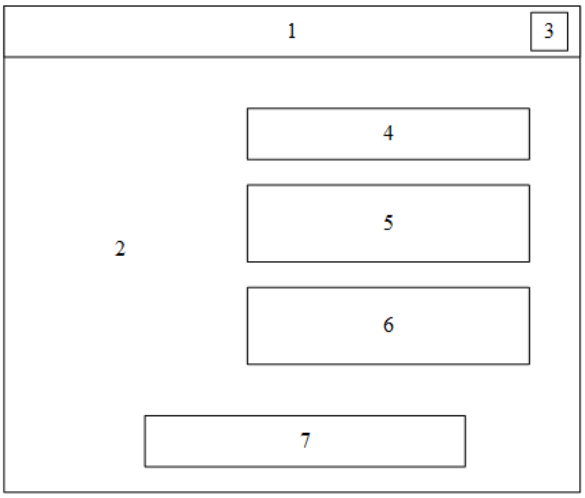


Рисунок 2.7 – Графічна схема діалогового вікна додавання нового рейсу

Елементи діалогового вікна додавання нового рейсу:

1. Стрічка заголовку;
2. Робоча область;
3. Унікальний номер літака;
4. Системна кнопка закрити;
5. Поле вводу унікального номеру літака;
6. Випадаючий список міста відправлення;
7. Випадаючий список міста прибуття;
8. Кнопка додати новий рейс.

Діалогове вікно [10] додавання нового пасажиру на борт літака складається з: поля унікального номеру літака, поле вводу ім’я пасажира, поле вводу віку, поле

вводу проживання пасажира. На рисунку 2.8 зображено графічну схему діалогового вікна додавання нового пасажира на борт літака.

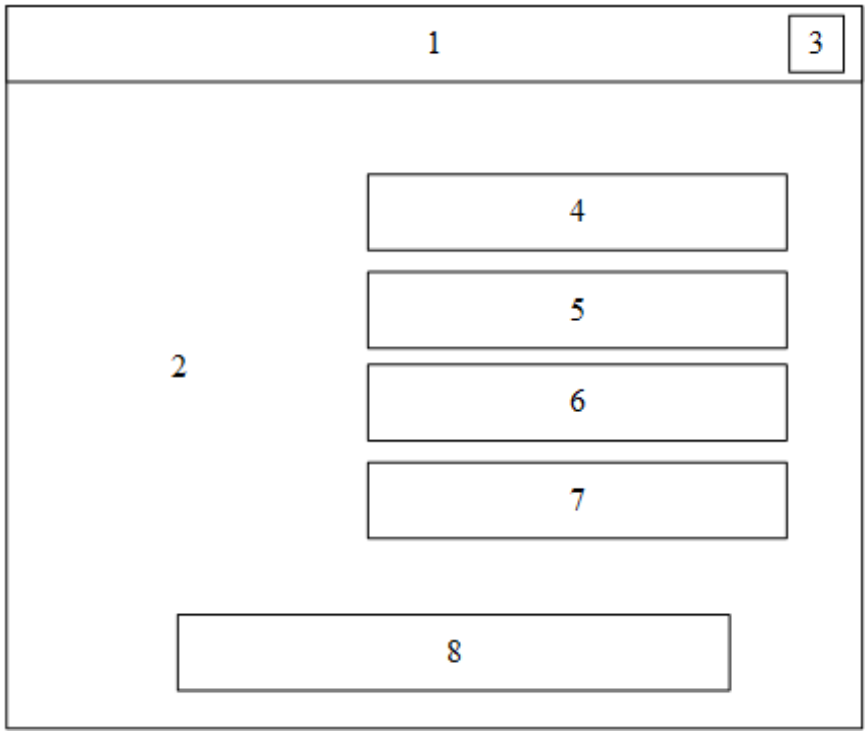


Рисунок 2.8 – Графічна схема діалогового вікна додавання нового рейсу

Елементи діалогового вікна додавання нового рейсу:

1. Стрічка заголовку;
2. Унікальний номер літака;
3. Робоча область;
4. Системна кнопка закрити;
5. Поле вводу унікального номеру літака;
6. Випадаючий список міста відправлення;
7. Випадаючий список міста прибуття;
8. Кнопка додати новий рейс.

Розробка графічного інтерфейсу додатку була проведена у середовищі розробки IntelliJ IDEA з використанням технології Swing GUI-builder.

2.3 Розробка методу обробки анімації літаків

Для того, щоб літаки коректно та відповідно до вимог відображались та графічно оброблювались на карті, було вирішено розробити метод обробки анімації літаків. Цей метод містить у собі те, що спочатку програма ініціалізує компоненти, перевіряє прибуття літака на карті, знаходить літак, який замінить поточний. Далі метод обробки анімації літаків встановлює число, яке більше ніж системний час, встановлює літак що прибув, на майбутній літак з найшвидшим часом віправлення та прибирає його з майбутніх площин. Тепер на карті генерується літак на поточному місці, встановлюється новий значок розташування та оновлюється таблиця з рейсами. Метод обробки анімації літаків зображено на рисунку 2.9.

Рисунок 2.9 – Метод обробки анімації літаків

Результатом виконання даного методу є згенерований значок літака, який переміщається по карті, та оновлена таблиця з авіарейсами.

2.4 Розробка алгоритмів роботи додатку

Щоб розробити додаток для моніторингу авіарейсів необхідно розробити загальний алгоритм [11] роботи додатку та алгоритм обробки анімації літаків.

Розроблено загальний алгоритм роботи додатку який полягає в наступному: згідно з обраним пунктом головного меню викликаються необхідні функції

базового класу головного вікна програми. Якщо користувач натискає на кнопку завантажити програму, викликається функція завантаження програми та користувач потрапляє на головне вікно програмного засобу, інакше користувач натискає системну кнопку вихід, та завершує користування програмним засобом. Далі аналогічною процедурою все відбувається з іншимми функціями Робота програми завершується коли користувач обрав відповідний пункт меню або натиснув кнопку вихід. Блок-схема алгоритму зображена на рисунку 2.9.

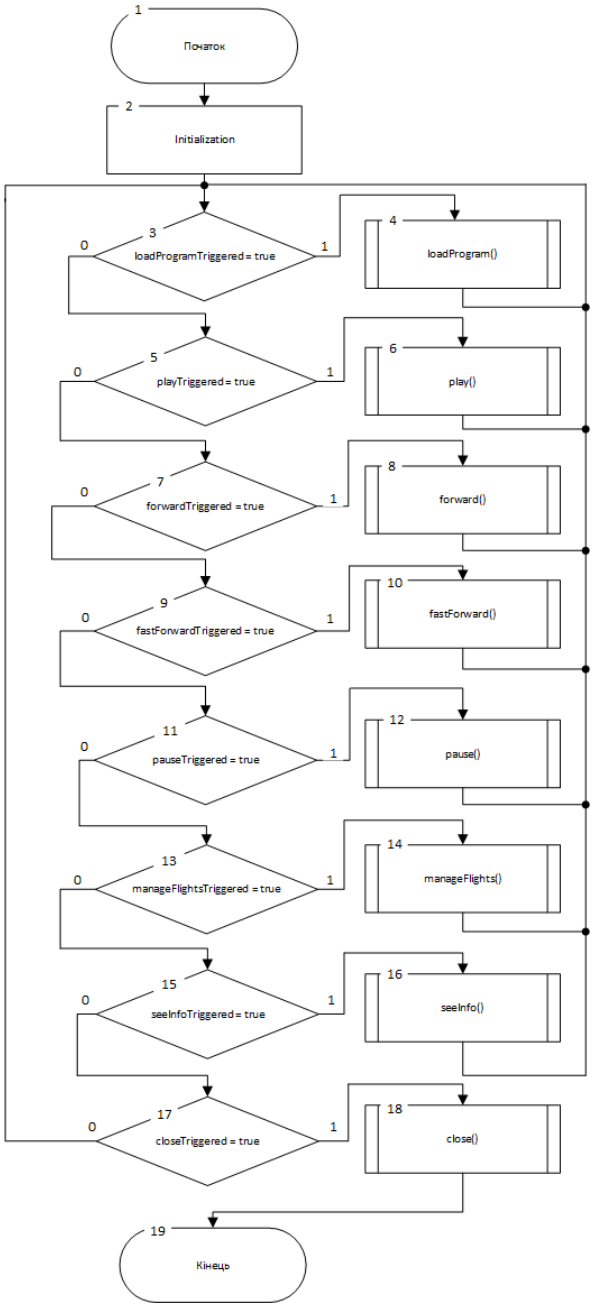


Рисунок 2.9 – Блок-схема алгоритму роботи додатку

Наступний етап – розробка алгоритму обробки анімації літаків. Відповідно до наступної блок-схеми спочатку ініціалізуються локації, літаки, потім запускається цикл та перевіряє чи прибув літак, якщо прибув то знаходить площину яка замінить поточну, далі визначає число, яке більше ніж будь-який можливий системний час. Блок-схема алгоритму зображена на рисунку 2.10.

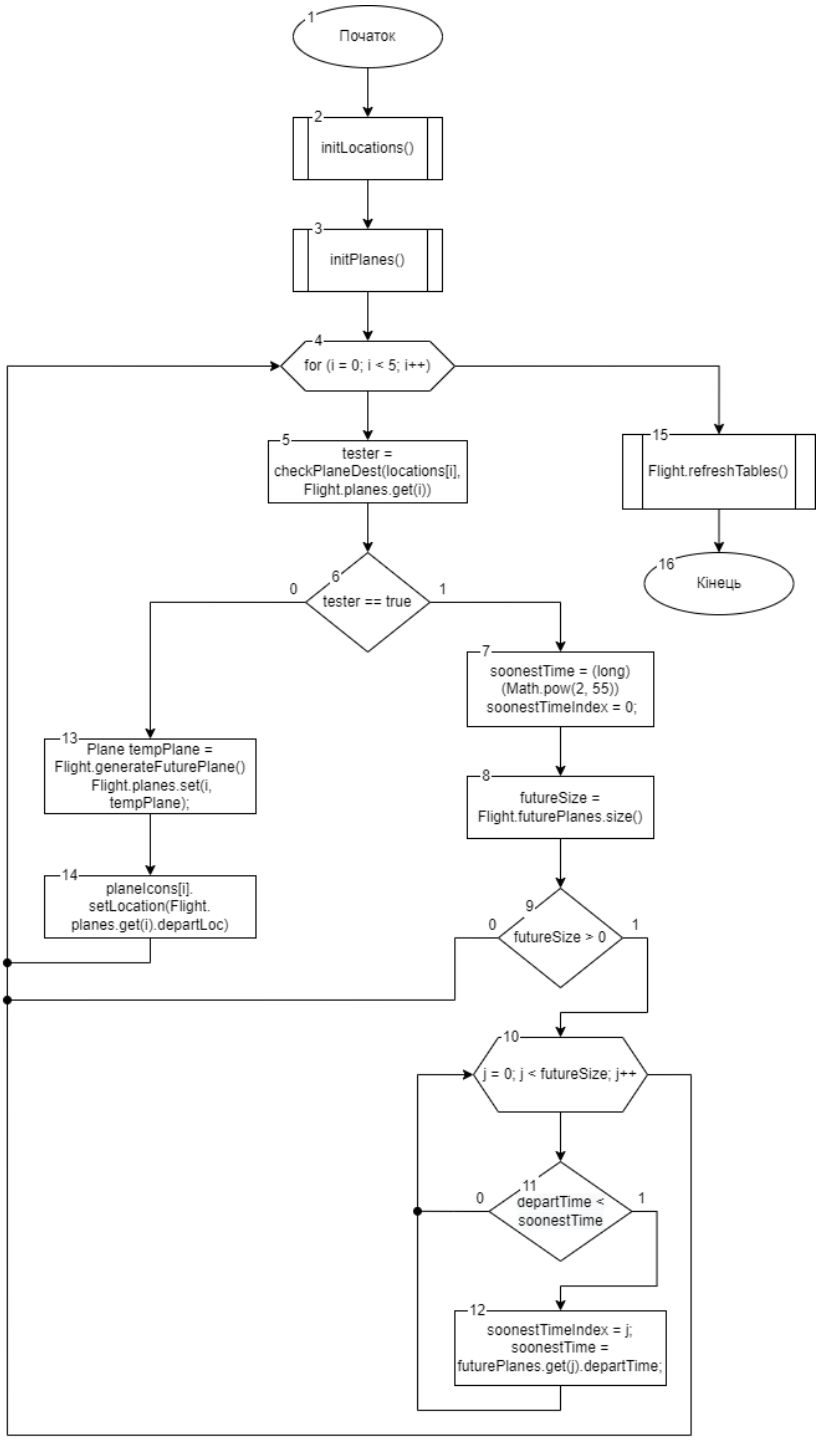


Рисунок 2.10 – Блок-схема алгоритму обробки анімації літаків

Визначає індекс, додає піктограму до масиву, отримує розмір масиву. Якщо найшвидший час менший поточного, встановлює індекс. Далі встановлює літак що прибув, на майбутній літак, із найшвидшим часом вильоту, прибирає його з майбутніх площин, видаляє рядок. Якщо майбутніх літаків немає, генерує літак на місці поточного, встановлює нове розташування значка та оновлює таблицю з рейсами.

2.5 Висновки

У другому розділі було проведено обґрунтування вибору інтерфейсу програмного засобу. У результаті чого було обрано графічний тип інтерфейсу.

Проведено розробку структури інтерфейсу.

Розроблено алгоритм загальної роботи додатку та алгоритм обробки анімації літаків.

Також розроблено метод обробки анімації літаків.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ МОНІТОРИНГУ АВІАРЕЙСІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу.

Для того, щоб створити програмний продукт на належному рівні потрібно зробити правильний вибір технології для його реалізації тому, що саме найбільш підходящі технології дадуть можливість значно полегшити процес реалізації та створення програмного продукту. Для розробки програмного засобу було прийнято рішення використовувати мову програмування Java.

Java [12] – високорівнева, заснована на класах, об’єктно-орієнтована мова програмування, яка розроблена так, щоб мати якомога менше залежностей реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники могли написати один раз, а потім запускати майже на будь-якому пристрої. Це пояснюється тим, що скомпільований код Java може працювати на всіх платформах, які підтримують Java, без необхідності перекомпіляції. Програми Java зазвичай компілюються у байт-код, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від базової архітектури комп’ютера. Синтаксис Java подібний до C і C++, але має менше засобів низького рівня, ніж будь-який з них. Середовище виконання Java надає динамічні можливості (такі як відображення та модифікація коду під час виконання), які зазвичай недоступні в традиційних скомпільованих мовах. Станом на 2019 рік Java була однією з найпопулярніших мов програмування, що використовуються особливо для веб-додатків, клієнт-серверів тощо. Головною особливістю цієї мови є те, що вона тісно пов’язана з платформою .NET, яка містить досить об’ємну стандартну бібліотеку класів, що значно спрощує написання програмного коду.

Однією з задач дизайну Java є переносимість, що означає, що програми, написані для платформи Java, повинні працювати подібним чином на будь-якій комбінації апаратного забезпечення та операційної системи з належною підтримкою часу виконання. Це досягається шляхом компіляції коду мови Java у

проміжне подання, яке називається байт-кодом Java, замість безпосереднього машинного коду, специфічного для архітектури. Інструкції байт-коду Java аналогічні машинному коду, але вони призначені для виконання віртуальною машиною (VM), написаною спеціально для апаратного забезпечення хоста. Кінцеві користувачі зазвичай використовують Java Runtime Environment (JRE), встановлену на своєму пристрої для автономних програм Java або веб-браузер для Java-аплетів. Стандартні бібліотеки надають загальний спосіб доступу до специфічних для хоста функцій, таких як графіка, потоки та мережа. Використання універсального байт-коду спрощує перенесення. Однак накладні витрати на інтерпретацію байт-коду в машинні інструкції змушували інтерпретовані програми майже завжди працювати повільніше, ніж рідні виконувані файли. Компілятори Just-in-time (JIT), які компілюють байт-код у машинний код під час виконання, були представлені на ранньому етапі. Компілятор Hotspot Java насправді є двома компіляторами в одному; і з GraalVM (включений, наприклад, у Java 11, але вилучений з Java 16), що дозволяє багаторівневу компіляцію. Сама Java не залежить від платформи і адаптована до конкретної платформи, на якій вона має працювати, за допомогою віртуальної машини Java (JVM), яка перекладає байт-код Java на машинну мову платформи.

Java використовує автоматичний збірник сміття для управління пам’яттю в життєвому циклі об’єкта. Програміст визначає час створення об’єктів, а середовище виконання Java відповідає за відновлення пам’яті, коли об’єкти більше не використовуються. Як тільки посилання на об’єкт не залишаються, недоступна пам’ять стає придатною для автоматичного звільнення збирачем сміття. Щось подібне до витоку пам’яті все ще може статися, якщо код програміста містить посилання на об’єкт, який більше не потрібен, як правило, коли об’єкти, які більше не потрібні, зберігаються в контейнерах, які все ще використовуються. Якщо викликаються методи для неіснуючого об’єкта, створюється виняток нульового покажчика. Одна з ідей, що лежить в основі моделі автоматичного управління пам’яттю Java, полягає в тому, що програмісти можуть позбутися тягаря

необхідності виконувати ручне керування пам’яттю. У деяких мовах пам’ять для створення об'єктів неявно виділяється в стеку або явно виділяється і звільняється з купи. В останньому випадку відповідальність за управління пам'яттю покладається на програміста. Якщо програма не звільняє об’єкт, відбувається витік пам’яті. Якщо програма намагається отримати доступ або звільнити пам’ять, яка вже була вивільнена, результат буде невизначеним і важко передбачити, і програма, швидше за все, стане нестабільною або аварійно завершить роботу. Це можна частково виправити за допомогою розумних покажчиків, але вони додають накладних витрат і ускладнюють. Збір сміття не запобігає витоку логічної пам’яті, тобто тих, де пам’ять все ще посилається, але ніколи не використовується. Збір сміття може статися в будь-який момент. В ідеалі це відбуватиметься, коли програма неактивна. Він гарантовано спрацьовує, якщо в купі недостатньо вільної пам’яті для виділення нового об’єкта; це може призвести до миттєвої зупинки програми. Явне керування пам'яттю неможливе в Java. Java не підтримує арифметику вказівників у стилі C/C++, де адресами об’єктів можна арифметично маніпулювати (наприклад, додаючи або віднімаючи зміщення). Це дозволяє збірнику сміття переміщувати об’єкти, на які посилаються, і гарантує безпеку та безпеку типів. Як і в C++ [13] та деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java зберігаються або безпосередньо в полях (для об'єктів), або в стеку (для методів), а не в купі, як це зазвичай вірно для непримітивних даних. типи (але див. аналіз escape). Це було свідоме рішення дизайнерів Java з міркувань продуктивності.

Також однією з вагомих частин в реалізації поставленого завдання стала розробка графічного інтерфейсу програми. Тому було вирішено використовувати бібліотеку Swing та Swing GUI-builder для швидкого створення графічного інтерфейсу на базі мови програмування Java.

Swing [14] – це набір інструментів віджетів GUI для Java. Він є частиною Java Foundation Classes (JFC) Oracle – API для надання графічного інтерфейсу користувача (GUI) для програм Java. Swing – відноситься до бібліотеки класів JFC, яка несе собою набір бібліотек для розробки графічних оболонок. Компоненти

бібліотеки підтримують специфічні модулі look-and-feel, що динамічно підключаються. За допомогою них можлива емуляція графічного інтерфейсу платформи (тобто до модуля можливо підключити інші, специфічні для даної операційної системи вигляд і поведінку). Це платформо-незалежна бібліотека, «модель–перегляд–контролер», яка дотримується моделі однопоточного програмування. Крім того, ця структура забезпечує рівень абстракції між структурою коду та графічним представленням графічного, що означає, що програму з використанням Swing можна запустити на всіх платформах, які підтримують JVM[15] також вона має дуже розподілену архітектуру, яка дозволяє підключати реалізації користувача вказаної інфраструктури інтерфейсів.

Сильна залежність Swing від механізмів виконання та непрямих шаблонів композиції дозволяє йому реагувати під час виконання на фундаментальні зміни в його налаштуваннях. Наприклад, програма на основі Swing здатна на гарячу заміну свого інтерфейсу користувача під час виконання. Крім того, користувачі можуть надати власну реалізацію зовнішнього вигляду та відчуття, що дозволяє одноманітно змінювати зовнішній вигляд існуючих додатків без будь-яких програмних змін у коді програми.

Високий рівень гнучкості Swing відображається в його властивій здатності замінювати елементи керування графічним інтерфейсом операційної системи (ОС) для відображення самого себе. Swing «малює» свої елементи керування за допомогою Java 2D API, замість того, щоб викликати набір інструментів власного інтерфейсу користувача. Таким чином, компонент Swing не має відповідного власного компонента графічного інтерфейсу ОС і може відтворювати себе будь-яким способом, можливим за допомогою базових графічних інтерфейсів.

Враховуючи всі вище перечисленні особливості та переваги, мови Java та бібліотеки Swing є найбільш доцільнішими для основної логіки програмного засобу та графічного інтерфейсу користувача.

3.2 Розробка основного модуля

Розробка основного модуля з бізнес логікою [16] – запорука реалізації програмного засобу, який буде працювати коректно та відповідатиме поставленим вимогам, як функціональним, так і нефункціональним. Даний модуль є найважливішим етапом створення додатку. Для створення бізнес логіки було використано мову програмування Java.

Цей метод створює об’єкти літака для польоту під час запуску. Метод initialGeneratePlanes – відповідає за створення об'єкту літака для польоту. Спочатку генерується унікальний номер літака та виконується п’ять разів. Потім обчислюється відстань та час польоту. Після цього генерується 66% пасажирів на борту літака. Код методу наведено на рисунку 3.1.

```
public static void initialGeneratePlanes(){
    for (int i = 0; i < 5; i++){
        String ID = Integer.toString((int)(Math.random()*900 + 100));
        int rand1 = (int)(Math.random() * 7);
        int rand2 = (int)(Math.random() * 7);
        while (rand1 == rand2){
            rand2 = (int)(Math.random() * 7);
        }
        Point departLoc = new Point(airports[0][rand1], airports[1][rand1]);
        Point arriveLoc = new Point(airports[0][rand2], airports[1][rand2]);

        long disX = Math.abs(arriveLoc.x - departLoc.x);
        long disY = Math.abs(arriveLoc.y - departLoc.y);
        long time = 0;
        if (disX >= disY){
            long dis = (disX - disY) + disY;
            time = dis * 1000;
        } else{
            long dis = (disY - disX) + disX;
            time = dis * 1000;
        }

        long departTime = System.currentTimeMillis();
        long arriveTime = departTime + time + totalFastForward;

        Passenger[] passengerArray = new Passenger[20];
        int[] seatingBooked = new int[20];
        for (int k = 0; k < seatingBooked.length; k++){
            seatingBooked[k] = 0;
        }
        for (int j = 0; j < 20; j++){ full)
            int rand = (int)(Math.random() * 3);
            if ((rand == 0) || (rand == 1)){
                passengerArray[j] = generatePassenger();
                seatingBooked[j] = 1;
            }
        }
        planes.add(new Plane(ID, departLoc, departTime, arriveLoc, arriveTime,
passengerArray, seatingBooked));
    }
```

Рисунок 3.1 – Код методу initialGeneratePlanes

Для генерації пасажирів на борт літака було розроблено метод generatePassangers. Спочатку оголошується масиви для можливих імен та країн, потім випадково обирається перше і останнє ім'я та країна. Код методу наведено на рисунку 3.2.

```
public static Passenger generatePassenger(){

    ArrayList <String> firstNames = new ArrayList<String>(Arrays.asList("Branden",
"John", "Jane", "Lucas", "Morgan", "Tera", "Alban", "Rosa", "Clayton", "Brent",
"Raleigh", "Dwain"));
    ArrayList <String> lastNames = new ArrayList<String>(Arrays.asList("Lisk", "Doe",
"Underhill", "Shine", "Thorley", "Winston", "King", "Janson", "Payton", "Low",
"Seward"));
    ArrayList <String> countries = new ArrayList<String>(Arrays.asList("Canada",
"United States", "Mexico"));

    String name = firstNames.get((int)(Math.random() * firstNames.size())) + " " +
lastNames.get((int)(Math.random() * lastNames.size()));
    String age = Integer.toString((int)(Math.random() * 90 + 10));
    String country = countries.get((int)(Math.random() * countries.size()));

    return (new Passenger(name, age, country));
}
```

Рисунок 3.2 – Код методу generatePassangers

Для оновлення таблиці з даними про рейси було створено метод refreshTables. Спочатку метод шукає літаки у масиві поточних рейсів, далі отримує інформацію від кожного об’єкту і встановлює таблиці на ці дані. Потім шукає у масиві майбутніх рейсів та додає новий рядок і нарешті отримує всю інформацію з масиву та відображає в таблиці. Код методу наведено на рисунку 3.3.

```
public static void refreshTables(){
    for (int i = 0; i < planes.size(); i++){
        currentFlightsTable.setValueAt(planes.get(i).ID, i, 0);
        currentFlightsBooking.setValueAt(planes.get(i).ID, i, 0);
        currentFlightsBooking.setValueAt(planes.get(i).getActualDepartLocation(), i,
1);
        currentFlightsBooking.setValueAt(sdf.format(new
Date(planes.get(i).departTime)), i, 2);
        currentFlightsBooking.setValueAt(planes.get(i).getActualArrivalLocation(), i,
3);
        currentFlightsBooking.setValueAt(sdf.format(new
Date(planes.get(i).arriveTime)), i, 4);
    }

    int max = dtm.getRowCount();
    for (int i = 0; i < max; i++){
        dtm.removeRow(0);
    }

    for (int i = 0; i < futurePlanes.size(); i++){
        dtm.addRow(new String[5]); //add a new row
        futureFlightsBooking.setValueAt(futurePlanes.get(i).ID, i, 0);
        futureFlightsBooking.setValueAt(futurePlanes.get(i).getActualDepartLocation(),
i, 1);
        futureFlightsBooking.setValueAt(sdf.format(new
Date(futurePlanes.get(i).departTime)), i, 2);

futureFlightsBooking.setValueAt(futurePlanes.get(i).getActualArrivalLocation(), i, 3);
        futureFlightsBooking.setValueAt(sdf.format(new
Date(futurePlanes.get(i).arriveTime)), i, 4);
    }
}
```

Рисунок 3.3 – Код методу refreshTables

3.3 Розробка модуля створення літаків

Наступним етапом розробки додатку для моніторнгу авіарейсів є розробка модуля створення літаків.

Перш за все необхідно створити клас Plane, який відповідає за створення літаків. Спочатку визначаємо класс, визначаємо змінні, та створюємо конструктор з параметарми. Частина коду класу наведено на рисунку 3.4.

```
public class Plane {

    String ID;
    Point departLoc;
    long departTime;
    Point arriveLoc;
    long arriveTime;
    Passenger[] passengerArray;
    int[] seatingBooked;


    public Plane(String myID, Point mydepartLoc, long mydepartTime, Point myarriveLoc,
long myarriveTime, Passenger[] myPassengerArray, int[] mySeatingBooked){
        ID = myID;
        departLoc = mydepartLoc;
        departTime = mydepartTime;
        arriveLoc = myarriveLoc;
        arriveTime = myarriveTime;
        passengerArray = myPassengerArray;
        seatingBooked = mySeatingBooked;
    }
```

Рисунок 3.4 – Частина коду класу Plane

Далі створюємо метод перетворення значення в аеропорт, потім ми визначаємо координати всіх аеропортів. Для цього був створений метод getActualDepartLocation. Частина коду методу наведено на рисунку 3.5.

```
public String getActualDepartLocation(){
    Point Vancouver = new Point(240, 540);
    Point Edmonton = new Point(370, 510);
    Point Iqualuit = new Point(770, 270);
    Point Winnipeg = new Point(560, 600);
    Point Toronto = new Point(790, 690);
    Point Ottawa = new Point(830, 650);
    Point Halifax = new Point(990, 590);
```

Рисунок 3.5 – Частина коду методу getActualDepartLocation

Тепер потрібно зробити тестування створених локацій за допогою умов. Код наведено нижче. Продовження частини коду методу наведено на рисунку 3.6.

```
        if (departLoc.equals(Vancouver)){
            return "Vancouver";
        } else if (departLoc.equals(Edmonton)){
            return "Edmonton";
        } else if (departLoc.equals(Iqualuit)){
            return "Iqualuit";
        } else if (departLoc.equals(Winnipeg)){
            return "Winnipeg";
        } else if (departLoc.equals(Toronto)){
            return "Toronto";
        } else if (departLoc.equals(Ottawa)){
            return "Ottawa";
        } else if (departLoc.equals(Halifax)){
            return "Halifax";
        } else{
            return "Unidentified Location";
        }
    }
```

Рисунок 3.6 – Продовження частини коду методу getActualDepartLocation.

Аналогічно створюєм метод getActualArrivalLocation, який відповідає за місце прибуття. Код методу наведено на рисунку 3.7.

```
    public String getActualArrivalLocation(){
        Point Vancouver = new Point(240, 540);
        Point Edmonton = new Point(370, 510);
        Point Iqualuit = new Point(770, 270);
        Point Winnipeg = new Point(560, 600);
        Point Toronto = new Point(790, 690);
        Point Ottawa = new Point(830, 650);
        Point Halifax = new Point(990, 590);

        if (arriveLoc.equals(Vancouver)){
            return "Vancouver";
        } else if (arriveLoc.equals(Edmonton)){
            return "Edmonton";
        } else if (arriveLoc.equals(Iqualuit)){
            return "Iqualuit";
        } else if (arriveLoc.equals(Winnipeg)){
            return "Winnipeg";
        } else if (arriveLoc.equals(Toronto)){
            return "Toronto";
        } else if (arriveLoc.equals(Ottawa)){
            return "Ottawa";
        } else if (arriveLoc.equals(Halifax)){
            return "Halifax";
        } else{
            return "Unidentified Location";
        }
    }
}
```

Рисунок 3.7 – Код методу getActualArrivalLocation

3.4 Розробка модуля обробки анімації літаків

У процесі використанням програмного засобу, настає час симулювання переміщення літаків на карті, тому для того, щоб це реалізувати нам потрібно створити класс AnimatePlanes. Далі нам потрібно створити метод run, який буде отримувати розташування піктограм площин та додавати їх до масиву. Частина коду класу AnimatePlanes наведено на рисунку 3.8.

```
public void run(){
    while(true){

        Point location1 = Flight.p1.getLocation();
        Point location2 = Flight.p2.getLocation();
        Point location3 = Flight.p3.getLocation();
        Point location4 = Flight.p4.getLocation();
        Point location5 = Flight.p5.getLocation();
        Point[] locations = {location1, location2, location3, location4, location5};

}
```

Рисунок 3.8 – Частина коду класу AnimatePlanes

Потім викликаємо метод анімації для кожної піктограми Код наведено нижче. Продовження частини коду класу наведено на рисунку 3.9.

```
Flight.p1.setLocation(animate(location1, Flight.planes.get(0).arriveLoc));
Flight.p2.setLocation(animate(location2, Flight.planes.get(1).arriveLoc));
Flight.p3.setLocation(animate(location3, Flight.planes.get(2).arriveLoc));
Flight.p4.setLocation(animate(location4, Flight.planes.get(3).arriveLoc));
Flight.p5.setLocation(animate(location5, Flight.planes.get(4).arriveLoc));
```

Рисунок 3.9 – Продовження частини коду класу AnimatePlanes

Після цього запускається цикл та перевіряє чи прибув літак, якщо прибув то знаходить літак який замінить поточний. Далі визначає число, яке більше ніж будь-який можливий системний час. Визначає індекс, додає піктограму до масиву, отримує розмір масиву. Якщо найшвидший час менший поточного, встановлює індекс. Продовження частини коду класу наведено на рисунку 3.10.

```
for (int i = 0; i < 5; i++){
    boolean tester = checkPlaneDest(locations[i], Flight.planes.get(i));
    if (tester == true){
        long soonestTime = (long)(Math.pow(2, 55));
        int soonestTimeIndex = 0;
        JLabel[] planeIcons = {Flight.p1, Flight.p2, Flight.p3, Flight.p4, Flight.p5};
        int futureSize = Flight.futurePlanes.size();
        if (futureSize > 0){
            for (int j = 0; j < futureSize; j++){
                if (Flight.futurePlanes.get(j).departTime < soonestTime){
                    soonestTimeIndex = j;
                    soonestTime = Flight.futurePlanes.get(j).departTime;
                }
            }
```

Рисунок 3.10 – Продовження частини коду класу AnimatePlanes

Далі встановлює літак що прибув, на майбутній літак, із найшвидшим часом вильоту, прибирає його з майбутніх рейсів, видаляє рядок. Якщо майбутніх літаків немає, генерує літак на місці поточного, встановлює нове розташування значка та оновлює таблицю з рейсами. Продовження частини коду класу наведено на рисунку 3.11.

```
                    Flight.planes.set(i, Flight.futurePlanes.get(soonestTimeIndex));
                    Flight.futurePlanes.remove(soonestTimeIndex);
Flight.dtm.removeRow(0);
                } else{
                    Plane tempPlane = Flight.generateFuturePlane();
                    Flight.planes.set(i, tempPlane);
                }
                planeIcons[i].setLocation(Flight.planes.get(i).departLoc); new
            }
        }
        Flight.refreshTables();
        try {
            Thread.sleep(1000 / Flight.animationSpeed);
        } catch (InterruptedException ex) {

        }
    }
}
```

Рисунок 3.11 – Продовження частини коду класу AnimatePlanes

Далі потрібно створити метод checkPlaneDest, який перевіряє, чи досяг літак місця призначення. Спочатку метод отримує місце прибуття літака, порівнює відстані по осі x і y. Якщо літак знаходиться в місці призначення, повертає true, інакше false. Код методу наведено на рисунку 3.12.

```
public boolean checkPlaneDest(Point location, Plane plane){
    Point airport = plane.arriveLoc;
    long differenceY = Math.abs(airport.y - location.y);
    long differenceX = Math.abs(airport.x - location.x);
    if(((differenceY > -3)&&(differenceY < 3)) && ((differenceX > -3)&&(differenceX <
3))){
        return true;
    }
    return false;
}
```

Рисунок 3.12 – Код методу checkPlaneDest

Метод animate створює нове розташування кожної піктограми. Спершу метод отримує місце призначення, знаходить відстань x і y.

Код методу наведено на рисунку 3.13.

```
public Point animate(Point location, Point destination){
    int destinationX = destination.x;
    int destinationY = destination.y;
    double distanceX = destinationX - location.x;
    double distanceY = destinationY - location.y;
    double difference = 0;
    double xMovement = 0;
    double yMovement = 0;

    if (Math.abs(distanceX) == Math.abs(distanceY)){
        if(distanceX >= 0){
            xMovement = 1;
        } else{
            xMovement = -1;
        }
        if (distanceY >= 0){
            yMovement = 1;
        } else{
            yMovement = -1;
        }
    } else if(Math.abs(distanceX) > Math.abs(distanceY)){
        if(distanceX >= 0){
            xMovement = 1;
        } else{
            xMovement = -1;
        }
```

Рисунок 3.13 – Код методу animate

Якщо відстані x і y однакові, літак рухається по діагоналі, якщо відстань x більша, переміщається за допомогою одного пробілу в цьому напрямку, аналогічно переміщається одним пробілом у напрямку y.

3.5 Розробка модуля внутрішнього годинника

Наступний етап розробки – створення модуля внутрішнього годинника clock, який забезпечує можливість пришвидшення часу вперед.

Модуль розроблено з використанням бібліотеки java.text.SimpleDateFormat та Java.util.Date.

При розробці модулю було використано потоки, тому було унаслідовано клас Thread.

Код класу clock наведено на рисунку 3.14.

```
public class Clock extends Thread{
    public void run(){

        while(true){
            Flight.totalFastForward = Flight.totalFastForward +
((Flight.animationSpeed * 1000) - 1000);
            long mlls = System.currentTimeMillis() + Flight.totalFastForward;
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            Date date = new Date(mlls);
            String output = sdf.format(date);
            Flight.clockLabel.setText(output);

            try {
                Thread.sleep(1080);
            } catch (InterruptedException ex) {

            }
        }
    }
}
```

Рисунок 3.14 – Код класу clock

Метод визначає значення totalFastForward, на основі швидкості анімації, отримує час з годинника, перетворює його у видимий час, та встановлює вихід. Запускається кожну секунду, але надає час для запуску.

3.6 Розробка модуля графічного інтерфейсу

Наступний етап розробки – створення классу Flight, який відповідає за графічний інтерфейс програми, та надає можливість відображувати програму. Він наслідує класс javax.swing.JFrame. Визначаємо усі глобальні змінні, та поточні дані рейсів, а також координати аеропорту.

Додаток розроблено з використанням мови програмування Java, бібліотек java.awt та javax.swing. та Swing-GUI builder.

Частина коду класу наведено на рисунку 3.15.

```
public class Flight extends javax.swing.JFrame{

    static DefaultTableModel dtm;
    static SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    static int selectedPlaneObject = -1;
    static long animationSpeed = 1;
    static long totalFastForward = 0;
    static String log = "";

    public static ArrayList<Plane> planes = new ArrayList<Plane>();
    public static ArrayList<Plane> futurePlanes = new ArrayList<Plane>();
    public static int[][] airports = new int[][]{
        {240, 370, 770, 560, 790, 830, 990},
        {540, 510, 270, 600, 690, 650, 590}
    };
    JButton[] seats;
```

Рисунок 3.15 – Частина коду класу Flight

Далі створюємо графічний інтерфейс нашого програмного засобу. Продовження частини коду класу наведено на рисунку 3.16.

```
passengerDialog.setAlwaysOnTop(true);
passengerDialog.setBounds(new java.awt.Rectangle(500, 200, 440, 300));
passengerDialog.setLocation(new java.awt.Point(500, 200));
passengerDialog.setResizable(false);

dialogTitle.setFont(new java.awt.Font("Lucida Grande", 0, 15));
dialogTitle.setText("Book New Passenger");

jPanel2.setLayout(new java.awt.GridLayout(4, 1));

idLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14));
idLabel.setText("Enter Plane ID #");
jPanel2.add(idLabel);

nameLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14));
nameLabel.setText("Enter Passenger Name:");
jPanel2.add(nameLabel);

ageLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14));
ageLabel.setText("Enter Age of Passenger:");
jPanel2.add(ageLabel);
```

Рисунок 3.16 – Продовження частини коду класу Flight

Далі створюємо усі необхідні групові макети, діалогові вікна, стилі, шрифти, розмір, пропуски, поля з інформацією, поля введення даних та інших компонентів.

Продовження частини коду класу наведено на рисунку 3.17.

```
javax.swing.GroupLayout passengerDialogLayout = new
javax.swing.GroupLayout(passengerDialog.getContentPane());
passengerDialog.getContentPane().setLayout(passengerDialogLayout);
passengerDialogLayout.setHorizontalGroup(

passengerDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(passengerDialogLayout.createSequentialGroup()

.addGroup(passengerDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(passengerDialogLayout.createSequentialGroup()
                .addGap(130, 130, 130)
                .addComponent(dialogTitle))
            .addGroup(passengerDialogLayout.createSequentialGroup()
                .addGap(12, 12, 12)

.addGroup(passengerDialogLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                    .addComponent(bookPassengerButton,
javax.swing.GroupLayout.PREFERRED SIZE, 409, javax.swing.GroupLayout.PREFERRED SIZE)
                    .addGroup(passengerDialogLayout.createSequentialGroup()
                        .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED_SIZE,
221, javax.swing.GroupLayout.PREFERRED_SIZE)))))
        .addGap(0, 0, Short.MAX_VALUE))
);
```

Рисунок 3.17 – Продовження частини коду класу Flight

У результаті створеного класу, отримуємо робочий графічний інтерфейс, до якого підключені усі функції програмного засобу моніторингу авіарейсів.

3.7 Висновки

У третьому розділі обґрунтовано вибір мови програмування, бібліотек та технологій, які будуть використовуватися при розробці програмного продукту та наведено основні їх переваги.

Проаналізувавши вимоги програмного продукту було вибрано мову програмування Java.

Для зручності розробки графічного інтерфейсу було обрано бібліотеку Swing та конструктор графічних інтерфейсів Swing GUI-builder.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Методи тестування

Тестування програмного забезпечення має широке поняття та багато визначень, але найосновніші з них це наступне:

- Тестування програмного забезпечення (Software Testing);
- Контроль якості (Quality Control, QC);
- Забезпечення якості (Quality Assurance, QA).

Тестування програмного забезпечення (Software Testing) [17] — це одна з технік контролю якості, що включає активності з планування робіт (Test Management), проєктування тестів (Test Design), виконання тестування (Test Execution) й аналізу отриманих результатів (Test Analysis).

Контроль якості (Quality Control) — це сукупність дій над продуктом у процесі розробки для отримання інформації про його актуальний стан, у розрізі готовності продукту до випуску, відповідності зафіксованим вимогам та заявленому рівню якості продукту.

Забезпечення якості (Quality Assurance) [18] — це сукупність заходів, що охоплюють всі технологічні етапи розробки, випуску й експлуатації ПЗ інформаційних систем на різних стадіях життєвого циклу для забезпечення необхідного рівня якості продукту, який випускається.

Тестування програмного забезпечення поділяється на види:

- Функціональні;
- Нефункціональні;
- Пов’язані зі змінами.

Функціональні різновиди тестування — базуються на функціях і особливостях, а також на взаємодії з іншими системами; можуть бути представлені на всіх рівнях тестування.

Нефункціональні різновиди тестування — описують тести, необхідні для визначення характеристик ПЗ, які можна виміряти в різних величинах. Загалом, це тестування того, як система працює.

Пов’язані зі змінами різновиди тестування — повторна перевірка роботи ПЗ після виправлення бага/дефекту, для підтвердження того, що проблема дійсно була вирішена і що виправлення не вплинуло на роботу інших систем.

Тестування програмного забезпечення поділяється на рівні:

- Компонентне тестування (Unit Testing);
- Інтеграційне тестування (Integration Testing);
- Системне тестування (System Testing);
- Приймальне тестування (Acceptance Testing).

Компонентне тестування (Unit Testing) — перевіряє функціональність і шукає дефекти в частинах програми, що доступні й можуть бути протестовані окремо (модулі програм, об'єкти, класи, функції та ін.). Це найдетальніший рівень тестування.

Інтеграційне тестування (Integration Testing) — призначене для перевірки взаємодії між компонентами, а також взаємодії з різними частинами системи (операційною системою, обладнанням або взаємодію між різними системами).

Системне тестування (System Testing) [19] — основне завдання, перевірка як функціональних так і нефункціональних вимог до системи в цілому. При цьому виявляються такі дефекти, як невірне використання ресурсів системи, непередбачені комбінації даних користувацького рівня, несумісність з оточенням, непередбачені сценарії використання та ін.

Приймальне тестування (Acceptance Testing) — формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою визначення, чи відповідає система приймальним критеріям та винесення рішення про прийом роботи замовником або іншою уповноваженою особою.

За знанням системи виділяють наступні види тестування ПЗ:

- Тестування чорної скриньки (black box);

- Тестування білої скриньки (white box);
- Тестування сірої скриньки(gray box).

Тестування без уявлення що знаходиться всередині програми називається тестування «чорної скриньки». Тестувальник не знає архітектуру продукту і не має доступу до вихідного коду. Зазвичай, тестуючи методом чорної скриньки, тестувальник буде працювати з графічним інтерфейсом користувача, незнаючи за якими принципами бізнес логіки працює програма, та нерозуміючи як вхідні дані обробляються, а отримувати тільки вже готовий результат, тобто вихідні дані.

Переваги:

- Продуктивний для великих об’ємів коду;
- Взаємодія з кодом непотрібна;
- Продукт можна протестувати без знань про внутрішню реалізацію програмного продукту.

Недоліки:

- Тестувальник має неповні знання про програму, тому на деякому етапі, тестування стає неефективним;
- Реалізується тільки фіксована кількість тестових випадків;
- Важко займатися дизайном тестових випадків.

Тестування білої скриньки – це детальний аналіз внутрішньої архітектури коду, логіки, структури. Тестування за допомогою білої скриньки також називається прозорим тестуванням, або відкритим тестуванням. Для того, щоб протестувати додаток за допомогою білої скриньки, тестувальник повинен знати внутрішню роботу програми.

Тестувальник повинен подивитись всередину вихідного коду і зрозуміти, який блок коду поводиться неправильно відповідно до вимог.

Переваги:

- Оптимізація коду;
- Видалення додаткових рядків коду, що може привести до ненайдених дефектів;

- Максимальне покриття сценаріїв використання.

Недоліки:

- Збільшення витрат фірми через необхідність відповідних кадрів;
- Важка підтримка такого тестування, оскільки потрібні відповідні інструменти для проведення таких видів робіт.

Тестування сірої скриньки – це метод тестування програмного продукту з неповним знанням внутрішньої системи.

Фахівець, який знає якийсь вузько-направлений домен, завжди буде актуальнішим ніж фахівець з обмеженим знанням домену. На відміну від тестування чорної скриньки, де робітник проводить тестування інтерфейсу користувача програми, при такому тестуванні він має доступ до проєктної документації та бази даних. Володіючи необхідними знаннями, він може підготувати кращі тестові дані та сценарії тестування, які з набагато більшою вірогідністю знайдуть дефект.

Переваги:

- Комбіновані методи тестування чорної скрииньки і білої скриньки;
- Проводиться тестування графічного інтерфейсу, та функціональної специфікації;
- Тест виконується з поглояду простого користувача, а не тестувальника.

Недоліки:

- Через те, що доступ до вихідного коду частково недоступний, область тестування звужується;
- Тести можуть бути зайвими, якщо розробник програмного забезпечення вже виконав тестовий приклад;
- Тестування всіх можливих вичерпних варіантів роботи програми майже неможливе, тому багато програмних розгалужень неперевірятимуться.

Також для того, щоб протестувати програмний засіб потрібно знати техніки тест дизайну, завдяки яким можна значно спростити роботу, знайти якомога більше дефектів, а головне зменшити час тестування.

Тест-дизайн [20] — це етап процесу тестування ПЗ, на якому проєктуються і створюються тестові випадки (тест-кейси) відповідно до визначених раніше критеріїв якості та цілей тестування.

До основних технік тест-дизайну відносять:

- Класи еквівалентності (Equivalence Partitioning) — розробка тестів методом чорного ящика, у якій тестові сценарії створюються для перевірки елементів еквівалентної області. Зазвичай тестові сценарії розробляються для покриття кожної області як мінімум один раз;
- Граничні значення (Boundary Value Analysis) — розробка тестів методом чорного ящика, коли тестові сценарії проєктуються на підставі граничних значень;
- Таблиця прийняття рішень (Decision Table Testing) — таблиця, яка відображає комбінації вхідних даних або причин із відповідними вихідними даними та діями (наслідками);
- Переходи станів (State Transition Testing) — розробка тестів методом чорного ящика, коли сценарії тестування будуються на основі виконання коректних і некоректних переходів станів;
- Сценарії використання (Use Case Testing) — розробка тестів методом чорного ящика, де тестові сценарії створюються для виконання сценаріїв використання;
- Чек-ліст (Checklist-based Testing) — метод створення тестів, який грунтується на досвіді, коли досвідчений тестувальник використовує високорівневі списки. Перелік містить пункти, які потрібно відмітити або запам'ятати, або складається з набору правил чи критеріїв,згідно з якими верифікується програмний продукт;
- Вгадування помилок (Error Guessing) — метод проєктування тестів, коли досвід тестувальника використовується для передбачення дефектів, які можуть бути у тестованому компоненті або системі в результаті помилок, а також для розробки тестів спеціально для виявлення дефектів.

Для тестування програмного засобу «Flight Monitoring» було обрано функціональне, системне тестування та тестування чорного ящика та техніку тест-дизайну вгадування помилок.

4.2 Тестування програмного засобу

Протестуємо відкриття програмного засобу моніторингу авіарейсів. Для цього необхідно запустити Executable Jar File у робочій директорії додатку. Робоча директорія додатку зображено на рисунку 4.1.

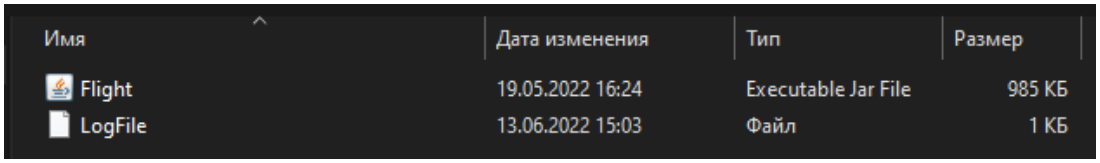


Рисунок 4.1 – Робоча директорія додатку

Протестуємо завантаження програмного засобу моніторингу авіарейсів. Для цього необхідно запустити програму з значка іконки «Flight» та натиснути кнопку «Load Program» завантажити програму. Вікно запуску програмного засобу зображено на рисунку 4.2.

Рисунок 4.2 – Екран запуску програмного засобу

Після вікна запуску програми ми потряпляємо у головне вікно нашого програмного засобу, де знаходиться весь основний функціонал, інтерфейси, кнопки для взаємодії користувача. Головне вікно запуску програмного засобу зображено на рисунку 4.3.

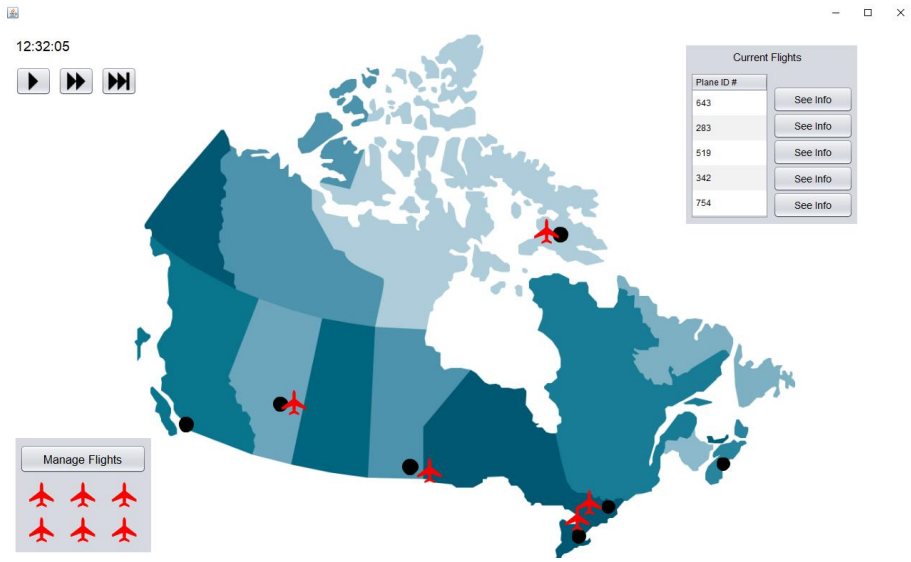


Рисунок 4.3 – Головне вікно програмного засобу

Для виклику функції відкриття поточних авіарейсів необхідно натиснути на кнопку «Manage Flights». У результаті відкривається меню керування поточними авіарейсами яке зображено на рисунку 4.4.

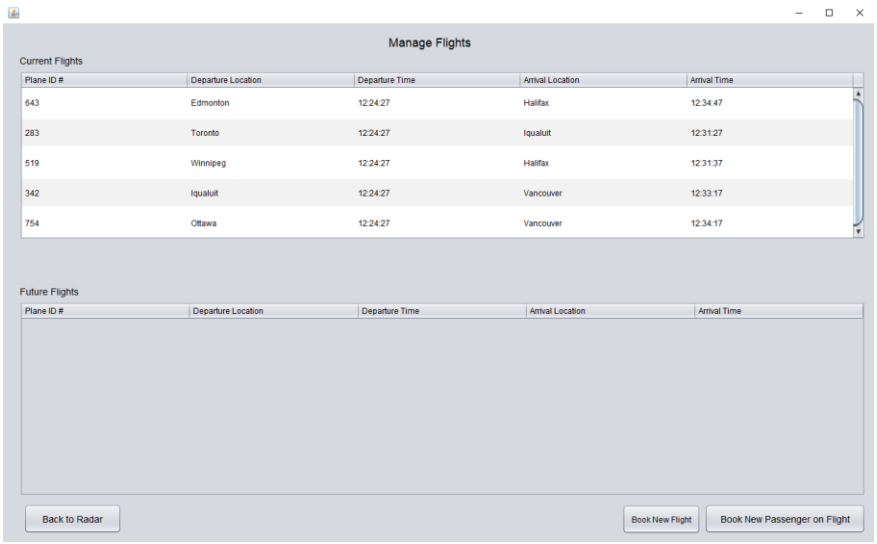


Рисунок 4.4 – Меню керування поточними та майбутніми рейсами

Протестуємо функцію додавання нового авіарейсу. Для цього нам потрібно натиснути на кнопку «Book New Flight». Після цього у нас з’являється діалогове вікно з додаванням нового авіарейсу в якому присутня можливість написання: ID номеру літака, вибір місця відправлення та вибір місця прибуття. Заповняєм стрічку «Enter Plane ID #» значенням «999», місце відправлення та місце прибуття вибираємо з випадаючого меню «Ottawa» та «Edmonton» відповідно і натискаємо кнопку «Book Flight». Діалогове вікно додавання нового авіарейсу зображено на рисунку 4.5.

Рисунок 4.5 – Діалогове вікно додавання нового авіарейсу

Після цього новий авіарейс додається у таблицю майбутні авіарейси «Future Flights». Таблиця з відображенням майбутніх авіарейсів зображена на рисунку 4.6.

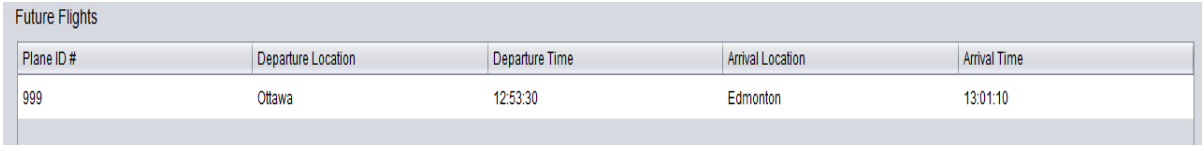


Рисунок 4.6 – Таблиця майбутніх авіарейсів

Після того як ми додали наступний авіарейс, коли настає година 13:01:10, цей літак з’являється у меню поточних авіарейсів під «ID номером 999». Меню поточних авіарейсів зображено на рисунку 4.7.

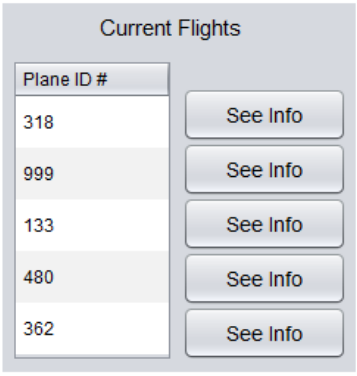


Рисунок 4.7 – Меню поточних авіарейсів

Натиснимо на кнопку «See Info», щоб подивитися інформацію про пасажирів на борту літака під унікальним номером 999. Після того як ми натиснули на кнопку, ми переходимо у вікно інформації про пасажирів. Тут ми бачимо що всі місця зображені зеленим кольором, тобто вони вільні тому, що ми не додали пасажирів.

Вікно інформації про пасажирів зображено на рисунку 4.8.

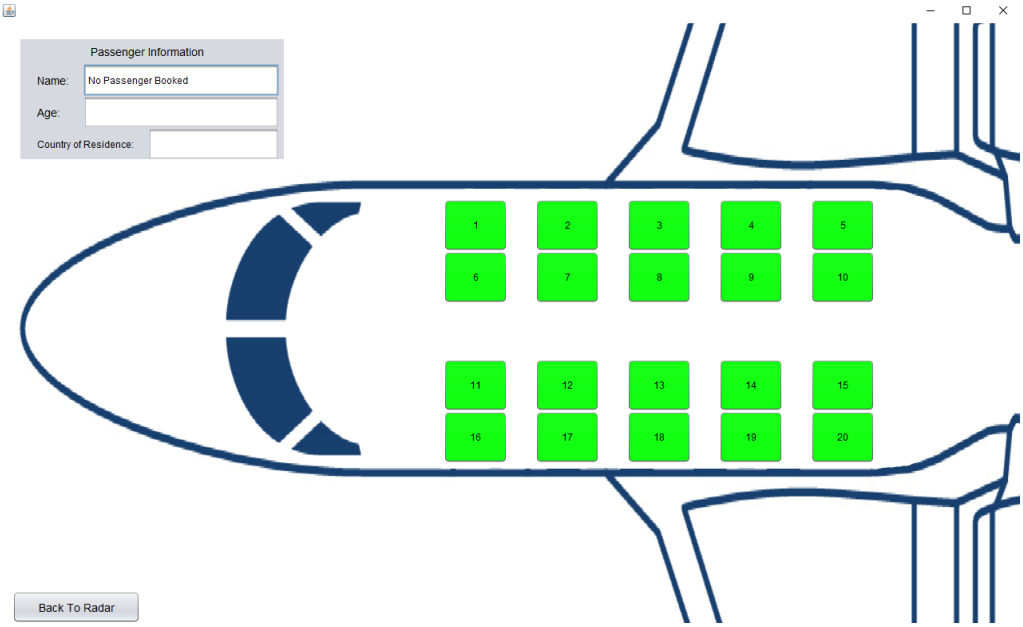


Рисунок 4.8 – Вікно інформації про пасажирів

Протестуємо функцію додавання нового пасажира на борт літака Для цього нам потрібно натиснути на кнопку «Book New Passenger on Flight». Після цього у нас з'являється діалогове вікно з додаванням нового пасажира на борт літака, в якому присутня можливість написання: ID номеру літака (в який саме літак ми хочемо додати пасажира), ім'я пасажира, вік та країну проживання. Заповняєм стрічку «Enter Plane ID #» значенням «999», ім'я заповнюємо значенням «Nazar Homeniuk», вік «21», країну проживання «Ukraine» відповідно і натискаємо кнопку «Book Passenger». Тепер місця які зображені червоним кольором, означають те що це місце вже зайнято.

Вікно інформації про пасажирів зображено на рисунку 4.9.

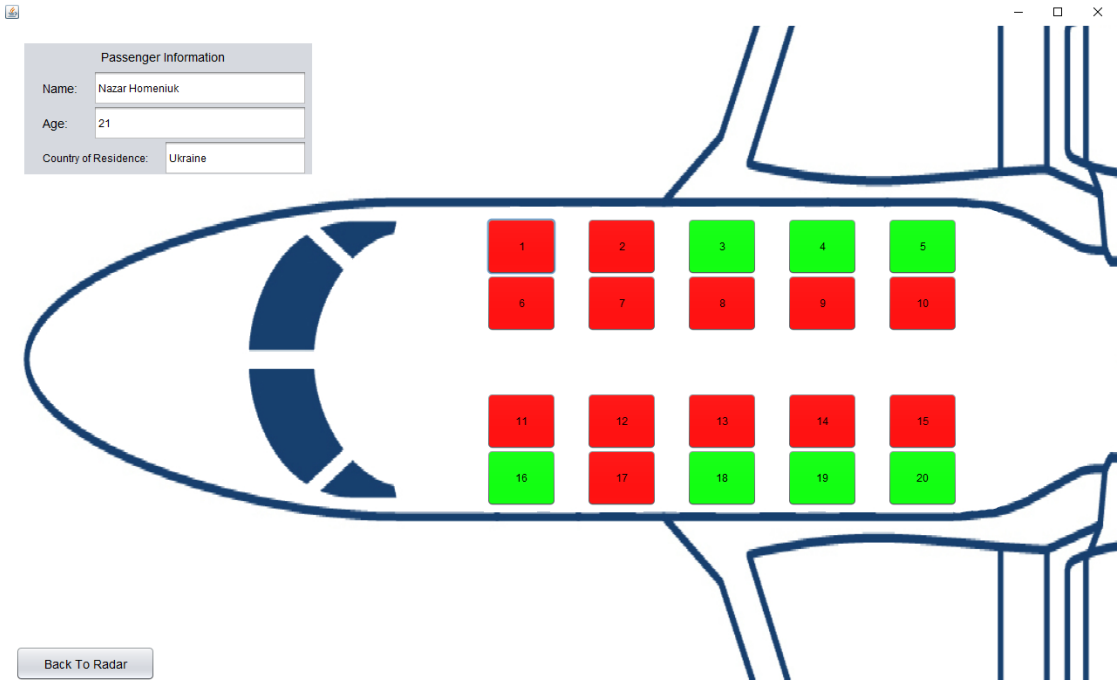


Рисунок 4.9 – Вікно інформації про пасажирів

Після того як ми додали пасажира на борт, він з'явився на борту літака, та підсвічується червоним. Уся інформація яку ми додавали у діалоговому вікні додавання нового пасажиру, тепер відображається, якщо натиснути на місце під номером «1».

Вікно інформації про пасажира зображено на рисунку 4.10.

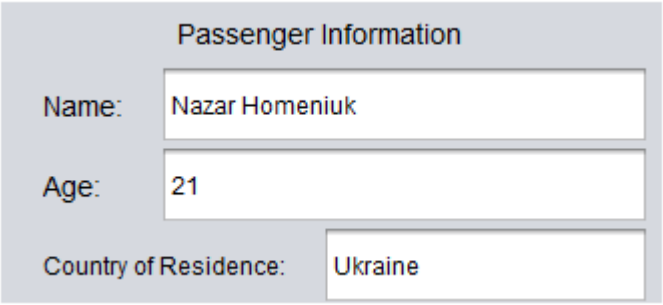


Рисунок 4.10 – Інформація про доданого пасажира

У результаті тестування додатку було доведено, що його функціонал працює відповідно до технічного завдання.

4.3 Розробка інструкції користувача

Першим етапом розробки інструкції користувача є визначення технічних вимог програмного продукту. У таблицях 4.1 та 4.2 наведено мінімальну та рекомендовану конфігурацію персонального комп’ютера для запуску програми.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1.2 ГГц
Об’єм оперативної пам’яті	2 ГБ для 32-розрядної системи і 4 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	500 МБ
Графічний прискорювач	Інтегрований графічний прискорювач, сумісний з DirectX9
Операційна система	Windows 10

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2 ГГц
Об’єм оперативної пам’яті	4 ГБ для 32-розрядної системи і 8 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	500МБ
Графічний прискорювач	Графічний прискорювач, сумісний з DirectX9 та об’ємом відеопам’яті 2 ГБ
Операційна система	Windows 10

Щоб почати користуватися програмним засобом моніторингу авіарейсів, потрібно запустити Flight.exe у робочій директорії програми.

Робоча директорія програмного засобу зображена на рисунку 4.11.

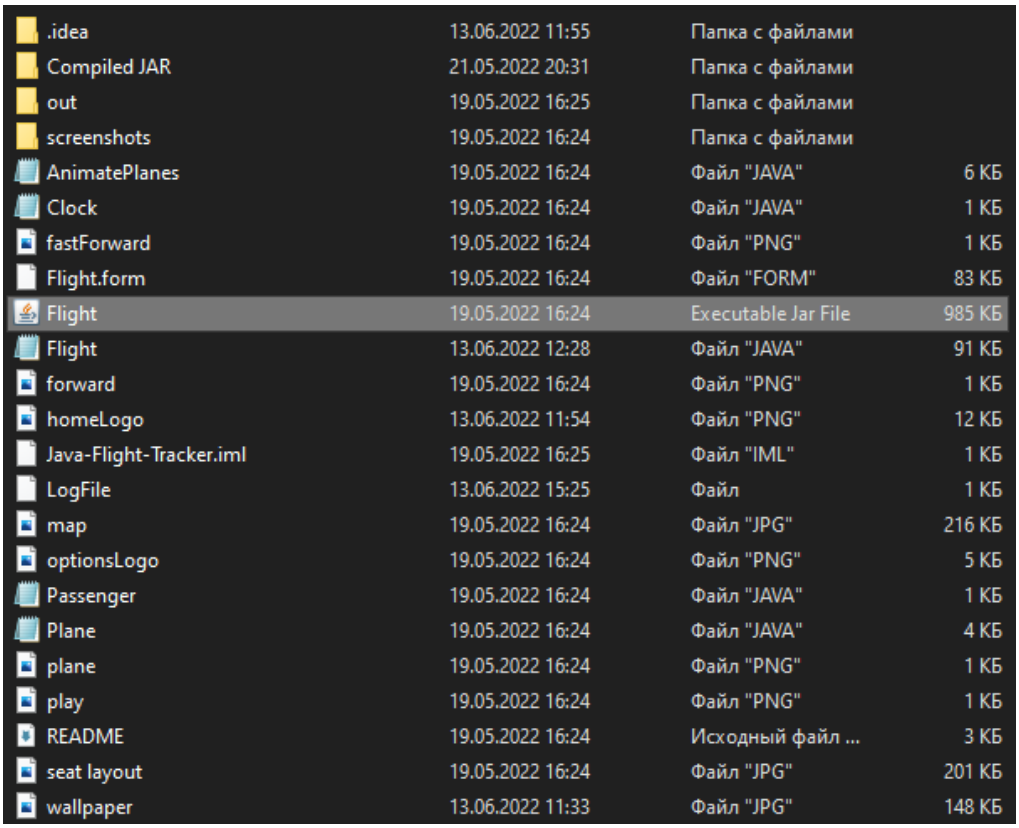


Рисунок 4.11 – Робоча директорія програмного засобу

Далі нам потрібно натиснути кнопку «Load» щоб завантажити програмний засіб, та потрапити у головне вікно користування програми.

Після того, як користувач потрапив у головне вікно додатку, він одразу бачить на карті поточні авіарейси. Для того, щоб додати новий рейс, необхідно натиснути на кнопку «Manage Flights», користувач потрапляє у вікно з таблицями поточних та майбутніх рейсів. Тепер потрібно натиснути на кнопку «Book New Flight», з’явиться діалогове вікно з полями які потрібно заповнити. Користувач заповнює поля валідними данми та натискає кнопку «Book Flight». Рейс який додав користувач, з’явився у таблиці майбутніх рейсів, коли час відльоту настав, доданий рейс з’являється у таблиці поточних авіарейсів та видаляється з попередньої таблиці. У результаті даних дій, на карті створюється піктограма літака, який вилітає з одного міста, та через деякий час переміститься у інше.

4.4 Висновки

У четвертому розділі було проведено тестування додатку, у результаті якого було доведено його повну працездатність та відповідність поставленому технічному завданню.

Розроблено інструкцію користувача по встановленню та використанню програмного продукту.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи було розроблено програмний засіб для моніторингу авіарейсів. Для розробки було використано середовище програмування IntelliJ IDEA. Робота розроблена згідно методичних вказівок.

Було проаналізовано стан даного питання на сьогоднішній день. Розглянуто основні аналоги, визначено їх особливості та недоліки і розроблено порівняння з власним програмним продуктом. У результаті аналізу обрано мову програмування Java, технологію Swing GUI-builler, та бібліотеку Swing для графічного інтерфейсу.

У процесі виконання роботи було зроблено: визначено найбільш ефективний метод моніторингу авіарейсів, розроблено загальний алгоритм роботи додатку, та алгоритм обробки анімації літаків, розроблено графічний інтерфейс користувача, розроблено програмний засіб моніторингу авіарейсів, проведено тестування програмного продукту.

Перед створенням програмного продукту було розроблено метод обробки анімації літаків, структуру графічного інтерфейсу.

Подальшого розвитку отримав метод обробки анімації літаків, на відміну від існуючих, використовує методи обробки у реальному часі, що дозволило підвищити продуктивність моніторингу авіарейсів та їх безпеку.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційні технології у сучасному світі: веб-сайт. URL: http://sophus.at.ua/publ/2013_12_19_20_kampodilsk/sekcija_7_2013_12_19_20/informacijni_tekhnologiji_v_suchasnomu_sviti/49-1-0-863 (дата звернення: 30.05.2022).

2. Гоменюк Н. В. Розробка програмного засобу моніторингу авіарейсів. Матеріали науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ). Вінниця, 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15624.

3. Авіадиспетчер: веб-сайт. URL: https://uk.wikipedia.org/wiki/Авіадиспетчер (дата звернення: 30.05.2022).

4. Моніторинг: веб-сайт. URL: http://multycourse.com.ua/ua/page/21/45 (дата звернення: 30.05.2022).

5. Автоматичне залежне спостереження – радіомовне (ADS-B): веб-сайт. URL: https://uk.wikipedia.org/wiki/Автоматичне_залежне_спостереження_-_радіомовне_(ADS–B) (дата звернення: 30.05.2022).

6. FlightAware: веб-сайт. URL: https://en.wikipedia.org/wiki/FlightAware (дата звернення: 30.05.2022).

7. Борис Погріщук., Ю. Паночишин. Комп’ютерна техніка та інформаційні технології : навч. посіб. Вид. 2-ге, переробл. і допов. Київ, 2012. 463 с.

8. What is A Web Interface: веб-сайт. URL: https://diib.com/learn/web-interface/ (дата звернення: 30.05.2022).

9. Інтерфейс користувача: веб-сайт. URL: https://ube.nlu.org.ua/article/Інтерфейс%20користувача (дата звернення: 30.05.2022).

10. Steven Miller. UH Design: A Field Good That Process And Methodology For Timeless Sir Experient : монографія. Берлін, 2021. 158 с.

11. What is an algorithm: веб-сайт. URL: https://www.techtarget.com/whatis/definition/algorithm#:~:text=An%20algorithm%20i

s%20a%20procedure,throughout%20all%20areas%20of%20IT (дата звернення: 30.05.2022).

12. Herbert Schildt. Java - The Complete Reference: монографія. USA McGraw-Hill Education, 11-те, 2018. 1208 с.

13. K. N. King. C++ Programming - A Modern Approach: монографія. Лондон : W W NORTON & CO, 2-ге, 2008. 832 с.

14. Java Swing Tutorial: веб-сайт. URL: https://www.javatpoint.com/java-swing (дата звернення: 30.05.2022).

15. What is Java Virtual Machine & its Architecture: веб-сайт. URL: https://www.guru99.com/java-virtual-machine-jvm.html (дата звернення: 30.05.2022).

16. Бізнес-логіка: веб-сайт. URL: https://uk.wikipedia.org/wiki/Бізнес-логіка (дата звернення: 30.05.2022).

17. How does software testing work: веб-сайт. URL: https://www.ibm.com/topics/softwaretesting#:~:text=Software%20testing%20is%20the%20process,development%20costs%20and%20improving%20performance (дата звернення: 30.05.2022).

18. Quality Assurance: веб-сайт. URL: https://www.techtarget.com/searchsoftwarequality/definition/quality-assurance (дата звернення: 30.05.2022).

19. What is System Testing: веб сайт. URL: https://www.guru99.com/system-testing.html (дата звернення: 30.05.2022).

20. When to create Test Design: веб-сайт. URL: http://tryqa.com/what-is-test-design-when-to-create-it/ (дата звернення: 30.05.2022).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф.
_____О. Н. Романюк
25 березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного засобу моніторингу авіарейсів»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., проф. Л.Б. Ліщинська
"____"_____2022 р.
Виконав:
_____студент гр. 1ПІ-18б Н.В. Гоменюк
"____"_____2022 р.

Вінниця – 2022 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного засобу моніторингу авіарейсів».

Галузь застосування – системи моніторингу авіарейсів.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 13 від « 07 » лютого 2022 р.

3. Мета та призначення розробки.

Метою роботи є підвищення продуктивності роботи користувача з засобом моніторингу авіарейсів за рахунок автоматизації та покращення алгоритму обробки анімації літаків.

Призначення роботи – підвищення ефективності та надійності засобу моніторингу авіарейсів.

4. Вихідні дані для проведення НДР

1. FlightAware: веб-сайт. URL: https://en.wikipedia.org/wiki/FlightAware (дата звернення: 30.05.2022).

2. What is A Web Interface: веб-сайт. URL: https://diib.com/learn/web-interface/ (дата звернення: 30.05.2022).

3. K. N. King. C++ Programming - A Modern Approach: монографія. Лондон : W W NORTON & CO, 2-ге, 2008. 832 с.

4. Steven Miller. UH Design: A Field Good That Process And Methodology For Timeless Sir Experient : монографія. Берлін, 2021. 158 с.

5. Технічні вимоги

Вхідні дані – текст англійською мовою; вихідні дані – карта та таблиці моніторингу авіарейсів.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного засобу моніторингу авіарейсів

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: д.т.н., проф. каф. ПЗ Ліщинська Л.Б.

Оригінальність	85,1%
Схожість	14,9%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Гоменюк Н. В.

Керівник роботи _____ Ліщинська Л. Б.

Додаток В – Лістинг програми

Flight.java

```
package flight;

//import all resources
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.table.DefaultTableModel;
import javax.swing.*;

public class Flight extends javax.swing.JFrame{

    //DEFINE ALL GLOBAL VARIABLES
    static DefaultTableModel dtm;
    static SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    static int selectedPlaneObject = -1;
    static long animationSpeed = 1;
    static long totalFastForward = 0;
    static String log = "";

    //all current flights data, as well as airport coordinate
locations
    public static ArrayList<Plane> planes = new ArrayList<Plane>();
    public static ArrayList<Plane> futurePlanes = new
ArrayList<Plane>();
    public static int[][] airports = new int[][]{
        {240, 370, 770, 560, 790, 830, 990},
        {540, 510, 270, 600, 690, 650, 590}
    };
```

```
    JButton[] seats;


    public Flight() {
        initComponents();
    }
     passengerDialog.setAlwaysOnTop(true);
passengerDialog.setBounds(new java.awt.Rectangle(500, 200, 440,
300));
passengerDialog.setLocation(new java.awt.Point(500, 200));
passengerDialog.setResizable(false);

dialogTitle.setFont(new java.awt.Font("Lucida Grande", 0, 15)); //
NOI18N
dialogTitle.setText("Book New Passenger");

jPanel2.setLayout(new java.awt.GridLayout(4, 1));

idLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
idLabel.setText("Enter Plane ID #");
jPanel2.add(idLabel);

nameLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
nameLabel.setText("Enter Passenger Name:");
jPanel2.add(nameLabel);

ageLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
ageLabel.setText("Enter Age of Passenger:");
jPanel2.add(ageLabel);

countryLabel.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
```

```
countryLabel.setText("Enter Country of Residence:");
jPanel2.add(countryLabel);

jPanel3.setLayout(new java.awt.GridLayout(4, 1, 0, 3));
jPanel3.add(passengerIdIn);
jPanel3.add(passengerNameIn);
jPanel3.add(passengerAgeIn);
jPanel3.add(passengerCountryIn);

bookPassengerButton.setFont(new java.awt.Font("Lucida Grande", 0,
15)); // NOI18N
bookPassengerButton.setText("Book Passenger");
bookPassengerButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        bookPassengerButtonActionPerformed(evt);
    }
});

javax.swing.GroupLayout passengerDialogLayout = new
javax.swing.GroupLayout(passengerDialog.getContentPane());
passengerDialog.getContentPane().setLayout(passengerDialogLayout);
passengerDialogLayout.setHorizontalGroup(

passengerDialogLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
    .addGroup(passengerDialogLayout.createSequentialGroup()

.addGroup(passengerDialogLayout.createParallelGroup(javax.swing.Grou
pLayout.Alignment.LEADING)
            .addGroup(passengerDialogLayout.createSequentialGroup()
                .addGap(130, 130, 130)
                .addComponent(dialogTitle))
            .addGroup(passengerDialogLayout.createSequentialGroup()
                .addGap(12, 12, 12)
```

```
.addGroup(passengerDialogLayout.createParallelGroup(javax.swing.Grou
pLayout.Alignment.LEADING)
                    .addComponent(bookPassengerButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 409,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(passengerDialogLayout.createSequentialGroup()
                        .addComponent(jPanel2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)
                        .addComponent(jPanel3,
javax.swing.GroupLayout.PREFERRED_SIZE, 221,
javax.swing.GroupLayout.PREFERRED_SIZE)))))
        .addGap(0, 0, Short.MAX_VALUE))
);
passengerDialogLayout.setVerticalGroup(

passengerDialogLayout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
    .addGroup(passengerDialogLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(dialogTitle)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATE
D)

.addGroup(passengerDialogLayout.createParallelGroup(javax.swing.Grou
pLayout.Alignment.LEADING, false)
            .addComponent(jPanel3,
javax.swing.GroupLayout.DEFAULT_SIZE, 178, Short.MAX_VALUE)
```

```
            .addComponent(jPanel2,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(bookPassengerButton,
javax.swing.GroupLayout.DEFAULT_SIZE, 58, Short.MAX_VALUE)
        .addContainerGap())
);

flightDialog.setAlwaysOnTop(true);
flightDialog.setLocation(new java.awt.Point(450, 200));
flightDialog.setResizable(false);
flightDialog.setSize(new java.awt.Dimension(380, 255));

titl.setFont(new java.awt.Font("Lucida Grande", 0, 15)); // NOI18N
titl.setText("Book New Flight");

jPanel5.setLayout(new java.awt.GridLayout(3, 1));

jLabel9.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
jLabel9.setText("Enter Plane ID #:");
jPanel5.add(jLabel9);

jLabel11.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
jLabel11.setText("Enter Departure Location:");
jPanel5.add(jLabel11);

jLabel12.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
jLabel12.setText("Enter Arrival Location:");
jPanel5.add(jLabel12);
```

```
jPanel6.setLayout(new java.awt.GridLayout(3, 1));
jPanel6.add(IdDialog);

depLocDialog.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Vancouver", "Edmonton", "Iqualuit", "Winnipeg",
"Toronto", "Ottawa", "Halifax" }));
jPanel6.add(depLocDialog);

arrLocDialog.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Vancouver", "Edmonton", "Iqualuit", "Winnipeg",
"Toronto", "Ottawa", "Halifax" }));
arrLocDialog.setSelectedIndex(1);
jPanel6.add(arrLocDialog);

bookFlightDialog.setFont(new java.awt.Font("Lucida Grande", 0, 15));
// NOI18N
bookFlightDialog.setText("Book Flight");
bookFlightDialog.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        bookFlightDialogActionPerformed(evt);
    }
});
     p1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/plane.png")));
// NOI18N
openingScreen.add(p1);
p1.setBounds(240, 540, 40, 40);

p2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/plane.png")));
// NOI18N
openingScreen.add(p2);
p2.setBounds(50, 170, 40, 50);
```

```
p3.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/plane.png")));
// NOI18N
openingScreen.add(p3);
p3.setBounds(50, 230, 40, 50);

p4.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/plane.png")));
// NOI18N
openingScreen.add(p4);
p4.setBounds(100, 330, 40, 40);

p5.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/plane.png")));
// NOI18N
openingScreen.add(p5);
p5.setBounds(10, 390, 40, 40);

clockLabel.setFont(new java.awt.Font("Sinhala Sangam MN", 0, 19));
// NOI18N
clockLabel.setText("21:45:16");
openingScreen.add(clockLabel);
clockLabel.setBounds(20, 10, 130, 40);

speedPanel.setBackground(new java.awt.Color(255, 255, 255));
speedPanel.setLayout(new java.awt.GridLayout(1, 3, 10, 0));

speed1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/play.png")));
// NOI18N
speed1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        speed1ActionPerformed(evt);
    }
});
```

```
speedPanel.add(speed1);

speed2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/forward.png"))
); // NOI18N
speed2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        speed2ActionPerformed(evt);
    }
});
speedPanel.add(speed2);

speed3.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/fastForward.pn
g"))); // NOI18N
speed3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        speed3ActionPerformed(evt);
    }
});
speedPanel.add(speed3);

openingScreen.add(speedPanel);
speedPanel.setBounds(20, 60, 170, 40);

mapPicture.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/flight/map.jpg")));
// NOI18N
openingScreen.add(mapPicture);
mapPicture.setBounds(0, 0, 1300, 771);

homePanel.add(openingScreen, "card1");

title.setFont(new java.awt.Font("Lucida Grande", 0, 18)); // NOI18N
title.setText("Manage Flights");
```

```
currentFlightsBooking.setModel(new
javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null},
        {null, null, null, null, null},
        {null, null, null, null, null},
        {null, null, null, null, null},
        {null, null, null, null, null}
    },
    new String [] {
        "Plane ID #", "Departure Location", "Departure Time",
"Arrival Location", "Arrival Time"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
currentFlightsBooking.setRowHeight(45);
currentFlightsBooking.setRowMargin(5);
jScrollPane1.setViewportView(currentFlightsBooking);

jLabel7.setFont(new java.awt.Font("Lucida Grande", 0, 14)); //
NOI18N
jLabel7.setText("Current Flights");
     //Main method:
public static void main(String args[]) {
    //add fist info to log file
    long time = System.currentTimeMillis();
    SimpleDateFormat sdf2 = new SimpleDateFormat("dd:MM:yyyy
```

```
HH:mm:ss");
    Date date = new Date(time);
    String output = sdf2.format(date);
    log = log + "Flight Monitoring v1.0 by Nazar Homeniuk \n" +
output + "\n";
    initialGeneratePlanes(); //generate the first planes

    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay
with the default look and feel.
     * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.
html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Flight.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Flight.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
```

```
java.util.logging.Logger.getLogger(Flight.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Flight.class.getName()).log(java.
util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Flight().setVisible(true);
        }
    });

    //When user closes the program
    Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
        public void run(){
            try{ //create the log file, copy the string data, and
close it
                log = log + sdf.format(new
Date(System.currentTimeMillis())) + " Program Terminated \n";

                OutputStream fout = new FileOutputStream("LogFile");
                OutputStream bout = new BufferedOutputStream(fout);
                OutputStreamWriter out = new
OutputStreamWriter(bout);

                out.write(log);
                out.flush();
                out.close();

            } catch (FileNotFoundException ex) {
                //
            } catch (IOException ex) {
```

```
                //
            }

        }
    }, "Shutdown-thread"));
}
```

AnimatePlanes.java

```
package flight;

import java.awt.Point;
import javax.swing.JLabel;

public class AnimatePlanes extends Thread{
    public void run(){
        while(true){
            //get locations of plane icons and add to array
            Point location1 = Flight.p1.getLocation();
            Point location2 = Flight.p2.getLocation();
            Point location3 = Flight.p3.getLocation();
            Point location4 = Flight.p4.getLocation();
            Point location5 = Flight.p5.getLocation();
            Point[] locations = {location1, location2, location3,
location4, location5};

            //call the animation methods for each icon
            Flight.p1.setLocation(animate(location1,
Flight.planes.get(0).arriveLoc));
            Flight.p2.setLocation(animate(location2,
Flight.planes.get(1).arriveLoc));
            Flight.p3.setLocation(animate(location3,
Flight.planes.get(2).arriveLoc));
            Flight.p4.setLocation(animate(location4,
Flight.planes.get(3).arriveLoc));
```

```
            Flight.p5.setLocation(animate(location5,
Flight.planes.get(4).arriveLoc));

            for (int i = 0; i < 5; i++){
                boolean tester = checkPlaneDest(locations[i],
Flight.planes.get(i)); //check if the plane arrived
                if (tester == true){ //if arrived:
                    //find which plane will replace the current one
                    long soonestTime = (long)(Math.pow(2, 55));
//define a number that is surely bigger then any possible system
time
                    int soonestTimeIndex = 0; //define the index
                    JLabel[] planeIcons = {Flight.p1, Flight.p2,
Flight.p3, Flight.p4, Flight.p5}; //add icons to array
                    int futureSize = Flight.futurePlanes.size();
//get the size of the array
                    if (futureSize > 0){ //if there are planes
upcoming:
                        for (int j = 0; j < futureSize; j++){
                            if
(Flight.futurePlanes.get(j).departTime < soonestTime){
                                //if the soonest time is less than
the current, set it, and set the index
                                soonestTimeIndex = j;
                                soonestTime =
Flight.futurePlanes.get(j).departTime;
                            }
                        }
                        //set the arrived plane to the future plane
with the soonest dpearture time
                        Flight.planes.set(i,
Flight.futurePlanes.get(soonestTimeIndex));

Flight.futurePlanes.remove(soonestTimeIndex); //remove it from the
future planes
```

```
                        Flight.dtm.removeRow(0); //remove the row
                    } else{ //if no upcoming planes, generate a
plane in the current one's place
                        Plane tempPlane =
Flight.generateFuturePlane();
                        Flight.planes.set(i, tempPlane);
                    }

planeIcons[i].setLocation(Flight.planes.get(i).departLoc); //set the
new location of the icon
                }
            }
            Flight.refreshTables(); //refresh the tables
            try {
                Thread.sleep(1000 / Flight.animationSpeed); //run x
times every second (x is the animation speed)
            } catch (InterruptedException ex) {
                //
            }
        }
    }

    //This method checks if the plane has reached its destination
    public boolean checkPlaneDest(Point location, Plane plane){
        Point airport = plane.arriveLoc; //get the arrival location
        long differenceY = Math.abs(airport.y - location.y);
//compare the distances in the x and y
        long differenceX = Math.abs(airport.x - location.x);
        if(((differenceY > -3)&&(differenceY < 3)) && ((differenceX
> -3)&&(differenceX < 3))){ //if the plane is at the destination,
return true
            return true;
        }
        return false; //else return false
    }
```

```
    //This method creates the new location for each icon
    public Point animate(Point location, Point destination){
        int destinationX = destination.x; //get the destination
location
        int destinationY = destination.y;

        double distanceX = destinationX - location.x; //find the x
and y distances
        double distanceY = destinationY - location.y;

        double difference = 0; //define variable
        double xMovement = 0;
        double yMovement = 0;

        //if the x and y distances are the same, the plane can move
diagonally
        if (Math.abs(distanceX) == Math.abs(distanceY)){
            if(distanceX >= 0){
                xMovement = 1;
            } else{
                xMovement = -1;
            }
            if (distanceY >= 0){
                yMovement = 1;
            } else{
                yMovement = -1;
            }
        } else if(Math.abs(distanceX) > Math.abs(distanceY)){ //if
the x distance is greater, move 1 space in that direction
            if(distanceX >= 0){
                xMovement = 1;
            } else{
                xMovement = -1;
            }
```

```
        } else if (Math.abs(distanceY) > Math.abs(distanceX)){ //if
the y distance is greater, move 1 space in that direction
            if (distanceY >= 0){
                yMovement = 1;
            } else{
                yMovement = -1;
            }
        }
        //create and return the generated point
        Point point = new Point((int)(location.x + xMovement),
(int)(location.y + yMovement));
        return point;
    }
}
```

Clock.java

```
package flight;

import java.text.SimpleDateFormat;
import java.util.Date;

public class Clock extends Thread{
    public void run(){

        while(true){
            Flight.totalFastForward = Flight.totalFastForward +
((Flight.animationSpeed * 1000) - 1000); //define the
totalFastForward value based on the animation speed
            long mlls = System.currentTimeMillis() +
Flight.totalFastForward; //get the clock time, convert to visible
time, set the output
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            Date date = new Date(mlls);
```

```
            String output = sdf.format(date);
            Flight.clockLabel.setText(output);

            try {
                Thread.sleep(1080); //run every second (but allow
time for the program to run)
            } catch (InterruptedException ex) {
                //
            }
        }
    }
}
```

Plane.java

```
package flight;

//import required resources
import java.awt.Point;
import java.util.ArrayList;

//define the class
public class Plane {
    //define the variables
    String ID;
    Point departLoc;
    long departTime;
    Point arriveLoc;
    long arriveTime;
    Passenger[] passengerArray;
    int[] seatingBooked;

    //create the contructor
    public Plane(String myID, Point mydepartLoc, long mydepartTime,
Point myarriveLoc, long myarriveTime, Passenger[] myPassengerArray,
int[] mySeatingBooked){
```

```
        ID = myID;
        departLoc = mydepartLoc;
        departTime = mydepartTime;
        arriveLoc = myarriveLoc;
        arriveTime = myarriveTime;
        passengerArray = myPassengerArray;
        seatingBooked = mySeatingBooked;
    }

    public void setPassengerArray(Passenger[] passengers){ //this
method changes the passengerArray
        passengerArray = passengers;
    }

    public void setSeatingBooked(int[] seating){ //this method
changes to seating array
        seatingBooked = seating;
    }

    //this method converts a point value to an airports
    public String getActualDepartLocation(){
        Point Vancouver = new Point(240, 540); //define coordinates
of all airports
        Point Edmonton = new Point(370, 510);
        Point Iqualuit = new Point(770, 270);
        Point Winnipeg = new Point(560, 600);
        Point Toronto = new Point(790, 690);
        Point Ottawa = new Point(830, 650);
        Point Halifax = new Point(990, 590);

        //test the locations
        if (departLoc.equals(Vancouver)){
            return "Vancouver";
        } else if (departLoc.equals(Edmonton)){
            return "Edmonton";
```

```
    } else if (departLoc.equals(Iqualuit)){
        return "Iqualuit";
    } else if (departLoc.equals(Winnipeg)){
        return "Winnipeg";
    } else if (departLoc.equals(Toronto)){
        return "Toronto";
    } else if (departLoc.equals(Ottawa)){
        return "Ottawa";
    } else if (departLoc.equals(Halifax)){
        return "Halifax";
    } else{
        return "Unidentified Location";
    }
}
//Same method as above except for an arrival location
public String getActualArrivalLocation(){
    Point Vancouver = new Point(240, 540);
    Point Edmonton = new Point(370, 510);
    Point Iqualuit = new Point(770, 270);
    Point Winnipeg = new Point(560, 600);
    Point Toronto = new Point(790, 690);
    Point Ottawa = new Point(830, 650);
    Point Halifax = new Point(990, 590);

    if (arriveLoc.equals(Vancouver)){
        return "Vancouver";
    } else if (arriveLoc.equals(Edmonton)){
        return "Edmonton";
    } else if (arriveLoc.equals(Iqualuit)){
        return "Iqualuit";
    } else if (arriveLoc.equals(Winnipeg)){
        return "Winnipeg";
    } else if (arriveLoc.equals(Toronto)){
        return "Toronto";
    } else if (arriveLoc.equals(Ottawa)){
```

```
            return "Ottawa";
        } else if (arriveLoc.equals(Halifax)){
            return "Halifax";
        } else{
            return "Unidentified Location";
        }
    }
}
```

Passenger.java

```
package flight;

public class Passenger {
    //define variables
    String name;
    String age;
    String country;

    //create constructors
    public Passenger(String myName, String myAge, String myCountry){
        name = myName;
        age = myAge;
        country = myCountry;
    }
}
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА ЗАСОБУ МОНІТОРИНГУ АВІАРЕЙСІВ

Рисунок Г.1 – Назва роботи

Рисунок Г.2 – Мета, об’єкт і предмет дослідження

Рисунок Г.3 – Наукова новизна та практична цінність

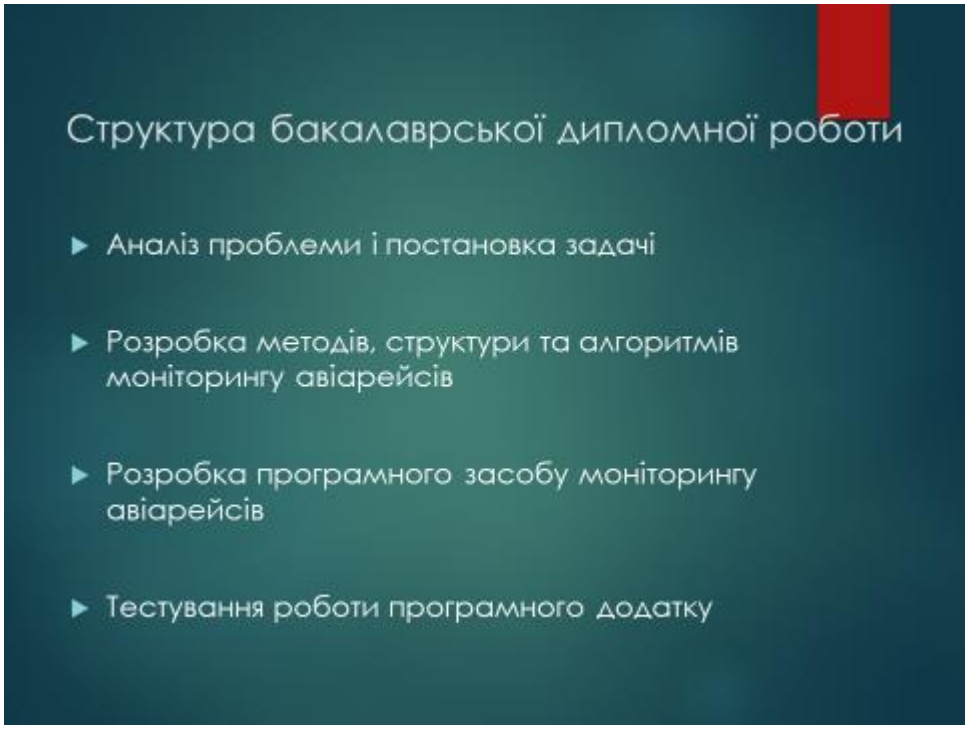


Рисунок Г.4 – Структура бакалаврської дипломної роботи

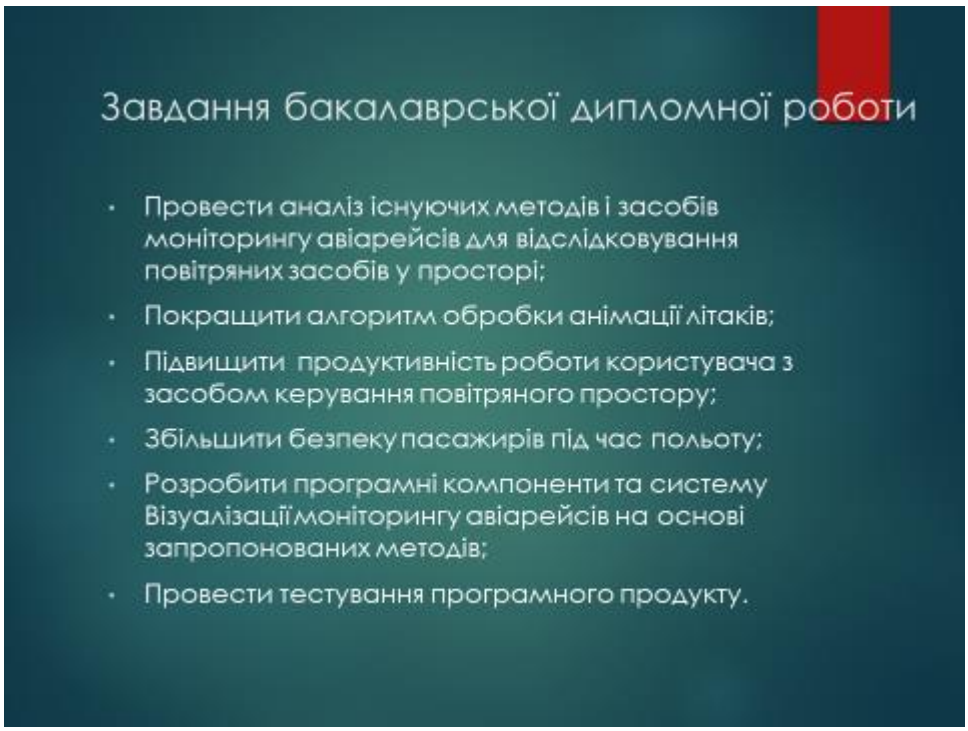


Рисунок Г.5 – Завдання бакалаврської дипломної роботи

Рисунок Г.6 – Аналоги

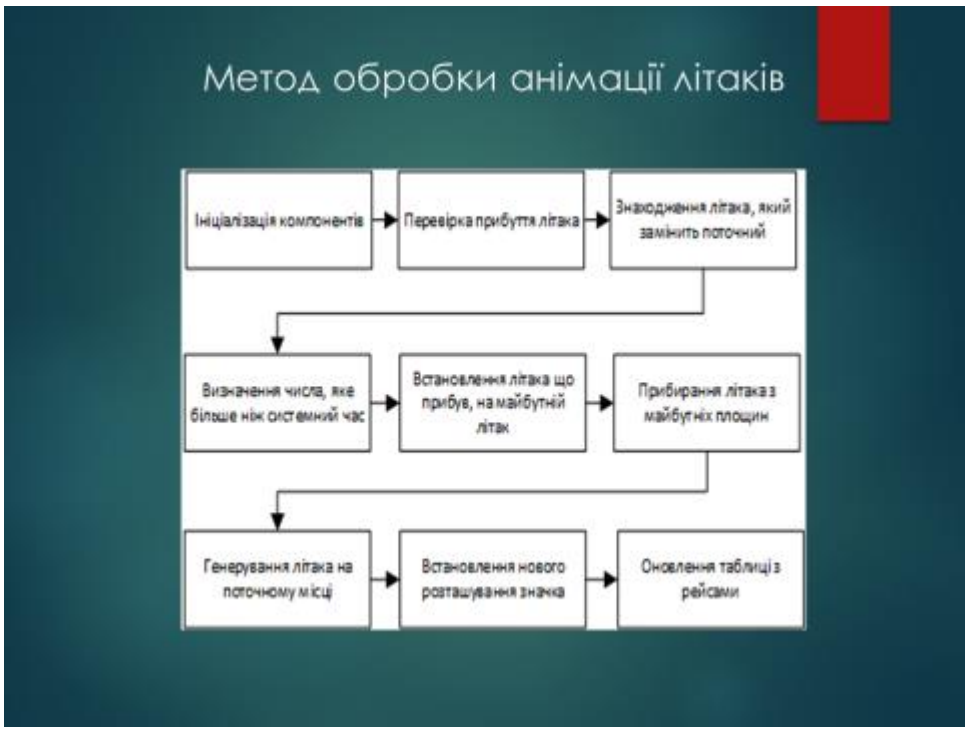


Рисунок Г.7 – Метод обробки анімації літаків

Рисунок Г.8 – Загальний алгоритм програмного засобу

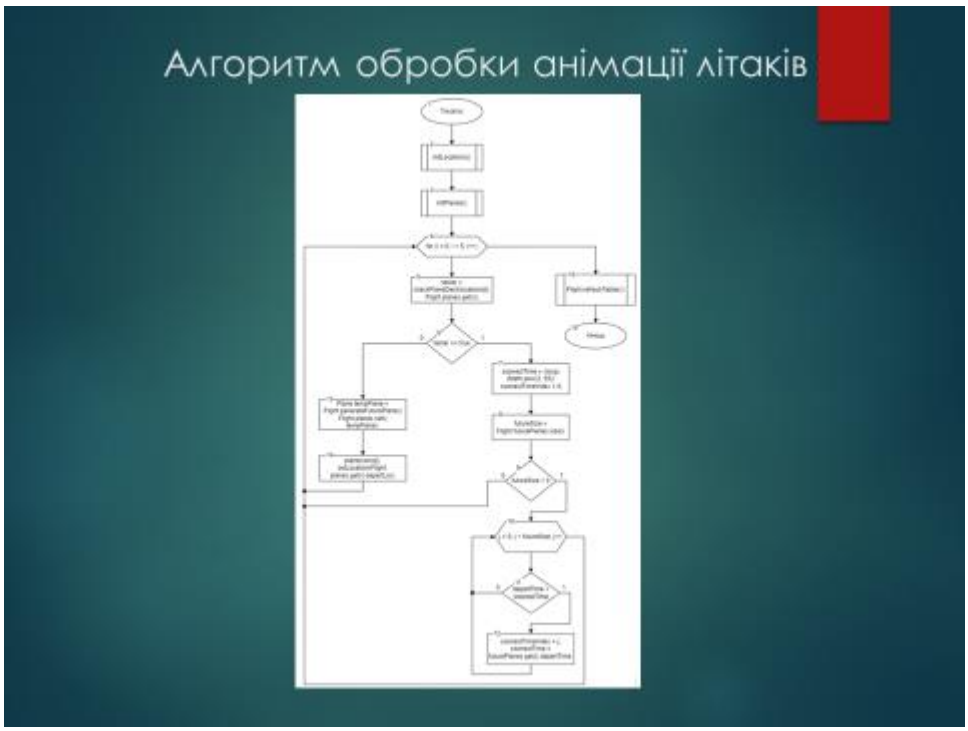


Рисунок Г.9 – Алгоритм обробки анімації літаків

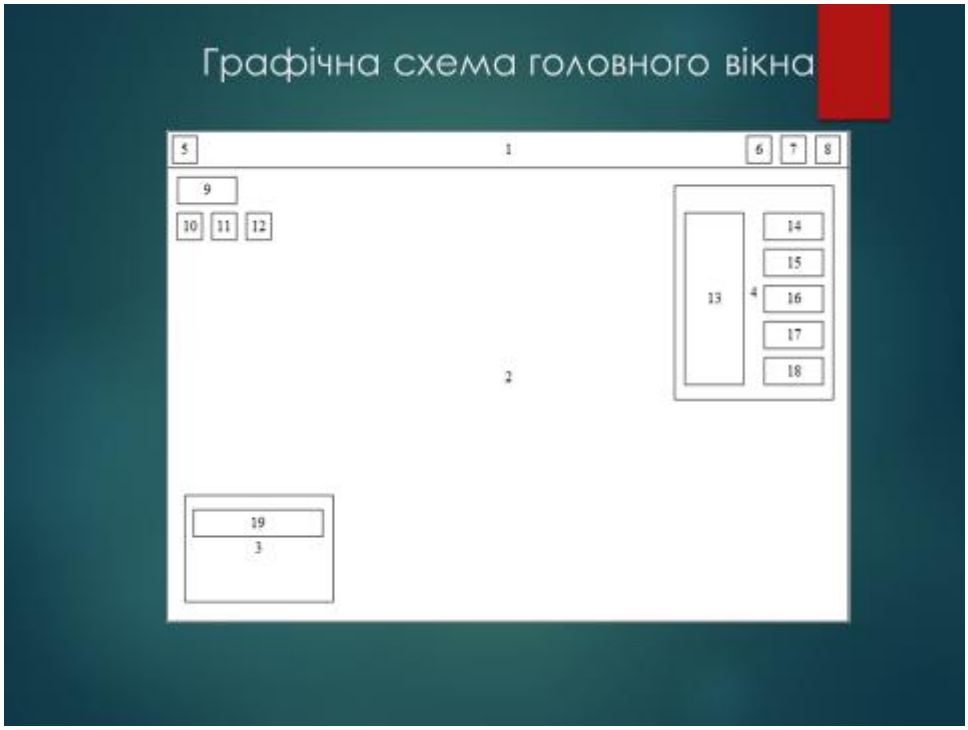


Рисунок Г.10 – Графічна схема головного вікна

Рисунок Г.11 – Тестування програмного засобу моніторингу авіарейсів

Рисунок Г.12 – Головне вікно програмного засобу

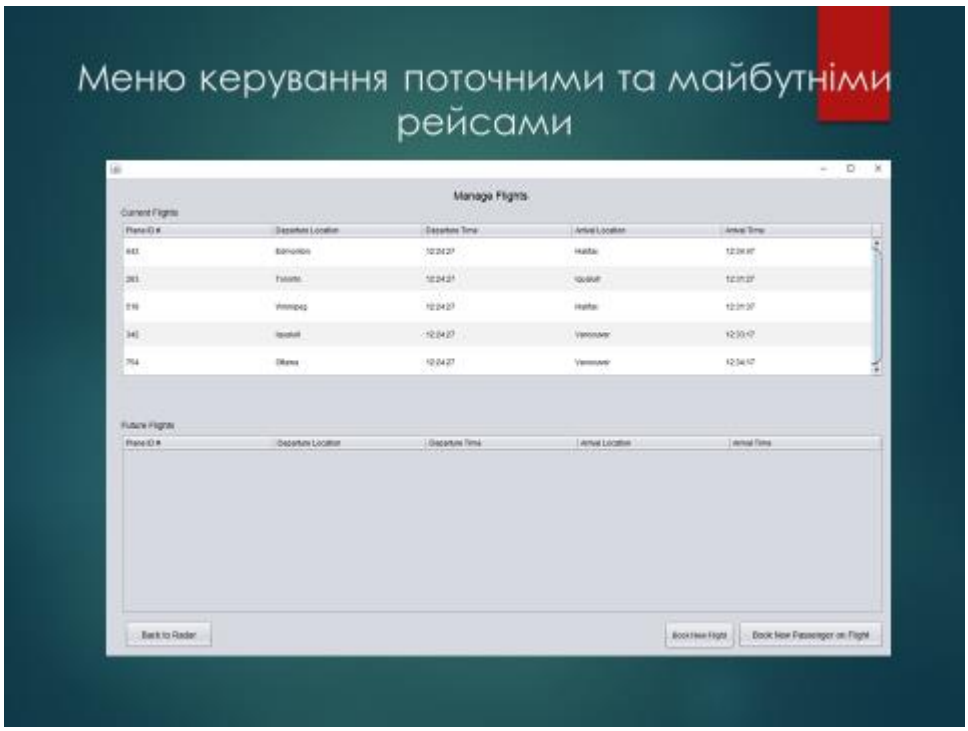


Рисунок Г.13 – Меню керування поточними та майбутніми рейсами

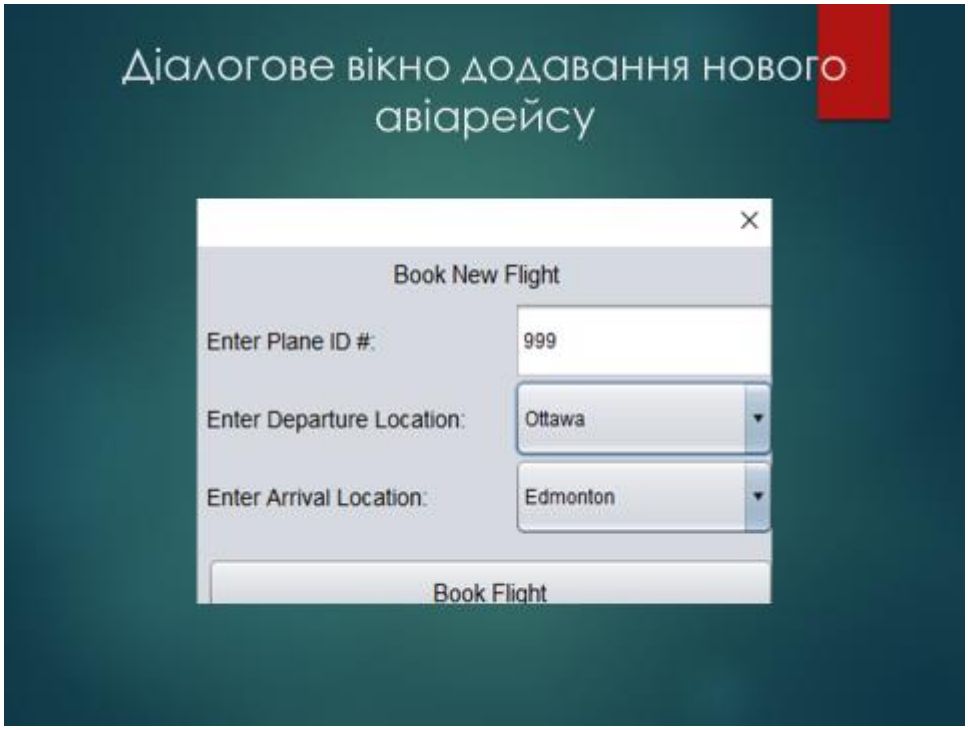


Рисунок Г.14 – Діалогове вікно додавання нового рейсу

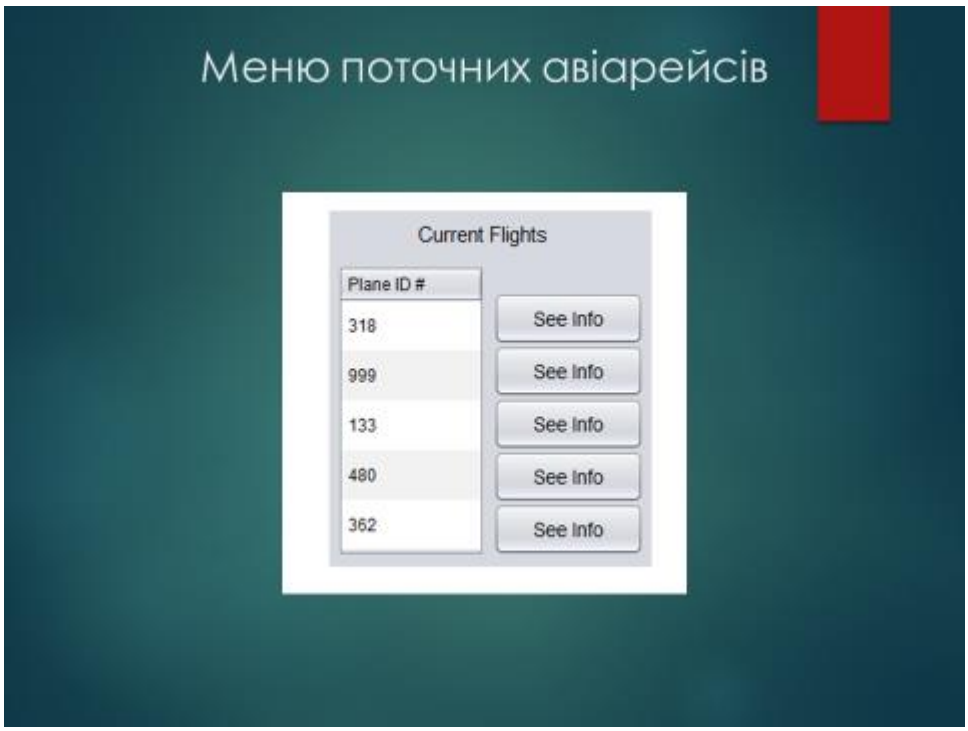


Рисунок Г.15 – Меню поточних авіарейсів

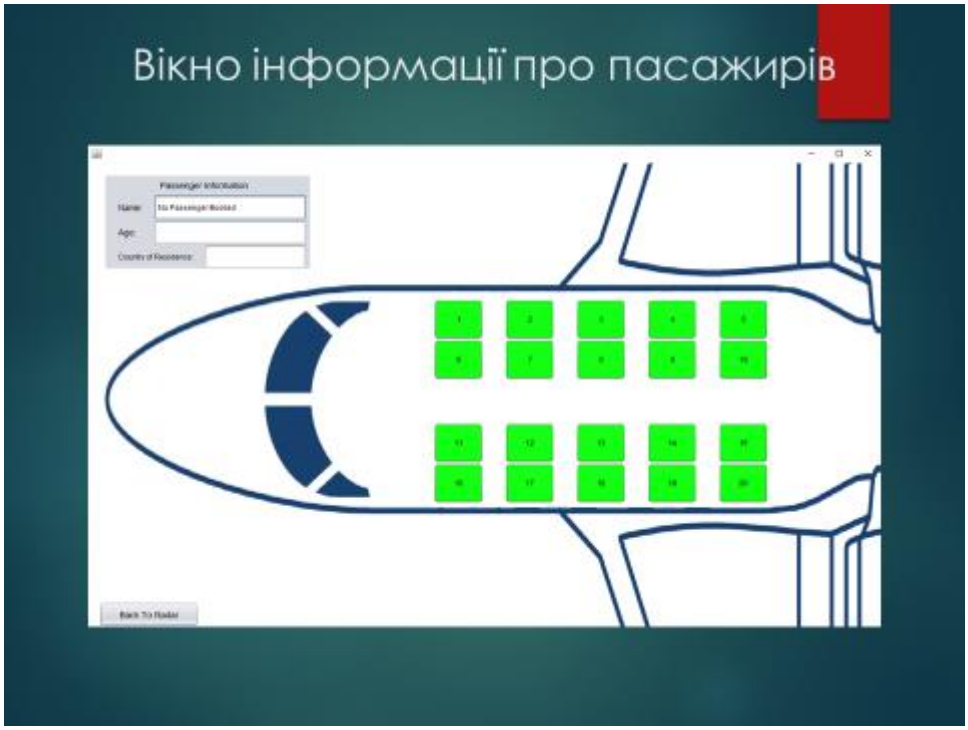


Рисунок Г.16 – Вікно інформації про пасажирів

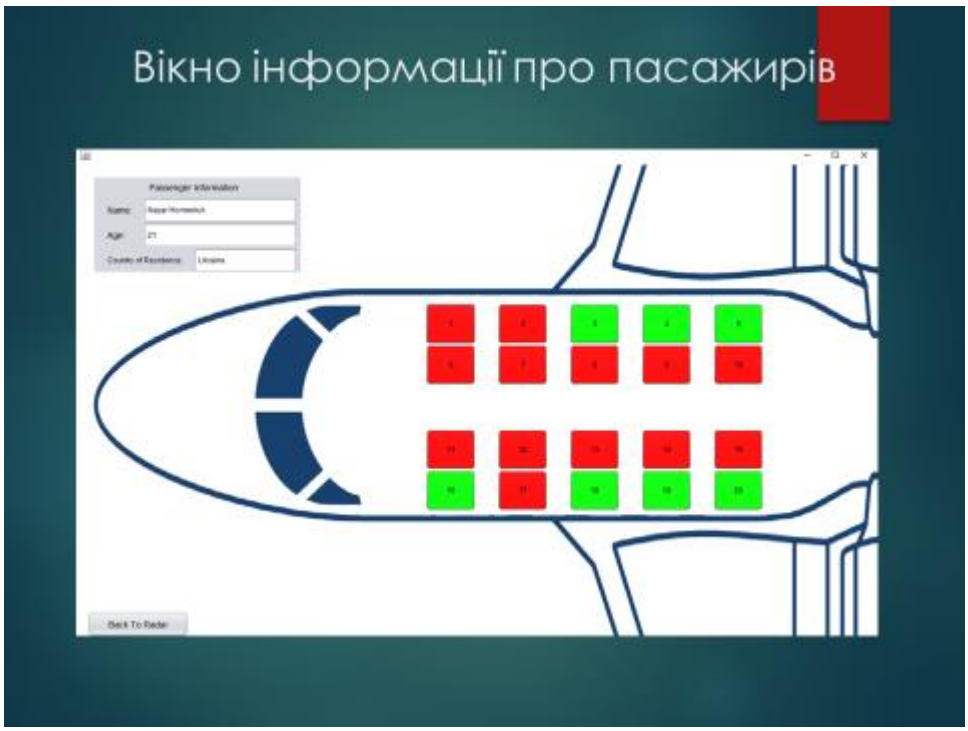


Рисунок Г.17 – Вікно інформації про пасажирів

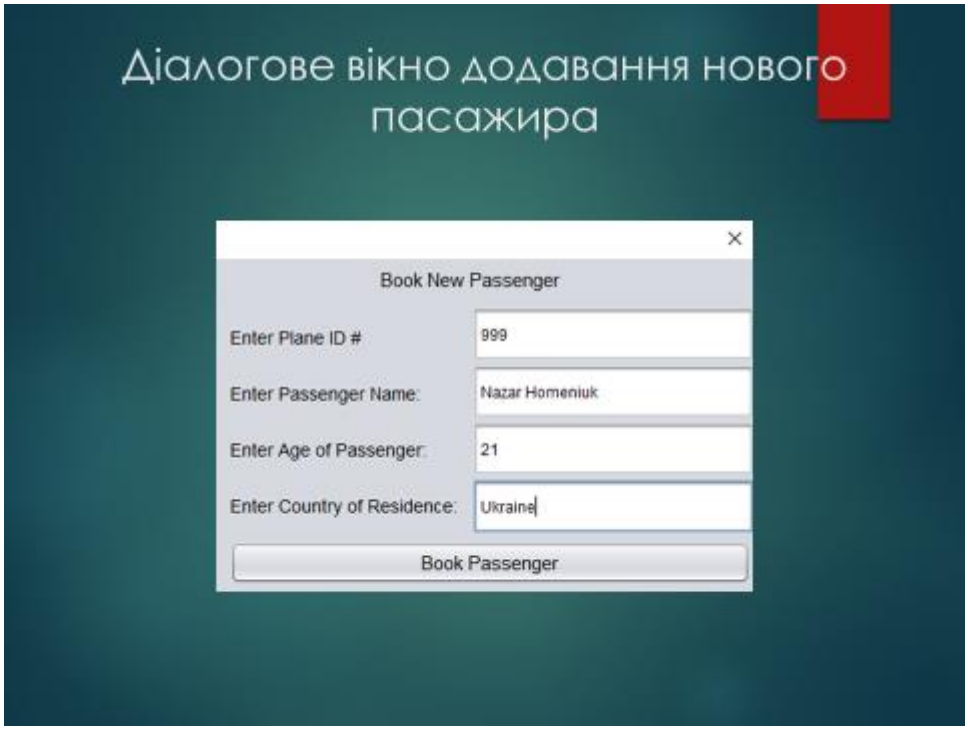


Рисунок Г.18 – Діалогове вікно додавання нового пасажира

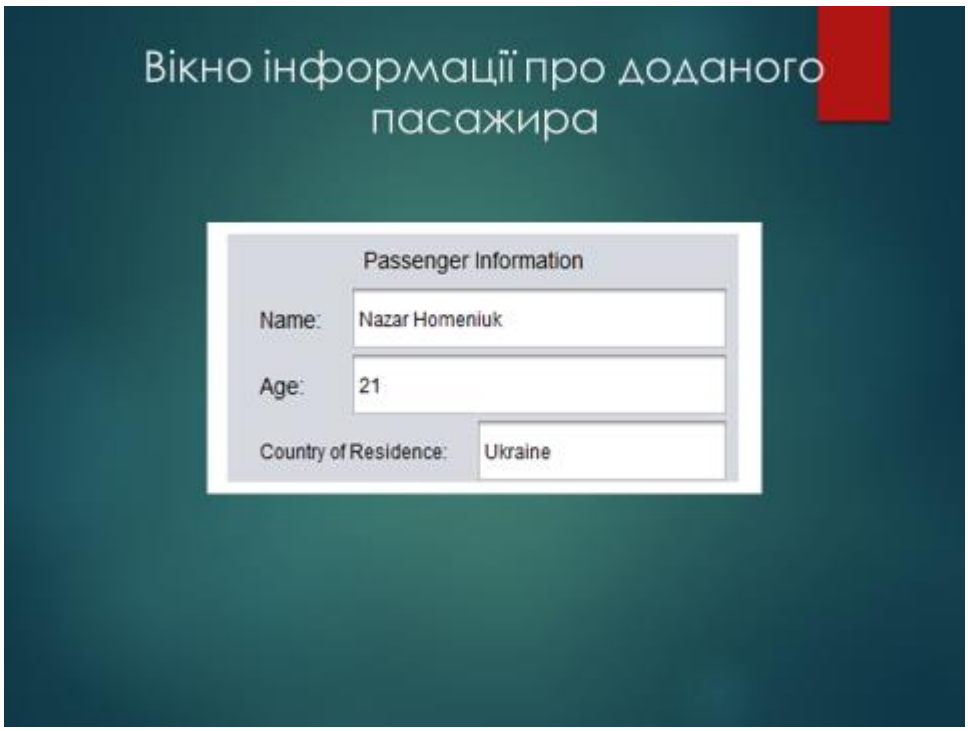


Рисунок Г.19 – Вікно інформації про доданого пасажира

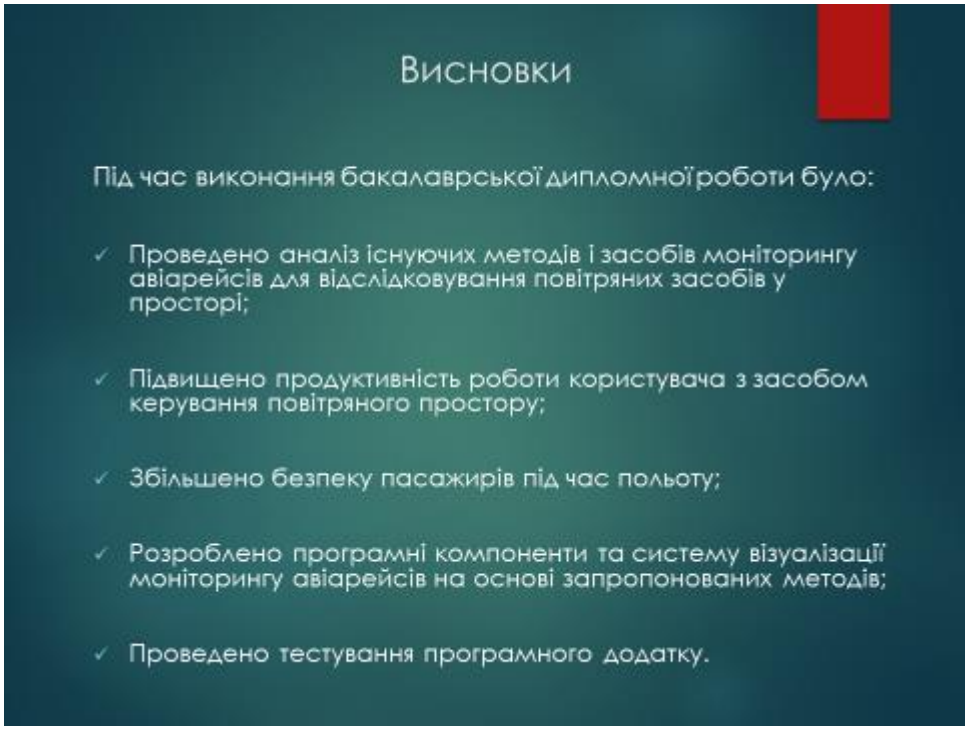


Рисунок Г.20 – Висновки

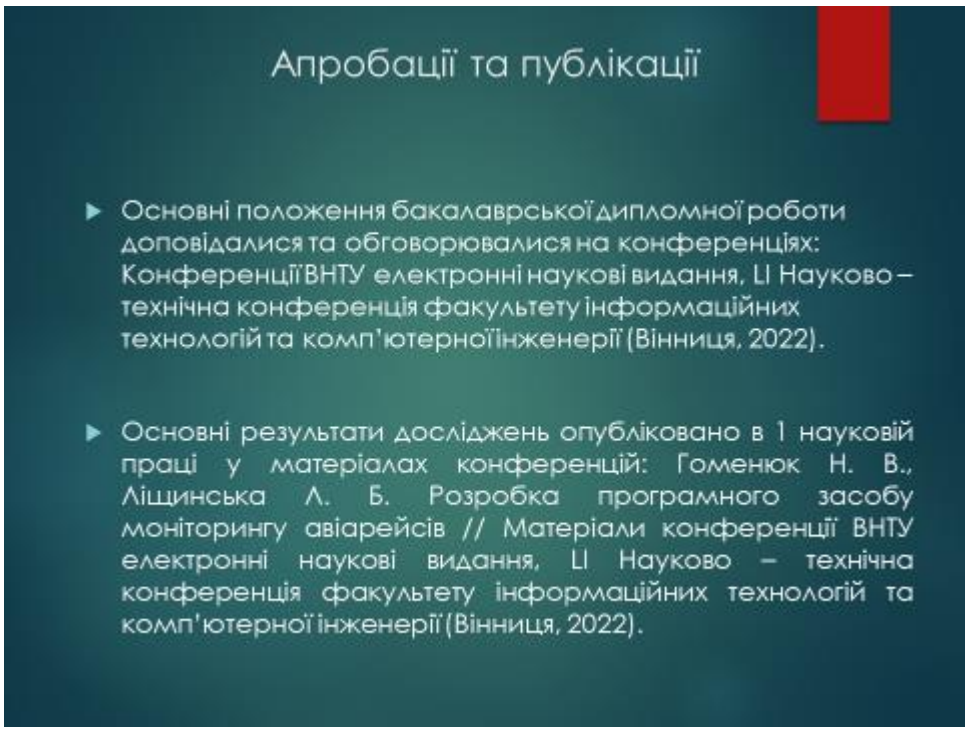


Рисунок Г.21 – Апробації та публікації

Рисунок Г.22 – Завершення