**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Природничо-гуманітарний фаховий коледж**

**ДВНЗ «Ужгородський національний університет»**

**Реєстраційний номер**

**Дата реєстрації**

Вінтоняк Василь Васильович

студент ІІІ курсу

групи КН-32

денної форми навчання

ІНП №253116

**Розробка консольного додатку   
«Система управління співробітниками компанії таксі»:**

**Навчальна практика з дисципліни**

**«**Об’єктно-орієнтоване програмування»

|  |  |
| --- | --- |
| **Допущено до захисту** | **Науковий керівник:** |
| Дата «  »   2024 р.  Підпис | **Викл. Владимир У.М.** |

**Робота захищена «**  **»**    **2024 р.**

**з оцінкою «**  **/**  **/**  **»**

**Підписи членів комісії**

Ужгород 2024

**ЗМІСТ**

ВСТУП 3

[1 ТЕОРЕТИЧНА ЧАСТИНА 4](#_Toc160890484)

[1.1 Загальні відомості про програмний продукт 4](#_Toc160890485)

[1.1.1 «Система управління співробітниками компанії таксі» 4](#_Toc160890486)

[1.1.2 ООП та тип данних JSON 5](#_Toc160890487)

[1.1.3 Google Code Style та Java SonarLint 5](#_Toc160890488)

[1.2 Огляд аналогів 6](#_Toc160890489)

[1.3 Опис предметної області та постановка задачі 7](#_Toc160890490)

[1.4 Обґрунтування вибору методу розв’язку задачі 8](#_Toc160890491)

[1.5 Огляд засобів розробки 10](#_Toc160890492)

[2 ПРАКТИЧНА ЧАСТИНА 12](#_Toc160890493)

[2.1 Інформаційна структура даних 12](#_Toc160890494)

[2.2 Функціональні частини програми 16](#_Toc160890495)

[Основні компоненти 16](#_Toc160890496)

[Використані інструменти 16](#_Toc160890497)

[Яку конвенцію іменування використали? 17](#_Toc160890498)

[Опис структури даних та використаних змінних 19](#_Toc160890499)

[Структура проекту 20](#_Toc160890500)

[2.3 Тестування проєкту 25](#_Toc160890501)

[2.4 Інтерфейс та керівництво користувача 28](#_Toc160890502)

[ВИСНОВКИ 30](#_Toc160890503)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 31](#_Toc160890504)

[Додаток 1. Структурна схема даних 32](#_Toc160890505)

[Додаток 2. Документація до програмного коду 34](#_Toc160890506)

[Додаток 3. Лістинг 46](#_Toc160890507)

Тема навчальної практики: "Розробка системи управління співробітниками компанії таксі"

Мета навчальної практики: Оволодіння навичками розробки програмного забезпечення для управління співробітниками таксі-компанії та вдосконалення знань, отриманих протягом курсу.

Задачі включають створення системи, що дозволяє ефективно керувати персоналом таксі-служби, включаючи прийняття замовлень, розподіл поїздок, ведення обліку зарплати та інші аспекти управління.

Актуальність: У сучасному світі, де таксі-індустрія стає все більш конкурентноспроможною та технологічно орієнтованою, розробка систем управління персоналом є ключовим елементом успішної діяльності таксі-компаній. Впровадження такої системи може сприяти підвищенню ефективності та якості обслуговування пасажирів.

Звітна робота складається з двох основних частин: теоретичної та практичної, кожна з яких включає кілька підпунктів. В кінці документа також наведені висновки, джерела та додатки.

Об'єктом дослідження є розробка програмного забезпечення для управління персоналом таксі-компанії. Предметами дослідження є мова програмування Java, середовище розробки IntelliJ IDEA, використання баз даних для зберігання інформації про співробітників та поїздки, а також розробка функціональності для автоматизації процесів управління персоналом.

# **ТЕОРЕТИЧНА ЧАСТИНА**

## **Загальні відомості про програмний продукт**

### «Система управління співробітниками компанії таксі»

Система управління співробітниками компанії таксі - це сукупність методів, технологій і процесів, які допомагають організувати, контролювати і мотивувати персонал, який займається перевезенням пасажирів. Така система повинна враховувати особливості роботи таксистів, їхні потреби, очікування і проблеми. Також вона повинна бути узгоджена зі стратегією і цілями бізнесу.

Система управління співробітниками компанії таксі включає такі функції:

* Набір і адаптація персоналу. Система допомагає знайти, оцінити і найняти кваліфікованих водіїв, а також забезпечити їхнє навчання, інструктаж і підтримку.
* Оцінка і розвиток персоналу. Система дозволяє встановити критерії оцінки роботи водіїв, проводити регулярні перевірки, аналізувати показники продуктивності, задоволення клієнтів і якості обслуговування. Також система сприяє розвитку професійних навичок і кар’єрного зростання персоналу [1].
* Мотивація і заохочення персоналу. Система допомагає створити ефективну систему оплати праці, яка враховує результати роботи, досвід, кваліфікацію і відгуки клієнтів. Також система надає можливість встановлювати премії, бонуси, гнучкий графік роботи, соціальні гарантії і інші форми заохочення персоналу.
* Комунікація і взаємодія персоналу. Система забезпечує ефективний обмін інформацією між водіями, диспетчерами, менеджерами і клієнтами. Також система сприяє створенню позитивної атмосфери в колективі, підтримці корпоративної культури і цінностей, вирішенню конфліктів і проблем [2].

### ООП та тип данних JSON

Об’єктно-орієнтоване програмування (ООП) - це парадигма програмування, яка використовує “об’єкти” - структури даних, що складаються з полів даних і методів разом з їхніми взаємодіями. ООП спрощує програмування шляхом поділу на частини, відомі як об’єкти.

JSON (JavaScript Object Notation) - це стандартний текстовий формат для представлення структурованих даних на основі синтаксису об’єкта JavaScript. Він зазвичай використовується для передачі даних в веб-додатках, наприклад, для відправки деяких даних від сервера до клієнта, таким чином, щоб це могло відображатися на веб-сторінці або навпаки.

JSON може бути збережений у власному файлі, який в основному є текстовим файлом з розширенням .json і MIME типом application/json. JSON існує як рядок, який необхідний при передачі даних через мережу. Він повинен бути перетворений на власний об’єкт JavaScript, якщо ви хочете отримати доступ до даних [3].

### Google Code Style та Java SonarLint

Google Code Style - це набір конвенцій, які використовуються для написання коду для проектів Google. Ці правила включають різні аспекти, від форматування коду до використання глобальних змінних і виключень. Вони допомагають зрозуміти велику кодову базу, коли весь код у ній написаний в однаковому стилі [4].

SonarLint - це безкоштовне розширення IDE, яке допомагає знаходити та виправляти проблеми кодування в реальному часі. Він прапорець проблеми, коли ви кодуєте, так само, як перевірка орфографії. Коли SonarLint використовується разом з SonarQube або SonarCloud, він стає потужною платформою для контролю якості коду. SonarLint підтримує багато мов, включаючи C, C++, Java, Go, JavaScript, TypeScript, Python, C#, Kotlin, Ruby, HTML, CSS, PHP & PL/SQL [5].

## Огляд аналогів

За аналог було взято 2 сайти «Taxi 838» та «Uklon»

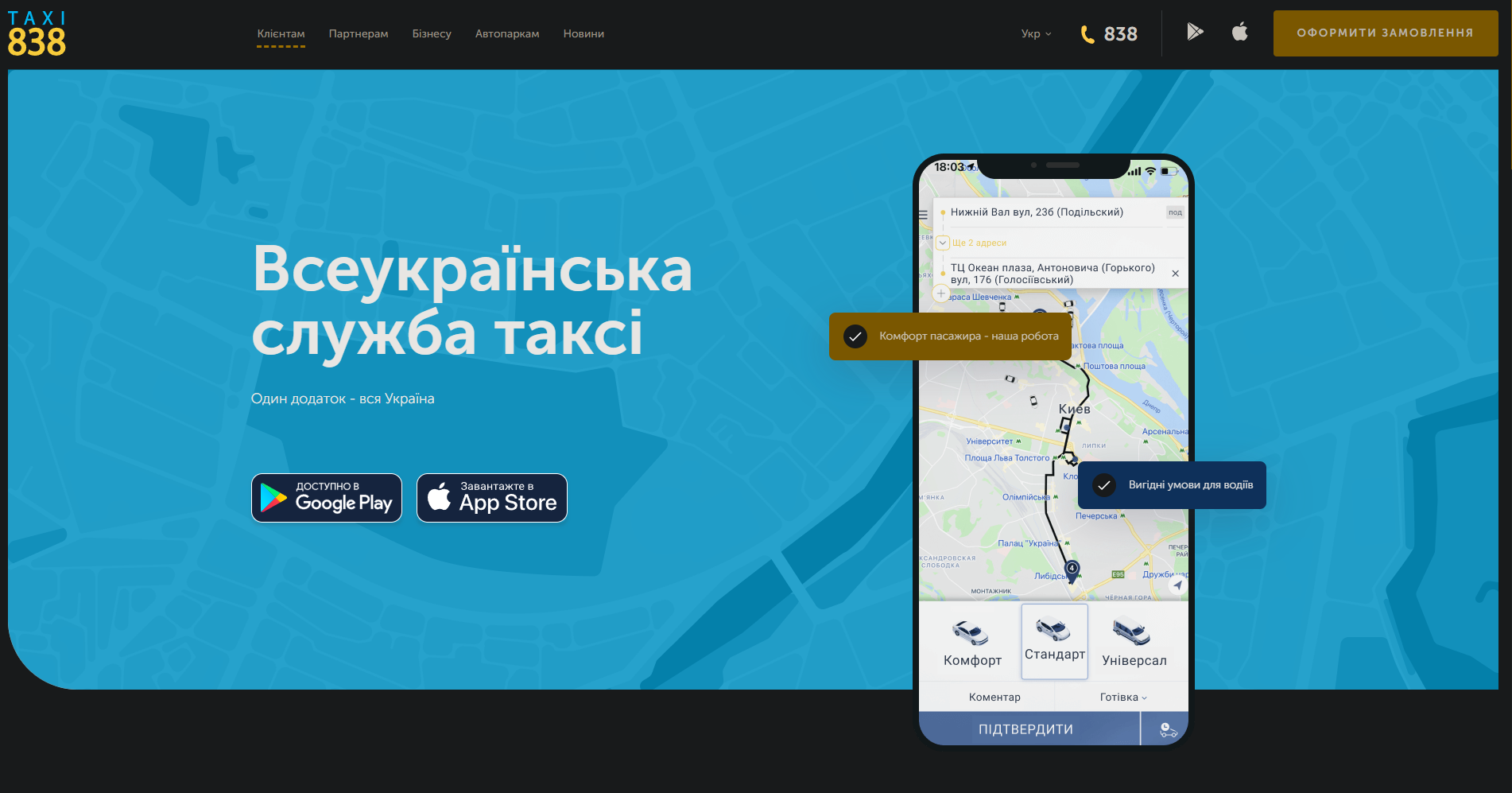


Рисунок 1.1 Ось такий дизайн на сайті «Taxi 838»

Плюси: досить простий доступ до інформації.

Мінуси: негарний дизайн та нема широкого функціоналу.

Посилання на сайт в додатках[6].

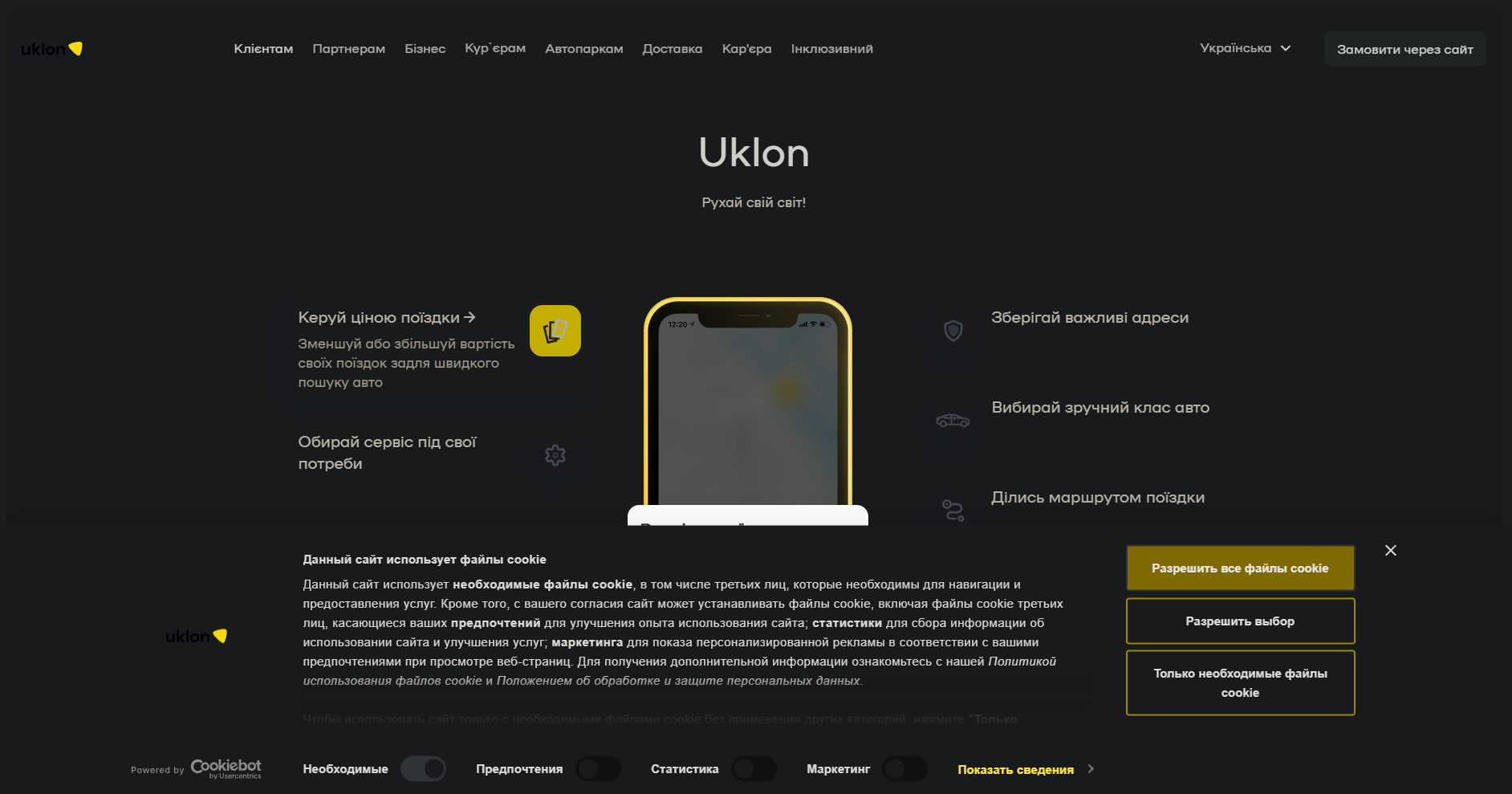


Рисунок 1.2 Ось такий дизайн на сайті «Uklon»

Плюси: стисла інформація, мінімалістичний дизайн.

Мінуси: мало функціоналу.

Посилання на сайт в додатках[7].

## **Опис предметної області та постановка задачі**

Головним завданням навчальної практики є створення консольного застосунку під назвою «Таксі компанія». Цей застосунок працює як база даних, зберігаючи в собі інформацію про співробітників компанії таксі, вільні автомобілі та графік роботи.

В даному додатку має бути реалізовано авторизацію, навігацію. Вирішено реалізувати взаємозв'язані логічні частини:

- Авторизація;

- Реєстрація;

- Водії;

- Графік роботи;

- Вільні автомобілі;

- Додавання інформації;

- Видалення інформації;

- Пошук;

- Вихід;

Для зручності користувача важливо розробити зрозумілий і простий інтерфейс програми з приємним дизайном і тимчасовим макетом. В даному випадку розробка інтерфейсу може бути виконана в середовищі розробки IntelliJ IDEA з використанням мови програмування Java.

Цільовою областю даного додатку є розробка програмного комплексу «Система управління співробітниками компанії таксі». Цей застосунок є важливим у сучасному світі, де таксі-індустрія стає все більш конкурентноспроможною та технологічно орієнтованою.

Розробка систем управління персоналом є ключовим елементом успішної діяльності таксі-компаній.

Цільова аудиторія програми - будь-який власник таксі компанії, якому необхідно переглянути список своїх співробітників та іншу інформацію. Вікове обмеження - від 18 років і старше.

## Обґрунтування вибору методу розв’язку задачі

**Чому версія JDK така?** Вибір версії JDK може бути обумовлений багатьма факторами, такими як сумісність з використовуваними бібліотеками та фреймворками, наявність нововведень у конкретній версії, або обрані стандарти для проекту.

**Чому файли саме такого розширення?** Вибір розширення файлів може бути обумовлений стандартами або конвенціями проекту. Наприклад, розширення .json може свідчити про те, що файли містять дані у форматі JSON.

**Чому архітектура даних саме така?** Архітектура даних може бути обрана відповідно до принципів нормалізації баз даних, ефективності читання чи запису даних, а також потреб бізнесу. Важливо враховувати складність операцій та швидкодію системи.

**Як розв'язали задачу авторизації і реєстрації?** Механізм авторизації та реєстрації може бути реалізований з використанням хешування паролів Bcrypt, перевірки унікальності імен користувачів та безпечних методів обробки даних форм.

**Чому CRUD реалізовано саме так?** Організація операцій CRUD (створення, читання, оновлення, видалення) у вказаному коді обумовлена вибором оптимального підходу до роботи з JSON-даними та принципами SOLID.

У реалізації використано принцип відокремлення обов'язків (Single Responsibility Principle - SRP). Кожен метод класу JsonEditor відповідає за конкретну операцію: завантаження, збереження, оновлення даних. Це сприяє збереженню чистоти коду та спрощує його розуміння.

**Які принципи програмування використорали! (SOLID, DRY…)**

У коді застосовано декілька принципів програмування:

**SOLID:**

**SRP (Single Responsibility Principle):** Клас JsonEditor має один конкретний обов'язок - редагування JSON-даних. Це сприяє високій зрозумілості та розширюваності коду.

**DRY (Don't Repeat Yourself):** Декілька методів в класі, такі як loadTopsData() та saveData(), використовуються для уникнення дублювання коду. Це полегшує зміни та підтримку коду.

**Коментарі та назви методів:** Код супроводжується коментарями, які пояснюють призначення класів і методів. Назви методів чітко вказують на їхню функціональність, що полегшує читання та розуміння коду.

Ці принципи спрощують розробку, підтримку та розширення коду, а також забезпечують його високу якість.

## Огляд засобів розробки

В рамках розробки проекту використовувалися різноманітні технології та програмні засоби для забезпечення ефективності, гнучкості та надійності системи. Серед основних технологій і інструментів можна виділити:

**Java:** Проєкт розроблявся мовою програмування Java, яка володіє високою переносимістю, об'єктно-орієнтованою архітектурою та широким спектром бібліотек.

**JSON:** Для зберігання та обміну даними використовувався формат JSON (JavaScript Object Notation). JSON є легким та зручним для використання засобом обміну структурованими даними.

**Gson:** бібліотека: Для роботи з JSON даними використовувалася бібліотека Gson, яка надає зручний інтерфейс для перетворення об'єктів Java в JSON і навпаки.

**Git:** Для контролю версій та спільної роботи над кодом використовувалася система контролю версій Git.

**IntelliJ IDEA:** Інтегроване середовище розробки (IDE), таке як IntelliJ IDEA, використовувалося для комфортної розробки, налагодження та управління проектом.

Таблица 1.1 Бібліотеки

|  |  |  |
| --- | --- | --- |
| Назва | Опис | Використання |
| **commons-lang3-3.5.jar** | Apache Commons Lang - це набір допоміжних класів для роботи з мовними операціями в Java | Включає в себе ряд корисних інструментів для роботи з рядками, масивами, числами, та ін. |
| **generex-1.0.2.jar** | Generex - це бібліотека для генерації рядків, які відповідають заданим регулярним виразам | Це може бути корисно, наприклад, при тестуванні. |
| **gson-2.10.1.jar** | Gson - бібліотека для роботи з JSON в Java | Вона надає можливість конвертації об'єктів Java в формат JSON та навпаки. |
| **javafaker-1.0.2.jar** | JavaFaker - це бібліотека для генерації випадкових даних, таких як імена, адреси, номери телефонів і т. д. | Це корисно для тестування та заповнення даних. |
| **snakeyaml-1.23.jar** | SnakeYAML - це бібліотека для обробки YAML-документів в Java | Вона може використовуватися для збереження конфігурацій та обміну структурованими даними у форматі YAML. |

# **ПРАКТИЧНА ЧАСТИНА**



## **Інформаційна структура даних**

В програмі використовуються лише json файли.

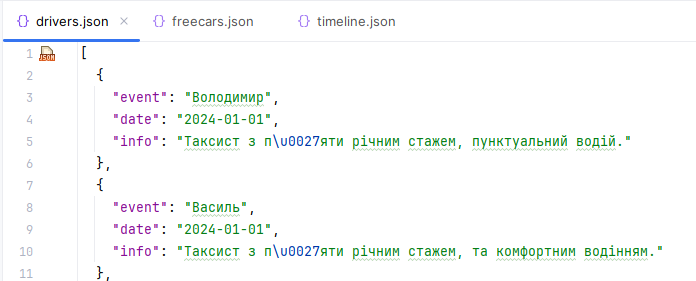


Рисунок 2.1 Файл де розписані водії

Таблица 2.1 Водії

|  |  |  |
| --- | --- | --- |
| **event (String)** | **date (String)** | **info (String)** |
| Володимир | 2024-01-01 | Таксист з п'яти річним стажем, пунктуальний водій. |
| Василь | 2024-01-01 | Таксист з п'яти річним стажем, та комфортним водінням. |
| Мага | 2024-01-01 | Таксист з великим стажем, цікавий співрозмовник. |
| Артур | 2024-01-01 | Таксист з восьми річним стажем, надійний водій. |
| Олександр | 2024-01-01 | Таксист з п'яти річним стажем, дуже добре орієнтується в місті. |
| Гога | 2024-01-01 | Таксист з п'ятнадцяти річним стажем, швидко доставляє. |

Ці елементи знаходяться в файлі drivers.json

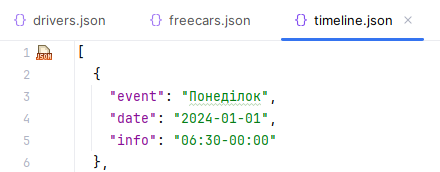


Рисунок 2.2 Файл де розписаний графік роботи

Таблица 2.2 Графік роботи

|  |  |  |
| --- | --- | --- |
| **event (String)** | **date (String)** | **info (String)** |
| Понеділок | 2024-01-01 | 06:30-00:00 |
| Вівторок | 2024-01-02 | 06:30-00:00 |
| Середа | 2024-01-03 | 06:30-00:00 |
| Четверг | 2024-01-04 | 06:30-00:00 |
| П’ятниця | 2024-01-05 | 06:30-00:00 |
| Субота | 2024-01-06 | Вихідний |
| Неділя | 2024-01-07 | Вихідний |

Ці елементи знаходяться в файлі timeline.json

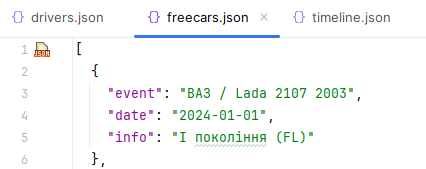


Рисунок 2.3 Файл де розписані вільні автомобілі

Таблица 2.3 Вільні автомобілі

|  |  |  |
| --- | --- | --- |
| **event (String)** | **date (String)** | **info (String)** |
| ВАЗ / Lada 2107 2003 | 2024-01-01 | I покоління (FL) |
| ВАЗ / Lada 2110 2007 | 2024-01-02 | I покоління (FL) |
| Tesla Model S 2022 | 2024-01-03 | I покоління (2nd FL) |
| Tesla Model S 2015 | 2024-01-04 | І покоління • P85 85 kWh (416 к.с.) |
| Mercedes-Benz E-Class 2013 | 2024-01-05 | W212 (FL) |
| Volvo S90 2017 | 2024-01-06 | II покоління |
| Volkswagen Passat Alltrack 2020 | 2024-01-07 | VIII покоління/B8 (FL) |

Ці елементи знаходяться в файлі freecars.json

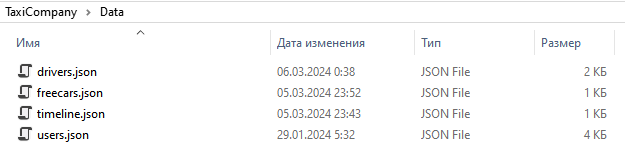


Рисунок 2.4 Всі json файли

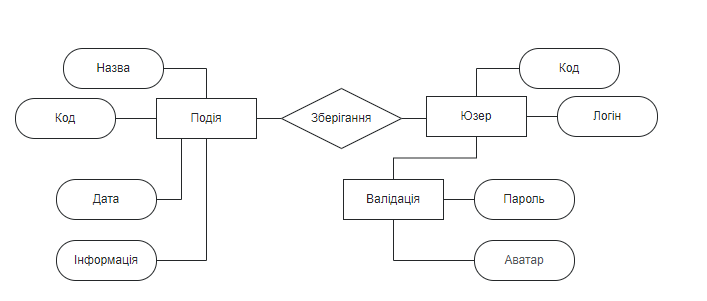


Рисунок 2.5 Концептуальна схема проекту

Функції які підтримуються в роботі з файлами:

* Читання файлу (Read): У класі Print використовується FileReader.
* Запис в файл (Create та Update): Gson для створення та оновлення JSON-файлу.
* Пошук у файлі: У класі Search реалізовано операцію пошуку у файлах за ключовим словом.)

## Функціональні частини програми

Основні компоненти

**LocalDateDeserializer та LocalDateSerializer:** LocalDateDeserializer та LocalDateSerializer - це класи для десеріалізації та серіалізації об'єктів типу LocalDate з JSON-представлення

**Registration:** Registration - клас для реєстрації нових користувачів. Використовується для отримання введених користувачем даних, перевірки унікальності імені та збереження нового користувача у файлі.

**Print:** клас для виведення інформації. Використовуються для зчитування JSON-даних та виведення структурованої інформації.

**Main:** Main - головний клас програми, який містить точку входу. Керує взаємодією з користувачем, викликає методи інших класів в залежності від вибору користувача.

Використані інструменти

Для розробки застосунку використовані наступні інструменти:

**Java:** Мова програмування для реалізації бізнес-логіки та функціональності програми.

**Gson Library:** Використовується для роботи з JSON-даними, зокрема для серіалізації та десеріалізації об'єктів.

**IntelliJ IDEA:** Інтегроване середовище розробки для написання, тестування та відлагодження коду.

Яку конвенцію іменування використали?

**Класи:** Ім'я класу починається з великої літери, а кожне нове слово також починається з великої літери (CamelCase). Наприклад, Main, Print та Other.

**Методи та Змінні:** Імена методів та змінних починаються з малої літери, а кожне нове слово починається з великої літери (CamelCase). Наприклад, printdriver, printfreecars, printEventInfo, printtimeline.

**Пакети:** Імена пакетів записані в нижньому регістрі, а якщо пакет складається з більше ніж одного слова, то кожне нове слово починається з великої літери. Наприклад, com.vintonyak.TaxiCompany.util.

**Змінні класу:** Ваш код використовує ініціалізацію полів класу при їх оголошенні, вказуючи змінні як private. Наприклад, private boolean isAuthenticated, private int userId.

**Коментарі:** Коментарі в коді також використовуються для пояснення функціоналу та роботи певних частин програми.

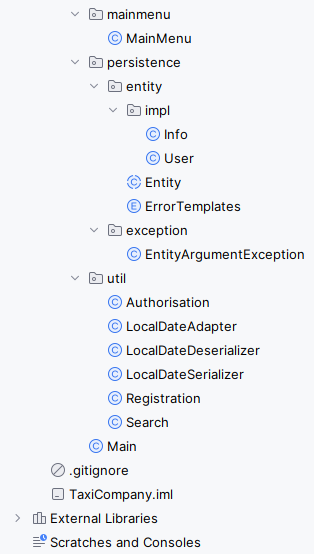
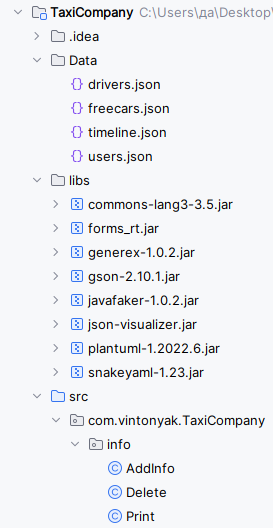


Рисунок 2.6 Фізична структура програми

При запуску додатку користувачу виводиться меню авторизації та реєстрації в якому користувач обера одну опцію яку йому потрібно та може або зареєструватись або авторизуватись. Після авторизації виводиться головне меню де є 7 основних пунктів

1. Водії (інформація про водіїв).
2. Графік роботи (інформація про графік роботи).
3. Вільні автомобілі (інформація про вільні авто).
4. Додавання інформації (функція з можливістю додавання).
5. Видалення інформації (функція з можливістю видалення).
6. Пошук (по всім файлам).

Після того як користувач завершив свою роботу в програмі він може натиснути вихід і програма завершить свою роботу

Опис структури даних та використаних змінних

**Main.java:**

* DATA\_PATH: Константа для шляху до папки з даними.
* scanner: Об'єкт для зчитування введення користувача.
* choice: Змінна для зберігання вибору користувача у головному меню.

**Print.java:**

* filePath: Шлях до JSON-файлу, що містить відповідну інформацію.

**User.java:**

* password: Змінна для зберігання паролю користувача.
* birthday: Змінна для зберігання дати народження користувача.
* email: Змінна для зберігання електронної пошти користувача.
* username: Змінна для зберігання логіну користувача.

**MainMenu.java:**

* DATA\_PATH: Константа для шляху до папки з даними.
* scanner: Об'єкт для зчитування введення користувача.
* userChoice: Змінна для зберігання вибору користувача у головному меню.
* print: Об'єкт для виклику методів обробки відповідної інформації.

Структура проекту

1. **Package: com.vintonyak.TaxiCompany**

* Main.java: Основний клас, який містить метод main для запуску додатка. Виводить головне меню, обробляє вибір користувача та викликає відповідні методи.

1. **Package: com.vintonyak.TaxiCompany.util**

* Registration.java: Клас для обробки реєстрації користувача.
* Authorisation.java: Клас для обробки авторизації користувача.
* Search.java: Клас, який містить метод для пошуку інформації в файловій системі.

1. **Package: com.vintonyak.TaxiCompany.info**

* Print.java: Клас для роботи з певною інформаціїю.
* AddInfo.java: Клас для додавання певної інформації.
* Delete.java: Клас для видалення вибраної інформації.

1. **Package: com.vintonyak.TaxiCompany.persistence.entity.impl**

* User.java: Клас, який представляє об'єкт користувача та містить відповідні методи для валідації полів.
* Info.java: Клас, який представляэ об’єкт інформації та містить відповідні методи для валідації полів.

1. **Package: com.vintonyak.TaxiCompany.mainmenu**

* MainMenu.java: Клас для представлення головного меню додатка та запуску відповідних функцій в залежності від вибору користувача.

Скріншоти внутрішньої структури класу;

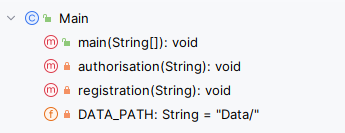


Рисунок 2.7 Внутрішня структура класу «Main»

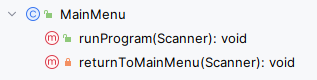


Рисунок 2.8 Внутрішня структура класу «MainMenu»

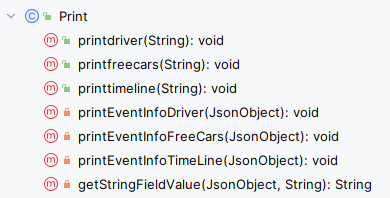


Рисунок 2.9 Внутрішня структура класу «Print»

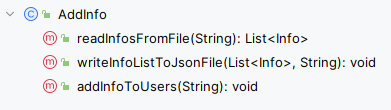


Рисунок 2.10 Внутрішня структура класу «AddInfo»

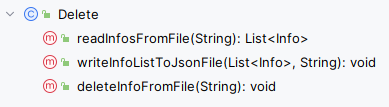


Рисунок 2.11 Внутрішня структура класу «Delete»

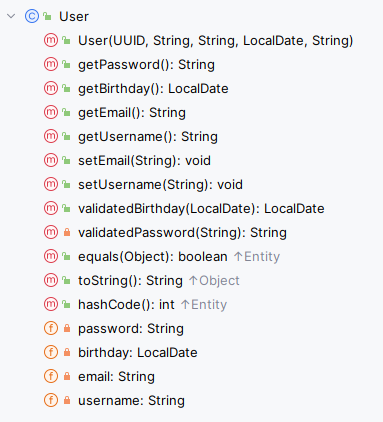


Рисунок 2.12 Внутрішня структура класу «User»

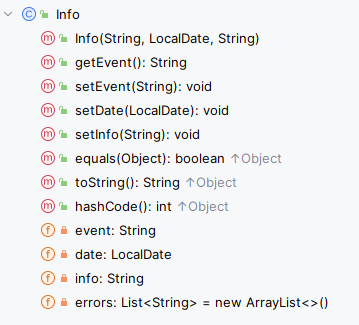


Рисунок 2.13 Внутрішня структура класу «Info»

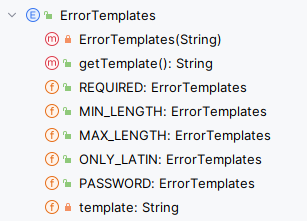


Рисунок 2.14 Внутрішня структура класу «ErroeTemplates»

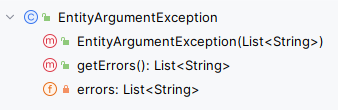


Рисунок 2.15 Внутрішня структура класу «EntityAtgumentException»



Рисунок 2.16 Внутрішня структура класу «Authorisation»

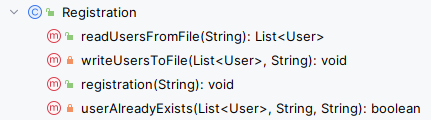


Рисунок 2.17 Внутрішня структура класу «Registration»

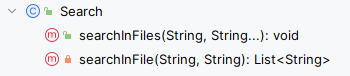


Рисунок 2.18 Внутрішня структура класу «Search»



Рисунок 2.19 Внутрішня структура класу «LocalDateDeserializer»

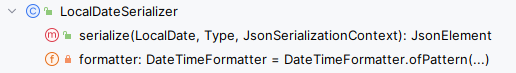


Рисунок 2.20 Внутрішня структура класу «LocalDateSerializer»

## Тестування проєкту

Додаток запускається з файлу та виводить початковий вибір для користувача.

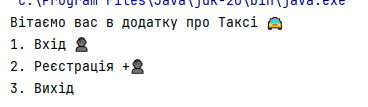


Рисунок 2.21 Запуск додатку

Якщо обрати не вірну цифру то додаток виведе цей початок знову.

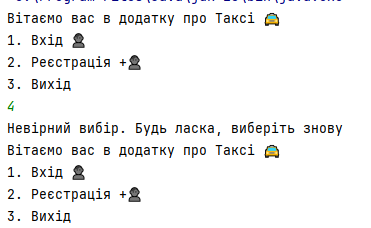


Рисунок 2.22 Не вірно обраний варіант.

Якщо в авторизації ввести не вірні дані додаток напише що дані були не вірними.

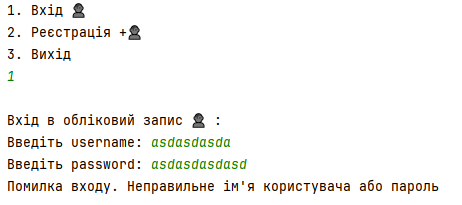


Рисунок 2.23 Валідація на вірність ведених даних в авторизації.

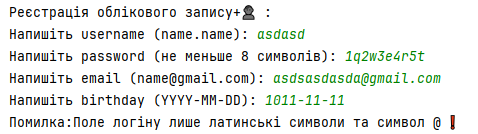


Рисунок 2.24 Валідація на вірність username.

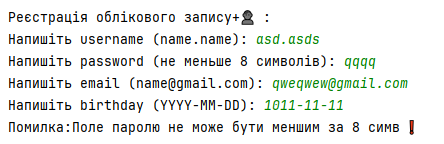


Рисунок 2.25 Валідація на вірність паролю.

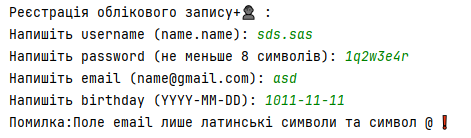


Рисунок 2.26 Валідація на вірність email.

Після вдалого входу в систему або вдалої реєстрації запускається головне меню, очищуючи консоль.

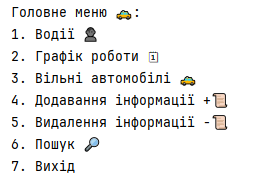


Рисунок 2.27 Головне меню

Якщо обрати 1 то виведеться весь вміст файлу drivers.json

Якщо обрати 2 то виведеться весь вміст файлу timeline.json

Якщо обрати 3 то виведеться весь вміст файлу freecars.json

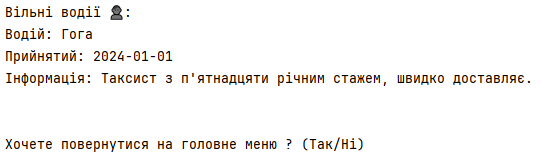


Рисунок 2.28 Вивід інформації з файлу «drivers.json»

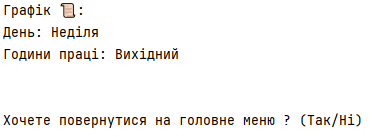


Рисунок 2.29 Вивід інформації з файлу «timeline.json»

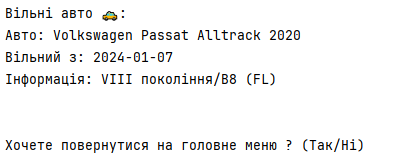


Рисунок 2.30 Вивід інформації з файлу «freecars.json»

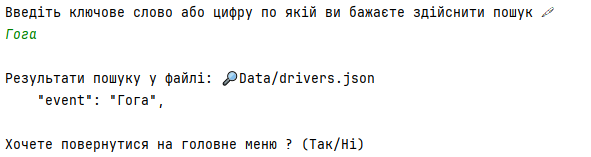


Рисунок 2.31 Вивід інформації через пошук в файлах

При натисканні Виходу програма закривається.

## Інтерфейс та керівництво користувача

Запуск додатку потребує «TaxiCompany.exe» та директорії «Data\» в якій знаходяться .json файли.

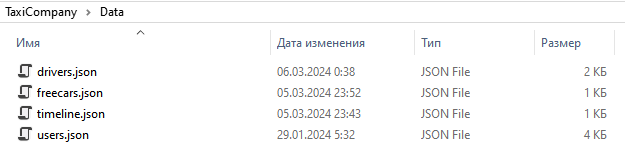


Рисунок 2.32 Вміст директорії «Data\».

При відсутності директорії «Data\» та/або її вмісту то програма після вибору авторизації або реєстрації вилітатиме. Коли в додатку потрібно вводити дату не за вказаним типом то додаток вилітатиме. Коли додаток очікує цифру а написати слово або букву то додаток вилітає.

* Протестувати на різних операційних системах! (напр. WSL Ubuntu).
* Протестувати вірність роботи програми у всіх можливих сценаріях. (напр. відсутність файлу даних, при любій модифікації даних у файлі).
* Протестувати валідацію даних при введені користувача!
* Якщо картинки повторюються в пункті 2.4, то посилатись до картинок, а не дублювати їх знову!

Для встановлення та запуску застосунка, вам знадобиться Java Runtime Environment (JRE). Ось кроки, які ви можете виконати:

**Завантажте та встановіть JRE:** Відвідайте офіційний веб-сайт Java: Java Download.

Натисніть на кнопку "Download" та виберіть версію JRE для своєї операційної системи (Windows, macOS, або Linux).

**Перевірте встановлення JRE:** Після завершення установки відкрийте командний рядок або термінал.

Введіть команду java -version і натисніть "Enter". Якщо JRE встановлено, ви побачите інформацію про версію Java.

**Запустіть застосунок:** Після встановлення JRE, ви можете запустити застосунок TaxiCompany, скомпільований за допомогою Java

Вимоги до Технічних Засобів для TaxiCompany:

**Java Runtime Environment (JRE):** Застосунок вимагає встановленого Java Runtime Environment для його коректного запуску та роботи.

**Операційна система:** Застосунок підтримує роботу на різних операційних системах, включаючи, але не обмежуючись, Windows, macOS та Linux.

**Локальне Зберігання:** Застосунок працює з локальними файлами для зберігання та обробки інформації.

Для запуску програми вам необхідно натиснути 2 рази по TaxiCompany.exe. Після запуску відкривається меню авторизації та реєстрації. Після авторизації в систему користувачеві доступні 7 основних пунктів

1. Водії (інформація про водіїв).
2. Графік роботи (інформація про графік роботи).
3. Вільні автомобілі (інформація про вільні авто).
4. Додавання інформації (функція з можливістю додавання).
5. Видалення інформації (функція з можливістю видалення).
6. Пошук (по всім файлам).

Коли користувач завершив роботу в програмі він може натиснути вихід і програма завершить свою роботу..

ВИСНОВКИ

Під час навчальної практики було обрано створення системи управління співробітниками компанії таксі. Програма розроблена з можливістю ефективної роботи з текстовими даними та забезпечує інтернаціоналізацію. У програмі реалізовано такі функції, як виведення даних з текстового файлу, операції з цими даними та управління персоналом.

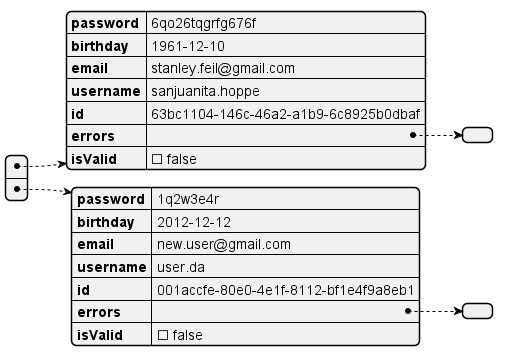
Дизайн програми ретельно продуманий і має просту, але зручну для користувача структуру. Використано емодзі та курсив для покращення візуального вигляду. Це дозволяє зробити взаємодію з програмою приємною для користувача.

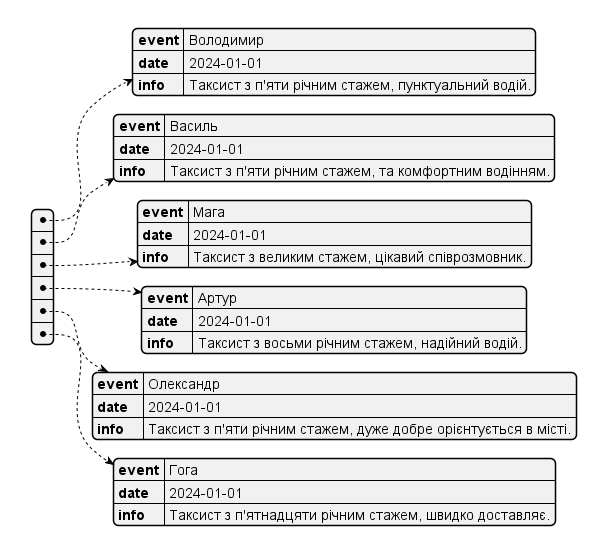
Програма створена з використанням мови програмування Java та відмінно працює в середовищі IntelliJ IDEA. Розробник має намір подальше вдосконалення програм та створення схожих проектів у майбутньому.

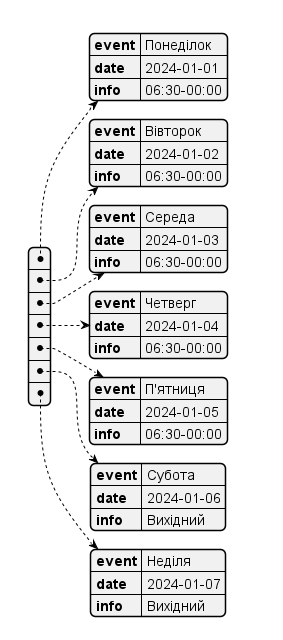
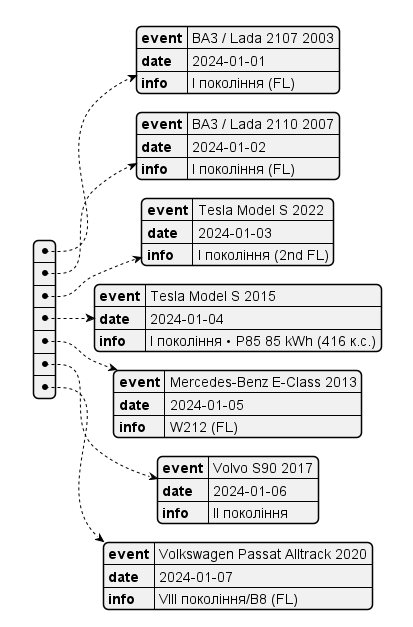
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке ERP-система управління підприємством та де її краще розмістити? URL: https://www.sim-networks.com/ukr/blog/enterprise-resource-planning-systems-and-cloud-infrastructure (дата звернення: 09.03.2024).
2. Що таке інформаційна система управління персоналом (HRIS)? URL: https://peopleforce.io/uk/hr-glossary/human-resource-information-system (дата звернення: 09.03.2024).
3. Работа с JSON. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON (дата звернення: 09.03.2024).
4. Google Style Guides. URL: https://google.github.io/styleguide/ (дата звернення: 09.03.2024).
5. SonarLint. URL: https://plugins.jetbrains.com/plugin/7973-sonarlint (дата звернення: 09.03.2024).
6. Всеукраїнська служба таксі. URL: https://taxi838.ua/ua/ (дата звернення: 09.03.2024).
7. Uklon. Рухай свій світ!. URL: https://uklon.com.ua/ (дата звернення: 09.03.2024).

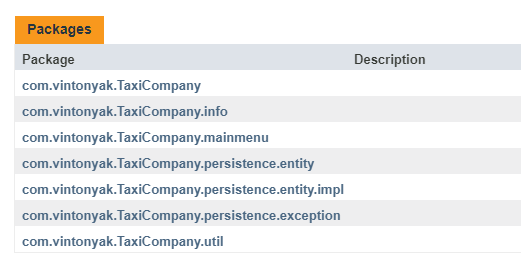
Додаток 1. Структурна схема даних



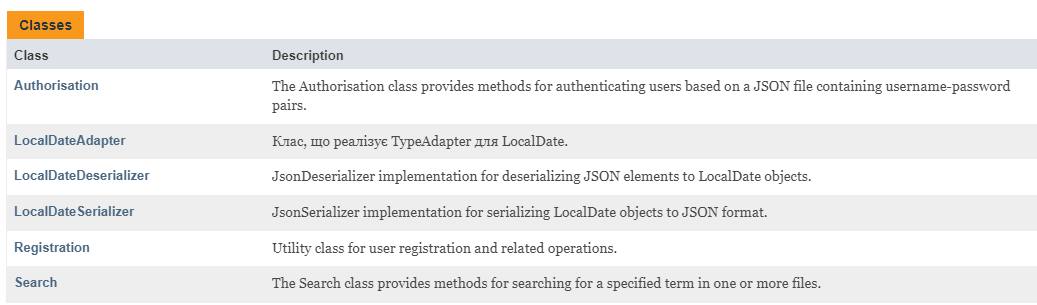


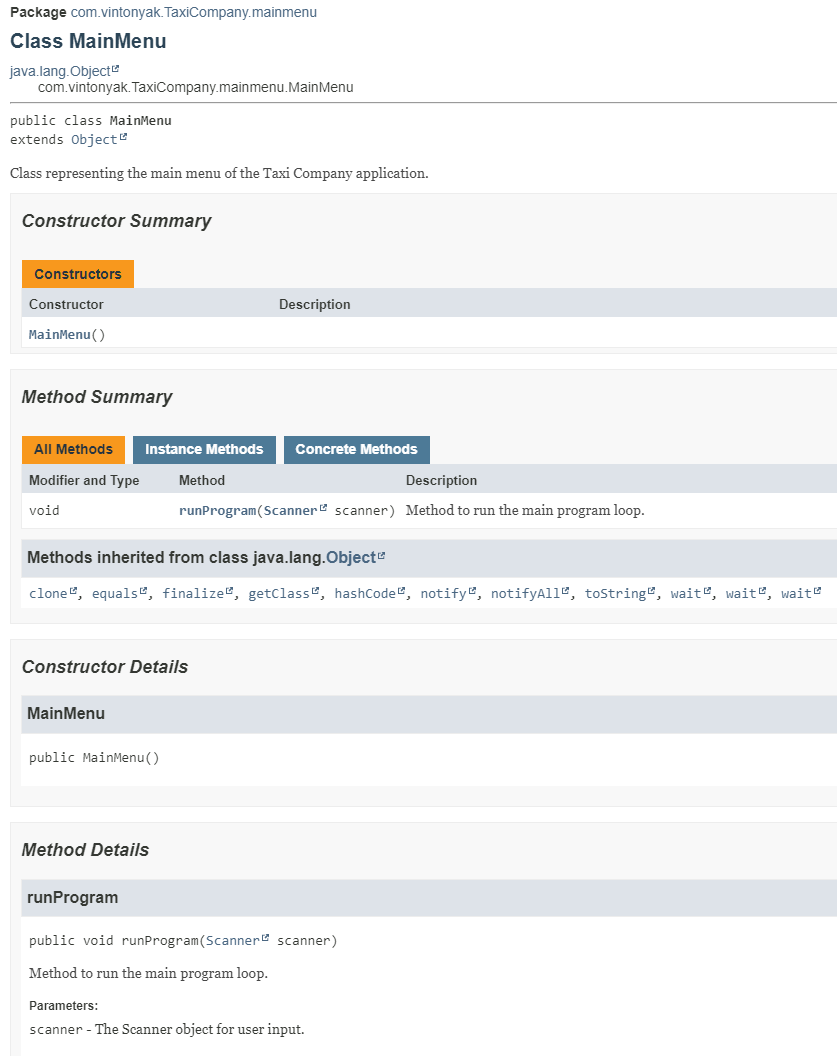


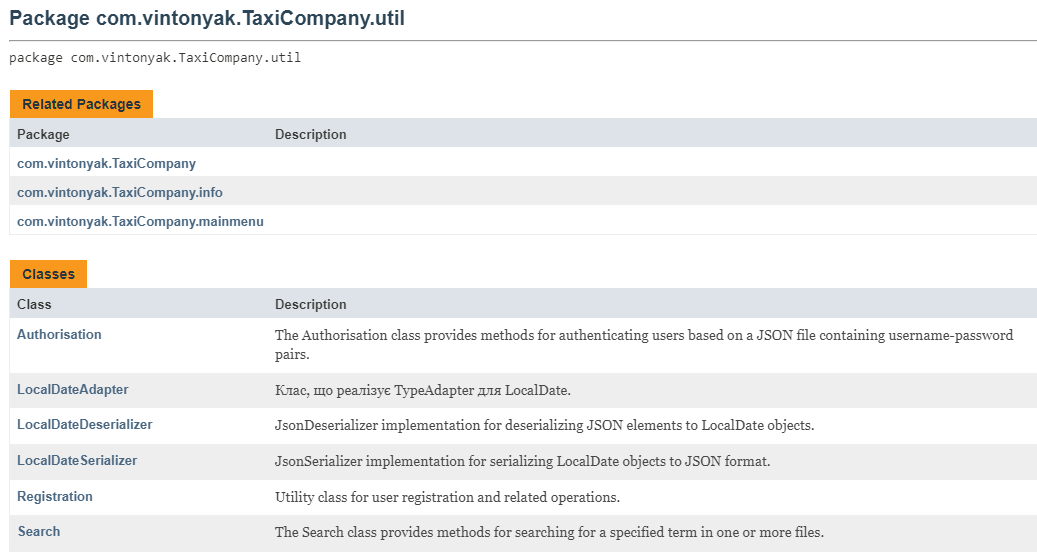
Додаток 2. Документація до програмного коду

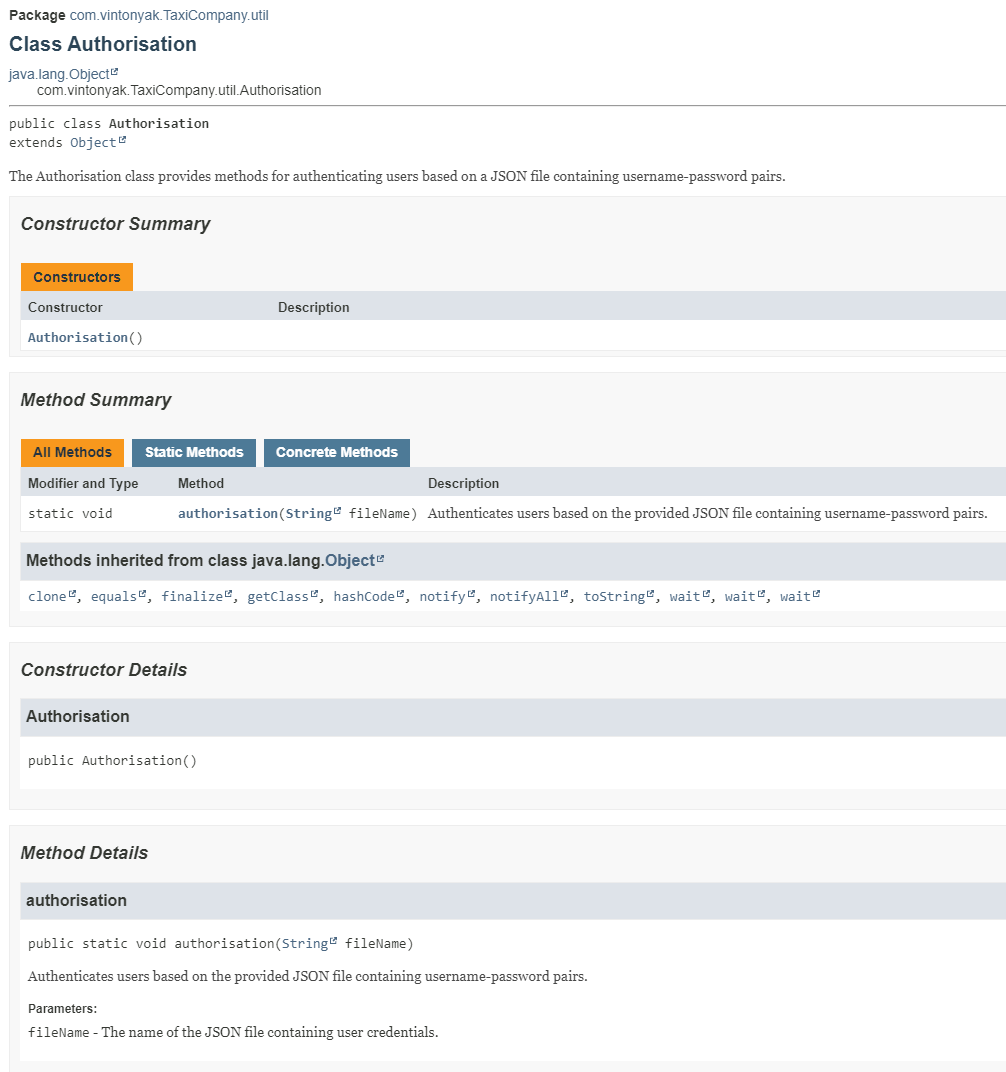


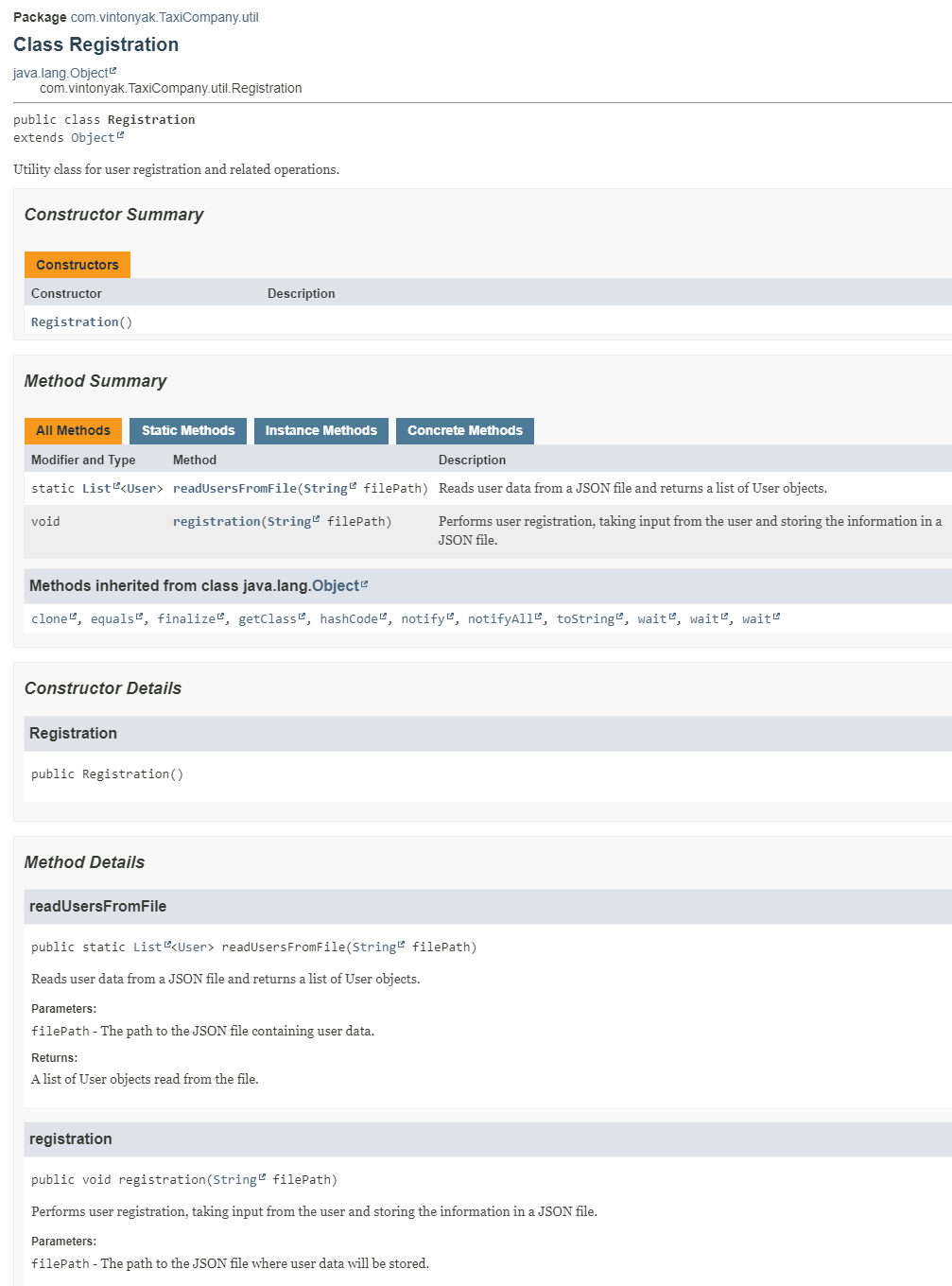


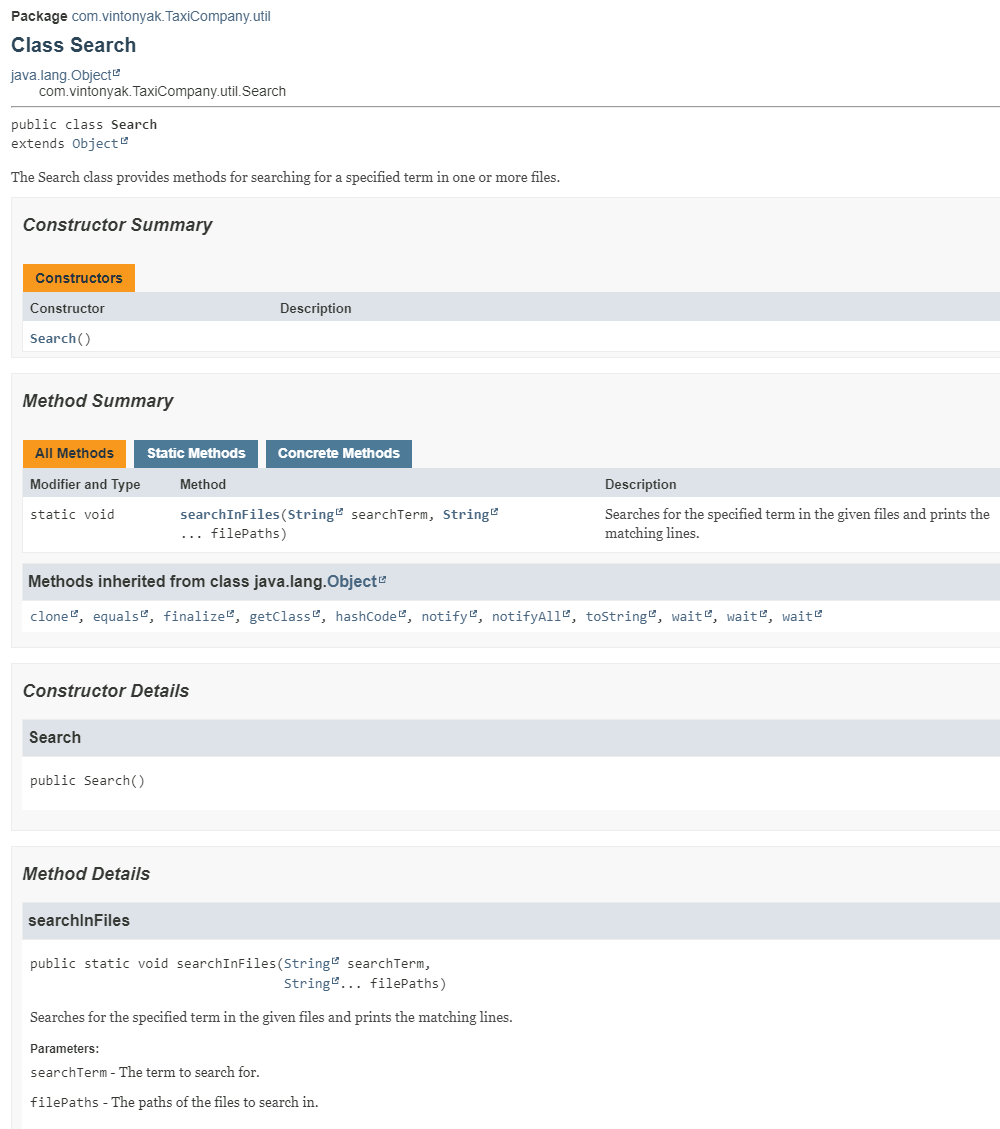




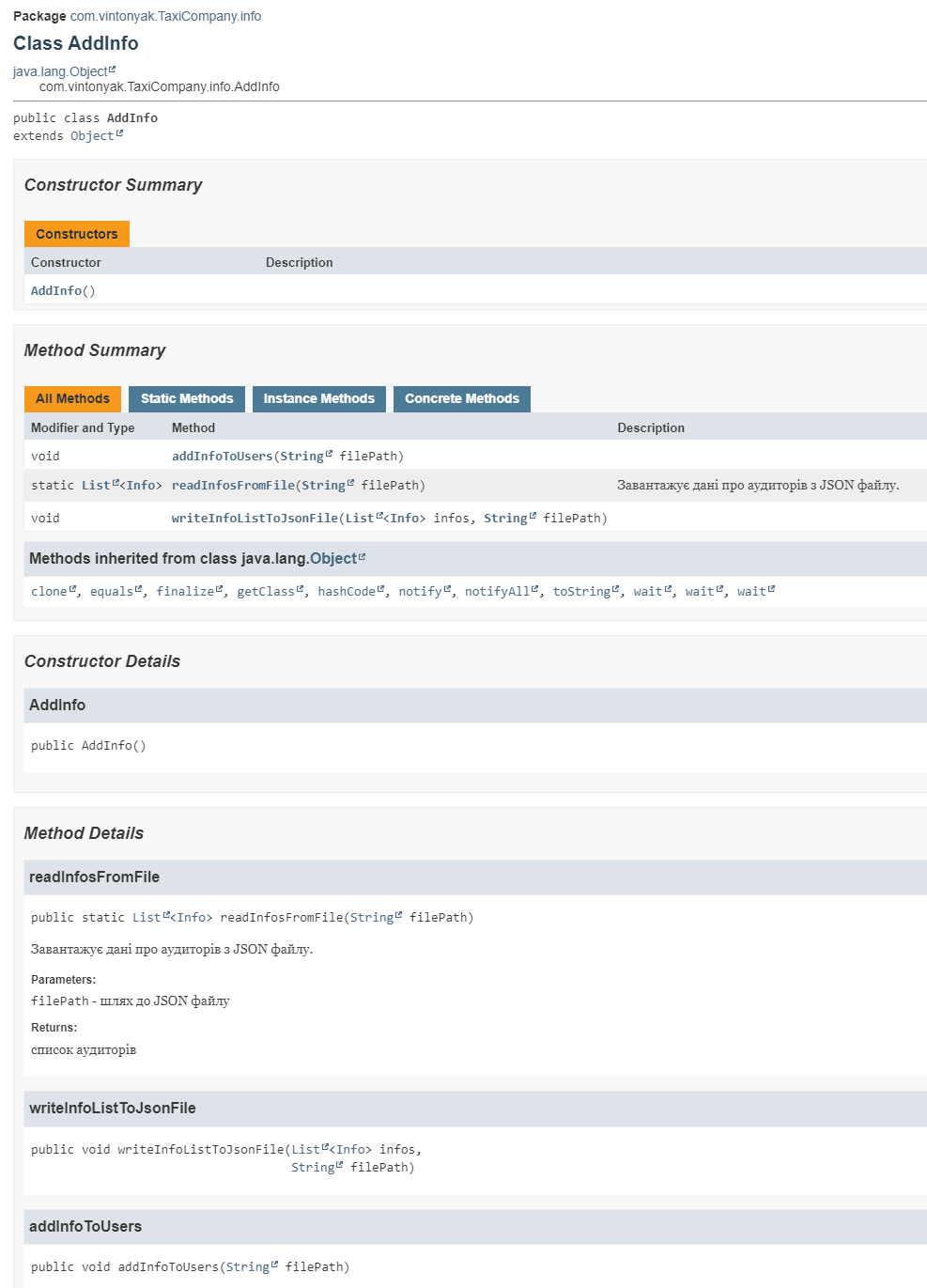


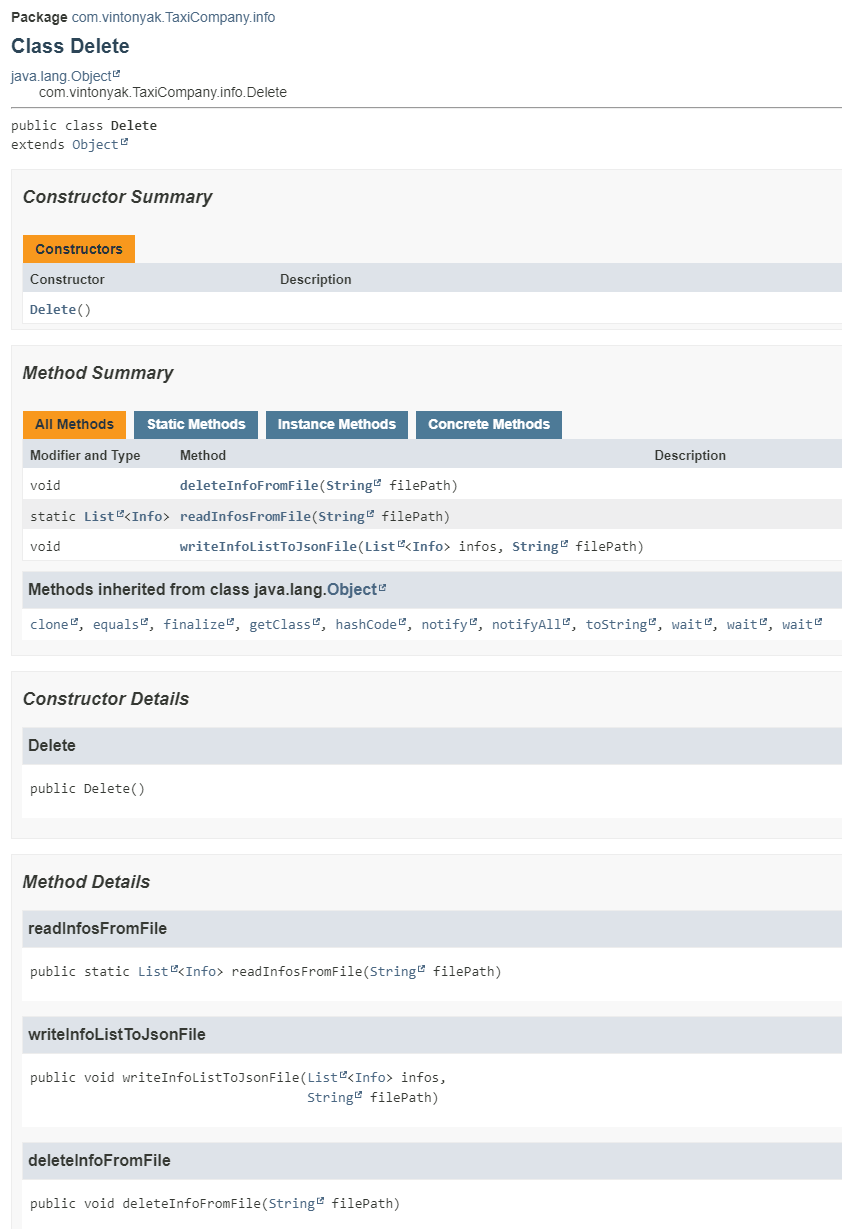


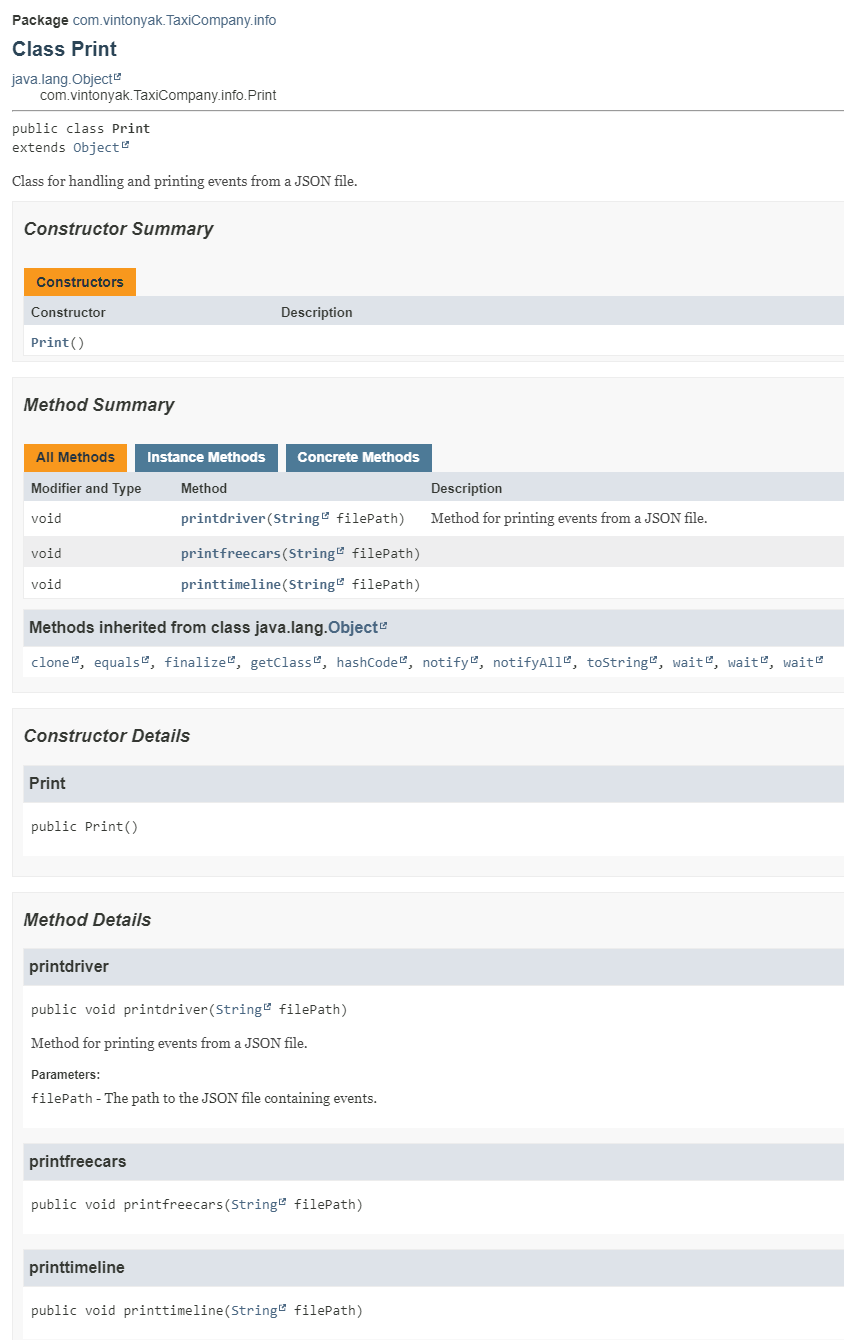


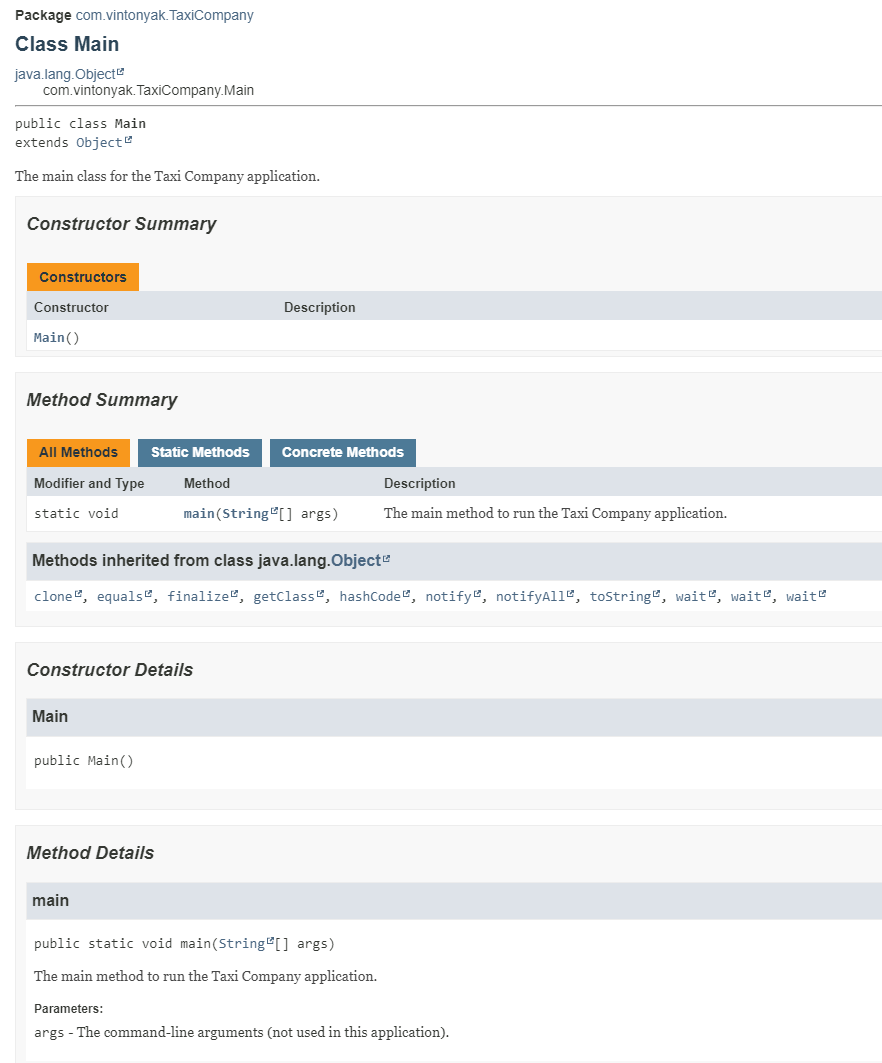


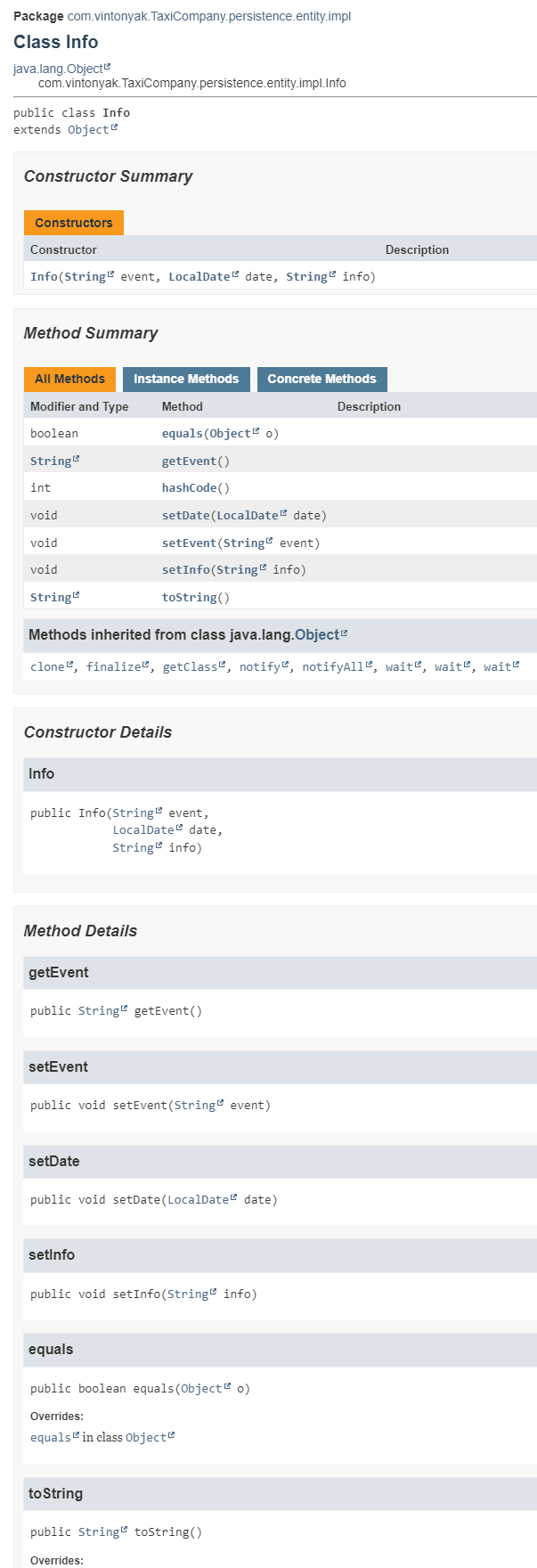


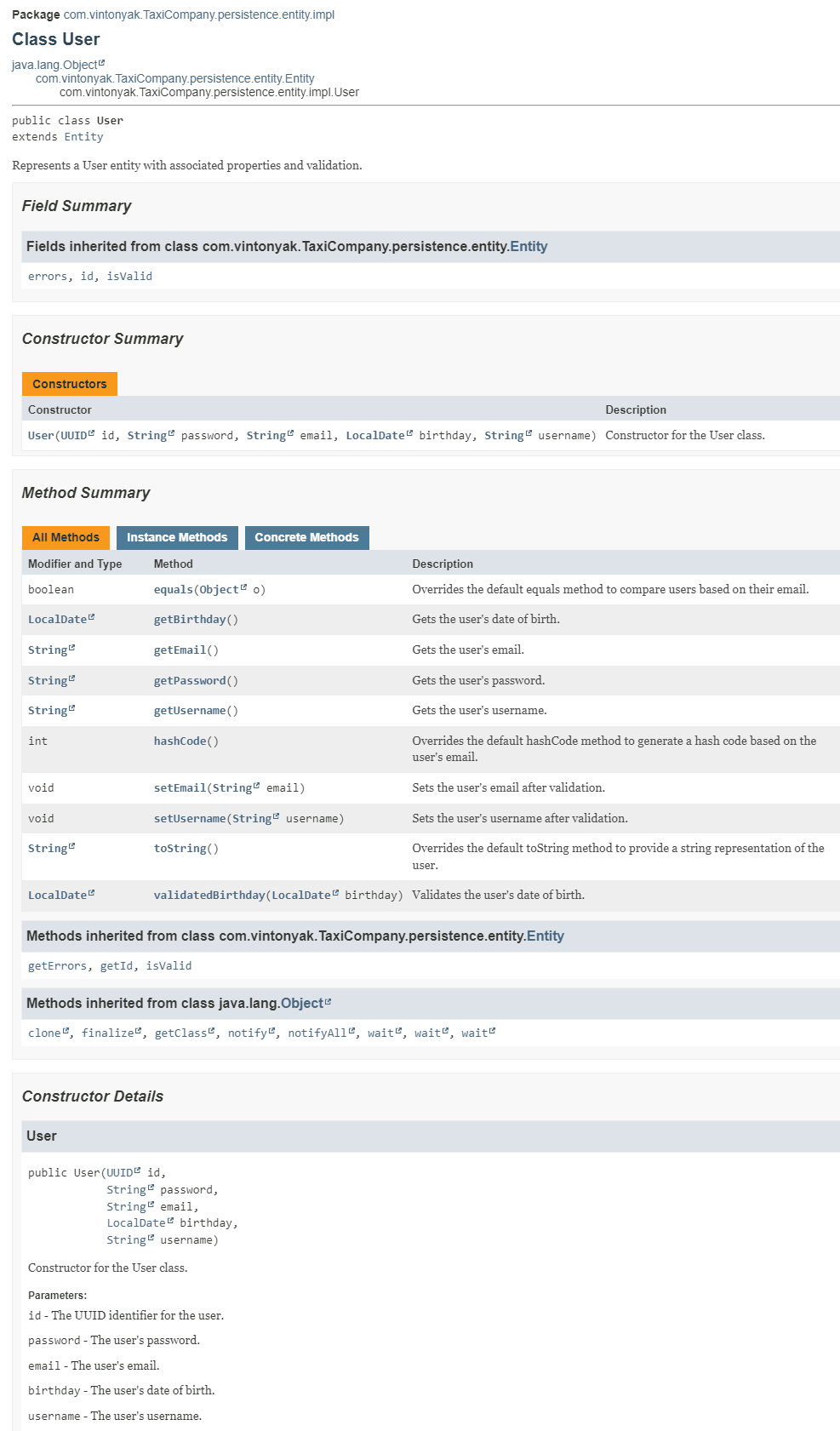


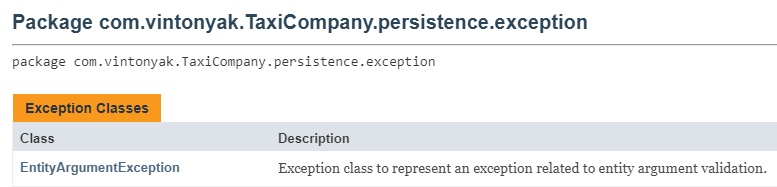


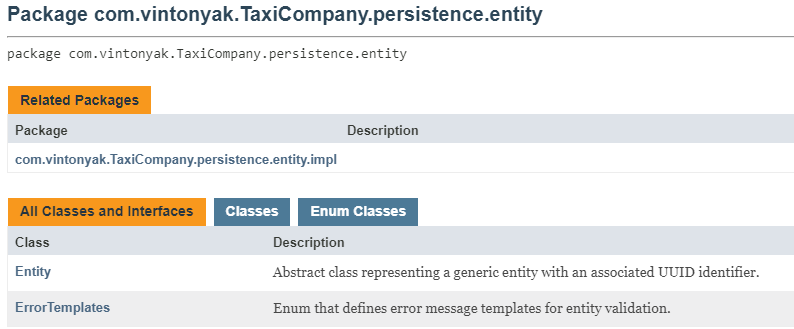


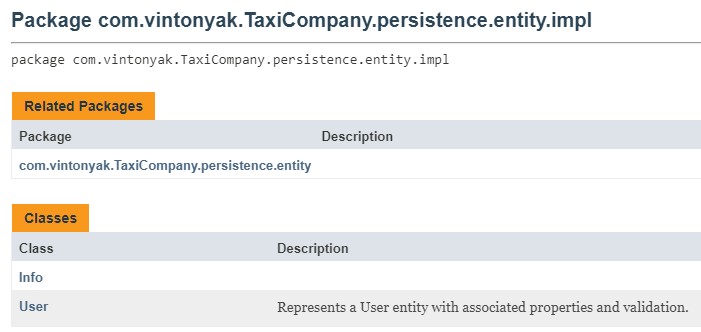












Додаток 3. Лістинг

Main.java

package com.vintonyak.TaxiCompany;  
  
import com.vintonyak.TaxiCompany.util.Registration;  
import com.vintonyak.TaxiCompany.util.Authorisation;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* The main class for the Taxi Company application.  
 \*/*public class Main {  
 private static final String *DATA\_PATH* = "Data/";  
  
 */\*\*  
 \* The main method to run the Taxi Company application.  
 \*  
 \* @param args The command-line arguments (not used in this application).  
 \*/* public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 while (true) {  
 System.*out*.println("Вітаємо вас в додатку про Таксі \uD83D\uDE96");  
 System.*out*.println("1. Вхід \uD83D\uDC64");  
 System.*out*.println("2. Реєстрація +\uD83D\uDC64");  
 System.*out*.println("3. Вихід");  
  
 int choice = scanner.nextInt();  
  
 switch (choice) {  
 case 1:  
 *authorisation*(*DATA\_PATH* + "users.json");  
 break;  
 case 2:  
 *registration*(*DATA\_PATH* + "users.json");  
 return;  
 case 3:  
 System.*out*.println("Дякуємо за використання додатка. До побачення! \uD83D\uDC4B\uD83C\uDFFC");  
 System.*exit*(0);  
 break;  
 default:  
 System.*out*.println("Невірний вибір. Будь ласка, виберіть знову");  
 }  
 }  
 }

*/\*\*  
 \* Method to handle user authorization.  
 \*  
 \* @param filePath The path to the file containing user data.  
 \*/* private static void authorisation(String filePath) {  
 Authorisation authorisation = new Authorisation();  
 authorisation.*authorisation*(filePath);  
 }  
  
 */\*\*  
 \* Method to handle user registration.  
 \*  
 \* @param filePath The path to the file containing user data.  
 \*/* private static void registration(String filePath) {  
 Registration registration = new Registration();  
 registration.registration(filePath);  
 }  
}

MainMenu.java

package com.vintonyak.TaxiCompany.mainmenu;  
import com.vintonyak.TaxiCompany.info.AddInfo;  
import com.vintonyak.TaxiCompany.info.Delete;  
import com.vintonyak.TaxiCompany.info.Print;  
import com.vintonyak.TaxiCompany.util.Search;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* Class representing the main menu of the Taxi Company application.  
 \*/*public class MainMenu {  
  
 */\*\*  
 \* Method to run the main program loop.  
 \*  
 \* @param scanner The Scanner object for user input.  
 \*/* public void runProgram(Scanner scanner) {  
 String DATA\_PATH = "Data/";  
  
 AddInfo addInfo;  
 Delete delete;  
  
 while (true) {  
 System.*out*.println("\nГоловне меню \uD83D\uDE95:");  
 System.*out*.println("1. Водії \uD83D\uDC64");  
 System.*out*.println("2. Графік роботи \uD83D\uDDD3");  
 System.*out*.println("3. Вільні автомобілі \uD83D\uDE95");  
 System.*out*.println("4. Додавання інформації +\uD83D\uDCDC");  
 System.*out*.println("5. Видалення інформації -\uD83D\uDCDC");  
 System.*out*.println("6. Пошук \uD83D\uDD0E");  
 System.*out*.println("7. Вихід");  
  
 String userChoice = scanner.nextLine();  
  
 switch (userChoice) {  
 case "1":  
 Print drivers = new Print();  
 drivers.printdriver(DATA\_PATH + "drivers.json");  
 returnToMainMenu(scanner);  
 break;  
 case "2":  
 Print timeline = new Print();  
 timeline.printtimeline(DATA\_PATH + "timeline.json");  
 returnToMainMenu(scanner);  
 break;  
 case "3":  
 Print freecars = new Print();  
 freecars.printfreecars(DATA\_PATH + "freecars.json");  
 returnToMainMenu(scanner);  
 break;  
 case "4":  
 System.*out*.println("\nОберіть файл для додавання інформації\uD83D\uDCDC:");  
 System.*out*.println("1. Водії");  
 System.*out*.println("2. Графік роботи");  
 System.*out*.println("3. Вільні авто");  
 System.*out*.println("4. Повернення в головне меню \uD83D\uDE95");  
  
 String fileChoice = scanner.nextLine();  
 String filePath;  
  
 switch (fileChoice) {  
 case "1":  
 filePath = DATA\_PATH + "drivers.json";  
 break;  
 case "2":  
 filePath = DATA\_PATH + "timeline.json";  
 break;  
 case "3":  
 filePath = DATA\_PATH + "freecars.json";  
 break;  
 case "4":  
 runProgram(scanner);  
 default:  
 System.*out*.println("Невірний вибір файлу. Використовуйте лише цифри 1-3. !");  
 returnToMainMenu(scanner);  
 return;  
 }  
  
 addInfo = new AddInfo();  
 addInfo.addInfoToUsers(filePath);  
  
 returnToMainMenu(scanner);  
 break;  
  
 case "5":  
 System.*out*.println("\nОберіть файл для видалення інформації \uD83D\uDDD1:");  
 System.*out*.println("1. Водії");  
 System.*out*.println("2. Графік роботи");  
 System.*out*.println("3. Вільні авто");  
 System.*out*.println("4. Повернення в головне меню \uD83D\uDE95");  
  
 String deleteFileChoice = scanner.nextLine();  
 String deleteFilePath;  
  
 switch (deleteFileChoice) {  
 case "1":  
 deleteFilePath = DATA\_PATH + "drivers.json";  
 break;  
 case "2":  
 deleteFilePath = DATA\_PATH + "timeline.json";  
 break;  
 case "3":  
 deleteFilePath = DATA\_PATH + "freecars.json";  
 break;  
 case "4":  
 runProgram(scanner);  
 default:  
 System.*out*.println("Невірний вибір файлу. Використовуйте лише цифри 1-3. !");  
 returnToMainMenu(scanner);  
 return;  
 }  
  
 delete = new Delete();  
 delete.deleteInfoFromFile(deleteFilePath);  
  
 returnToMainMenu(scanner);  
 break;  
 case "6":  
 System.*out*.println("\nВведіть ключове слово або цифру по якій ви бажаєте здійснити пошук \uD83D\uDD8B");  
 String searchTerm = scanner.nextLine();  
 Search.*searchInFiles*(searchTerm, DATA\_PATH + "drivers.json", DATA\_PATH + "timeline.json", DATA\_PATH + "freecars.json");  
 returnToMainMenu(scanner);  
 break;  
  
 case "7":  
 System.*out*.println("Дякуємо за використання додатка. До побачення! \uD83D\uDC4B\uD83C\uDFFC");  
 System.*exit*(0);  
 break;  
  
 default:  
 System.*out*.println("Невірний вибір. Будь ласка, введіть коректний номер опції !");  
 }  
 }  
 }  
  
 */\*\*  
 \* Method to prompt the user to return to the main menu.  
 \*  
 \* @param scanner The Scanner object for user input.  
 \*/* private void returnToMainMenu(Scanner scanner) {  
 System.*out*.println("\nХочете повернутися на головне меню ? (Так/Ні)");  
 String returnToMainMenuChoice = scanner.nextLine();  
 if (!returnToMainMenuChoice.equalsIgnoreCase("Так")) {  
 System.*out*.println("Дякуємо за використання додатка. До побачення! \uD83D\uDC4B\uD83C\uDFFC");  
 System.*exit*(0);  
 }  
 }  
}

Print.java

package com.vintonyak.TaxiCompany.info;  
  
import com.google.gson.Gson;  
import com.google.gson.JsonArray;  
import com.google.gson.JsonElement;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParseException;  
import java.io.FileReader;  
import java.io.IOException;  
  
*/\*\*  
 \* Class for handling and printing events from a JSON file.  
 \*/*public class Print {  
  
 */\*\*  
 \* Method for printing events from a JSON file.  
 \*  
 \* @param filePath The path to the JSON file containing events.  
 \*/* public void printdriver(String filePath) {  
 try {  
 *// Зчитування JSON з файлу як масив* JsonArray jsonArray = new Gson().fromJson(new FileReader(filePath), JsonArray.class);  
  
 if (jsonArray != null) {  
 for (JsonElement jsonElement : jsonArray) {  
  
 JsonObject eventObject = jsonElement.getAsJsonObject();  
 printEventInfoDriver(eventObject);  
 }  
 } else {  
 System.*out*.println("Помилка читання файлу або файл порожній !");  
 }  
 } catch(JsonParseException | IOException e){  
 e.printStackTrace();  
 }  
 }  
 public void printfreecars(String filePath) {  
 try {  
 *// Зчитування JSON з файлу як масив* JsonArray jsonArray = new Gson().fromJson(new FileReader(filePath), JsonArray.class);  
  
 if (jsonArray != null) {  
 for (JsonElement jsonElement : jsonArray) {  
  
 JsonObject eventObject = jsonElement.getAsJsonObject();  
 printEventInfoFreeCars(eventObject);  
 }  
 } else {  
 System.*out*.println("Помилка читання файлу або файл порожній !");  
 }  
 } catch(JsonParseException | IOException e){  
 e.printStackTrace();  
 }  
 }  
 public void printtimeline(String filePath) {  
 try {  
 *// Зчитування JSON з файлу як масив* JsonArray jsonArray = new Gson().fromJson(new FileReader(filePath), JsonArray.class);  
  
 if (jsonArray != null) {  
 for (JsonElement jsonElement : jsonArray) {  
  
 JsonObject eventObject = jsonElement.getAsJsonObject();  
 printEventInfoTimeLine(eventObject);  
 }  
 } else {  
 System.*out*.println("Помилка читання файлу або файл порожній !");  
 }  
 } catch(JsonParseException | IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Method for printing information about a tragic event.  
 \*  
 \* @param eventObject The JSON object representing event.  
 \*/* private void printEventInfoDriver(JsonObject eventObject) {  
 System.*out*.println("Вільні водії \uD83D\uDC64:");  
 System.*out*.println("Водій: " + getStringFieldValue(eventObject, "event"));  
 System.*out*.println("Прийнятий: " + getStringFieldValue(eventObject, "date"));  
 System.*out*.println("Інформація: " + getStringFieldValue(eventObject, "info"));  
 System.*out*.println();  
 }  
 private void printEventInfoFreeCars(JsonObject eventObject) {  
 System.*out*.println("Вільні авто \uD83D\uDE95:");  
 System.*out*.println("Авто: " + getStringFieldValue(eventObject, "event"));  
 System.*out*.println("Вільний з: " + getStringFieldValue(eventObject, "date"));  
 System.*out*.println("Інформація: " + getStringFieldValue(eventObject, "info"));  
 System.*out*.println();  
 }  
 private void printEventInfoTimeLine(JsonObject eventObject) {  
 System.*out*.println("Графік \uD83D\uDCDC:");  
 System.*out*.println("День: " + getStringFieldValue(eventObject, "event"));  
 System.*out*.println("Години праці: " + getStringFieldValue(eventObject, "info"));  
 System.*out*.println();  
 }  
 private String getStringFieldValue(JsonObject jsonObject, String fieldName) {  
 JsonElement jsonElement = jsonObject.get(fieldName);  
 if (jsonElement != null && jsonElement.isJsonPrimitive()) {  
 return jsonElement.getAsString();  
 }  
 return ""; *// Або інше значення за замовчуванням, якщо поле відсутнє або не є JSON-примітивом* }  
}

AddInfo.java

package com.vintonyak.TaxiCompany.info;  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.reflect.TypeToken;  
import com.vintonyak.TaxiCompany.persistence.entity.impl.Info;  
import com.vintonyak.TaxiCompany.persistence.exception.EntityArgumentException;  
import com.vintonyak.TaxiCompany.util.LocalDateAdapter;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.lang.reflect.Type;  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Scanner;  
  
public class AddInfo {  
  
 */\*\*  
 \* Завантажує данi про аудиторiв з JSON файлу.  
 \*  
 \* @param filePath шлях до JSON файлу  
 \* @return список аудиторiв  
 \*/* public static List<Info> readInfosFromFile(String filePath) {  
 List<Info> infos;  
  
 try (FileReader reader = new FileReader(filePath)) {  
 *// Створюємо iнстанцiю Gson з власним TypeAdapter для LocalDate* Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateAdapter())  
 .create();  
  
 *// Визначаємо тип для десерiалiзацiї* Type infoListType = new TypeToken<List<Info>>() {  
 }.getType();  
  
 *// Читаємо вмiст JSON файлу та десерiалiзуємо його у список аудиторiв* infos = gson.fromJson(reader, infoListType);  
 } catch (IOException e) {  
 *// Обробка виняткiв (наприклад, файл не знайдено, проблеми з форматом JSON)* System.*err*.println("Error loading data from JSON: " + e.getMessage());  
 infos = List.*of*(); *// Альтернативно, можна iнiцiалiзувати порожнiм списком* }  
  
 return infos;  
 }  
  
  
 public void writeInfoListToJsonFile(List<Info> infos, String filePath) {  
  
 try (FileWriter writer = new FileWriter(filePath)) {  
 Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateAdapter())  
 .setPrettyPrinting().create();  
  
 gson.toJson(infos, writer);  
  
 System.*out*.println("Інформацію збережено в файл " + filePath);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void addInfoToUsers(String filePath) {  
 List<Info> infos = *readInfosFromFile*(filePath);  
  
 Scanner scanner = new Scanner(System.*in*);  
 try {  
 *// Введення інформації* System.*out*.println("\nДодавання інформації +\uD83D\uDCDC:");  
 System.*out*.print("Введіть подію: ");  
 String event = scanner.nextLine();  
 System.*out*.print("Введіть інформацію: ");  
 String info = scanner.nextLine();  
 System.*out*.print("Введіть дату (YYYY-MM-DD): ");  
 LocalDate date = LocalDate.*parse*(scanner.nextLine());  
  
 *// Створення об'єкта Info та додавання його до списку* Info newInfo = new Info(event, date, info);  
 infos.add(newInfo);  
 writeInfoListToJsonFile(infos, filePath);  
  
 System.*out*.println("Інформацію успішно додано");  
  
 } catch (EntityArgumentException e) {  
 for (String error : e.getErrors()) {  
 System.*out*.println("Error: " + error);  
 }  
 }  
 }  
}

Delete.java

package com.vintonyak.TaxiCompany.info;  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.reflect.TypeToken;  
import com.vintonyak.TaxiCompany.persistence.entity.impl.Info;  
import com.vintonyak.TaxiCompany.util.LocalDateAdapter;  
  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.lang.reflect.Type;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class Delete {  
 public static List<Info> readInfosFromFile(String filePath) {  
 List<Info> infos;  
  
 try (FileReader reader = new FileReader(filePath)) {  
 *// Створюємо iнстанцiю Gson з власним TypeAdapter для LocalDate* Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateAdapter())  
 .create();  
  
 *// Визначаємо тип для десерiалiзацiї* Type infoListType = new TypeToken<List<Info>>() {  
 }.getType();  
  
 *// Читаємо вмiст JSON файлу та десерiалiзуємо його у список аудиторiв* infos = gson.fromJson(reader, infoListType);  
 } catch (IOException e) {  
 *// Обробка виняткiв (наприклад, файл не знайдено, проблеми з форматом JSON)* System.*err*.println("Error loading data from JSON: " + e.getMessage());  
 infos = List.*of*(); *// Альтернативно, можна iнiцiалiзувати порожнiм списком* }  
  
 return infos;  
 }  
  
  
 public void writeInfoListToJsonFile(List<Info> infos, String filePath) {  
  
 try (FileWriter writer = new FileWriter(filePath)) {  
 Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateAdapter())  
 .setPrettyPrinting().create();  
  
 gson.toJson(infos, writer);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void deleteInfoFromFile(String filePath) {  
 List<Info> infos = *readInfosFromFile*(filePath);  
  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.println("Введіть подію для видалення:");  
 String eventToDelete = scanner.nextLine();  
  
 List<Info> updatedInfos = new ArrayList<>();  
  
 for (Info info : infos) {  
 if (!info.getEvent().equals(eventToDelete)) {  
 updatedInfos.add(info);  
 }  
 }  
  
 if (!updatedInfos.isEmpty()) {  
 System.*out*.println("Інформацію успішно видалено");  
  
 *// Зберегти оновлений список у JSON файлі* writeInfoListToJsonFile(updatedInfos, filePath);  
 scanner.nextLine();  
 } else {  
 System.*out*.println("Події не знайдено.");  
 scanner.nextLine();  
 }  
 }  
}

User.java

package com.vintonyak.TaxiCompany.persistence.entity.impl;  
  
import com.vintonyak.TaxiCompany.persistence.entity.Entity;  
import com.vintonyak.TaxiCompany.persistence.entity.ErrorTemplates;  
import com.vintonyak.TaxiCompany.persistence.exception.EntityArgumentException;  
  
import java.time.LocalDate;  
import java.util.Objects;  
import java.util.UUID;  
import java.util.regex.Pattern;  
  
*/\*\*  
 \* Represents a User entity with associated properties and validation.  
 \*/*public class User extends Entity {  
  
 private final String password;  
 private final LocalDate birthday;  
 private String email;  
 private String username;  
  
 */\*\*  
 \* Constructor for the User class.  
 \*  
 \* @param id The UUID identifier for the user.  
 \* @param password The user's password.  
 \* @param email The user's email.  
 \* @param birthday The user's date of birth.  
 \* @param username The user's username.  
 \*/* public User(UUID id, String password, String email, LocalDate birthday, String username) {  
 super(id);  
 this.password = validatedPassword(password);  
 setEmail(email);  
 this.birthday = validatedBirthday(birthday);  
 setUsername(username);  
 }  
  
 */\*\*  
 \* Gets the user's password.  
 \*  
 \* @return The user's password.  
 \*/* public String getPassword() {  
 return password;  
 }  
  
 */\*\*  
 \* Gets the user's date of birth.  
 \*  
 \* @return The user's date of birth.  
 \*/* public LocalDate getBirthday() {  
 return birthday;  
 }  
  
 */\*\*  
 \* Gets the user's email.  
 \*  
 \* @return The user's email.  
 \*/* public String getEmail() {  
 return email;  
 }  
  
 */\*\*  
 \* Gets the user's username.  
 \*  
 \* @return The user's username.  
 \*/* public String getUsername() {  
 return username;  
 }  
  
 */\*\*  
 \* Sets the user's email after validation.  
 \*  
 \* @param email The user's email to be set.  
 \*/* public void setEmail(String email) {  
 final String templateName = "email";  
  
 if (email.isBlank()) {  
 errors.add(ErrorTemplates.*REQUIRED*.getTemplate().formatted(templateName));  
 }  
 var pattern = Pattern.*compile*("^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}$");  
 if (!pattern.matcher(email).matches()) {  
 errors.add(ErrorTemplates.*ONLY\_LATIN*.getTemplate().formatted(templateName, 24));  
 }  
  
 if (!this.errors.isEmpty()) {  
 throw new EntityArgumentException(errors);  
 }  
  
 this.email = email;  
 }  
  
 */\*\*  
 \* Sets the user's username after validation.  
 \*  
 \* @param username The user's username to be set.  
 \*/* public void setUsername(String username) {  
 final String templateName = "логіну";  
  
 if (username.isBlank()) {  
 errors.add(ErrorTemplates.*REQUIRED*.getTemplate().formatted(templateName));  
 }  
 if (username.length() < 4) {  
 errors.add(ErrorTemplates.*MIN\_LENGTH*.getTemplate().formatted(templateName, 4));  
 }  
 if (username.length() > 24) {  
 errors.add(ErrorTemplates.*MAX\_LENGTH*.getTemplate().formatted(templateName, 24));  
 }  
 var pattern = Pattern.*compile*("^[a-zA-Z0-9\_]+$");  
 if (pattern.matcher(username).matches()) {  
 errors.add(ErrorTemplates.*ONLY\_LATIN*.getTemplate().formatted(templateName, 24));  
 }  
  
 if (!this.errors.isEmpty()) {  
 throw new EntityArgumentException(errors);  
 }  
  
 this.username = username;  
 }  
  
 */\*\*  
 \* Validates the user's date of birth.  
 \*  
 \* @param birthday The user's date of birth to be validated.  
 \* @return The validated date of birth.  
 \*/* public LocalDate validatedBirthday(LocalDate birthday) {  
 final String templateName = "дня народження";  
  
 if (password.isBlank()) {  
 errors.add(ErrorTemplates.*REQUIRED*.getTemplate().formatted(templateName));  
 }  
  
 var pattern = Pattern.*compile*("^\\d{4},\\d{2},\\d{2}$");  
 if (pattern.matcher(password).matches()) {  
 errors.add(ErrorTemplates.*PASSWORD*.getTemplate().formatted(templateName, 24));  
 }  
  
 if (!this.errors.isEmpty()) {  
 throw new EntityArgumentException(errors);  
 }  
  
 return birthday;  
 }  
  
 */\*\*  
 \* Validates and returns the user's password.  
 \*  
 \* @param password The user's password to be validated.  
 \* @return The validated password.  
 \*/* private String validatedPassword(String password) {  
 final String templateName = "паролю";  
  
 if (password.isBlank()) {  
 errors.add(ErrorTemplates.*REQUIRED*.getTemplate().formatted(templateName));  
 }  
 if (password.length() < 8) {  
 errors.add(ErrorTemplates.*MIN\_LENGTH*.getTemplate().formatted(templateName, 8));  
 }  
 if (password.length() > 32) {  
 errors.add(ErrorTemplates.*MAX\_LENGTH*.getTemplate().formatted(templateName, 32));  
 }  
 var pattern = Pattern.*compile*("\"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\\\d).+$\"");  
 if (pattern.matcher(password).matches()) {  
 errors.add(ErrorTemplates.*PASSWORD*.getTemplate().formatted(templateName, 24));  
 }  
  
 if (!this.errors.isEmpty()) {  
 throw new EntityArgumentException(errors);  
 }  
  
 return password;  
 }  
  
 */\*\*  
 \* Overrides the default equals method to compare users based on their email.  
 \*  
 \* @param o The object to compare with this user.  
 \* @return True if the users are equal, false otherwise.  
 \*/* @Override  
 public boolean equals(Object o) {  
 if (this == o) {  
 return true;  
 }  
 if (o == null || getClass() != o.getClass()) {  
 return false;  
 }  
 User user = (User) o;  
 return Objects.*equals*(email, user.email);  
 }  
  
 */\*\*  
 \* Overrides the default toString method to provide a string representation of the user.  
 \*  
 \* @return A string representation of the user.  
 \*/* @Override  
 public String toString() {  
 return "User{" +  
 "password='" + password + '\'' +  
 ", birthday=" + birthday +  
 ", email='" + email + '\'' +  
 ", username='" + username + '\'' +  
 ", id=" + id +  
 '}';  
 }  
  
 */\*\*  
 \* Overrides the default hashCode method to generate a hash code based on the user's email.  
 \*  
 \* @return The hash code value for this user.  
 \*/* @Override  
 public int hashCode() {  
 return Objects.*hash*(email);  
 }  
}

Entity.java

package com.krnelx.Japaneseguide.persistence.entity;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
import java.util.UUID;  
  
*/\*\*  
 \* Abstract class representing a generic entity with an associated UUID identifier.  
 \*/*public abstract class Entity {  
  
 */\*\*  
 \* The unique identifier for the entity.  
 \*/* protected final UUID id;  
  
 */\*\*  
 \* List of error messages associated with the entity's validation.  
 \*/* protected List<String> errors;  
  
 */\*\*  
 \* Flag indicating whether the entity is considered valid.  
 \*/* protected boolean isValid;  
  
 */\*\*  
 \* Constructor for the Entity class.  
 \*  
 \* @param id The UUID identifier for the entity.  
 \*/* protected Entity(UUID id) {  
 errors = new ArrayList<>();  
 this.id = id;  
 }  
  
 */\*\*  
 \* Gets the UUID identifier of the entity.  
 \*  
 \* @return The UUID identifier.  
 \*/* public UUID getId() {  
 return id;  
 }  
  
 */\*\*  
 \* Checks if the entity is considered valid based on validation errors.  
 \*  
 \* @return True if the entity is valid, false otherwise.  
 \*/* public boolean isValid() {  
 return errors.isEmpty();  
 }  
  
 */\*\*  
 \* Gets the list of error messages associated with the entity's validation.  
 \*  
 \* @return The list of error messages.  
 \*/* public List<String> getErrors() {  
 return errors;  
 }  
  
 */\*\*  
 \* Overrides the default equals method to compare entities based on their UUID.  
 \*  
 \* @param o The object to compare with this entity.  
 \* @return True if the entities are equal, false otherwise.  
 \*/* @Override  
 public boolean equals(Object o) {  
 if (this == o) {  
 return true;  
 }  
 if (o == null || getClass() != o.getClass()) {  
 return false;  
 }  
 Entity entity = (Entity) o;  
 return Objects.*equals*(id, entity.id);  
 }  
  
 */\*\*  
 \* Overrides the default hashCode method to generate a hash code based on the entity's UUID.  
 \*  
 \* @return The hash code value for this entity.  
 \*/* @Override  
 public int hashCode() {  
 return Objects.*hash*(id);  
 }  
}

ErrorTemplates.java

package com.krnelx.Japaneseguide.persistence.entity;  
  
*/\*\*  
 \* Enum that defines error message templates for entity validation.  
 \*/*public enum ErrorTemplates {  
  
  
 */\*\*  
 \* Error template for required field validation.  
 \*/  
 REQUIRED*("Поле %s є обов'язковим до заповнення ❗"),  
  
 */\*\*  
 \* Error template for minimum length validation.  
 \*/  
 MIN\_LENGTH*("Поле %s не може бути меншим за %d симв ❗"),  
  
 */\*\*  
 \* Error template for maximum length validation.  
 \*/  
 MAX\_LENGTH*("Поле %s не може бути бiльшим за %d симв ❗"),  
  
 */\*\*  
 \* Error template for allowing only Latin characters.  
 \*/  
 ONLY\_LATIN*("Поле %s лише латинськi символи та символ @ ❗"),  
  
 */\*\*  
 \* Error template for password validation.  
 \*/  
 PASSWORD*("Поле %s латинськi миволи, хочаб одна буква з великоi, одна з малоi та хочаб одна цифра ❗");  
  
 */\*\*  
 \* The error message template.  
 \*/* private String template;  
  
 */\*\*  
 \* Constructor for ErrorTemplates enum.  
 \*  
 \* @param template The error message template.  
 \*/* ErrorTemplates(String template) {  
 this.template = template;  
 }  
  
 */\*\*  
 \* Gets the error message template.  
 \*  
 \* @return The error message template.  
 \*/* public String getTemplate() {  
 return template;  
 }  
}

EntityAtgumentException.java

package com.krnelx.Japaneseguide.persistence.exception;  
  
import java.util.List;  
  
*/\*\*  
 \* Exception class to represent an exception related to entity argument validation.  
 \*/*public class EntityArgumentException extends IllegalArgumentException {  
  
 */\*\*  
 \* List of error messages associated with the exception.  
 \*/* private final List<String> errors;  
  
 */\*\*  
 \* Constructs a new EntityArgumentException with the specified list of errors.  
 \*  
 \* @param errors The list of error messages associated with the exception.  
 \*/* public EntityArgumentException(List<String> errors) {  
 this.errors = errors;  
 }  
  
 */\*\*  
 \* Gets the list of error messages associated with the exception.  
 \*  
 \* @return The list of error messages.  
 \*/* public List<String> getErrors() {  
 return errors;  
 }  
}

Authorisation.java

package com.vintonyak.TaxiCompany.util;  
  
import com.google.gson.\*;  
import com.vintonyak.TaxiCompany.mainmenu.MainMenu;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.Scanner;  
  
*/\*\*  
 \* The Authorisation class provides methods for authenticating users based on a JSON file containing username-password pairs.  
 \*/*public class Authorisation {  
 */\*\*  
 \* Authenticates users based on the provided JSON file containing username-password pairs.  
 \*  
 \* @param fileName The name of the JSON file containing user credentials.  
 \*/* public static void authorisation(String fileName) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 boolean authenticationSuccessful = false;  
  
 while (!authenticationSuccessful) {  
 try {  
 *// Зчитуємо JSON з файлу* JsonArray jsonArray = new Gson().fromJson(new FileReader(fileName), JsonArray.class);  
 System.*out*.println("\nВхід в обліковий запис \uD83D\uDC64 :");  
 System.*out*.print("Введіть username: ");  
 String enteredUsername = scanner.nextLine();  
  
 System.*out*.print("Введіть password: ");  
 String enteredPassword = scanner.nextLine();  
  
 *// Проходимо по всіх об'єктах масиву* for (JsonElement jsonElement : jsonArray) {  
 JsonObject jsonObject = jsonElement.getAsJsonObject();  
  
 *// Отримуємо інформацію з JSON* String storedUsername = jsonObject.get("username").getAsString();  
 String storedPassword = jsonObject.get("password").getAsString();  
  
 *// Перевірка введених даних* if (enteredUsername.equals(storedUsername) && enteredPassword.equals(storedPassword)) {  
 System.*out*.println("Вхід успішний");  
  
 MainMenu mainApp = new MainMenu();  
 mainApp.runProgram(scanner);  
  
 authenticationSuccessful = true;  
 break; *// Вихід із циклу, оскільки аутентифікація вдалася* }  
 }  
 *// Якщо цей код викликався, то аутентифікація не вдалася* System.*out*.println("Помилка входу. Неправильне ім'я користувача або пароль");  
 } catch (JsonParseException | IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

Registration.java

package com.vintonyak.TaxiCompany.util;  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.reflect.TypeToken;  
import com.vintonyak.TaxiCompany.persistence.entity.impl.User;  
import com.vintonyak.TaxiCompany.persistence.exception.EntityArgumentException;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.lang.reflect.Type;  
import java.time.LocalDate;  
import java.time.format.DateTimeParseException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
import java.util.UUID;  
  
*/\*\*  
 \* Utility class for user registration and related operations.  
 \*/*public class Registration {  
  
 */\*\*  
 \* Reads user data from a JSON file and returns a list of User objects.  
 \*  
 \* @param filePath The path to the JSON file containing user data.  
 \* @return A list of User objects read from the file.  
 \*/* public static List<User> readUsersFromFile(String filePath) {  
 List<User> userList = new ArrayList<>();  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  
 *// Створюємо Gson з можливістю десеріалізації LocalDate* Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateDeserializer())  
 .create();  
  
 *// Визначаємо тип для десеріалізації списку користувачів* Type userListType = new TypeToken<List<User>>() {  
 }.getType();  
  
 *// Читаємо вміст файлу та десеріалізуємо його у список користувачів* userList = gson.fromJson(reader, userListType);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 return userList;  
 }  
  
 */\*\*  
 \* Writes a list of User objects to a JSON file.  
 \*  
 \* @param users The list of User objects to be written to the file.  
 \* @param filePath The path to the JSON file where user data will be stored.  
 \*/* private static void writeUsersToFile(List<User> users, String filePath) {  
 try (FileWriter writer = new FileWriter(filePath)) {  
 *// Створюємо Gson з красивим виведенням* Gson gson = new GsonBuilder()  
 .registerTypeAdapter(LocalDate.class, new LocalDateSerializer())  
 .setPrettyPrinting().create();  
  
 *// Перетворюємо список користувачів в JSON та записуємо у файл* gson.toJson(users, writer);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Performs user registration, taking input from the user and storing the information in a JSON file.  
 \*  
 \* @param filePath The path to the JSON file where user data will be stored.  
 \*/* public void registration(String filePath) {  
 Scanner scanner = new Scanner(System.*in*);  
 List<User> users = *readUsersFromFile*(filePath);  
  
 try {  
 *// Введення даних користувача* UUID userId = UUID.*randomUUID*();  
 System.*out*.println("\nРеєстрація облікового запису+\uD83D\uDC64 :");  
 System.*out*.print("Напишіть username (name.name): ");  
 String username = scanner.nextLine();  
 System.*out*.print("Напишіть password (не меньше 8 символів): ");  
 String password = scanner.nextLine();  
 System.*out*.print("Напишіть email (name@gmail.com): ");  
 String email = scanner.nextLine();  
 System.*out*.print("Напишіть birthday (YYYY-MM-DD): ");  
 LocalDate birthday = LocalDate.*parse*(scanner.nextLine());  
  
 if (userAlreadyExists(users, email, username)) {  
 System.*out*.println("Помилка: Користувач з такою ж адресою електронної пошти або ім'ям користувача вже існує");  
 registration(filePath);  
 }  
  
 *// Створення об'єкта користувача та додавання його до списку* User user = new User(userId, password, email, birthday, username);  
 users.add(user);  
  
 *// Перезапис файлу JSON з оновленим списком користувачів  
 writeUsersToFile*(users, filePath);  
  
 System.*out*.println("Реєстрація успішна");  
  
 *// Автоматично викликаємо метод авторизації* Authorisation.*authorisation*(filePath); *// Передайте шлях до файла, де зберігаються дані користувачів* } catch (EntityArgumentException e) {  
 *// Обробка помилок, якщо вони виникли під час створення користувача* for (String error : e.getErrors()) {  
 System.*out*.println("Помилка:" + error);  
 }  
 } catch (DateTimeParseException e) {  
 *// Обробка помилок, якщо дата введена невірно* System.*out*.println(  
 "Помилка: Неправильний формат дати. Будь ласка, введіть дату у форматі YYYY-MM-DD");  
 } finally {  
 scanner.close();  
 }  
 }  
  
 */\*\*  
 \* Checks if a user with the given email or username already exists in the list of users.  
 \*  
 \* @param users The list of User objects to check against.  
 \* @param email The email to check for duplication.  
 \* @param username The username to check for duplication.  
 \* @return true if a user with the given email or username already exists, false otherwise.  
 \*/* private boolean userAlreadyExists(List<User> users, String email, String username) {  
 for (User user : users) {  
 if (user.getEmail().equals(email) || user.getUsername().equals(username)) {  
 return true;  
 }  
 }  
 return false;  
 }  
}

Search.java

package com.vintonyak.TaxiCompany.util;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* The Search class provides methods for searching for a specified term in one or more files.  
 \*/*public class Search {  
  
 */\*\*  
 \* Searches for the specified term in the given files and prints the matching lines.  
 \*  
 \* @param searchTerm The term to search for.  
 \* @param filePaths The paths of the files to search in.  
 \*/* public static void searchInFiles(String searchTerm, String... filePaths) {  
 for (String filePath : filePaths) {  
 List<String> matchingLines = *searchInFile*(searchTerm, filePath);  
 if (!matchingLines.isEmpty()) {  
 System.*out*.println("\nРезультати пошуку у файлі: \uD83D\uDD0E" + filePath);  
 for (String line : matchingLines) {  
 System.*out*.println(line);  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Searches for the specified term in the given file and returns the matching lines.  
 \*  
 \* @param searchTerm The term to search for.  
 \* @param filePath The path of the file to search in.  
 \* @return A list of lines containing the specified term.  
 \*/* private static List<String> searchInFile(String searchTerm, String filePath) {  
 List<String> matchingLines = new ArrayList<>();  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 if (line.contains(searchTerm)) {  
 matchingLines.add(line);  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 return matchingLines;  
 }  
}

Info.java

package com.vintonyak.TaxiCompany.persistence.entity.impl;  
  
import com.vintonyak.TaxiCompany.persistence.entity.ErrorTemplates;  
import com.vintonyak.TaxiCompany.persistence.exception.EntityArgumentException;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class Info {  
  
 private String event;  
 private LocalDate date;  
 private String info;  
 private final List<String> errors = new ArrayList<>();  
  
 public Info(String event, LocalDate date, String info) {  
 setEvent(event);  
 setDate(date);  
 setInfo(info);  
 }  
  
 public String getEvent() {  
 return event;  
 }  
 public void setEvent(String event) {  
 this.event = event;  
 }  
  
 public void setDate(LocalDate date) {  
 this.date = date;  
 }  
  
 public void setInfo(String info) {  
 final String templateName = "події";  
  
 if (info.isBlank()) {  
 errors.add(ErrorTemplates.*REQUIRED*.getTemplate().formatted(templateName));  
 }  
 if (info.length() < 4) {  
 errors.add(ErrorTemplates.*MIN\_LENGTH*.getTemplate().formatted(templateName, 4));  
 }  
 if (info.length() > 24) {  
 errors.add(ErrorTemplates.*MAX\_LENGTH*.getTemplate().formatted(templateName, 24));  
 }  
  
 if (!this.errors.isEmpty()) {  
 throw new EntityArgumentException(errors);  
 }  
  
 this.info = info;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) {  
 return true;  
 }  
 if (o == null || getClass() != o.getClass()) {  
 return false;  
 }  
 Info infoObj = (Info) o;  
 return Objects.*equals*(event, infoObj.event) &&  
 Objects.*equals*(date, infoObj.date) &&  
 Objects.*equals*(info, infoObj.info);  
 }  
  
 @Override  
 public String toString() {  
 return "Info{" +  
 "event='" + event + '\'' +  
 ", date=" + date +  
 ", info='" + info + '\'' +  
 '}';  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(event, date, info);  
 }  
}

LocalDateDeserializer.java

package com.krnelx.Japaneseguide.util;  
  
import com.google.gson.JsonDeserializationContext;  
import com.google.gson.JsonDeserializer;  
import com.google.gson.JsonElement;  
import com.google.gson.JsonParseException;  
  
import java.lang.reflect.Type;  
import java.time.LocalDate;  
  
*/\*\*  
 \* JsonDeserializer implementation for deserializing JSON elements to LocalDate objects.  
 \*/*public class LocalDateDeserializer implements JsonDeserializer<LocalDate> {  
  
 */\*\*  
 \* Deserializes a JsonElement to a LocalDate object.  
 \*  
 \* @param json The JsonElement to be deserialized.  
 \* @param typeOfT The type of the object to be deserialized.  
 \* @param context The context for deserialization.  
 \* @return A LocalDate object representing the deserialized date.  
 \* @throws JsonParseException if the JsonElement cannot be deserialized to a LocalDate.  
 \*/* @Override  
 public LocalDate deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context)  
 throws JsonParseException {  
 return LocalDate.*parse*(json.getAsJsonPrimitive().getAsString());  
 }  
}

LocalDateSerializer.java

package com.krnelx.Japaneseguide.util;  
  
import com.google.gson.JsonElement;  
import com.google.gson.JsonPrimitive;  
import com.google.gson.JsonSerializationContext;  
import com.google.gson.JsonSerializer;  
  
import java.lang.reflect.Type;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
  
*/\*\*  
 \* JsonSerializer implementation for serializing LocalDate objects to JSON format.  
 \*/*public class LocalDateSerializer implements JsonSerializer<LocalDate> {  
  
 */\*\*  
 \* DateTimeFormatter for formatting LocalDate objects to the specified pattern.  
 \*/* private static final DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("yyyy-MM-dd");  
  
 */\*\*  
 \* Serializes a LocalDate object to a JsonElement with the specified date format.  
 \*  
 \* @param localDate The LocalDate object to be serialized.  
 \* @param type The type of the object to be serialized.  
 \* @param jsonSerializationContext The context for serialization.  
 \* @return A JsonElement representing the serialized LocalDate.  
 \*/* @Override  
 public JsonElement serialize(LocalDate localDate, Type type,  
 JsonSerializationContext jsonSerializationContext) {  
 return new JsonPrimitive(*formatter*.format(localDate));  
 }  
}