

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Кафедра програмних засобів

(найменування кафедри)

**КУРСОВИЙ ПРОЄКТ
(РОБОТА)**

з дисципліни «Об'єктно-орієнтоване програмування»

(назва дисципліни)

на тему: Розробка програмного забезпечення автоматизації праці диспетчера в касах аерофлоту з використання принципів ООП

Студента 1 курсу КНТ-113сп групи
спеціальності 121 Інженерія
програмного забезпечення
освітня програма (спеціалізація) інженерія
програмного забезпечення
Бедських А. В.

(прізвище та ініціали)

Керівник асистент, Короткий О. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала
Кількість балів: Оцінка: ECTS

Члени комісії

| | |
|----------|--|
| (підпис) | Короткий О. В. (прізвище та ініціали) |
| (підпис) | Каплієнко Т.І. (прізвище та ініціали) |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет Запорізький національний університет НУ «Запорізька політехніка». Факультет комп'ютерних наук і технологій ;
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2022 року

З А В Д А Н Н Я

НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Бедських Артем Віталійович
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка програмного забезпечення автоматизації праці диспетчера в касах аерофлоту з використання принципів ООП
 керівник проекту (роботи) Короткий Олександр Володимирович, асистент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від _____
2. Строк подання студентом проекту (роботи) 21 грудня 2022 року
3. Вихідні дані до проекту (роботи) розробити додаток згідно теми курсової роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення; 2. Проєктування програмного забезпечення системи; 3. Розробка програмного забезпечення системи; 4. Аналіз ефективності програмного забезпечення; 5. Розробка документів на супроводження програмного забезпечення.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---------------------|---|----------------|---------------------------|
| | | завдання видав | прийняв виконане завдання |
| 1-5 Основна частина | Короткий О.В., асистент | | |
| | | | |
| | | | |

7. Дата видачі завдання 06 листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів курсового проєкту (роботи) | Строк виконання етапів проєкту (роботи) | Примітка |
|-------|--|---|----------------|
| 1. | Аналіз індивідуального завдання. | 1 тиждень | |
| 2. | Аналіз програмних засобів, що будуть використовуватись в роботі. | 2 тиждень | |
| 3. | Аналіз структур даних, що необхідно використати в курсовій роботі. | 3 тиждень | |
| 4. | Затвердження завдання | 4 тиждень | |
| 5. | Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача. | 5-9 тиждень | |
| 6. | Аналіз вимог до апаратних засобів | 9 тиждень | |
| 7. | Розробка програмного забезпечення | 9-13 тиждень | |
| 8. | Проміжний контроль | 10 тиждень | Розділи 1-5 ПЗ |
| 9. | Оформлення, відповідних пунктів пояснювальної записки. | 10-14 тиждень | Розділи 1-2 ПЗ |
| 10. | Захист курсової роботи. | 15 тиждень | |

Студент _____ Бедських А.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Короткий О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до курсової роботи містить 90 сторінок, 16 рисунків, 0 таблиць, 2 додатки, 8 джерел.

Пояснювальна записка складається з шести розділів.

Розділ «Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення» містить в собі аналіз сучасних методів до проектування програмного забезпечення та аналіз різних сфер життя.

Розділ «Проектування програмного забезпечення системи» містить аналіз функцій системи, розробку UML діаграм використання, проектування графічного інтерфейсу та постановку мети.

Розділ «Розробка програмного забезпечення системи» містить розробку структури програми, розробку UML діаграми класів та опис класів програмного забезпечення.

Розділ «Аналіз ефективності програмного забезпечення» містить аналіз застосунку на швидкодія та масштабованість, аналіз ефективності кожного компоненту системи, а також тестування всього програмного забезпечення.

Розділ «Розробка документів на супроводження програмного забезпечення» містить інструкцію для програміста, де є розгортка застосунку на локальній машині, а також інструкцію користувача.

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ДИСПЕЧЕР АЕРОФЛОТУ, РОБОТА З ДАНИМИ, TYPESCRIPT, TAILWIND CSS, VITE, GIT, АЛГОРИТМИ, ОПТИМІЗАЦІЯ, ТЕСТУВАННЯ

ЗМІСТ

| | |
|--|----|
| Вступ..... | 6 |
| 1 Огляд та аналіз сучасних методів та засобів проектування програмного забезпечення..... | 7 |
| 1.1 Огляд сучасного стану питання | 7 |
| 1.2 Основні шляхи вирішення поставленої задачі | 8 |
| 1.3 Формулювання мети досліджень | 9 |
| 2 Проектування програмного забезпечення системи..... | 10 |
| 2.1 Постановка мети | 10 |
| 2.2 Аналіз функцій системи | 10 |
| 2.3 Проектування графічного інтерфейсу | 11 |
| 2.4 Розробка структури системи..... | 13 |
| 3 Розробка програмного забезпечення системи | 15 |
| 3.1 Розробка UML діаграми класів | 15 |
| 3.2 Розробка UML діаграми використання | 15 |
| 3.3 Опис класів програмного комплексу | 16 |
| 4 Аналіз ефективності програмного забезпечення..... | 25 |
| 4.1 Базовий аналіз ефективності програмного забезпечення..... | 25 |
| 4.2 Тестування програмного забезпечення | 26 |
| 5 Розробка документів на супроводження програмного забезпечення | 30 |
| 5.1 Інструкція програміста | 30 |
| 5.2 Інструкція користувачеві | 32 |
| Висновки | 35 |
| Перелік джерел посилання | 36 |
| ДОДАТОК А | 37 |
| ДОДАТОК Б..... | 87 |

ВСТУП

В умовах сучасного технологічного розвитку, де авіаційна індустрія відіграє ключову роль у забезпеченні глобальної мобільності, важливою стає автоматизація та оптимізація робочих процесів в обслуговуванні пасажирського повітряного транспорту. Зокрема, диспетчери у касах авіакомпаній стикаються з великим обсягом інформації та завдань, що вимагають швидкої та точної обробки. У цьому контексті розробка програмного забезпечення, яке базується на принципах об'єктно-орієнтованого програмування (ООП), стає ключовою для досягнення ефективної автоматизації та підвищення якості обслуговування.

Ця курсова робота націлена на вивчення та розробку програмного забезпечення для автоматизації роботи диспетчерів у касах авіакомпаній, з використанням принципів ООП. Індустрія авіаційного транспорту потребує ефективних інструментів для оптимізації процесів продажу та обслуговування квитків, а також для взаємодії з різними аспектами логістики та пасажирськими даними. Використання принципів ООП не лише забезпечить високий рівень функціональності програми, але і забезпечить гнучкість та легкість в обслуговуванні, що особливо важливо в динамічному середовищі авіаційного бізнесу.

Основною метою дослідження є створення програмного продукту, який забезпечить ефективну роботу диспетчерів, автоматизуючи рутинні операції та спрощуючи взаємодію з базами даних, бронюванням та іншими системами. У цьому контексті, конкретні завдання включають в себе реалізацію функцій створення квитків, моніторингу доступності місць, а також оптимізацію обробки фінансових та пасажирських даних.

Ця робота має важливе значення для подальшого розвитку авіаційного сектору, сприяючи підвищенню ефективності та забезпеченню високого рівня обслуговування пасажирів [1].

1 ОГЛЯД ТА АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд сучасного стану питання

Однією з ключових складових успіху в розробці програмного забезпечення для автоматизації диспетчерської діяльності в касах авіакомпаній є глибокий огляд та аналіз поточного стану цієї галузі. У зв'язку зі стрімким розвитком авіаційного сектору та зростанням пасажиропотоку, виникає необхідність у вдосконаленні систем обробки інформації та оптимізації робочих процесів.

Сучасні тенденції вказують на активне впровадження цифрових технологій та програмних рішень для автоматизації діяльності диспетчерів. Велика увага приділяється розробці систем, які забезпечують швидку та точну обробку великого обсягу інформації, а також поліпшують взаємодію з пасажиром та іншими службами авіакомпаній.

Зростання конкуренції в авіаційній галузі покладає великий тиск на оптимізацію робочих процесів та підвищення якості обслуговування. Системи автоматизації повинні враховувати різноманітні вимоги та стандарти, а також бути готовими до масштабування в разі зростання обсягів роботи.

За останні роки відзначається тенденція до використання принципів об'єктно-орієнтованого програмування (ООП) у розробці програмного забезпечення для авіаційного сектору. Це забезпечує більш гнучку та розширювану архітектуру, спрощує супровід та розвиток систем.

Огляд сучасного стану не може обійти увагою міжнародний досвід у розробці програмного забезпечення для авіаційного сектору. Аналіз кращих практик та впровадження передових рішень, які успішно використовуються у світі, дозволяє враховувати найефективніші підходи та уникнути можливих помилок.

1.2 Основні шляхи вирішення поставленої задачі

При аналізі основних напрямків вирішення проблем в цій сфері можна виділити кілька перспективних та потенційно вдалих підходів.

Використання інтегрованих систем управління є одним із перших шляхів у вирішенні завдань автоматизації диспетчерської діяльності. Ці системи забезпечують централізоване керування різними процесами та функціями, що полегшує моніторинг та взаємодію з пасажирями, рейсами та білетами.

Використання технологій штучного інтелекту, таких як машинне навчання та аналітичні алгоритми, відкриває нові можливості для покращення рішень у сфері авіаційної диспетчерської. Автоматизовані системи можуть прогнозувати попит, вирішувати проблеми та оптимізувати робочі процеси.

Створення інтуїтивних графічних інтерфейсів є ключовим напрямком для полегшення взаємодії диспетчерів з програмним забезпеченням. Зручний та легкий у використанні інтерфейс сприяє швидкому доступу до необхідних функцій та редагуванню даних.

Впровадження шаблонів проєктування, зокрема абстрактної фабрики та адаптера, може значно полегшити розробку та управління програмним комплексом. Ці шаблони дозволяють створювати гнучкі та розширювані системи [2].

Використання принципів об'єктно-орієнтованого програмування (ООП) є ключовим аспектом при розробці програмного забезпечення для автоматизації диспетчерської роботи. Це сприяє створенню модульних та гнучких систем, які легко підтримувати та розширювати.

Важливо враховувати баланс між використанням інноваційних технологій та забезпеченням стабільності системи. Інновації повинні вирішувати конкретні завдання та допомагати підвищити ефективність робочих процесів.

Розробка та впровадження ефективних заходів забезпечення безпеки є невід'ємною частиною вирішення завдань у галузі авіаційного програмного забезпечення.

Усі ці основні шляхи вирішення проблем автоматизації диспетчерської роботи в касах авіакомпаній взаємодіють та доповнюють один одного, створюючи комплексний та ефективний підхід до створення програмного забезпечення, що відповідає вимогам сучасності [3].

1.3 Формулювання мети досліджень

Головною метою є створення програмного продукту, що відповідає високим стандартам ефективності та зручності в користуванні для диспетчерів авіакас, а також покращення загальної продуктивності та точності роботи у цій сфері. Дослідження буде спрямоване на вивчення поточних проблем та визначення ключових аспектів, які можуть бути покращені за допомогою програмного забезпечення.

Мета включає в себе також аналіз існуючих методів та засобів автоматизації в диспетчерській діяльності, враховуючи як національний, так і міжнародний досвід. Отримання конкретних результатів, які будуть враховувати потреби та виклики авіаційного сектору, є важливою частиною мети дослідження. Конкретизація висновків та розробка рекомендацій стануть основою для подальшої реалізації програмного забезпечення та внесення позитивних змін у сферу диспетчерського обслуговування в авіакасах

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Постановка мети

Основною метою дослідження є розробка програмного забезпечення, який забезпечить ефективну роботу диспетчерів, автоматизуючи рутинні операції та спрощуючи взаємодію з зовнішнім сховищем даних, бронюванням та іншими системами. Програмний продукт повинен демонструвати високий рівень швидкодії та забезпечувати зручний інтерфейс для користувача [1].

2.2 Аналіз функцій системи

Аналіз функцій системи є важливим етапом у розробці програмного забезпечення для забезпечення автоматизації праці диспетчера в касах аерофлоту. Цей етап дозволяє визначити основні функціональні вимоги до системи, які є необхідними для її подальшого проектування та реалізації.

Виокремлені функції враховують потреби диспетчера та надають йому можливість здійснювати весь спектр операцій, необхідних для безперебійної та ефективної роботи. Аналіз функцій системи дозволяє визначити їх характеристики та особливості для подальшого реалізації та інтеграції у програмному забезпеченні [4].

У рамках курсової роботи передбачається розробка програмного забезпечення для забезпечення автоматизації праці диспетчера в касах аерофлоту з використанням об'єктно-орієнтованого програмування. Основними функціями системи є:

- функція відображення даних про рейс, пасажирів та білет в вигляді таблиці;
- функція зміни рейсу, пасажирів та білету через форму;
- функція створення нового рейсу, пасажирів та білету через форму;
- функція пошуку по даним;

– функція видалення даних.

Однією з ключових функцій системи - функція відображення даних про рейс, пасажирів та білет в вигляді таблиці. Ця функція передбачає зручний та легкий доступ до важливої інформації через візуальне представлення у вигляді таблиці. Диспетчер може швидко переглядати та оцінювати дані про рейси, пасажирів та білети, що сприяє прийняттю обґрунтованих рішень та оперативному реагуванню на можливі зміни.

Другою важливою функцією є функція зміни рейсу, пасажирів та білету через форму. Ця опція надає можливість диспетчеру вносити зміни в існуючі дані зручним та інтуїтивно зрозумілим способом. Використання форми дозволяє введення необхідної інформації про рейс, пасажирів та білет, а також забезпечує контроль за правильністю та цілісністю змінених даних.

Третьою ключовою функцією є функція створення нового рейсу, пасажирів та білету через форму. Ця функція важлива для оперативного введення нових даних, які можуть виникнути в результаті розширення аерофлоту або зміни розкладу. Використання форми для створення нових записів забезпечує однообразність та стандартизацію введення даних [5].

Останньою, але не менш важливою, функцією є функція пошуку по даним. Ця опція дозволяє диспетчеру швидко та ефективно знаходити необхідну інформацію в системі. Пошук може бути здійснений за різними параметрами, такими як ім'я пасажирів, номер рейсу або інші ключові атрибути, що полегшує роботу диспетчера та дозволяє швидко реагувати на поточні ситуації.

2.3 Проектування графічного інтерфейсу

Розробка графічного інтерфейсу (GUI) є ключовим етапом у створенні програмного забезпечення для автоматизації праці диспетчера аерофлоту. Інтерфейс повинен бути зручним, інтуїтивно зрозумілим та забезпечувати зручність взаємодії користувача з системою.

Основні елементи інтерфейсу:

- таби, за допомогою яких можна переходити між сторінками пасажирів, рейсів та білетів;
- кнопка створення пасажирів, рейсів або білетів;
- форма для створення та редагування пасажирів, рейсів або білетів;
- таблиця для показу даних;
- кнопка видалення елемента;
- кнопка редагування елемента.

На інтерфейсі передбачено використання табів для категорій пасажирів, рейсів і білетів, що дозволяє диспетчеру легко перемикатися між різними частинами системи. Кожен таб має відповідний розділ, де можна переглядати та редагувати відомості.

Зокрема, важливою функцією є наявність кнопок для створення нових об'єктів - пасажирів, рейсів або білетів. Ці кнопки надають диспетчеру можливість легко додавати нові записи до системи. Крім того, для зручності диспетчера передбачено використання форм для створення та редагування даних. Це дозволяє вводити необхідну інформацію про пасажирів, рейси та білети у вигляді структурованої форми [2].

Однією з важливих частин інтерфейсу є таблиця для відображення даних. Ця таблиця дозволяє користувачу переглядати інформацію про пасажирів, рейси та білети у зручному форматі. Таблиця може включати важливі атрибути, такі як ім'я пасажирів, номер рейсу, дата вильоту та інші відомості, які полегшують сприйняття інформації.

Для управління об'єктами в системі передбачено кнопки видалення та редагування елементів. Це надає користувачу можливість легко видаляти або вносити зміни до існуючих записів. Кнопка видалення дозволяє користувачу видаляти з системи застарілу чи непотрібну інформацію, а кнопка редагування - вносити корективи у вже існуючі дані.

2.4 Розробка структури системи

Структурна модель системи, показана на рисунку 2.1, визначає основні компоненти та їх взаємозв'язки.

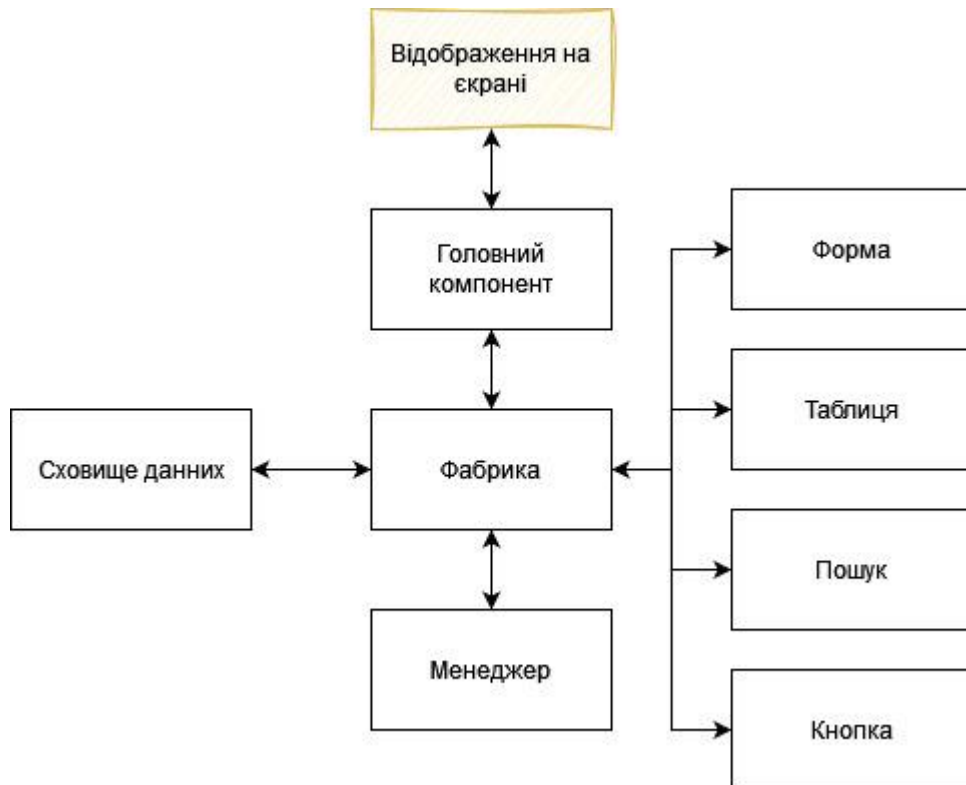


Рисунок 2.1 – Структурна модель застосунку

Система складається з основних компонентів, що взаємодіють між собою:

- головний компонент;
- фабрика. Оскільки застосунок побудований на основі абстрактної фабрики ця структура нам потрібна;
- менеджер який локально зберігає данні про пасажирів, рейси та білети та може додавати/видаляти, а також змінювати їх;
- сховище даних;
- форма. Форма використовується для додавання та зміни елементів;

- таблиця. Використовується для відображення даних в зручний спосіб;
- пошук. Використовується для зручного пошуку по елементам;
- кнопка. Окрема структура кнопки для зручності.

Головний компонент є осердям системи, відповідальним за ініціалізацію та координацію роботи інших компонентів.

Фабрика використовується для реалізації абстрактної фабрики, яка визначає інтерфейс для створення родини взаємодіючих об'єктів без конкретизації їхніх класів.

Менеджер виступає як центральний об'єкт, що управляє локальним зберіганням даних про пасажирів, рейси та білети. Він забезпечує можливість додавання, видалення та зміни цих об'єктів.

Сховище даних використовується для зберігання інформації та забезпечення доступу до неї з інших компонентів системи.

Форма представляє інтерфейс для додавання та зміни елементів. Таблиця використовується для відображення даних у зручному форматі, забезпечуючи користувачеві зручний перегляд інформації [6].

Пошук дозволяє здійснювати швидкий та ефективний пошук по елементам системи.

Окремою структурою є кнопка, яка забезпечує зручний та інтуїтивний спосіб виклику певних функцій чи дій в системі. Цей компонент спрощує взаємодію користувача з програмою, забезпечуючи легкий доступ до необхідних опцій.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Розробка UML діаграми класів

На рисунку 3.1 представлена UML діаграма класів.

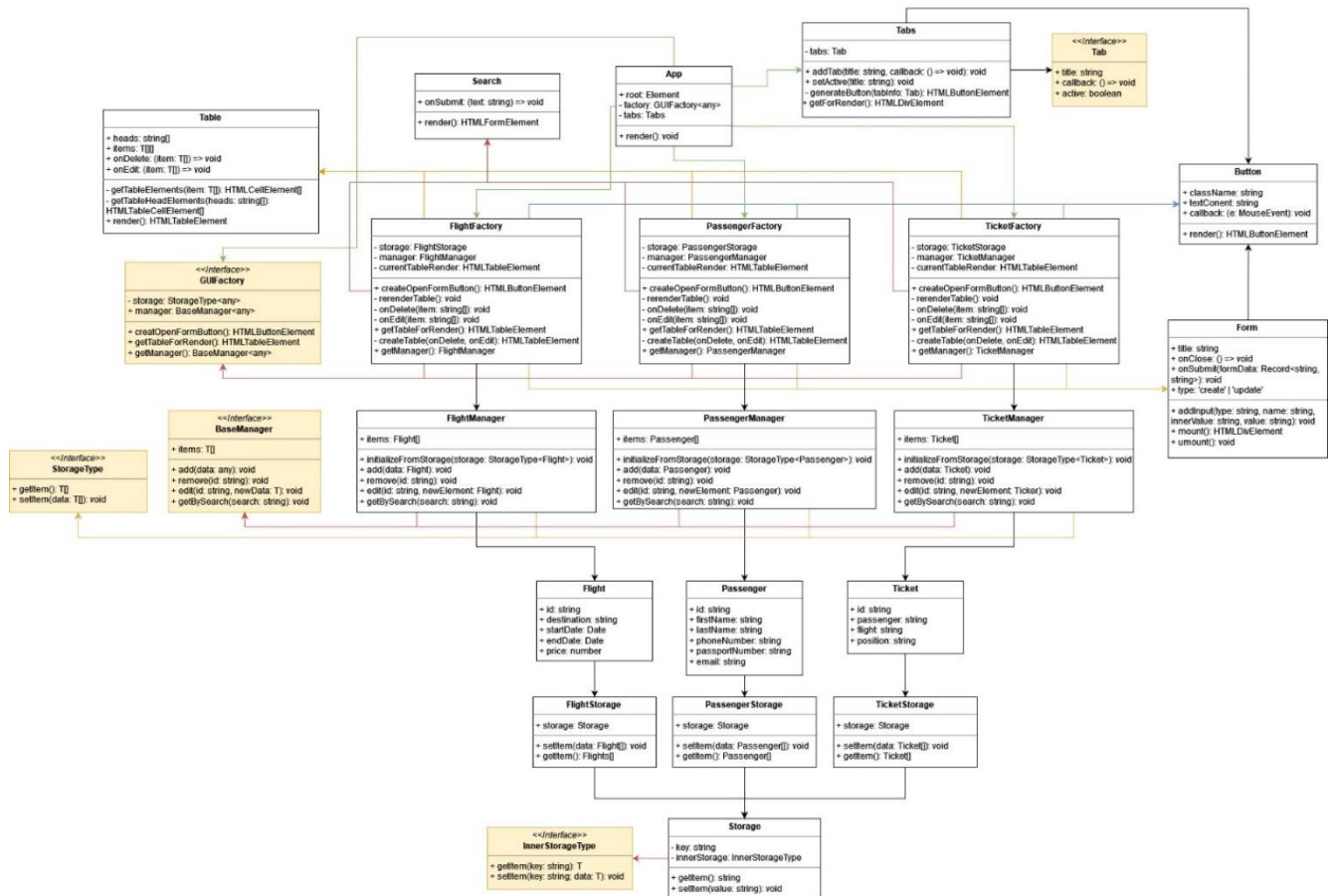


Рисунок 3.1 – UML діаграма класів

3.2 Розробка UML діаграми використання

UML діаграма використання визначає, які функції можуть бути використані користувачами. До основних входять створення та редагування пасажирів, рейсів та білетів, а також пошук [3].

На рисунку 3.2 показана UML діаграма використання.

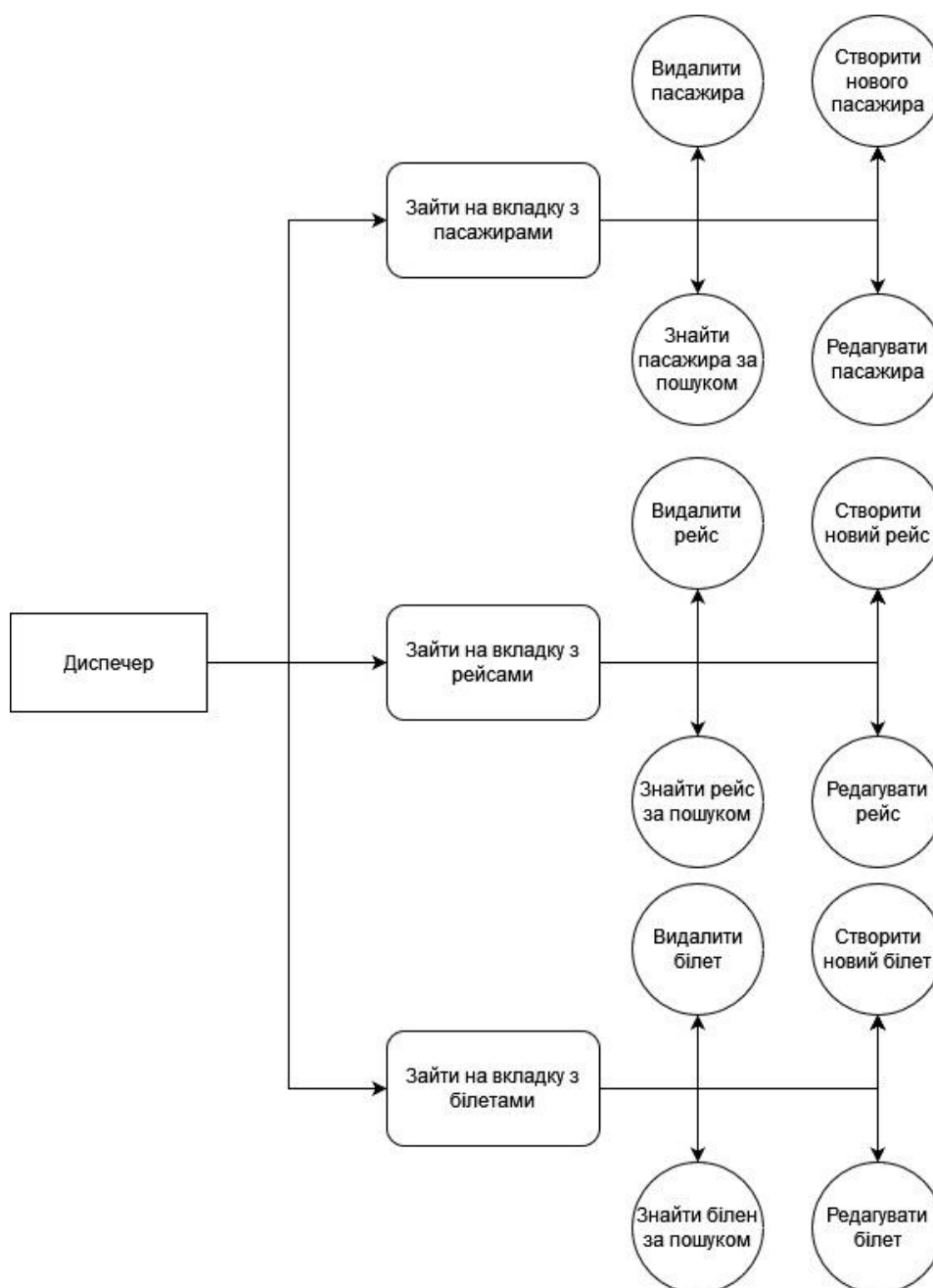


Рисунок 3.2 – UML діаграма використання

3.3 Опис класів програмного комплексу

Опис класів починається з збереження даних в сховищах. Механізм роботи зі сховищем має в собі 1 інтерфейс та 3 класи.

Почнемо з інтерфейса `InnerStorageType` який описує інтерфейс зовнішнього сховища в якому можуть зберігатись дані. Система зроблена таким

чином, що ми можемо перед виконання коду під'єднати якесь інше сховище через клас адаптер, паттерн проєктування адаптер.

Інтерфейс `InnerStorageType` має два методи:

- `getItem` який приймає ключ та видає результат;
- `setItem` який приймає ключ та значення та видає результат.

Головним класом який пов'язує окремі сховища даних та зовнішнє є `Storage`. Цей клас має наступні методи та властивості:

- `key` – приватне поле для ключа;
- `innerStorage` – приватне поле для зовнішнього сховища, яке має тип `InnerStorageType`;

- `getItem()` – метод для отримання даних зі сховища;
- `setItem(value: string)` – приймає строку та встановлює її в сховище.

Клас `Storage` потрібен, оскільки кожне окреме сховище може мати своє окреме зовнішнє сховище. Тобто, у нас можуть білети зберігатись на одному сховищі з одним адаптером, а рейси – на іншому з іншим адаптером.

Розглянемо класи сховищ. Перший клас сховища – `FlightStorage` потрібен для зберігання даних про рейси. Цей клас має наступні властивості та методи:

- `storage` – приватне поле яке вказує яке саме сховище ми використовуємо;
- `setItem(data: Flight[])` – приймає масив рейсів та записує в сховище;
- `getItem(): Flight[]` – повертає масив рейсів зі сховища.

Наступне сховище – `PassengerStorage` потрібен для зберігання даних про пасажирів. Цей клас має наступні властивості та методи:

- `storage` – приватне поле яке вказує яке саме сховище ми використовуємо;
- `setItem(data: Passenger[])` – приймає масив пасажирів та записує в сховище;
- `getItem(): Passenger[]` – повертає масив пасажирів зі сховища.

Останнє сховище `TicketStorage` потрібен для зберігання даних про білети. Цей клас має наступні властивості та методи:

– storage – приватне поле яке вказує яке саме сховище ми використовуємо;

- setItem(data: Ticket[]) – приймає масив білетів та записує в сховище;
- getItem(): Ticket [] – повертає масив білетів зі сховища.

Розглянемо основні базові застосунку.

Клас Flight – клас, який зберігає данні про рейс та має наступні поля:

- id – Унікальний ідентифікатор;
- fromCity – Місто, з якого вилітає літак;
- destination – Місто, в яке прилетить літак;
- startDate – Дата початку польоту;
- endDate – Дата кінця польоту;
- price – Ціна;
- seatsCount – Кількість місць для пасажирів.

Клас Passenger – клас, який відповідає за базові дані пасажирів та має наступні поля:

- id – Унікальний ідентифікатор;
- firstName – Ім'я пасажирів;
- lastName – Прізвище пасажирів;
- phoneNumber – Номер телефону;
- passportNumber – Номер паспорта;
- email - Пошта.

Клас Ticket – клас, який відповідає за базові дані квитка та має наступні поля:

- id – Унікальний ідентифікатор;
- passenger – Ідентифікатор пасажирів;
- flight – Ідентифікатор рейсу;
- position – Позиція в літаку.

Головним інтерфейсом для пов'язування сховищ даних та основної логіки є StorageType. Цей інтерфейс має наступні поля:

- getItem(): T[];
- setItem(data: T[]): void.

Базовий клас – один елемент. Для роботи з масивом існує спеціальний менеджер, який має імплементує інтерфейс BaseManager та має наступні властивості та методи:

- items – Поле для збереження елементів;
- initializeFromStorage(storage: StorageType<any>) – Метод для ініціалізації даних зі сховища;
- add(data: any) – Метод для додавання елементів;
- remove(id: string) – Метод для видалення елементів;
- edit(id: string, newData: T) – Метод для редагування елементів;
- getBySearch(search: string) – Метод для пошуку елементів.

Клас FlightManager – клас для управління рейсами, цей клас має наступні поля та методи:

- items – Поле для збереження елементів рейсів;
- initializeFromStorage(storage: StorageType<Flight>) – Метод для ініціалізації даних зі сховища;
- add(data: Flight) – Метод для додавання елементів;
- remove(id: string) – Метод для видалення елементів;
- edit(id: string, newData: Flight) – Метод для редагування елементів;
- getBySearch(search: string) – Метод для пошуку елементів.

Клас PassengerManager – клас для управління пасажирами, цей клас має наступні поля та методи:

- items – Поле для збереження елементів рейсів;
- initializeFromStorage(storage: StorageType<Passenger>) – Метод для ініціалізації даних зі сховища;
- add(data: Passenger) – Метод для додавання елементів;
- remove(id: string) – Метод для видалення елементів;

- `edit(id: string, newData: Passenger)` – Метод для редагування елементів;
- `getBySearch(search: string)` – Метод для пошуку елементів.

Клас `TicketManager` – клас для управління білетами, цей клас має наступні поля та методи:

- `items` – Поле для збереження елементів рейсів;
- `initializeFromStorage(storage: StorageType<Ticket>)` – Метод для ініціалізації даних зі сховища;
- `add(data: Ticket)` – Метод для додавання елементів;
- `remove(id: string)` – Метод для видалення елементів;
- `edit(id: string, newData: Ticket)` – Метод для редагування елементів;
- `getBySearch(search: string)` – Метод для пошуку елементів.

В проєкті реалізований паттерн проєктування абстрактна фабрика. Основним інтерфейсом цієї фабрики є `GUIFactory`, який має наступні поля та властивості:

- `storage: StorageType` - Приватне поле сховища даних;
- `manager: BaseManager` – Приватне поле менеджера;
- `creatOpenFormButton()` – Метод, який повертає кнопку для додавання нових елементів;
- `getTableForRender()` – Метод, який повертає таблицю для відображення елементів, а також редагування та видалення;
- `getManager()` – Метод для взяття менеджера.

Першою фабрикою є фабрика рейсів – `FlightFactory`, яка має наступні властивості та методи:

- `storage: FlightStorage` - Приватне поле сховища даних;
- `manager: FlightManager` - Приватне поле менеджера;
- `currentTableRender` – Приватне поле HTML елемент таблиці, використовується для ререндеру;

- `createOpenFormButton()` - Метод, який повертає кнопку для додавання нових елементів;
- `getTableForRender()` - Метод, який повертає таблицю для відображення елементів, а також редагування та видалення;
- `getManager()` - Метод для взяття менеджера;
- `rerenderTable()` – Приватний метод, який потрібний для перерендеру таблиці;
- `onDelete(item: string[])` - Приватний метод який викликається при видаленні елемента в таблиці;
- `onEdit(item: string[])` - Приватний метод який викликається при редагуванні елемента в таблиці та автоматично відкриває форму редагування;
- `createTable(onDelete, onEdit)` - Приватний метод який створює таблицю.

Другою фабрикою є фабрика пасажирів – `PassengerFactory`, яка має наступні властивості та методи:

- `storage: PassengerStorage` - Приватне поле сховища даних;
- `manager: PassengerManager` - Приватне поле менеджер;
- `currentTableRender` – Приватне поле HTML елемент таблиці, використовується для ререндеру;
- `createOpenFormButton()` - Метод, який повертає кнопку для додавання нових елементів;
- `getTableForRender()` - Метод, який повертає таблицю для відображення елементів, а також редагування та видалення;
- `getManager()` - Метод для взяття менеджера;
- `rerenderTable()` – Приватний метод, який потрібний для перерендеру таблиці;
- `onDelete(item: string[])` - Приватний метод який викликається при видаленні елемента в таблиці;

- `onEdit(item: string[])` - Приватний метод який викликається при редагуванні елемента в таблиці та автоматично відкриває форму редагування;
- `createTable(onDelete, onEdit)` - Приватний метод який створює таблицю.

Третьою фабрикою є фабрика білетів – `TicketFactory`, яка має наступні властивості та методи:

- `storage: TicketStorage` - Приватне поле сховища даних;
- `manager: TicketManager` - Приватне поле менеджера;
- `currentTableRender` – Приватне поле HTML елемент таблиці, використовується для ререндеру;
- `createOpenFormButton()` - Метод, який повертає кнопку для додавання нових елементів;
- `getTableForRender()` - Метод, який повертає таблицю для відображення елементів, а також редагування та видалення;
- `getManager()` - Метод для взяття менеджера;
- `rerenderTable()` – Приватний метод, який потрібний для перерендеру таблиці;
- `onDelete(item: string[])` - Приватний метод який викликається при видаленні елемента в таблиці;
- `onEdit(item: string[])` - Приватний метод який викликається при редагуванні елемента в таблиці та автоматично відкриває форму редагування;
- `createTable(onDelete, onEdit)` - Приватний метод який створює таблицю.

Для пошуку існує клас `Search`. Цей клас має наступні методи та властивості:

- `onSubmit` – Поле, яке буде викликане коли користувач закінчить вводити текст для пошуку;
- `render()` – Метод для рендеру пошуку на сторінку.

Для створення та редагування елементів використовується форма – клас `Form`. Цей клас має наступні властивості та методи:

- `title` – Назва форми;
- `onClose` – Властивість, яка буде викликана при закритті форми;
- `onSubmit(formData: Record<string, string>)` – Властивість, яка буде викликана при успішному введенні даних користувачем;
- `type` – Тип форми, створення нових елементів або ж оновлення;
- `addInput` – Метод для додавання нових полів в форму;
- `mount()` – Монтування форми на сторінку;
- `umount()` – Видалення форми зі сторінки.

В проєкті використовуються багато однакових кнопок, тому для неї був створений клас `Button`, цей клас має наступні властивості та методи:

- `className` – Назва класу кнопки для стилізації;
- `textContent` – Текст кнопки;
- `callback` – Поле, яке буде викликано при;
- `render()` – Метод для рендера компонента.

Для таблиці також розроблений окремий клас – `Table`. Цей клас має наступні властивості та методи:

- `heads` – Заголовки таблиці;
- `items` – Елементи таблиці;
- `onDelete` – Поле, яке буде викликано при видаленні елемента;
- `onEdit` – Поле, яке буде викликано при редагуванні елемента;
- `getTableElements(item: T[])` – Приватне поле для генерування елементів таблиці для рендера;
- `getTableHeadElements(heads: string[])` – Приватне поле для генерування елементів заголовків таблиці для рендера;
- `render()` – Метод для повернення таблиці для рендера.

Для створення табуляції було розроблено клас `Tabs`, а також для одного елемента таба окремий інтерфейс – `Tab`.

Інтерфейс Tab має наступні поля:

- title – Назва табу;
- callback – Поле яке буде викликано при натисканні на таб;
- active – Чи активний таб.

Клас Tabs має наступні властивості та методи:

- tabs – Приватне поле масивів Tab;
- addTab – Метод створення нових табів;
- setActive – Метод зміни активності таба;
- generateButton(tabInfo: Tab) – Приватний метод генерації кнопки, потрібний для рендера;
- getForRender() – Взяти таби для рендера.

А також головний компонент – App, який має наступні властивості та метод:

- root – HTML елемент в який буде рендеритись контент;
- factory: GUIFactory – Приватне поле. Фабрика яка зараз відображається;
- tabs – Приватне поле. Екземпляр класу Tabs;
- render() – Рендер на сторінку.

4 АНАЛІЗ ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Базовий аналіз ефективності програмного забезпечення

Основні аспекти базового аналізу ефективності програмного забезпечення:

- швидкодія;
- масштабованість;
- надійність;
- зручність використання;

Швидкодія є ключовим аспектом в розробці програмного забезпечення для автоматизації роботи диспетчера в касах аерофлоту. Для досягнення високого рівня швидкодії потрібно зосередити увагу на реалізації ефективного механізму відображення графічних даних, що забезпечить високу швидкість рендерингу та мінімізацію витрат ресурсів.

Масштабованість системи є важливим аспектом, оскільки програма повинна ефективно працювати з великими обсягами інформації. Забезпечення плавного та стабільного відображення графічних елементів є основною ціллю.

Надійність є ще одним ключовим критерієм ефективності програмного забезпечення, особливо для систем, які використовуються в диспетчерській роботі кас аерофлоту. Надійність вказує на стійкість та стабільність програми під час роботи, а також на її здатність уникати виникнення помилок чи збоїв.

Зручність використання це інша важлива характеристика ефективності програмного забезпечення, оскільки вона визначає, наскільки зручно та ергономічно користувач може взаємодіяти з системою. Потрібно враховувати якість та зручність використання графічного інтерфейсу програми, його логічність та інтуїтивність. Також легкість навчання користуванню програмою є немаловажливим [4].

Враховуючи ці аспекти, забезпечується не тільки висока швидкодія програмного забезпечення, але й низький поріг для використання програми. Аналіз ефективності компонентів програмного забезпечення.

4.2 Тестування програмного забезпечення

Тестування програмного забезпечення дозволяє підтвердити функціональність та надійність розробленого програмного продукту.

Були протестовані наступні функції:

- функція відображення даних про рейс, пасажирів та білет в вигляді таблиці (дивитись рисунок 4.1);
- функція зміни рейсу, пасажирів та білету за допомоги форми (дивитись рисунок 4.2);
- функція створення нового рейсу, пасажирів та білету через форму (дивитись рисунок 4.3);
- функція видалення рейсу, пасажирів та білету (дивитись рисунок 4.4);
- функція пошуку по даним (дивитись рисунок 4.5).

Passenger

Ticket

Flight

Create flight

Search

Search

| ID | City from | Destination | Start date | End date | Price | Seats count | |
|----|--------------------|------------------|------------|------------|-------|-------------|---------------------------|
| 0 | East Bernitaview | Millcreek | 2021-09-09 | 2019-05-12 | 1296 | 950 | <div>D</div> <div>E</div> |
| 1 | Lake Nevaport | Yorba Linda | 2005-08-02 | 2051-02-12 | 19272 | 163 | <div>D</div> <div>E</div> |
| 2 | Sunrise Manor | Cutler Bay | 2047-08-21 | 2080-07-29 | 76912 | 747 | <div>D</div> <div>E</div> |
| 3 | Noeliatown | Elliottstad | 2016-11-10 | 2089-01-27 | 81161 | 85 | <div>D</div> <div>E</div> |
| 4 | East Lamar | Vinniebury | 1993-08-11 | 2048-08-17 | 77532 | 393 | <div>D</div> <div>E</div> |
| 5 | San Bruno | Lubowitzland | 1992-03-09 | 2031-01-04 | 32647 | 370 | <div>D</div> <div>E</div> |
| 6 | Tremblayfort | East Kadenfurt | 2057-08-13 | 1995-06-15 | 3299 | 2260 | <div>D</div> <div>E</div> |
| 7 | Jodyburgh | Decatur | 2032-03-28 | 2095-11-14 | 60310 | 884 | <div>D</div> <div>E</div> |
| 8 | Vickieport | South Jayceeberg | 2070-09-16 | 2051-06-01 | 7128 | 2600 | <div>D</div> <div>E</div> |
| 9 | North Graciefort | Leannonberg | 2096-02-01 | 2033-10-22 | 4446 | 9210 | <div>D</div> <div>E</div> |
| 10 | Turlock | Ceciliafurt | 2042-08-22 | 2084-03-03 | 31426 | 934 | <div>D</div> <div>E</div> |
| 11 | East Joanie | Winston-Salem | 2067-07-27 | 2031-08-24 | 4565 | 5890 | <div>D</div> <div>E</div> |
| 12 | South Ivaville | Round Rock | 2028-06-19 | 2077-09-16 | 2528 | 9830 | <div>D</div> <div>E</div> |
| 13 | Lake Sandrine | Rancho Cucamonga | 2070-11-04 | 2030-07-23 | 60581 | 530 | <div>D</div> <div>E</div> |
| 14 | West Cassandrefort | West Reed | 2026-05-02 | 2007-07-04 | 99981 | 583 | <div>D</div> <div>E</div> |
| 15 | New Ismael | Aylatown | 2035-08-05 | 2019-02-14 | 7433 | 9600 | <div>D</div> <div>E</div> |

Рисунок 4.1 – Тестування відображення інформації

Passenger

Ticket

Flight

Create flight

Search

Search

| ID | City from | Destination | Start date | End date | Price | Seats count | |
|----|--------------------|------------------|------------|------------|-------|-------------|---------------------------|
| 0 | East Bernitaview | Millcreek | 2021-09-09 | 2019-05-12 | 1296 | 950 | <div>D</div> <div>E</div> |
| 1 | Lake Nevaport | Yorba Linda | 2005-08-02 | 2051-02-12 | 19272 | 163 | <div>D</div> <div>E</div> |
| 2 | Sunrise Manor | Cutler Bay | 2047-08-21 | 2080-07-29 | 76912 | 747 | |
| 3 | Noeliatown | Elliottstad | 2016-11-10 | 2089-01-27 | 81161 | 85 | |
| 4 | East Lamar | Vinniebury | 1993-08-11 | 2048-08-17 | 77532 | 393 | |
| 5 | San Bruno | Lubowitzland | 1992-03-09 | 2031-01-04 | 32647 | 370 | |
| 6 | Tremblayfort | East Kadenfurt | 2057-08-13 | 1995-06-15 | 3299 | 226 | |
| 7 | Jodyburgh | Decatur | 2032-03-28 | 2095-11-14 | 60310 | 884 | |
| 8 | Vickieport | South Jayceeberg | 2070-09-16 | 2051-06-01 | 7128 | 260 | |
| 9 | North Graciefort | Leannonberg | 2096-02-01 | 2033-10-22 | 4446 | 921 | |
| 10 | Turlock | Ceciliafurt | 2042-08-22 | 2084-03-03 | 31426 | 934 | |
| 11 | East Joanie | Winston-Salem | 2067-07-27 | 2031-08-24 | 4565 | 5890 | <div>D</div> <div>E</div> |
| 12 | South Ivaville | Round Rock | 2028-06-19 | 2077-09-16 | 2528 | 9830 | <div>D</div> <div>E</div> |
| 13 | Lake Sandrine | Rancho Cucamonga | 2070-11-04 | 2030-07-23 | 60581 | 530 | <div>D</div> <div>E</div> |
| 14 | West Cassandrefort | West Reed | 2026-05-02 | 2007-07-04 | 99981 | 583 | <div>D</div> <div>E</div> |
| 15 | New Ismael | Aylatown | 2035-08-05 | 2019-02-14 | 7433 | 9600 | <div>D</div> <div>E</div> |

Update flight

From city:

East Bernitaview

Destination:

Millcreek

Start date:

09.09.2021

End date:

12.05.2019

Price:

1296

Seats count:

950

Update

Рисунок 4.2 – Тестування редагування інформації

Passenger Ticket Flight

Create flight

East Bernitaview Search

| ID | City from | Destination | Start date | End date | Price | Seats count |
|----|-----------|-------------|------------|----------|-------|-------------|
|----|-----------|-------------|------------|----------|-------|-------------|

Рисунок 4.3 – Тестування видалення інформації

Після видалення відправлення, його миттєво було прибрано із бази даних. Продемонстровано за рахунок пошуку за одним із критеріїв видаленого відправлення.

Passenger Ticket Flight

Create flight

East Bernitaview Search

| ID | City from | Destination | Start date | End date | Price | Seats count |
|----|-----------|-------------|------------|----------|-------|-------------|
|----|-----------|-------------|------------|----------|-------|-------------|

Add new flight

From city: 23

Destination: 1436

Start date: 01.02.2024

End date: 10.02.2024

Price: 4565

Seats count: 436

Create

Рисунок 4.4 – Тестування додавання інформації

У тестуванні було додано дані з відповідними полями, що можна побачити на рисунку 4.5.

Passenger

Ticket

Flight

Create flight

4565

Search

| ID | City from | Destination | Start date | End date | Price | Seats count | | |
|----|-------------|---------------|------------|------------|-------|-------------|---|---|
| 11 | East Joanie | Winston-Salem | 2067-07-27 | 2031-08-24 | 4565 | 5890 | D | E |
| 16 | 23 | 1436 | 2024-02-01 | 2024-02-10 | 4565 | 436 | D | E |

Рисунок 4.5 – Тестування пошуку інформації

Після проведення тестування застосунку виявлено відсутність будь-яких помилок чи несправностей в його роботі. Всі функціональні та інтерфейсні елементи працюють вірно. Тестування вказує на високу якість програмного забезпечення, адекватну відповідь на користувацькі взаємодії та відсутність технічних недоліків.

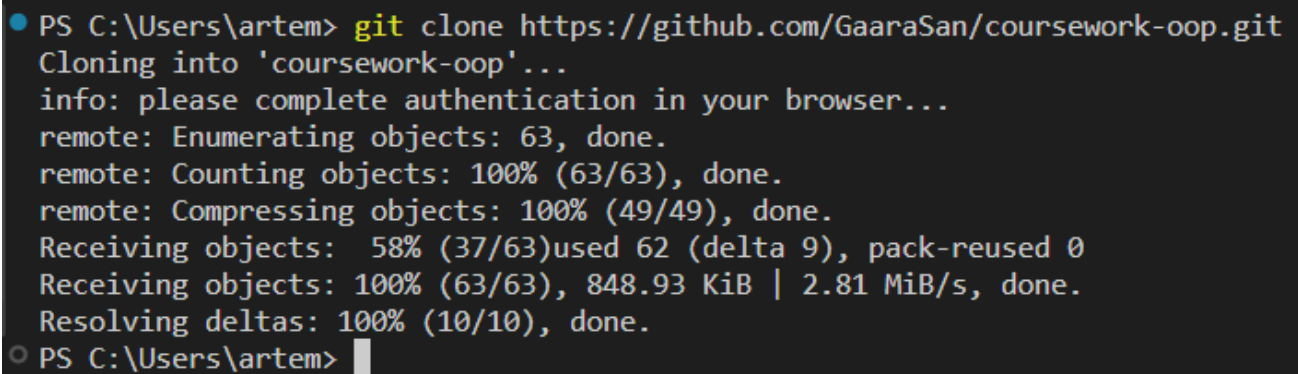
5 РОЗРОБКА ДОКУМЕНТІВ НА СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Інструкція програміста

Ця інструкція відповідає на запитання як розгорнути проєкт локально, як в нього вносити зміни та як тестувати застосунок.

Розглянемо як розгорнути проєкт локально.

Спочатку потрібно клонувати репозиторій застосунку на вашому робочому пристрої. Використовуйте команду `git clone` та вказуйте URL репозиторію. На рисунку 5.1 показаний приклад клонування репозиторію.



```
PS C:\Users\artem> git clone https://github.com/GaaraSan/coursework-oop.git
Cloning into 'coursework-oop'...
info: please complete authentication in your browser...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (49/49), done.
Receiving objects: 58% (37/63)used 62 (delta 9), pack-reused 0
Receiving objects: 100% (63/63), 848.93 KiB | 2.81 MiB/s, done.
Resolving deltas: 100% (10/10), done.
PS C:\Users\artem>
```

Рисунок 5.1 – Клонування репозиторію

Встановлення залежностей. Переконайтеся, що у вас встановлений пакетний менеджер `pnpm`. Використовуйте команду `pnpm install`, щоб встановити всі необхідні залежності. Це забезпечить коректну роботу застосунку. На рисунку 5.2 показаний приклад встановлення залежностей [8].

```
● PS C:\Users\artem\coursework-oop> pnpm install
Lockfile is up to date, resolution step is skipped
Packages: +111
+++++
Progress: resolved 111, reused 111, downloaded 0, added 111, done

devDependencies:
+ eslint 8.56.0
+ prettier 3.2.4
+ typescript 5.3.3
+ vite 5.0.12

Done in 1.2s
○ PS C:\Users\artem\coursework-oop> █
```

Рисунок 5.2 – Встановлення залежностей

Запуск застосунку для розробки. Використовуйте команду `npm run dev`, щоб запустити застосунок у режимі розробки. Це дозволить вам спостерігати за змінами в реальному часі та вносити власні виправлення. На рисунку 5.3 показаний приклад запуску застосунку.

```
○ PS C:\Users\artem\coursework-oop> npm run dev

> dispatcher-coursework-oop@0.0.0 dev
> vite

VITE v5.0.12 ready in 197 ms

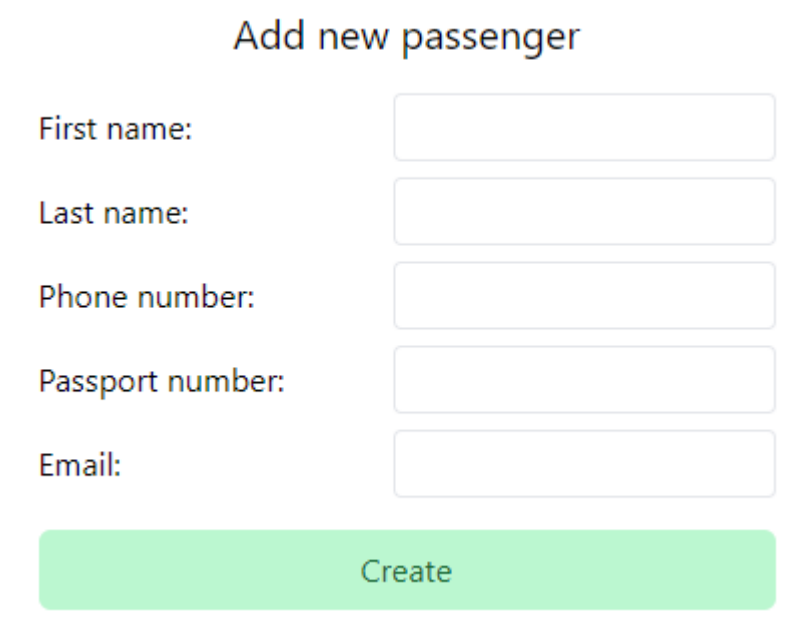
→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
█
```

Рисунок 5.3 – Запуск застосунку

Тепер проєкт розгорнутий локально та готовий до редагування або ж використання.

5.2 Інструкція користувачеві

Заходячи на сайт користувачу буде доступні функції для додання пасажирів, квитків та відправлень. Додання інформації виконується за допомогою спеціальної форми, яка зображена на рисунку 5.4. Після цього потрібно натиснути кнопку «Create» для підтвердження додання інформації у базу даних або можна натиснути поза межами форми для того щоб закрити її.



The image shows a web form titled "Add new passenger". It contains five input fields stacked vertically, each with a label to its left: "First name:", "Last name:", "Phone number:", "Passport number:", and "Email:". Below these fields is a large green button with the text "Create". The entire form is enclosed in a thin grey border.

Рисунок 5.4 – Форма для додання даних

Далі користувач може переключатися між вкладками для відображення таблиць з пасажирями, заброньованими квитками та відправленнями. Також виконувати пошук по таблиці та видаляти або редагувати дані (дивитись рисунок 5.5).

Passenger
Ticket
Flight

Create passenger

Search

Search

| ID | First name | Last name | Phone number | Passport number | Email | | |
|----|------------|-----------|-----------------------|-----------------|----------------------|---|---|
| 0 | Verda | MacGyver | 1-954-379-8257 | 7175509 | verda@gmail.com | D | E |
| 1 | Earnestine | Boehm | 1-568-497-4202 x250 | 8706309 | earnestine@gmail.com | D | E |
| 2 | Tyrell | Koss | 463-689-1237 x23775 | 4307909 | tyrell@gmail.com | D | E |
| 3 | Haylee | Mosciski | (357) 853-4342 x3754 | 4986509 | haylee@gmail.com | D | E |
| 4 | Tyreek | King | 278.972.6995 x15797 | 1370109 | tyreek@gmail.com | D | E |
| 5 | Dean | Barrows | (200) 922-6742 x240 | 1919709 | dean@gmail.com | D | E |
| 6 | Brock | Runte | 998-864-1474 | 853909 | brock@gmail.com | D | E |
| 7 | Pearline | Mante | (965) 882-8129 | 399409 | pearline@gmail.com | D | E |
| 8 | Alana | Jenkins | 741.656.7526 x707 | 2441809 | alana@gmail.com | D | E |
| 9 | Chadd | Fahey | 1-550-220-7432 x0477 | 7869409 | chadd@gmail.com | D | E |
| 10 | Kim | King | 314.884.7259 | 9571309 | kim@gmail.com | D | E |
| 11 | Fidel | Leannon | 836-875-1771 x501 | 380409 | fidel@gmail.com | D | E |
| 12 | Haylee | Dibbert | 441-567-3039 | 74709 | haylee@gmail.com | D | E |
| 13 | Catharine | Dickens | 1-480-801-7570 x86205 | 3432109 | catharine@gmail.com | D | E |
| 14 | Cathrine | Kihn | 367-720-3214 | 5930709 | cathrine@gmail.com | D | E |
| 15 | Loma | Torp | 948-416-1388 x7160 | 9519509 | loma@gmail.com | D | E |
| 16 | Freida | Connelly | 226.537.8980 | 4314609 | freida@gmail.com | D | E |
| 17 | Lucile | Crooks | 972.265.5430 | 9493409 | lucile@gmail.com | D | E |

Рисунок 5.5 – Основний вид

Пошук можливий за усіма полями. Потрібно ввести у текстове поле потрібне значення та натиснути кнопку пошуку. Усі знайдені результати будуть відображені у таблиці нижче, як зображено на рисунку 5.6.

Passenger
Ticket
Flight

Create passenger

Verda

Search

| ID | First name | Last name | Phone number | Passport number | Email | | |
|----|------------|-----------|----------------|-----------------|-----------------|---|---|
| 0 | Verda | MacGyver | 1-954-379-8257 | 7175509 | verda@gmail.com | D | E |

Рисунок 5.6 – Реалізація пошуку даних

Для видалення даних потрібно натиснути відповідну кнопку «D», після чого дані будуть миттєво видалені із бази даних (дивитись рисунок 5.7).

| ID | First name | Last name | Phone number | Passport number | Email |
|----|------------|-----------|--------------|-----------------|-------|
|----|------------|-----------|--------------|-----------------|-------|

Рисунок 5.7 – Демонстрація видалення даних

Редагування даних виконується при натисканні кнопки «E». Диспетчеру потрібно буде змінити потрібні дані у спливаючому вікні. Інформація для редагування буде сама «підтянута» у поля форми, як зображено на рисунку 5.8. Після введення нових даних, потрібно натиснути кнопку «Update» для підтвердження редагування.

| ID | First name | Last name | Phone number | Passport number | Email |
|----|------------|-----------|-----------------------|-----------------|----------------------|
| 1 | Earnestine | Boehm | 1-568-497-4202 x250 | 8706309 | earnestine@gmail.com |
| 2 | Tyrell | Koss | 463-689-1237 x23775 | 4307909 | tyrell@gmail.com |
| 3 | Haylee | Mosciski | (357) 853-4342 x3754 | 4986509 | haylee@gmail.com |
| 4 | Tyreek | King | 278.972.6995 x15797 | 1370109 | tyreek@gmail.com |
| 5 | Dean | Barrows | (200) 922-6742 x240 | 1919709 | dean@gmail.com |
| 6 | Brock | Runte | 998-864-1474 | 853909 | brock@gmail.com |
| 7 | Pearline | Mante | (965) 882-8129 | 399409 | pearline@gmail.com |
| 8 | Alana | Jenkins | 741.656.7526 x707 | 2441809 | alana@gmail.com |
| 9 | Chadd | Fahey | 1-550-220-7432 x0477 | 7869409 | chadd@gmail.com |
| 10 | Kim | King | 314.884.7259 | 9571309 | kim@gmail.com |
| 11 | Fidel | Leannon | 836-875-1771 x501 | 380409 | fidel@gmail.com |
| 12 | Haylee | Dibbert | 441-567-3039 | 74709 | haylee@gmail.com |
| 13 | Catharine | Dickens | 1-480-801-7570 x86205 | 3432109 | catharine@gmail.com |

Рисунок 5.8 – Редагування даних

ВИСНОВКИ

В ході виконання даної курсової роботи було розроблено програмне забезпечення, яке спрямоване на автоматизацію робочих процесів диспетчерів у касах авіакомпаній, використовуючи принципи об'єктно-орієнтованого програмування (ООП). Розроблений продукт відзначається високою ефективністю та зручністю взаємодії, спрощуючи завдання обробки пасажирських даних та оптимізації продажу квитків.

У результаті огляду сучасних технологій та методів у галузі авіаційного обслуговування, встановлено, що висока ступінь конкуренції та зростання обсягів пасажиропотоку потребують вдосконалення системи обробки інформації у касовому відділенні. Розроблена програма має за мету полегшити та прискорити роботу диспетчерів, а також забезпечити точність та швидкість виконання операцій.

У ході проектування програмного забезпечення було проведено докладний аналіз функціональності системи, розроблено діаграми використання та створено інтуїтивний графічний інтерфейс. Здійснено визначення основних функцій програми та структури системи, розроблено діаграму класів, а також використано паттерни проектування абстрактна фабрика, адаптер та singleton.

Аналіз ефективності програмного забезпечення показав його високу швидкодію та масштабованість. Тестування дозволило оцінити ефективність окремих компонентів програми та впевнитися у стабільності та надійності розробленого застосунку.

Окрім того, була розроблена методика взаємодії користувачів з системою, враховуючи особливості як розробників, так і звичайних користувачів. Ця методика сприятиме покращенню робочого процесу та забезпечить максимальний комфорт при використанні програмного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hatch, S.V. Computerized Engine Controls / S.V. Hatch. – Boston: Cengage Learning, 2016. – 688 p.
2. Czichos, H. Measurement, Testing and Sensor Technology. Fundamentals and Application to Materials and Technical Systems / H. Czichos. – Berlin: Springer, 2018. – 213 p.
3. Kaźmierczak, J. Data Processing and Reasoning in Technical Diagnostics / J. Kaźmierczak, W. Cholewa. – Warszawa: Wydawnictwa Naukowo-Techniczne, 1995. – 186 p.
4. Diagnostics as a Reasoning Process: From Logic Structure to Software Design / [M. Cristani, F. Olivieri, C. Tomazzoli, L. Vigano, M. Zorzi] // Journal of Computing and Information Technology. – 2018. – Vol. 27 (1). – P. 43-57.
5. Wieczorek, A.N. Analysis of the Possibility of Integrating a Mining Right-Angle Planetary Gearbox with Technical Diagnostics Systems / A.N. Wieczorek // Scientific Journal of Silesian University of Technology. Series Transport. – 2016. – Vol. 93. – P. 149-163.
6. Tso, B. Classification Methods for Remotely Sensed Data / B. Tso, P.M. Mather. – Boca Raton : CRC Press, 2016. – 352 p.
7. Oppermann, A. Regularization in Deep Learning – L1, L2, and Dropout [Electronic resource]. – Access mode: <https://www.deeplearning-academy.com/p/ai-wiki-regularization>.
8. Classic Regularization Techniques in Neural Networks [Electronic resource]. – Access mode: <https://medium.com/@ODSC/classic-regularization-techniques-in-neural-networks-68bccee03764>.

```
import { FlightFactory } from './other/FlightFactory'
import { GUIFactory } from './other/GUIFactory'
import { PassengerFactory } from './other/PassengerFactory'
import { Tabs } from './other/Tabs'
import { TicketFactory } from './other/TicketFactory'

export class App {
  root: Element

  private factory: GUIFactory<any>
  private tabs: Tabs

  constructor(rootId: string) {
    this.root = document.querySelector(rootId)!

    const tabs = new Tabs()

    const addPassengerTab = () => {
      this.factory = new PassengerFactory()
      this.tabs.setActive('Passenger')
      this.render()
    }

    const addTicketTab = () => {
      this.factory = new TicketFactory()
      this.tabs.setActive('Ticket')
      this.render()
    }
  }
}
```

```

const addFlightTab = () => {
    this.factory = new FlightFactory()
    this.tabs.setActive('Flight')
    this.render()
}

tabs.addTab('Passenger', addPassengerTab)
tabs.addTab('Ticket', addTicketTab)
tabs.addTab('Flight', addFlightTab)

this.tabs = tabs

this.factory = new PassengerFactory()
this.tabs.setActive('Passenger')
this.render()
}

render() {
    this.root.innerHTML = "
    this.root.className = 'm-4'
    this.root.append(this.tabs.getForRender())
    this.root.append(this.factory.createOpenFormButton())
    this.root.append(this.factory.getSearchForRender())
    this.root.append(this.factory.getTableForRender())
}
}

import { App } from './App'

```

```
const app = new App('#app')
```

```
app.render()
```

```
/// <reference types="vite/client" />
```

```
export interface BaseManager<T> {
    items: T[]
    add(data: unknown): void
    remove(id: string): void
    edit(id: string, newData: T): void
    getBySearch(search: string): void
}
```

```
export class Flight {
    id: string
    fromCity: string
    destination: string
    startDate: Date
    endDate: Date
    price: number
    seatsCount: number

    constructor(
        id: string,
        fromCity: string,
```

```

        destination: string,
        startDate: Date,
        endDate: Date,
        price: number,
        seatsCount: number
    ) {
        this.id = id
        this.fromCity = fromCity
        this.destination = destination
        this.startDate = startDate
        this.endDate = endDate
        this.price = price
        this.seatsCount = seatsCount
    }
}

import { StorageType } from '../utils/Storage'
import { BaseManager } from './BaseManager'
import { Flight } from './Flight'

export class FlightManager implements BaseManager<Flight> {
    items: Flight[] = []

    initializeFromStorage(
        storage: StorageType<
            {
                fromCity: string
                destination: string
                startDate: string

```



```

        endDate: string
        price: number
        seatsCount: number
    }[]
>
) {
    const itemsFromStorage = storage.getItem()

    if (!itemsFromStorage) return

    let id = 0

    this.items = itemsFromStorage.map(item => {
        const newItem = new Flight(
            String(id),
            item.fromCity,
            item.destination,
            new Date(item.startDate),
            new Date(item.endDate),
            Number(item.price),
            Number(item.seatsCount)
        )

        id++

        return newItem
    })
}

add({

```

```

        fromCity,
        destination,
        startDate,
        endDate,
        price,
        seatsCount
    }: {
        fromCity: string
        destination: string
        startDate: Date
        endDate: Date
        price: number
        seatsCount: number
    }) {
        const lastIdString = this.items.at(-1)?.id

        const lastId = lastIdString ? Number(lastIdString) + 1 : 0

        this.items.push(
            new Flight(
                String(lastId),
                fromCity,
                destination,
                startDate,
                endDate,
                price,
                seatsCount
            )
        )
    }

```

```

remove(id: string) {
    this.items = this.items.filter(item => item.id !== id)
}

edit(id: string, newElement: Flight) {
    const itemIndex = this.items.findIndex(item => item.id === id)

    if (itemIndex === undefined) return

    this.items[itemIndex] = newElement
}

getBySearch(search: string) {}
}

```

```

export class Passenger {
    id: string
    firstName: string
    lastName: string
    phoneNumber: string
    passportNumber: number
    email: string

    constructor(
        id: string,
        firstName: string,
        lastName: string,
        phoneNumber: string,

```

```

        passportNumber: number,
        email: string
    ) {
        this.id = id
        this.firstName = firstName
        this.lastName = lastName
        this.phoneNumber = phoneNumber
        this.passportNumber = passportNumber
        this.email = email
    }
}

```

```

import { StorageType } from '../utils/Storage'
import { BaseManager } from './BaseManager'
import { Passenger } from './Passenger'

```

```

export class PassengerManager implements BaseManager<Passenger> {
    items: Passenger[] = [
        new Passenger(
            '1',
            'Delphine',
            'Kovacek',
            '385-201-3239',
            53529,
            'delphine@gmail.com'
        ),
        new Passenger(
            '2',
            'Joan',

```

```
'Huels',  
'894-736-9782',  
62907,  
'joan@gmail.com'  
,  
new Passenger(  
    '3',  
    'Graciela',  
    'Botsford',  
    '875-892-5002',  
    59441,  
    'graciela@gmail.com'  
,  
new Passenger(  
    '4',  
    'Marisol',  
    'Hegmann',  
    '890-237-1592',  
    82568,  
    'marisol@gmail.com'  
,  
new Passenger(  
    '5',  
    'Carolina',  
    'Kertzmann',  
    '995-805-2774',  
    72128,  
    'carolina@gmail.com'  
,  
new Passenger(  

```

```

        '6',
        'Melvin',
        'Jaskolski',
        '891-381-4774',
        92615,
        'melvin@gmail.com'
    ),
    new Passenger(
        '7',
        'Corrine',
        'Schulist',
        '589-497-9412',
        87695,
        'corrine@gmail.com'
    ),
    new Passenger(
        '8',
        'Derick',
        'Glover',
        '680-974-4140',
        96553,
        'derick@gmail.com'
    )
]

initializeFromStorage(
    storage: StorageType<
    {
        firstName: string
        lastName: string
    }
)

```

```

        phoneNumber: string
        passportNumber: number
        email: string
    }[]
>
) {
    const itemsFromStorage = storage.getItem()

    if (!itemsFromStorage) return

    let id = 0

    this.items = itemsFromStorage.map(item => {
        const newItem = new Passenger(
            String(id),
            item.firstName,
            item.lastName,
            item.phoneNumber,
            item.passportNumber,
            item.email
        )

        id++

        return newItem
    })
}

add({
    firstName,

```

```

        lastName,
        phoneNumber,
        passportNumber,
        email
    }: {
        firstName: string
        lastName: string
        phoneNumber: string
        passportNumber: number
        email: string
    }) {
        const lastIdString = this.items.at(-1)?.id

        const lastId = lastIdString ? Number(lastIdString) + 1 : 0

        this.items.push(
            new Passenger(
                String(lastId),
                firstName,
                lastName,
                phoneNumber,
                passportNumber,
                email
            )
        )
    }

    remove(id: string) {
        this.items = this.items.filter(item => item.id !== id)
    }

```



```

edit(id: string, newPassenger: Passenger) {
    const itemIndex = this.items.findIndex(item => item.id === id)

    console.log(itemIndex, id, this.items)

    if (itemIndex === undefined) return

    this.items[itemIndex] = newPassenger
}

getBySearch(search: string) {}
}

export class Ticket {
    id: string
    passenger: string
    flight: string
    position: string

    constructor(id: string, passenger: string, flight: string, position: string) {
        this.id = id
        this.passenger = passenger
        this.flight = flight
        this.position = position
    }
}

```

```

import { StorageType } from '../utils/Storage'
import { BaseManager } from './BaseManager'
import { Ticket } from './Ticket'

export class TicketManager implements BaseManager<Ticket> {
  items: Ticket[] = []

  initializeFromStorage(
    storage: StorageType<
      {
        passenger: string
        flight: string
        position: string
      }[]
    >
  ) {
    const itemsFromStorage = storage.getItem()

    if (!itemsFromStorage) return

    let id = 0

    this.items = itemsFromStorage.map(item => {
      const newItem = new Ticket(
        String(id),
        item.passenger,
        item.flight,
        item.position
      )
      id++
    })
  }
}

```

```

        return newItem
    })
}

add({
    passenger,
    flight,
    position
}): {
    passenger: string
    flight: string
    position: string
}) {
    const lastIdString = this.items.at(-1)?.id

    const lastId = lastIdString ? Number(lastIdString) + 1 : 0

    this.items.push(new Ticket(String(lastId), passenger, flight, position))
}

remove(id: string) {
    this.items = this.items.filter(item => item.id !== id)
}

edit(id: string, newElement: Ticket) {
    const itemIndex = this.items.findIndex(item => item.id === id)

    if (itemIndex === undefined) return

```

```

        this.items[itemIndex] = newElement
    }

    getBySearch(search: string) {}
}

export class Button {
    className: string
    textContent: string
    callback: (e: MouseEvent) => void

    constructor(
        textContent: string,
        callback: (e: MouseEvent) => void,
        className: string | null = null
    ) {
        this.className =
            className ??
            `px-4 py-2 rounded-lg text-slate-800 bg-slate-200 w-auto
            hover:bg-slate-300 transition`

        this.textContent = textContent
        this.callback = callback
    }

    render() {
        const button = document.createElement('button')
        button.className = this.className
        button.textContent = this.textContent
    }
}

```

```

        button.addEventListener('click', this.callback)

        return button
    }
}

import { Flight } from '../classes/Flight'
import { FlightManager } from '../classes/FlightManager'
import { FlightStorage } from '../utils/FlightStorage'
import { Button } from './Button'
import { Form } from './Form'
import { GUIFactory } from './GUIFactory'
import { Search } from './Search'
import { Table } from './Table'

export class FlightFactory implements GUIFactory<unknown> {
    private storage
    private manager
    private search
    private currentTableRender: HTMLTableElement | null = null

    constructor() {
        this.manager = new FlightManager()
        this.storage = new FlightStorage()
        this.search = new Search(this.onSearchChange.bind(this))

        this.manager.initializeFromStorage(this.storage as any)
    }
}

```

```

private onSearchChange(text: string) {
  const itemsForRender = this.manager.items.filter(item => {
    const BIG_BIG_STRING = Object.values(item)
      .map(element => {
        if (element instanceof Date) {
          return element.toISOString().split('T')[0]
        }

        return String(element)
      })
      .join("")
      .toLowerCase()

    return BIG_BIG_STRING.includes(text.toLowerCase())
  })

  this.renderTable(itemsForRender)
}

getSearchForRender() {
  return this.search.render()
}

createOpenFormButton() {
  const callback = () => {
    const form = new Form(
      'Add new flight',
      () => {
        form.umount()
      },

```

```

    formData => {
      this.manager.add({
        fromCity: formData.fromCity,
        destination: formData.destination,
        startDate: new Date(formData.startDate),
        endDate: new Date(formData.endDate),
        price: Number(formData.price),
        seatsCount: Number(formData.seatsCount)
      })

      this.renderTable()
      this.storage.setItem(this.manager.items)

      form.unmount()
    }
  )

  form.addInput('text', 'From city', 'fromCity')
  form.addInput('text', 'Destination', 'destination')
  form.addInput('date', 'Start date', 'startDate')
  form.addInput('date', 'End date', 'endDate')
  form.addInput('number', 'Price', 'price')
  form.addInput('number', 'Seats count', 'seatsCount')

  form.mount()
}

return new Button(
  'Create flight',
  callback,

```

```

        `mt-2 px-4 py-2 rounded-lg text-slate-800 bg-slate-200 w-auto
        hover:bg-slate-300 transition`

```

```

        ).render()
    }

```

```

private rerenderTable(items?: Flight[]) {
    const itemsForRender = items ?? this.manager.items

    const newTable = this.createTable(
        itemsForRender,
        this.onDelete.bind(this),
        this.onEdit.bind(this)
    )

    this.currentTableRender?.replaceWith(newTable)
    this.currentTableRender = newTable
}

```

```

private onDelete = (item: (string | number)[]) => {
    console.log('onDelete')
    const id = item[0]

    this.manager.remove(id as string)
    this.rerenderTable()
    this.storage.setItem(this.manager.items)
}

```

```

private onEdit = (item: (string | number | Date)[]) => {
    const [id, fromCity, destination, startDate, endDate, price, seatsCount]

```

=

item

```

const form = new Form(
  'Update flight',
  () => {
    form.umount()
  },
  formData => {
    this.manager.edit(
      String(id),
      new Flight(
        String(id),
        formData.fromCity,
        formData.destination,
        new Date(formData.startDate),
        new Date(formData.endDate),
        Number(formData.price),
        Number(formData.seatsCount)
      )
    )

    this.rerenderTable()
    this.storage.setItem(this.manager.items)

    form.umount()
  },
  'update'
)

console.log(startDate, new Date(startDate))

```

```

form.addInput('string', 'From city:', 'fromCity', String(fromCity))
form.addInput('string', 'Destination:', 'destination', String(destination))
form.addInput(
    'date',
    'Start date:',
    'startDate',
    new Date(startDate).toISOString().split('T')[0]
)
form.addInput(
    'date',
    'End date:',
    'endDate',
    new Date(endDate).toISOString().split('T')[0]
)
form.addInput('number', 'Price:', 'price', String(price))
form.addInput('number', 'Seats count:', 'seatsCount', String(seatsCount))

form.mount()
}

getTableForRender() {
    this.currentTableRender = this.createTable(
        this.manager.items,
        this.onDelete.bind(this),
        this.onEdit.bind(this)
    )

    return this.currentTableRender
}

```

```

private createTable(
    items: Flight[],
    onDelete: (item: (string | number)[]) => void,
    onEdit: (item: (string | number)[]) => void
) {
    const HEADS = [
        'ID',
        'City from',
        'Destination',
        'Start date',
        'End date',
        'Price',
        'Seats count'
    ]

    const ITEMS = items.map(item => {
        return [
            item.id,
            item.fromCity,
            item.destination,
            item.startDate.toISOString().split('T')[0],
            item.endDate.toISOString().split('T')[0],
            item.price,
            item.seatsCount
        ]
    })

    const table = new Table(HEADS, ITEMS, onDelete, onEdit)

```

```

        return table.render()
    }

    getManager() {
        return this.manager
    }
}

export class Form {
    title: string
    onClose: () => void
    onSubmit: (formData: Record<string, string>) => void
    type: 'create' | 'update' = 'create'
    initialData: Record<string, string>

    private inputs: HTMLLabelElement[] = []
    private mountedForm: HTMLDivElement | null = null
    private root = document.documentElement

    constructor(
        title: string,
        onClose: () => void,
        onSubmit: (formData: Record<string, string>) => void,
        type: 'create' | 'update' = 'create',
        initialData: Record<string, string> = {}
    ) {
        this.title = title
        this.onClose = onClose
        this.onSubmit = onSubmit
    }
}

```

```

    this.type = type
    this.initialData = initialData
  }

```

```

addInput(type: string, name: string, innerValue: string, value: string = "") {
  const labelElement = document.createElement('label')
  labelElement.className = 'flex justify-between items-center gap-x-2'

  const labelTextNode = document.createTextNode(name)
  labelElement.appendChild(labelTextNode)

  const inputElement = document.createElement('input')
  inputElement.className = 'p-1 border rounded'

  inputElement.type = type
  inputElement.name = innerValue
  inputElement.value = value

  labelElement.appendChild(inputElement)
  this.inputs.push(labelElement)
}

```

```

mount() {
  const wrapper = document.createElement('div')

  const content = document.createElement('div')
  content.className =
    'm-0 absolute top-1/2 left-1/2 -translate-x-1/2 -translate-y-1/2 w-
[400px]'

```

```

const backdrop = document.createElement('div')
backdrop.className =
    'absolute top-0 left-0 w-full h-full bg-slate-500 opacity-40'

backdrop.onclick = () => this.onClose()

wrapper.append(backdrop)

const formElement = document.createElement('form')
formElement.className =
    'flex flex-col gap-y-4 p-4 rounded-md shadow-md bg-white'

const title = document.createElement('h2')
title.className = 'text-xl text-center'
title.textContent = this.title

formElement.append(title)

const inputGroupElement = document.createElement('div')
inputGroupElement.className = 'flex flex-col gap-y-2'

inputGroupElement.append(...this.inputs)
formElement.appendChild(inputGroupElement)

formElement.onsubmit = e => {
    e.preventDefault()

    const data = Object.fromEntries(new
FormData(formElement).entries())

```

```

        this.onSubmit(data as Record<string, string>)
    }

    const buttonElement = document.createElement('button')
    buttonElement.className =
        'rounded-md bg-green-200 text-green-800 py-2 hover:bg-green-
300'

    buttonElement.appendChild(
        document.createTextNode(this.type === 'update' ? 'Update' :
'Create')
    )
    formElement.appendChild(buttonElement)
    content.appendChild(formElement)

    wrapper.append(content)

    this.mountedForm = wrapper
    this.root.append(wrapper)

    return wrapper
}

    unmount() {
        this.mountedForm?.remove()
    }
}

```

```
import { BaseManager } from '../classes/BaseManager'
```

```
export interface GUIFactory<T> {
    createOpenFormButton(): string | Node
    getSearchForRender(): HTMLFormElement
    getTableForRender(): HTMLTableElement
    getManager(): BaseManager<T>
}
```

```
import { Passenger } from '../classes/Passenger'
import { PassengerManager } from '../classes/PassengerManager'
import { PassengerStorage } from '../utils/PassengerStorage'
import { Button } from './Button'
import { Form } from './Form'
import { GUIFactory } from './GUIFactory'
import { Search } from './Search'
import { Table } from './Table'
```

```
export class PassengerFactory implements GUIFactory<Passenger> {
    private manager
    private storage
    private search
    private currentTableRender: HTMLTableElement | null = null

    constructor() {
        this.manager = new PassengerManager()
        this.storage = new PassengerStorage()
        this.search = new Search(this.onSearchChange.bind(this))

        this.manager.initializeFromStorage(this.storage as any)
    }
}
```



```
}
```

```
private onSearchChange(text: string) {
    const itemsForRender = this.manager.items.filter(item => {
        const BIG_BIG_STRING = Object.values(item)
            .map(element => {
                if (element instanceof Date) {
                    return element.toISOString().split('T')[0]
                }

                return String(element)
            })
            .join("")
            .toLowerCase()

        return BIG_BIG_STRING.includes(text.toLowerCase())
    })

    this.renderTable(itemsForRender)
}
```

```
getSearchForRender() {
    return this.search.render()
}
```

```
createOpenFormButton() {
    const callback = () => {
        const form = new Form(
            'Add new passenger',
            () => {
```

```

        form.umount()
    },
    formData => {
        this.manager.add({
            firstName: formData.firstName,
            lastName: formData.lastName,
            phoneNumber: formData.phoneNumber,
            passportNumber:
Number(formData.passportNumber),
            email: formData.email
        })

        this.renderTable()
        this.storage.setItem(this.manager.items)

        form.umount()
    }
)

form.addInput('text', 'First name:', 'firstName')
form.addInput('text', 'Last name:', 'lastName')
form.addInput('tel', 'Phone number:', 'phoneNumber')
form.addInput('number', 'Passport number:', 'passportNumber')
form.addInput('email', 'Email:', 'email')

form.mount()
}

return new Button(
    'Create passenger',

```

```

        callback,
        `mt-2 px-4 py-2 rounded-lg text-slate-800 bg-slate-200 w-auto
        hover:bg-slate-300 transition`
    ).render()
}

```

```

private rerenderTable(items?: Passenger[]) {
    const itemsForRender = items ?? this.manager.items

    const newTable = this.createTable(
        itemsForRender,
        this.onDelete.bind(this),
        this.onEdit.bind(this)
    )

    this.currentTableRender?.replaceWith(newTable)
    this.currentTableRender = newTable
}

```

```

private onDelete = (item: (string | number)[]) => {
    console.log('onDelete')
    const id = item[0]
    this.manager.remove(id as string)

    this.rerenderTable()
    this.storage.setItem(this.manager.items)
}

```

```

private onEdit = (item: (string | number)[]) => {

```

```
const [id, firstName, lastName, phoneNumber, passportNumber, email]
= item
```

```
const form = new Form(
  'Update passenger',
  () => {
    form.umount()
  },
  formData => {
    this.manager.edit(
      String(id),
      new Passenger(
        String(id),
        formData.firstName,
        formData.lastName,
        formData.phoneNumber,
        Number(formData.passportNumber),
        formData.email
      )
    )

    this.rerenderTable()
    this.storage.setItem(this.manager.items)

    form.umount()
  },
  'update'
)
```

```
form.addInput('text', 'First name:', 'firstName', String(firstName))
```

```

        form.addInput('text', 'Last name:', 'lastName', String(lastName))
        form.addInput('tel', 'Phone number:', 'phoneNumber',
String(phoneNumber))
        form.addInput(
            'number',
            'Passport number:',
            'passportNumber',
            String(passportNumber)
        )
        form.addInput('email', 'Email:', 'email', String(email))

        form.mount()
    }

    getTableForRender() {
        this.currentTableRender = this.createTable(
            this.manager.items,
            this.onDelete.bind(this),
            this.onEdit.bind(this)
        )

        return this.currentTableRender
    }

    private createTable(
        items: Passenger[],
        onDelete: (item: (string | number)[]) => void,
        onEdit: (item: (string | number)[]) => void
    ) {
        const HEADS = [

```

```

        'ID',
        'First name',
        'Last name',
        'Phone number',
        'Passport number',
        'Email'
    ]

    const ITEMS = items.map(item => {
        return [
            item.id,
            item.firstName,
            item.lastName,
            item.phoneNumber,
            item.passportNumber,
            item.email
        ]
    })

    const table = new Table(HEADS, ITEMS, onDelete, onEdit)

    return table.render()
}

getManager() {
    return this.manager
}
}

```

```

import { Button } from './Button'

export class Search {
  onSubmit: (text: string) => void

  constructor(onSubmit: (text: string) => void) {
    this.onSubmit = onSubmit
  }

  render() {
    const form = document.createElement('form')
    form.className = 'flex gap-x-2'

    const input = document.createElement('input')
    input.type = 'text'
    input.placeholder = 'Search'

    const button = new Button('Search', () => {})

    form.append(input)
    form.append(button.render())

    form.onsubmit = e => {
      e.preventDefault()

      this.onSubmit(input.value ?? "")
    }

    return form
  }
}

```

```
}
```

```
import { Button } from './Button'
```

```
export class Table<T> {
```

```
  heads: string[]
```

```
  items: T[][]
```

```
  onDelete: (item: T[]) => void
```

```
  onEdit: (item: T[]) => void
```

```
  constructor(
```

```
    heads: string[],
```

```
    items: T[][],
```

```
    onDelete: (item: T[]) => void,
```

```
    onEdit: (item: T[]) => void
```

```
  ) {
```

```
    this.heads = heads
```

```
    this.items = items
```

```
    this.onDelete = onDelete
```

```
    this.onEdit = onEdit
```

```
  }
```

```
  private getTableElements(item: T[]) {
```

```
    return item.map(elementText => {
```

```
      const td = document.createElement('td')
```

```
      td.className = 'border px-4 py-2'
```

```
      td.textContent = String(elementText)
```



```

        return td
    })
}

private getTableHeadElements = (heads: string[]) => {
    return heads.map(head => {
        const th = document.createElement('th')

        th.className = 'bg-slate-100 font-medium border px-4 py-2'
        th.textContent = String(head)

        return th
    })
}

render() {
    const itemsForRender = this.items.map(item => {
        const tableElements = this.getTableElements(item)

        const deleteButton = new Button(
            'D',
            () => {
                this.onDelete(item)
            },
            'ml-1 w-[30px] h-[30px] bg-red-200 rounded-md border
border-red-400 hover:bg-red-300 hover:bg-red-300 transition '
        )

        const editButton = new Button(
            'E',

```

```

        () => {
            this.onEdit(item)
        },
        'w-[30px] h-[30px] bg-sky-200 rounded-md border border-
sky-400 hover:bg-sky-300 hover:bg-red-300 transition '
    )

```

```
const tr = document.createElement('tr')
```

```
tr.append(...tableElements)
```

```
const td = document.createElement('td')
```

```
td.className = 'px-1'
```

```
const div = document.createElement('div')
```

```
div.className = 'flex gap-x-1 items-center justify-center'
```

```
div.append(deleteButton.render())
```

```
div.append(editButton.render())
```

```
td.append(div)
```

```
tr.append(td)
```

```
return tr
```

```
})
```

```
const table = document.createElement('table')
```

```
table.className = 'mt-2'
```

```
const headsForRender = this.getTableHeadElements(this.heads)
```

```

    const thead = document.createElement('thead')
    thead.append(...headsForRender)

    const tbody = document.createElement('tbody')
    tbody.append(...itemsForRender)

    table.append(thead, tbody)

    return table
  }
}

```

```
import { Button } from './Button'
```

```

interface Tab {
  title: string
  callback: () => void
  active: boolean
}

```

```

export class Tabs {
  private tabs: Tab[] = []

  addTab(title: string, callback: () => void) {
    this.tabs.push({
      title,
      callback,
      active: false
    })
  }
}

```

```
    })
  }

```

```
setActive(title: string) {
  this.tabs = this.tabs.map(tab => {
    if (tab.title === title) {
      return {
        ...tab,
        active: true
      }
    }
  })

  return {
    ...tab,
    active: false
  }
}
})
}

```

```
private generateButton(tabInfo: Tab) {
  return new Button(
    tabInfo.title,
    tabInfo.callback,
    `px-4 py-2 rounded-lg text-slate-800 bg-slate-200 w-auto
    hover:bg-slate-300 transition ${tabInfo.active ? 'bg-red-200' : ''}`
  ).render()
}

```

```
getForRender() {
  const div = document.createElement('div')

```

```

        div.className = 'flex gap-x-2'

        const buttons = this.tabs.map(tab => this.generateButton(tab))

        div.append(...buttons)

        return div
    }
}

```

```

import { Ticket } from '../classes/Ticket'
import { TicketManager } from '../classes/TicketManager'
import { TicketStorage } from '../utils/TicketStorage'
import { Button } from './Button'
import { Form } from './Form'
import { GUIFactory } from './GUIFactory'
import { Search } from './Search'
import { Table } from './Table'

export class TicketFactory implements GUIFactory<unknown> {
    private manager
    private storage
    private search
    private currentTableRender: HTMLTableElement | null = null

    constructor() {
        this.manager = new TicketManager()
        this.storage = new TicketStorage()
        this.search = new Search(this.onSearchChange.bind(this))
    }
}

```

```

        this.manager.initializeFromStorage(this.storage as any)
    }

    private onSearchChange(text: string) {
        const itemsForRender = this.manager.items.filter(item => {
            const BIG_BIG_STRING = Object.values(item)
                .map(element => {
                    if (element instanceof Date) {
                        return element.toISOString().split('T')[0]
                    }

                    return String(element)
                })
                .join("")
                .toLowerCase()

            return BIG_BIG_STRING.includes(text.toLowerCase())
        })

        this.rerenderTable(itemsForRender)
    }

    getSearchForRender() {
        return this.search.render()
    }

    createOpenFormButton() {
        const callback = () => {
            const form = new Form(

```

```

      'Add new ticket',
      () => {
        form.umount()
      },
      formData => {
        this.manager.add({
          passenger: formData.passenger,
          flight: formData.flight,
          position: formData.position
        })

        this.renderTable()
        this.storage.setItem(this.manager.items)

        form.umount()
      }
    )

    form.addInput('text', 'Passenger:', 'passenger')
    form.addInput('text', 'Flight:', 'flight')
    form.addInput('tel', 'Position:', 'position')

    form.mount()
  }

  return new Button(
    'Create ticket',
    callback,
    `mt-2 px-4 py-2 rounded-lg text-slate-800 bg-slate-200 w-auto
    hover:bg-slate-300 transition`
  )
}

```

```

        ).render()
    }

    private rerenderTable(items?: Ticket[]) {
        const itemsForRender = items ?? this.manager.items

        const newTable = this.createTable(
            itemsForRender,
            this.onDelete.bind(this),
            this.onEdit.bind(this)
        )

        this.currentTableRender?.replaceWith(newTable)
        this.currentTableRender = newTable
    }

    private onDelete = (item: (string | number)[]) => {
        console.log('onDelete')
        const id = item[0]

        this.manager.remove(id as string)
        this.storage.setItem(this.manager.items)

        this.rerenderTable()
    }

    private onEdit = (item: (string | number)[]) => {
        const [id, passenger, flight, position] = item

        const form = new Form(

```



```

    'Update ticket',
    () => {
        form.umount()
    },
    formData => {
        this.manager.edit(
            String(id),
            new Ticket(
                String(id),
                formData.passenger,
                formData.flight,
                formData.position
            )
        )

        this.rerenderTable()
        this.storage.setItem(this.manager.items)

        form.umount()
    },
    'update'
)

form.addInput('text', 'Passenger', 'passenger', String(passenger))
form.addInput('text', 'Flight', 'flight', String(flight))
form.addInput('tel', 'Position', 'position', String(position))

form.mount()
}

```

```

getTableForRender() {
    this.currentTableRender = this.createTable(
        this.manager.items,
        this.onDelete.bind(this),
        this.onEdit.bind(this)
    )

    return this.currentTableRender
}

private createTable(
    items: Ticket[],
    onDelete: (item: (string | number)[]) => void,
    onEdit: (item: (string | number)[]) => void
) {
    const HEADS = ['ID', 'Passenger', 'Flight', 'Position']

    const ITEMS = items.map(item => {
        return [item.id, item.passenger, item.flight, item.position]
    })

    const table = new Table(HEADS, ITEMS, onDelete, onEdit)

    return table.render()
}

getManager() {
    return this.manager
}
}

```

```

import { Flight } from '../classes/Flight'
import { Storage } from './Storage'

export class FlightStorage {
    storage: Storage

    constructor() {
        this.storage = new Storage('flight')
    }

    setItem(data: Flight[]) {
        this.storage.setItem(JSON.stringify(data))
    }

    getItem() {
        const data = this.storage.getItem()

        return data ? (JSON.parse(data) as Flight[]) : []
    }
}

```

```

import { Passenger } from '../classes/Passenger'
import { Storage } from './Storage'

```

```

export class PassengerStorage {
    storage: Storage

```

```

constructor() {
    this.storage = new Storage('passenger')
}

setItem(data: Passenger[]) {
    this.storage.setItem(JSON.stringify(data))
}

getItem() {
    const data = this.storage.getItem()

    return data ? (JSON.parse(data) as Passenger[]) : []
}
}

```

```

export interface StorageType<T> {
    getItem: () => T
    setItem: (data: T[]) => void
}

```

```

export interface InnerStorageType {
    getItem: (key: string) => string
    setItem: (key: string, value: string) => void
}

```

```

export class Storage {
    key: string
    innerStorage: InnerStorageType
}

```

```

    constructor(
        key: string,
        innerStorage: InnerStorageType = localStorage as unknown as
InnerStorageType
    ) {
        this.key = key
        this.innerStorage = innerStorage
    }

    getItem() {
        return this.innerStorage.getItem(this.key)
    }

    setItem(value: string) {
        this.innerStorage.setItem(this.key, value)
    }
}

```

```
import { Storage } from './Storage'
```

```

export class TicketStorage {
    storage: Storage

    constructor() {
        this.storage = new Storage('ticket')
    }

    setItem(
        data: {

```

```

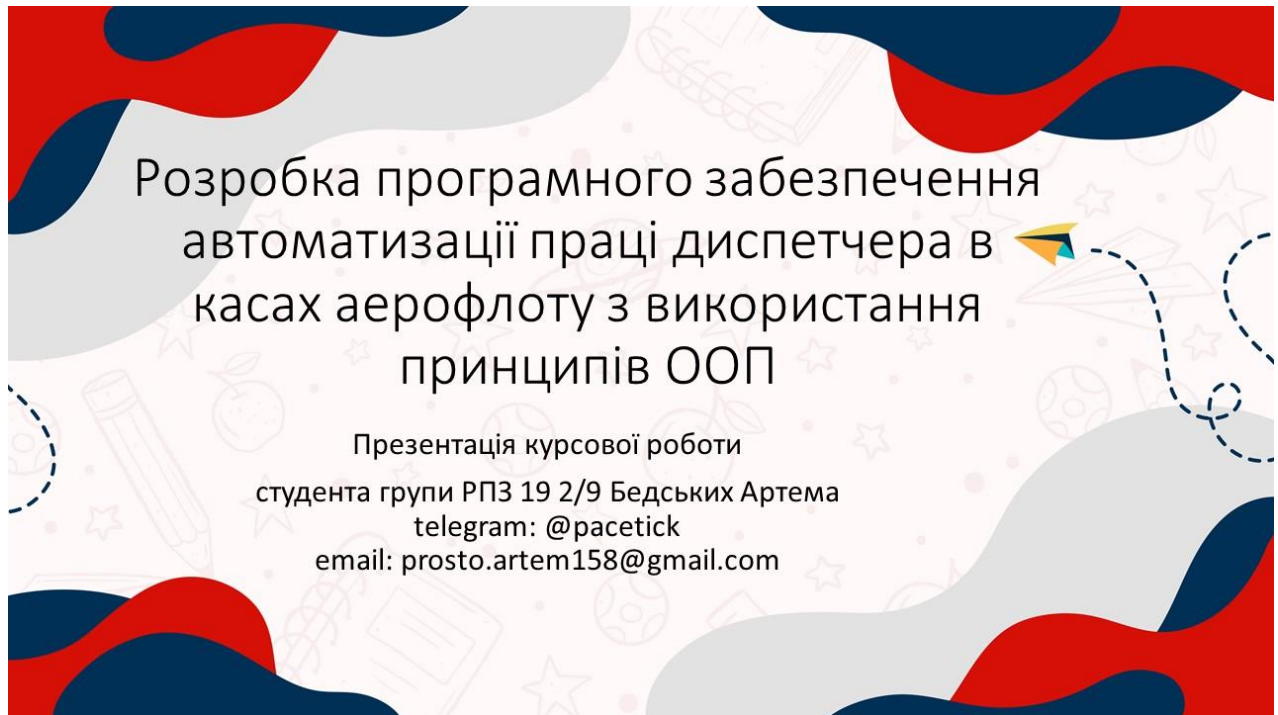
        passenger: string
        flight: string
        position: string
    }[]
) {
    this.storage.setItem(JSON.stringify(data))
}

getItem() {
    const data = this.storage.getItem()

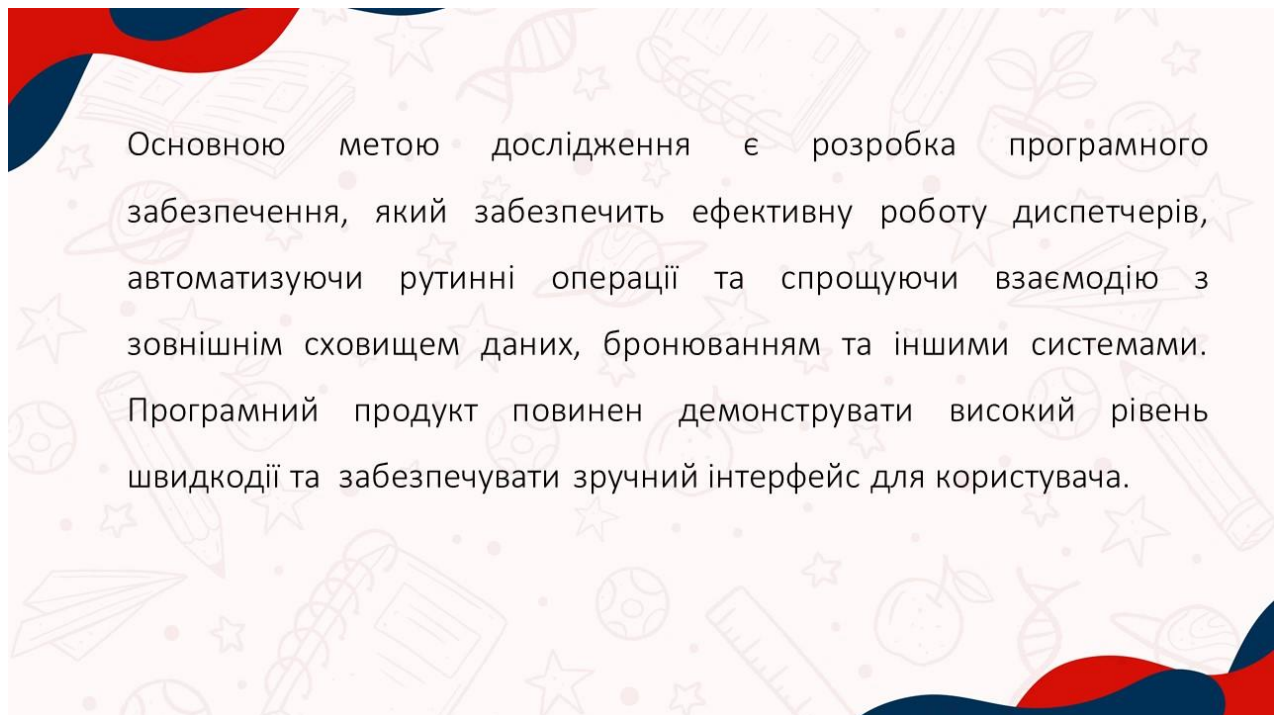
    return data
        ? (JSON.parse(data) as {
            passenger: string
            flight: string
            position: string
        })[]
        : []
}
}
```

Слайди презентації:

Слайд 1



Слайд 2



Слайд 3

Основні функції програми:

- Функція відображення даних про рейс, пасажера та білет в вигляді таблиці
- Функція зміни рейсу, пасажера та білету через форму
- Функція створення нового рейсу, пасажера та білету через форму
- Функція пошуку по даним
- Функція видалення даних

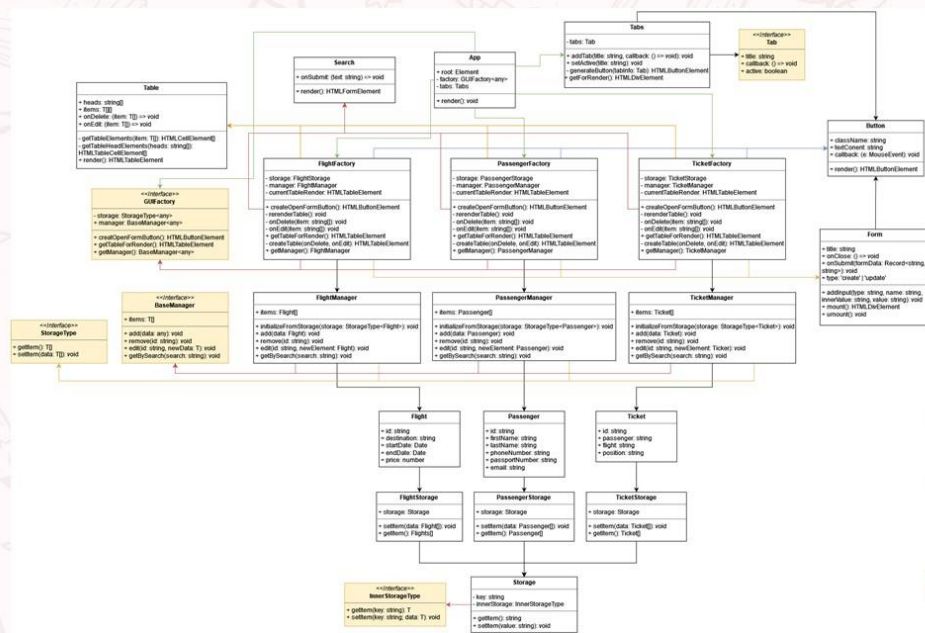
Слайд 4

Що було використано при розробці?



Слайд 5

Діаграма класів



Слайд 6

Перейдемо до демонстрації програми



Слайд 7

Висновки

В ході виконання даної курсової роботи було розроблено програмне забезпечення, яке спрямоване на автоматизацію робочих процесів диспетчерів у касах авіакомпаній, використовуючи принципи об'єктно-орієнтованого програмування (ООП). Розроблений продукт відзначається високою ефективністю та зручністю взаємодії, спрощуючи завдання обробки пасажирських даних та оптимізації продажу квитків.

Слайд 8

Дякую за увагу

