НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО:
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Веб-застосунок для предиктивного аналізу даних дорожньо-транспортних пригод»

Виконав:
студент IV курсу, групи ТІ-91
Березовський Іван Микитович _____

Керівник:
Асистент
Швайко Валерій Григорович _____

Рецензент: _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Рівень вищої освіти перший (бакалаврський)

спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)
”___”_____2023р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Березовському Івану Микитовичу__
(прізвище, ім’я, по батькові)

1. Тема роботи _Веб-застосунок для предиктивного аналізу даних дорожньо-транспортних пригод_____
_____

керівник роботи_____Швайко Валерій Григорович, асистент__
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ” ” червня 2023р. № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _мова програмування Java, фреймворки Spring Boot, Spring Data JPA, Vaadin, бібліотека Weka, середовище розробки IntelliJ IDEA Community_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Провести аналіз аналогічних веб-застосунків з предиктивного аналізу дорожньо-транспортних пригод, спроектувати структуру майбутнього датасету, вибрати цільові змінні та алгоритм класифікації. Розробити опис архітектури бази даних та модулів веб-застосунку, а саме модулю машинного навчання, імпорту та експорту, роботи з базою даних, безпеки та інтерфейсу. Надати інструкцію до розгортання та використання системи._____

5. Перелік ілюстративного матеріалу
Існуючі програмні засоби, Засоби розробки, Діаграма компонентів, Діаграма прецедентів, Архітектура бази даних, Вбудований набір даних, Сторінки реєстрації, логіну, перегляду, додавання, редагування та завантаження датасетів, Сторінки передбачень та візуалізації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "_10_" __вересня__2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	25.05.2023	виконано
2.	Дослідження предметної області	10.09.2022 – 01.02.2023	виконано
3.	Постановка вимог до проектування системи	02.02.2023 – 20.03.2023	виконано
4.	Дослідження існуючих рішень	21.03.2023 – 01.04.2023	виконано
5.	Розробка програмного продукту	02.04.2023 – 10.05.2023	виконано
6.	Тестування	11.05.2023 – 13.05.2023	виконано
7.	Захист програмного продукту	19.05.2023	виконано
8.	Оформлення дипломної роботи	13.05.2023- 30.05.2023	виконано
9.	Передзахист	06.06.2023	виконано
10.	Захист	19.06.2023	виконано

Студент _____ _____Березовський І.М._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Швайко В. Г._____
(підпис) (прізвище та ініціали,)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 60 сторінок, 33 рисунки, 2 формули, 1 додаток та 16 бібліотечних найменувань.

Метою роботи є надання статистичної інформації і аналітичних висновків для індивідуальних користувачів та компаній, таких як таксі або каршерінгові компанії, страхові, виробники авто, власники декількох автівок та інші.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано та обрано класифікаційний алгоритм машинного навчання і змінні для датасету;

- проаналізовано схожі додатки та визначено додатковий функціонал веб-застосунку.

Розроблено модель здатну:

- передбачити ступень тяжкості дорожньо-транспортної пригоди;

- передбачити кількість постраждалих.

Розроблено веб-застосунок, який надає користувачеві графічний інтерфейс для роботи з наборами даних та аваріями, отримання передбачення, імпорту дорожньо-транспортних пригод та експорту передбачень, переглядання наборів даних у вигляді графіку.

Практичне значення одержаних результатів полягає в отриманні передбачень ступеню тяжкості аварії та кількості потерпілих з натренованої моделі. Зроблено огляд існуючих систем та алгоритмів класифікації. Розроблено архітектуру веб-застосунку. Реалізовано інтерфейс для демонстрації роботи моделі.

Запропоновані методи дають можливість компаніям отримувати передбачення на своїх наборах даних, завдяки чому вони можуть редагувати ціни на свої послуги з урахуванням ризиків.

Ключові слова: класифікаційні алгоритми, дорожньо-транспортні пригоди, машинне навчання, наївний Байєс, архітектура бази даних.

ABSTRACT

Structure and scope of the thesis. The work contains 60 pages, 33 figures, 2 formulas, 1 appendix and 16 bibliographic titles.

The purpose of the work is to provide statistical information and analytical conclusions for individual users and companies, such as taxi or car sharing companies, insurance companies, car manufacturers, owners of several cars and others.

To achieve the set goal, the following tasks were completed:

- the machine learning classification algorithm and variables for the dataset were analyzed and selected;

- similar applications were analyzed and additional functionality of the web application was determined.

The model is capable of:

- predicting the degree of severity of a traffic accident;

- predicting the number of victims.

The web application that has been developed provides the user with a graphical interface to work with datasets and accidents, obtain predictions, import traffic accidents and export predictions, view datasets as graphs.

The practical significance of the obtained results lies in obtaining predictions of the severity of the accident and the number of victims from the trained model. An overview of existing classification systems and algorithms was made. The architecture of the web application has been developed. An interface for demonstrating the model's operation has been implemented.

The proposed methods enable companies to obtain predictions from their datasets, allowing them to adjust the prices of their services based on risk.

Keywords: classification algorithms, traffic accidents, machine learning, naive Bayes, database architecture.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

IDE — Integrated Development System (Інтегрована система розробки)

JPA — Java Persistence API

СУБД — Система управління базами даних

SQL — Structured Query Language

ВСТУП

Впродовж останнього сторіччя автомобіль став доступним та швидким засобом переміщення. Потреба в громадському або персональному транспорті є спільною для мешканців як невеликого села, так і шумного мегаполісу. Водночас подібна зручність має свої проблеми. Так, станом на 2022-й рік дорожньо-транспортні пригоди є найпоширенішою причиною загибелі молоді від 5 до 29 років. За даними всесвітньої організації з охорони здоров'я, кожного року у ДТП помирає близько 1.3 мільйони людей та від 20 до 50 мільйонів людей отримують травми різних ступенів тяжкості. Найбільшої небезпеці на дорозі себе наражають пішоходи, велосипедисти та мотоциклісти [1].

Незважаючи на те, що статистично кількість дорожньо-транспортних пригод в Україні зменшилась на 24% за 2022 рік, причини такого спаду не є оптимістичними, оскільки підрахунки велись без урахування тимчасово окупованих територій. Іншою причиною таких змін в статистиці стало загальне зменшення дорожньо-транспортного руху в деяких містах [2]. Таким чином, відкритим залишається питання покращення ситуації на дорогах.

Метою даної дипломної роботи є створення веб-застосунку для предиктивного аналізу даних дорожньо-транспортних пригод. Програма використовує класифікаційний алгоритм наївного Байєса для передбачення ступеню тяжкості аварії та визначення кількості постраждалих. Предиктивний аналіз проводиться на 18-ти атрибутах, таких як причина та тип зіткнення, стан та тип дороги, погодні умови, тощо. Користувач може створити передбачення на вбудованому наборі даних, або ж створити власний. Заповнення може відбуватись через інтерфейс програми, або шляхом завантаження файлу у форматі .csv. Для тренування набору даних користувач має натиснути відповідну кнопку інтерфейсу, після чого буде створена, натренована та збережена модель. Дана класифікаційна модель буде завантажена з бази даних щойно користувач створить нове передбачення.

Окрім основного функціоналу, в веб-застосунку передбачена можливість переглядання графіків для кращого аналізу залежності ступеню тяжкості аварії від певного змінної набору даних. Для генерації графіку користувач повинен обрати натренований набір даних та бажану змінну. Також користувач може завантажити список передбачень для обраного набору даних у форматі .csv.

Слід зазначити що веб-застосунок є доступним лише для авторизованих користувачів, оскільки набори даних можуть бути чутливою інформацією, до якої повинен мати доступ лише власник. У процесі реєстрації система має отримати унікальне ім’я користувача, електронну адресу, пароль та його підтвердження. Пароль зберігається зашифрованим.

Користувачами даного веб-застосунку можуть бути каршерінгові компанії, таксі, страхові компанії та виробники автомобілів. За допомогою наданих системою передбачень компанії зможуть оцінити ризики та з їх урахуванням сформувати ціну на свої послуги. Також можливе застосування системи власником декількох автомобілів, але у такому випадку створення нового повноцінного набору даних буде складнішим.

1 ЗАДАЧА ПРЕДИКТИВНОГО АНАЛІЗУ ДАНИХ ДОРОЖНЬО-ТРАНСПОРТНИХ ПРИГОД

Метою роботи є надання аналітичного висновку щодо ступеню тяжкості та кількості постраждалих у дорожньо-транспортній пригоді на основі заданих атрибутів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Дослідити алгоритми класифікації.
2. Дослідити та обрати основні атрибути, що можуть впливати на тяжкість ДТП.
3. Обрати та використати класифікаційний алгоритм в програмі.
4. Розробити структуру бази даних.
5. Створити веб-застосунок для надання програмі набору даних для навчання.
6. Протестувати веб-застосунок.

Користувачами даного веб-застосунку будуть каршерінгові компанії, таксі, страхові, виробники транспортних засобів, власники багатьох автомобілів та інші. Нижче наведено список функцій програми.

1. Можливість переглядати, додавати та видаляти наявні набори даних.
2. Можливість переглядати, додавати, редагувати та видаляти дорожньо-транспортні пригоди у кожному наборі даних.
3. Можливість отримувати передбачення ступеню тяжкості аварії та кількості постраждалих для певного набору даних.
4. Можливість імпорту та експорту даних у форматі .csv.
5. Можливість переглядання графіків для аналізу залежності ступеню тяжкості аварії або кількості постраждалих від певного атрибуту.
6. Можливість авторизуватись в системі.

На вході програми – набір даних у форматі .csv або введений безпосередньо в інтерфейс веб-застосунку. На виході – передбачення у форматі .csv.

Висновки до розділу 1

У даному розділі було поставлено мету програмного продукту, виділено ключові задачі які він має виконувати та основний функціонал веб-застосунку. Визначено можливих користувачів, описано вхідні та вихідні дані програми.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ АНАЛІЗУ ДАНИХ ДОРОЖНЬО-ТРАНСПОРТНИХ ПРИГОД

2.1 Аналіз існуючих систем

Незважаючи на велику кількість спроектованих моделей машинного навчання на тему передбачення тяжкості дорожньо-транспортних пригод, реалізація у відкритому доступі всього одна.

Так, для визначення тяжкості аварії наявний застосунок, що використовує випадкові ліси. Застосунок має назву road-accidents-prediction-and-classification та основним його фунціоналом є сторінка, що на основі вбудованого набору даних пропонує отримати передбачення (рисунок 2.1) [3].

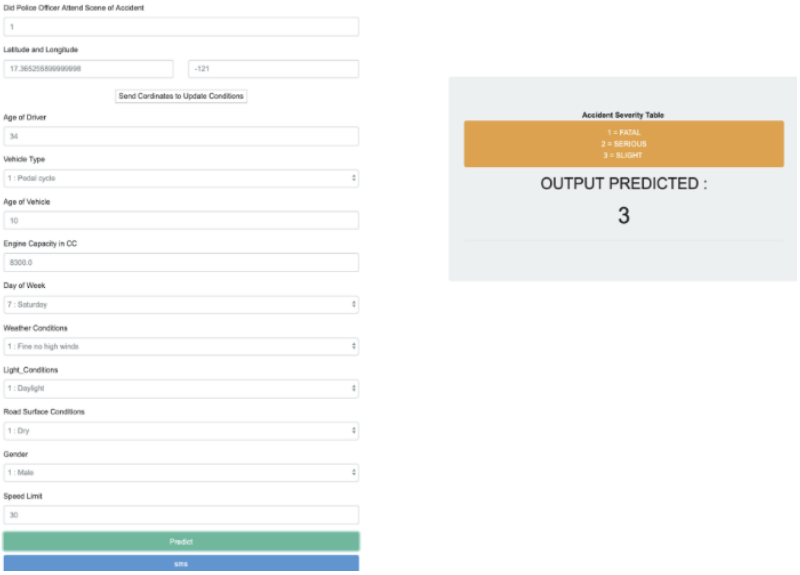


Рисунок 2.1 – Отримання передбачення ступеню тяжкості

Оскільки застосунок надає можливість передбачити ступінь тяжкості дорожньо-транспортної пригоди для конкретної локації, він також може відправляти SMS до користувача або поліції з даними про передбачення. Це зроблено для того щоб запобігти появі смертельної аварії.

Іншим корисним функціоналом сайту є карта з зображенням передбачень (рисунок 2.2).

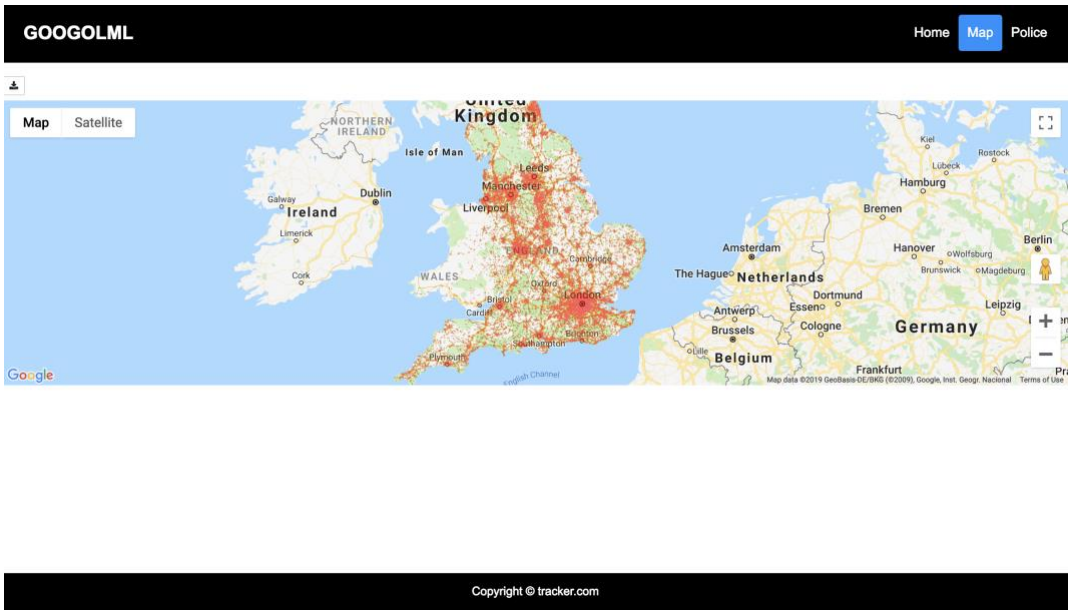


Рисунок 2.2 – Карта можливих аварій

Чим більша ступінь тяжкості аварії для певної локації, тим більша та яскравіша точка на карті. Таким чином користувач може проаналізувати траси з потенціально найсерйознішими аваріями. Користувач також може завантажити дані з карти, натиснувши на відповідну кнопку у лівому верхньому кутку сторінки.

З переваг даного веб-застосунку можна виділити наступні:

1. Можливість передбачення для конкретної широти та довготи.

2. Швидке надання даних до відповідних органів.

3. Графік для кращого розуміння ступеню тяжкості аварії.

Однак дана система має свої недоліки. Виділимо основні:

1. Невелика точність на обмеженому наборі даних. Для передбачення за широтою і довготою повинен бути великий та збалансований за місцевістю датасет.

2. Неможливість додати власний набір даних, користувач обов’язково повинен довіряти якості вбудованого.

3. Відсутність авторизації та реєстрації, що ставить під сумнів захищеність чутливих даних в передбаченні, оскільки всі вони додаються до публічної карти.

4. Єдиним результатом є ступень тяжкості аварії, який є неповним без додаткових показників, наприклад, без кількості постраждалих.

Таким чином, існує потреба у створенні захищеного застосунку, який дозволяє робити передбачення як на вбудованому наборі даних, так і на власному.

2.2 Фактори впливу на тяжкість дорожньо-транспортної пригоди

Дорожньо-транспортні пригоди можуть бути спричинені низкою причин. Згідно комплексного дослідження, проведеного Зої Кристофороу та Симоном Кохеном, на ступень тяжкості аварії мають найбільший вплив такі фактори як п’яне водіння, старіння, наявність лобового зіткнення, наявність зіткнення з мотоциклістом або навпаки, зі значно більшим транспортним засобом, погане освітлення, вертикальна чи горизонтальна кривизна дороги, тип дороги, перевищення швидкості [4].

Водночас інші фактори, хоча й мають прямий вплив, не є стабільними та не можуть надати точних, незалежних від конкретного датасету прогнозів. До таких факторів можна віднести характеристики як транспортного засобу, так і водія, погодні та дорожні умови, наявність невеликих дефектів в дорозі.

Також на тяжкість дорожньо-транспортної пригоди може впливати час поїздки. Так, інше дослідження свідчить що ключовими характеристиками тяжких або смертельних аварій є їзда вранці або вночі при поганому освітленні, між п’ятницею та суботою, на основних дорогах міст та під час спроби обгону іншого транспортного засобу [5].

Підсумовуючи наведені вище дослідження та опираючись на модифіковані дані з датасету зі схожою предметною областю [6], складемо список факторів, що будуть використані як змінні для навчання класифікаційного алгоритму:

1. Час дорожньо-транспортної пригоди, у 24-годинному форматі.
2. День тижня.
3. Вікова група водія.
4. Гендер водія.

5. Досвід водія.

6. Тип транспортного засобу, що спричинив аварію.

7. Наявність попереднього дефекту у транспортного засобу.

8. Вирівнювання дороги

9. Тип та стан дороги.

10. Дорожні умови.

11. Умови освітлення.

12. Погодні умови.

13. Тип зіткнення.

14. Кількість транспортних засобів в аварії.

15. Кількість постраждалих.

16. Тип руху транспортного засобу перед зіткненням.

17. Причина зіткнення.

18. Ступінь тяжкості аварії.

Слід зазначити, що з наведених атрибутів цільовими змінними аналізу є два: кількість потерпілих та тяжкість аварії. До списку атрибутів була також додана невелика кількість «нестабільних» факторів впливу.

2.3 Класифікаційні алгоритми машинного навчання

Навчання з учителем, або кероване навчання, є типом машинного навчання в якому модель тренується на наборі підібраних даних з відомим результатом. Дані з результатом діють у якості «вчителя». Кероване навчання може бути основою для таких задач як фільтрування спаму, класифікація рисунків, надання рекомендацій для перегляду, тощо [7].

Існує два типи алгоритмів для навчання з учителем. Перший – регресія, використовується у випадку коли між вхідними даними та результатом є безпосередній зв’язок та змінна результату є безперервною. Тобто, результат не має

строго визначеної невеликої кількості значень (класів). Прикладами задач для цього типу алгоритмів є прогноз погоди, трендів продаж, тощо.

Другий тип алгоритмів – класифікація, застосовується у випадку, коли змінна результату може бути поділена на категорії. Класифікаційні алгоритми можуть визначити наявність знайомих об’єктів на фото, класифікувати обличчя людини на фотографії, визначити чи є надіслане повідомлення спамом, тощо.

Результатом роботи класифікаційного алгоритму у даній системі мають бути ступінь тяжкості та кількість постраждалих. На виході першої моделі має бути одне з трьох значень: легке, тяжке або смертельне поранення. На виході другої – кількість постраждалих від 0 до 10. Оскільки змінна результату в обох випадках не є безперервною, а є обмеженою чіткими значеннями, то обидві задачі є задачами класифікації.

Слід зазначити, що хоча основний вбудований датасет веб-застосунку матиме великий розмір (10000+ даних), користувач не завжди зможе надати настільки ж великий датасет. Це означає що основна вимога до алгоритму класифікації – вміння видавати відносно точні результати на невеликих наборах даних. Роздивимось найбільш поширені алгоритми класифікації та визначимо, чи підходять вони для даної роботи.

Дерево вибору містить вітки у вигляді ієрархії та кожна вітка є if-else оператором. Вітки поділяють датасет за найважливішими атрибутами. Листи дерева при цьому є можливими значеннями класифікації. Лист, до якого дійшов алгоритм, є результатом його роботи.

Випадковий ліс є сукупністю (ансамблем) дерев вибору. Кожне дерево тренується на частині датасету та видає результат. Результатом роботи алгоритму при цьому є значення, що набрало найбільше «голосів» від дерев вибору. Даний алгоритм є більш точним, але й складнішим, потребуючим більше даних для тренування. Це робить його менш привабливим для даної роботи.

Метод k-найближчого сусіда намагається передбачити значення результату алгоритму шляхом обчислення дистанції між новими даними, та вже збереженими

в нейронній мережі точками натренованих даних. Якщо є декілька близьких точок, то значення вирішується «голосуванням» між цими точками.

Наївний Байєс є алгоритмом, що базується на теоремі Байєса. Слід зазначити, що дана теорема припускає, що змінні датасету не є залежними одна від одної. Хоча це припущення не завжди є правдивим, в структурі датасету даної роботи майже немає залежності однієї змінної від іншої (за виключенням двох змінних, що є результатом). Іншою перевагою алгоритму для цієї програми є швидкість та точність на невеликих наборах даних.

Отже, роздивимось теорему Байєса. Представимо множину описів аварії як X (x_1, x_2… x_n), тоді Y – множина класів результату. Для класифікації необхідно знайти такий клас результату, для якого ймовірність P(Y|X) буде максимальною. Знаходження даної ймовірності можна побачити у формулі (2.1).

$$P(Y|X) = P(X|Y) * P(Y) / P(X) \quad (2.1)$$

Ліва частина даної формули має назву апостеріорної ймовірності, тобто ймовірність того, що Y зумовлено певним X. Аналогічно і з P(X|Y) – це ймовірність того, що X зумовлено певним Y.

P(Y) в свою чергу – це апріорна ймовірність, тобто ймовірність виконання Y в незалежності від будь-яких чинників. P(X) є постійною для кожного Y, що означає, що знаменник не має значення для визначення максимального P(Y|X). Отже, формула може бути спрощена до числівника.

Для визначення P(Y) необхідно кількість подій класу Y поділити на загальну кількість подій. Наприклад, якщо у наборі даних 500 аварій, 400 з яких є легкими, то апріорна ймовірність виникнення легкої аварії складає 400/500 = 0.8.

Беручи до уваги незалежність змінних, складемо формулу 2.2 для знаходження P(X|Y).

$$P(X|Y) = P(x_1|Y) * P(x_2|Y) * \dots * P(x_n|Y). \quad (2.2)$$

Після обчислення формули 2.1 для кожного з класів, результатом роботи алгоритму буде максимальне значення апостеріорної ймовірності [8].

Отже, у якості класифікаційного алгоритму було обрано алгоритм наївного Байєса через простоту реалізації, можливість застосування теореми Байєса для змінних даного набору даних та достатню точність навіть на невеликих користувацьких датасетах.

Висновки до розділу 2

У даному розділі було проведено аналіз схожого за призначенням веб-застосунка, виділено його переваги та недоліки, обґрунтовано розробку даного програмного продукту. Було наведено основні фактори, що впливають на тяжкість дорожньо-транспортної пригоди, спроектовано структуру майбутнього набору даних, обрано класифікаційний алгоритм машинного навчання.

3 ЗАСОБИ РОЗРОБКИ СИСТЕМИ

3.1 Мова програмування Java

Java – об’єктно-орієнтована мова програмування високого рівня, основним застосуванням якої є написання серверної частини великих та складних веб-застосунків. Основними характеристиками даної мови є простота, безпека, надійність, розподіленість та динамічність. Також слід зазначити, що Java є багатопоточною, завдяки чому програми здатні обробляти велику кількість даних одночасно [9].

Ще одна особливість Java полягає в кросплатформності, тобто незалежності коду від пристрою, на якому він виконується. Це досягається шляхом виконання коду у віртуальній машині замість самої операційної системи. Повну архітектуру виконання .java коду можна побачити на рисунку 3.1.

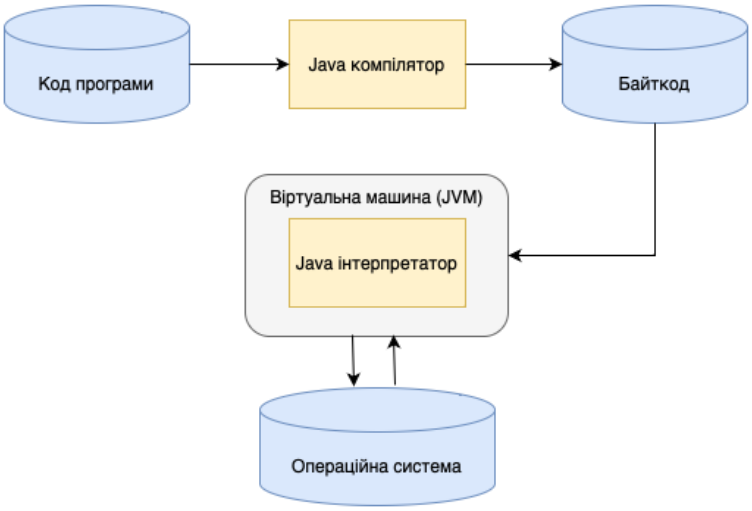


Рисунок 3.1 – Архітектура виконання Java коду

Основною причиною використання Java у даній роботі стала наявність численної спільноти розробників, а отже великого набору фреймворків та бібліотек для задач будь-якої складності. Так, в даному програмному продуктові були використані бібліотеки Java для створення сторінок сайту та реалізації алгоритму класифікації.

3.2 Середовище розробки IntelliJ IDEA

Існує велика кількість як середовищ розробки, так і текстових редакторів адаптованих під Java. Вибір IntelliJ IDEA Community був обумовлений швидкістю написання коду, відсутністю додаткових витрат та попереднім досвідом. На рисунку 3.2 зображено інтерфейс та роботу в IntelliJ IDEA Community.

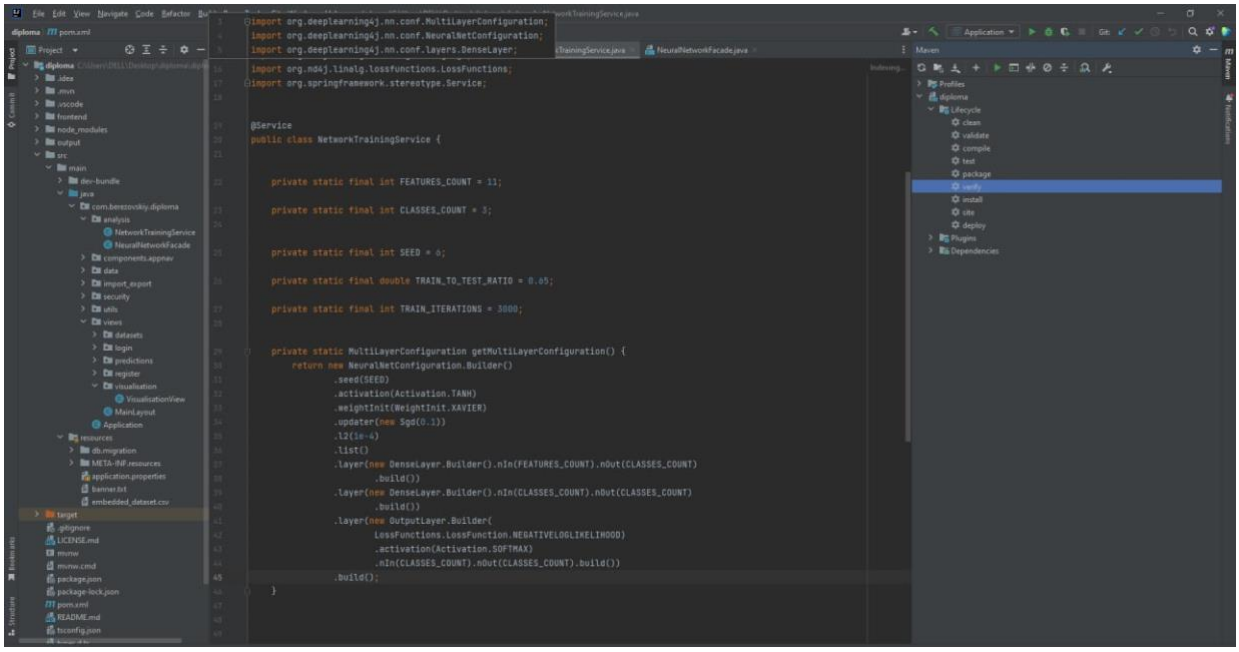


Рисунок 3.2 – Інтерфейс та робота в IntelliJ IDEA Community

Оскільки дане середовище розробки є безкоштовною версією з відкритим кодом, деякі можливості повної версії відсутні. Так, Community Edition не має вбудованої підтримки фронтенд мов програмування, не підтримує фреймворки екосистеми Spring, Java Enterprise Edition, мову структурованих запитів SQL та інші поширені технології для програмування серверної частини веб-застосунків. Також повна версія надає можливість працювати з базою даних не виходячи з середовища розробки [10].

Незважаючи на обмеженість обраної версії, вона має певні засоби для швидкої та комфортної розробки, а саме підтримку Java, Maven, Git та Docker. За необхідності можуть бути додані плагіни для інших технологій.

3.3 Фреймворки екосистеми Spring

Кожен написаний на Java веб-застосунок складається з великої кількості об’єктів. Частина цих об’єктів слугує як обгортка над даними, що отримуються з бази даних, файлової системи або ж з запиту користувача. Вони називаються моделлю та створюються кожен раз, коли програмі необхідно передавати та оброблювати дані. Друга частина об’єктів веб-застосунку складається з сервісів, що не мають стану та лише обробляють модель. Вони також можуть бути залежні від інших сервісів, що ускладнює їхнє створення та роботу з ними [11].

Основна задача Spring Framework – керування такими об’єктами (бінами). Після реєстрації в Spring, налаштований бін може бути отриманий з контейнера та вставлений в необхідне місце програми. Такий підхід дозволяє не шукати залежності та не створювати сервіс кожен раз, коли його необхідно викликати. Однак реєстрація та отримання бінів напряму через виклик ApplicationContext також є досить громіздким шляхом, що потребує налаштування та додавання бінів у контекст вручну.

Сучасні застосунки поєднують Spring Framework зі Spring Boot, що надає можливості для швидкого конфігурування та розгортання веб-застосунку. Опис бінів при цьому має бути налаштований за допомогою XML, java або анотацій [12]. При запуску вбудованого в Spring Boot серверу Tomcat відбувається створення та завантаження бінів в контейнер.

Окрім Spring Boot та Spring Framework веб-застосунок також використовує Spring Data JPA для взаємодії з реляційною базою даних. Даний фреймворк є Spring-абстракцією специфікації Java Persistence API. «Під капотом» даної абстракції за замовченням знаходиться Hibernate, що трансформує притаманну реляційній базі даних модель у об’єктно-орієнтовану [13]. Більш детальна архітектура знаходиться на рисунку 3.3.

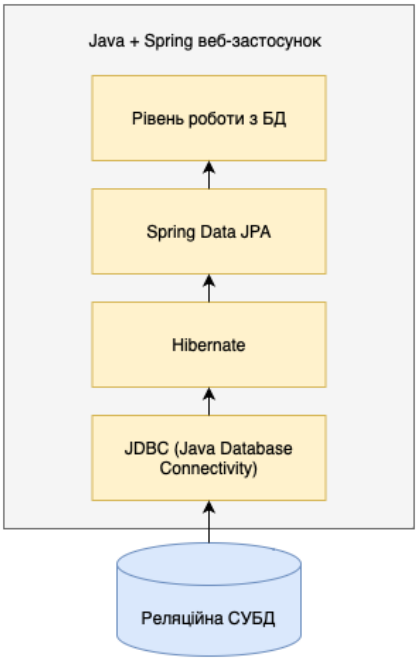


Рисунок 3.3 – Архітектура Spring Data JPA

Spring Data JPA надає інтерфейс JpaRepository, унаслідувавши який інтерфейс веб-застосунку отримує можливість звертатись до бази даних. Розробник може писати власні sql-запити, використовувати наявні у інтерфейсі методи або створювати нові. Spring Data JPA звільняє від необхідності трансформування об’єктів в строку таблиці, замість цього програміст може одразу перейти до написання логіки запитів. Запити можуть написані як за допомогою PostgreSQL чи JPQL, так і з використанням іменування методів інтерфейсу.

Ще одним фреймворком екосистеми Spring, що застосовується в роботі, є Spring Security. Дана технологія дозволяє реалізувати аутентифікацію та авторизацію в системі. Для аутентифікації (представлення користувача системі) Spring Security надає можливість логіну та реєстрації, а для авторизації (надання прав авторизованому користувачеві) використовує ролі. Слід зазначити, що оскільки в системі лише одна роль – звичайний користувач, то в системі наявна лише аутентифікація.

Як результат використання екосистеми фреймворків Spring ми маємо розгорнутий та захищений веб-застосунок з доступом до бази даних.

3.4 Фреймворк Vaadin

Vaadin є фреймворком з відкритим кодом що дозволяє програмувати фронтенд веб-застосунку використовуючи лише Java. Це дозволяє пришвидшити та здешевити процес розробки. Vaadin є раціональним вибором для невеликих веб-застосунків з нескладним дизайном.

Кожен елемент у даному фреймворку є компонентом. Це означає що кожен клас, що представляє сторінку за певною адресою, має наслідувати клас Component [14]. Також компонентами є додані на сторінку блоки, текст, кнопки, тощо. До кожного компоненту можуть бути додані стилі, а також функціонал в залежності від його типу. Так, для кнопки може бути задано текст, іконку та додано ComponentEventListener.

Під капотом фреймворку відбувається трансформація java коду у HTML, CSS та JavaScript. При розгортанні веб-застосунку відбувається завантаження або оновлення залежностей, які завантажуються в папку node_modules. Після цього програма розгортається на заданому в конфігурації порту та браузер автоматично відкривається на адресі за замовченням.

Хоча Vaadin може бути використаний без Spring Boot і вбудованого в нього серверу Tomcat, для цього необхідна велика кількість додаткової конфігурації. Завдяки доданої в проект залежності vaadin-spring-boot-starter, майже вся конфігурація відбувається автоматично, звільнивши розробника від зайвих дій.

Окрім ядра фреймворку Vaadin, у даній роботі також були застосовані Vaadin ApexCharts, що представляють собою набір графіків з відкритим кодом. Дана бібліотека написана на JavaScript, але адаптована і для інтерфейсу на Java. На рисунку 3.1 наведено використаний у програмі компонент.

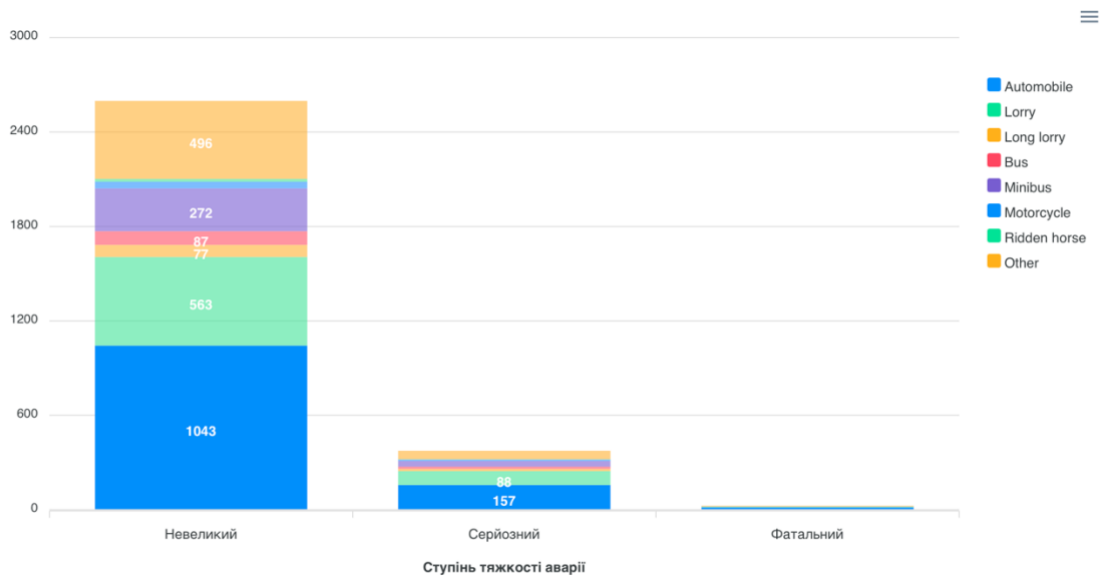


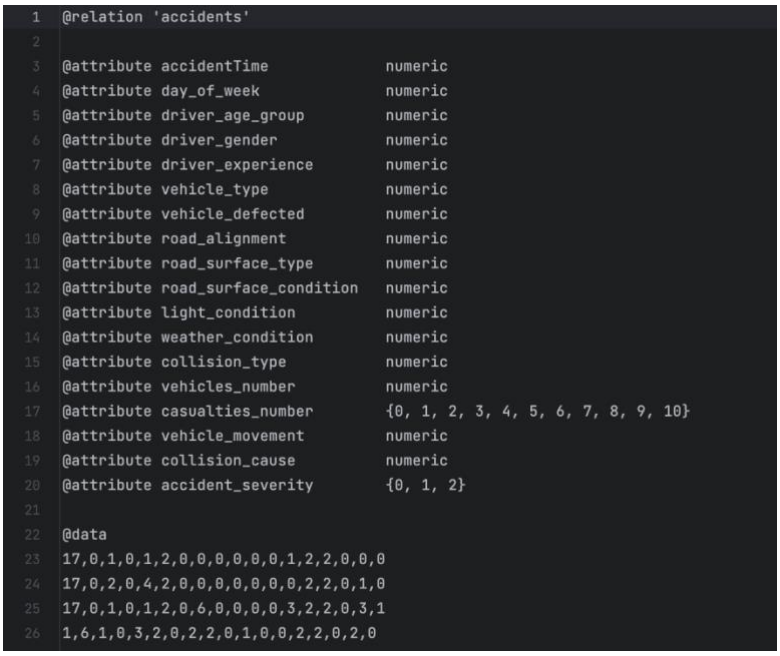
Рисунок 3.4 – Графік бібліотеки Vaadin ApexCharts

3.5 Бібліотека алгоритмів машинного навчання Weka

Weka є колекцією алгоритмів машинного навчання та інструментів попередньої обробки даних. За допомогою даної бібліотеки розробник може підготувати дані, вставити їх в класифікатор (модель) певного алгоритму та проаналізувати отриманий класифікатор за точністю та швидкістю. Weka представлена у вигляді як бібліотеки Java, так і графічного інтерфейсу.

У Weka наявні реалізації наступних алгоритмів: наївний Байєс, дерево вибору, випадковий ліс, таблиця вибору, лінійна регресія, тощо. Також у бібліотеці присутня підтримка нейронних мереж та глибокого навчання [15].

Датасет зазвичай завантажується з файлу формату .arff. Початок такого файлу зображено на рисунку 3.5. Файл логічно поділяється на три частини: назва датасету (@relation), назва та опис кожної змінної датасету (@attribute) та сам набір даних у форматі .csv (@data). Тип кожної змінної в розділі @data має співпадати з типом у розділі @attribute та мати аналогічний порядок.



```
@relation 'accidents'

@attribute accidentTime             numeric
@attribute day_of_week              numeric
@attribute driver_age_group         numeric
@attribute driver_gender            numeric
@attribute driver_experience        numeric
@attribute vehicle_type             numeric
@attribute vehicle_defected         numeric
@attribute road_alignment           numeric
@attribute road_surface_type        numeric
@attribute road_surface_condition   numeric
@attribute light_condition          numeric
@attribute weather_condition        numeric
@attribute collision_type           numeric
@attribute vehicles_number          numeric
@attribute casualties_number        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
@attribute vehicle_movement         numeric
@attribute collision_cause          numeric
@attribute accident_severity        {0, 1, 2}

@data
17,0,1,0,1,2,0,0,0,0,0,0,1,2,2,0,0,0
17,0,2,0,4,2,0,0,0,0,0,0,0,2,2,0,1,0
17,0,1,0,1,2,0,6,0,0,0,0,3,2,2,0,3,1
1,6,1,0,3,2,0,2,2,0,1,0,0,2,2,0,2,0
```

Рисунок 3.5 – Початок файлу формату .arff

Weka використовує клас Instances для представлення датасету у програмі. Відповідно Instance – одна строка з набору даних. Отриманий з ArffLoader об’єкт класу Instances передається до алгоритму класифікації, що є однією з реалізацій інтерфейсу Classifier. Для побудови моделі викликається метод buildClassifier.

Отримати результат на основі класифікатору можливо, викликавши метод classify з набором даних. На виході методу – цільова змінна у форматі double.

Для даного веб-застосунку важливим є також збереження класифікаторів у базі даних з їх подальшим отриманням. Для цього Weka надає допоміжний клас SerializationHelper, що читає та пише в потік даних.

3.6 Система керування базами даних PostgreSQL

PostgreSQL є об’єктно-реляційною системою управління базами даних з відкритим кодом. Наряду з такими СУБД як Oracle, MySQL, Microsoft SQL Server та інші, вона використовується в розробці складних корпоративних застосунків. PostgreSQL поєднує в собі великий набір функціоналу та типів, але при цьому не є повільнішою за аналоги [16].

На відміну від інших систем управління базами даних, PostgreSQL надає можливість користувачеві створювати власні типи даних, а не лише використовувати вбудовані. Слід зазначити, що й набір вбудованих типів є досить широким. Так, для даної роботи є суттєвим зберігання натренованих нейронних мереж в масивах байтів. У MySQL для цього необхідно було б переводити байти у строку та навпаки. У PostgreSQL необхідно лише зазначити тип поля як bytea, що є вбудованим масивом байтів.

PostgreSQL, на відміну від Oracle, не є комерційною базою даних. Це означає повну відсутність додаткових витрат, пов’язаних з придбанням ліцензій, незалежність продукту від одного вендору.

Оскільки IDE IntelliJ IDEA Community не має підтримки роботи з базою даних, для цього було використано pgAdmin 4 (рисунок 3.6).

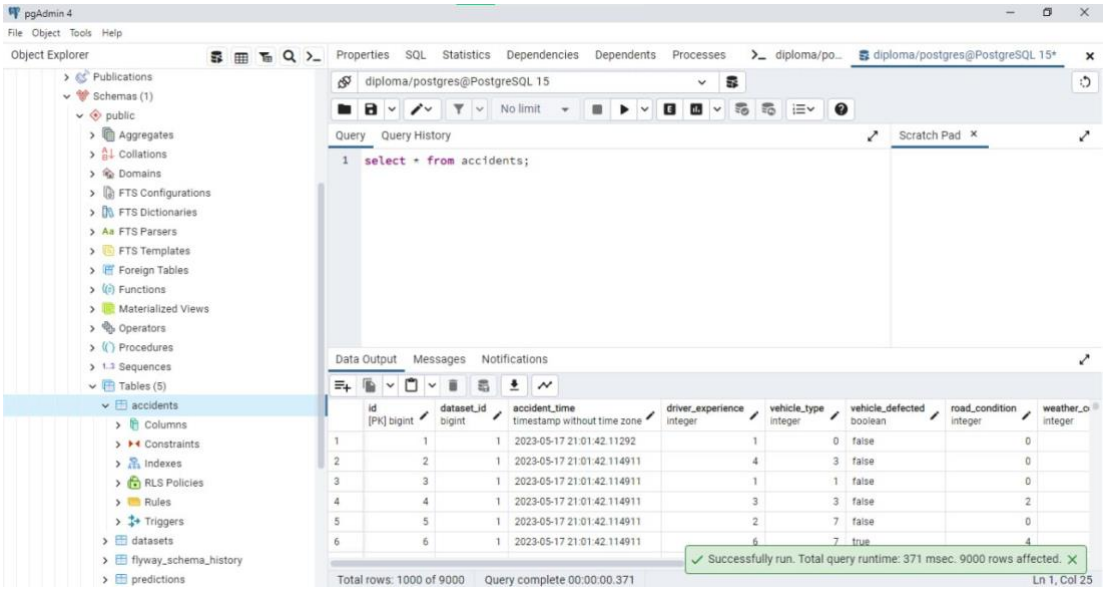


Рисунок 3.6 – Робота в pgAdmin 4.

3.7 Інші бібліотеки

Окрім наведених фреймворків та бібліотек, були використані такі допоміжні технології як Maven, Flyway та Lombok. Мета їх використання – полегшення та пришвидшення роботи розробника, а не пряме виконання задач.

Maven є інструментом автоматизації зборки, що зазвичай використовується з Java та супутніми технологіями. Дана технологія завантажує залежності з віддаленого репозиторію до локальної машини, на якій буде розгорнуто систему. Окрім керування залежностями, Maven збирає та тестує проект, створює .jar або .war файли, додає плагіни, тощо.

Flyway це система міграції бази даних з відкритим кодом. Дана технологія дозволяє легко змінювати структуру таблиць бази даних, додавати поля та функції.

Єдиною роботою розробника при цьому є написання міграцій – файлів формату .sql, що становлять скрипти всього функціоналу, що має бути доданий до бази даних. Також необхідно створити саму базу даних та написати конфігурацію для Flyway, вказавши строку підключення, користувача, пароль, базу даних та папку, де зберігаються міграції. Процес роботи з Flyway після підключення до бази даних можна побачити на рисунку 3.7.

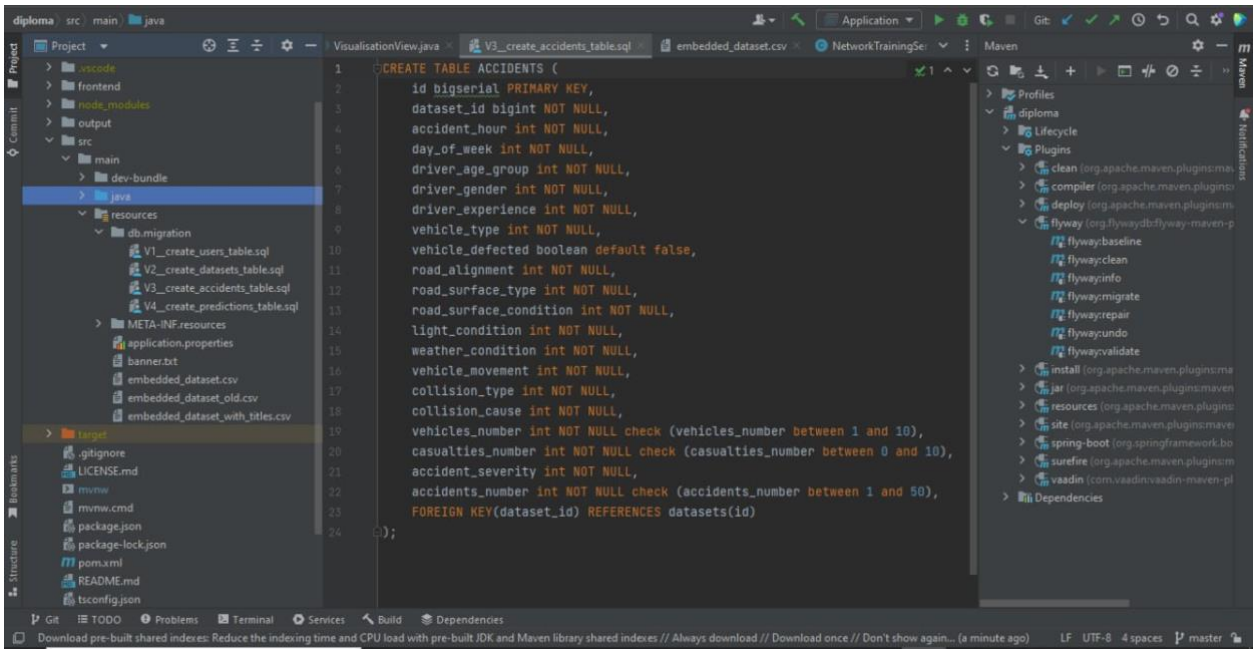


Рисунок 3.7 – Робота з Flyway

Так, для виконання міграції необхідно виконати команду mvn flyway:migrate. При наявності змін до структури бази даних розробник має додати до папки db.migrations скрипт з префіксом «V5__», після чого знов запустити команду mvn flyway:migrate. Старі скрипти при цьому не будуть виконані.

Для генерації коду в даному проекті було використано Lombok. За допомогою анотацій даної бібліотеки розробник може уникнути необхідності писати геттери, сеттери, конструктори та інші типові методи вручну. Lombok додає необхідні методи вже у .class файл, згенерований під час компіляції. Це не тільки дозволяє пришвидшити процес розробки, а й робить кодову базу значно меншою та зрозумілішою.

Типовий клас моделі, до якого додано Lombok, матиме анотації @Getter, @Setter, @NoArgsConstructor, @AllArgsContructor, @Builder, тощо. Слід зауважити, що використання @EqualsAndHashcode, @ToString та @Data повинно бути обмежене в JPA моделі. Дані анотації можуть автоматично завантажувати дані, що збільшує кількість запитів до бази.

Висновки до розділу 3

У цьому розділі детально описані використані у проекті технології, обґрунтовано вибір багатьох з них, наведено базові приклади використання як в сучасному світі розробки, так і в даній роботі. Окрім основних засобів розробки, а саме мови програмування Java, екосистеми фреймворків Spring, фреймворку для розробки інтерфейсу Vaadin, бібліотеки Weka, СУБД PostgreSQL, були використані інструменти для збільшення ефективності розробки. Прикладами таких інструментів є IntelliJ IDEA Community, pgAdmin 4, Maven та Flyway.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура системи додатку

Створений веб-застосунок логічно складається з п’яти модулів: аналізу даних, імпорту та експорту даних, роботи з базою даних, безпеки та відображення. Коротко опишемо кожен з них.

Модуль аналізу даних є ядром системи та містить в собі код що тренує алгоритм класифікації для обчислення тяжкості аварії та кількості постраждалих. Після того як модель натренована на наявних в датасеті даних, модуль може зробити передбачення. В якості класифікаційного алгоритму використано наївний Байєс, оскільки він здатен відпрацьовувати на невеликій кількості даних.

Алгоритм класифікації тренується окремо на кожному наборі даних за викликом користувача. Після тренування модель зберігається у базі даних, звідки витягується коли необхідно зробити передбачення.

Модуль імпорту та експорту виконує дві важливі для веб-застосунку задачі. Першою є перетворення файлів формату .csv в дані про дорожньо-транспортні пригоди програми. Друга задача полягає в отриманні передбачень програми у форматі .csv. Виконання обох задач є результатом взаємодії з модулем роботи з базою даних.

Модуль роботи з базою даних складається з рівня Repository та моделей. Рівень Repository є найнижчим рівнем веб-застосунку та майже повністю реалізований фреймворком Spring Data JPA. Розробникові необхідно лише прописати запити до бази даних або скористуватись вбудованими запитами. В модулі наявні чотири моделі зв’язані з відповідними таблицями бази даних (User, Dataset, Accident, Prediction) та велика кількість перерахувань, що є типами даних атрибутів нейронної мережі (AccidentSeverity, WeatherCondition, VehicleType, LightCondition, CollisionCause, тощо).

Модуль безпеки створено за допомогою фреймворку Spring Security. Даний модуль убезпечує сторінки веб-застосунку від неавторизованого доступу та додає дані про авторизованого користувача в програму в якості керованого Spring Framework об’єкта.

Модуль відображення є найбільшим модулем в системі та складається з восьмох сторінок (сторінка логіну, реєстрації, датасетів, передбачень, графіків, переглядання одного датасету, сторінок створення та завантаження датасету). Кожна сторінка є компонентом фреймворку Vaadin, який трансформує код з мови програмування Java на фронтенд мови (HTML, CSS, JavaScript та його фреймворки).

Нижче наведена діаграма компонентів веб-застосунку (рисунок 4.1).

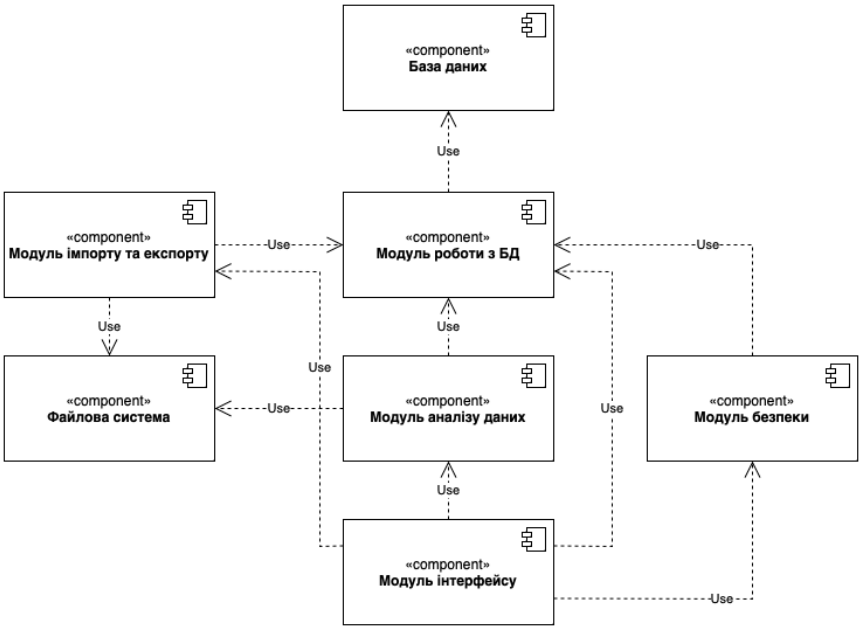


Рисунок 4.1 – Діаграма компонентів

Як показано на діаграмі, модуль відображення використовує інші чотири модулі для створення інтерфейсу веб-застосунку. Модуль роботи з БД є повною абстракцією над базою даних та використовується усіма іншими модулями. Файлова система є тимчасовим сховищем документів, таких як .csv та .arff файли передбачень та датасетів.

4.2 Модуль взаємодії з базою даних

Як було зазначено у попередньому пункті, модуль взаємодії з базою даних може бути поділений на дві частини: доменну модель та рівень репозиторію.

Доменна модель застосунку складається з чотирьох моделей та 14ти перерахувань. Один об’єкт класу моделі дорівнює одному записові в таблиці бази даних. Розглянемо кожен клас окремо.

Клас User є представленням в програмі таблиці users, що містить всіх користувачів системи. Об’єкт цього класу дорівнює одному зареєстрованому користувачеві. Даний клас не призначений для модифікації після створення, оскільки він лише містить дані для авторизації. Полями класу є id (унікальний ідентифікатор користувача, генерується базою даних), username (унікальне ім’я користувача), email (електронна адреса користувача) та password (пароль, зашифрований за допомогою криптографічної функції формування ключа bcrypt).

Кожен користувач може мати декілька наборів даних. Набори даних зберігаються в таблиці datasets. Об’єкт відповідного класу Dataset є представленням набору даних у програмі. Датасет може бути створено або користувачем, або автоматично після реєстрації, для вбудованого набору даних. Полями класу є id, title (назва набору даних), description (додатковий опис набору), user (об’єкт класу User, до якого належить набір даних), isTrainedOn (чи присутня модель класифікаційного алгоритму, натренована на даному наборі даних), accidentSeverityPredictionClassifier (натренована модель, що передбачає тяжкість аварії), casualtiesNumberPredictionClassifier (натренована модель, що передбачає кількість постраждалих).

Один набір даних може мати велику кількість дорожньо-транспортних пригод та передбачень. Об’єкт класу Accident є представленням однієї строки таблиці accidents та є аварією, яку надає програмі користувач для тренування алгоритму. Клас має наступні поля: id, accidentHour (година, в яку трапилась аварія), dayOfWeek (один з семи днів тижня), driverAgeGroup (вікова група водія),

driverGender (гендер водія), driverExperience (приблизна кількість років досвіду), vehicleType (тип транспортного засобу), vehicleDefected (наявність серйозного пошкодження транспортного засобу), roadAlignment (пряма чи крива лінія дороги), roadSurfaceType (тип дорожнього покриття), roadSurfaceCondition (дорожні умови), lightCondition (освітлення дороги), weatherCondition (погодні умови), collisionType (з чим або ким відбулось зіткнення), vehiclesNumber (кількість транспортних засобів, задіяних в аварії), casualtiesNumber (кількість постраждалих), vehicleMovement (як рухався транспортний засіб, що спричинив аварію), collisionCause (причина зіткнення), accidentSeverity (ступінь тяжкості аварії), accidentsNumber (кількість схожих аварій).

Клас Prediction, що є представленням передбачення, має майже аналогічну структуру полів до класу Accident. Єдина відмінність – відсутність у передбачень поля accidentsNumber. Дане поле є лише у аварій та додане з метою уникнення дублювання записів в базі, а також для зручності користувачів, оскільки додавання кожної схожої аварії вручну займає час.

У класів User та Dataset також є додаткові, згенеровані Spring Data JPA, списки об’єктів. Так, User має список датасетів, а Dataset – списки аварій та передбачень. Дані поля завантажуються лише за потреби використання, оскільки завантаження великого списку даних без нагальної потреби не є хорошою практикою. Такий підхід має назву ліниве завантаження (lazy loading).

Більша кількість полів у класах Accident та Prediction базуються на перерахуваннях (enumeration). Розглянемо всі наявні в доменній моделі перерахування.

1. DayOfWeek. Може набувати значень MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY та OTHER. Останнє значення є спільним для багатьох перерахувань та застосовується у випадку, якщо поле не задано.

2. DriverAgeGroup. Значеннями є CHILD (вік водія до 18 років), YOUNG (18-30), MIDDLE_AGED (31-50), SENIOR (51 та старше) та OTHER.

3. DriverGender. Можливими значеннями перерахування є MALE, FEMALE, OTHER.

4. DriverExperience. Може набувати значень BELOW_YEAR, ONE_TO_TWO_YEARS, TWO_TO_FIVE_YEARS, FIVE_TO_TEN_YEARS, ABOVE_TEN_YEARS, NO_LICENSE та OTHER.

5. VehicleType. Значеннями даного перерахування є AUTOMOBILE, LORRY, LONG_LORRY, BUS, MINIBUS, MOTORCYCLE, RIDDEN_HORSE та OTHER.

6. RoadAlignment. Може набувати значень TANGENT_ROAD_WITH_ FLAT_TERRAIN (дотична дорога з рівнинною місцевістю), TANGENT_ROAD_WITH_MOUNTAINOUS_TERRAIN (дотична дорога з гірською місцевістю), TANGENT_ROAD_WITH_MILD_GRADE_FLAT_ TERRAIN (дотична дорога з помірним ухилом і рівнинною місцевістю), STEEP_GRADE_DOWNWARD_WITH_MOUNTAINOUS_TERRAIN (крутий схил вниз з гірською місцевістю), GENTLE_HORIZONTAL_CURVE (полога горизонтальна крива), ESCARPMENT (відкіс) та OTHER.

7. RoadSurfaceType. Значеннями є ASPHALT_ROAD (асфальтна дорога), BAD_ASPHALT_ROAD (асфальтна дорога з суттєвими недоліками), EARTH_ROAD (земляна тропка), GRAVEL_ROAD (гравійна дорога) та OTHER.

8. RoadSurfaceCondition. Може набувати значень DRY (суха дорога), WET_OR_DAMP (мокра дорога) та OTHER.

9. LightCondition. Можливі значення складаються з DAYLIGHT (сонячне, денне світло), DARKNESS_WITH_STREET_LIGHTNING (нічна темрява, дорога освітлена ліхтарями), DARKNESS_WITHOUT_STREET_LIGHTNING (нічна темрява без додаткового освітлення).

10. WeatherCondition. Може набувати значень NORMAL, RAINING (дощ), RAINING_AND_WINDY (дощ та вітер), CLOUDY (похмуро), FOG_OR_MIST (туман), SNOW (сніг) та OTHER.

11. CollisionType. Значеннями є COLLISION_WITH_VEHICLE (зіткнення з іншим транспортним засобом, COLLISION_WITH_PARKED_VEHICLE (зіткнення

з припаркованим транспортним засобом, COLLISION_WITH_PEDESTRIAN (зіткнення з пішоходом), COLLISION_WITH_OBJECTS (зіткнення з нерухомим об’єктом), COLLISION_WITH_ANIMALS (зіткнення з твариною), FALL_FROM_VEHICLE (падіння з транспортного засобу), ROLLOVER (перевертання транспортного засобу) та OTHER. Зазначимо, що зіткнення може відбуватись як з одним, так і з декількома об’єктами.

12. VehicleMovement. Може набувати таких значень як GOING_STRAIGHT (прямий рух), MOVING_BACKWARD (рух назад), U_TURN (розворот), TURNOVER (перекидання), WAITING (очікування), REVERSING (рух заднім ходом), STOPPING (зупиняючись), OVERTAKING (переганяючи) та OTHER.

13. CollisionCause. Значеннями є MOVING_BACKWARD, OVERTAKING, CHANGING_LINE_TO_RIGHT, CHANGING_LINE_TO_LEFT, NO_DISTANCE, NO_PRIORITY_TO_VEHICLE, NO_PRIORITY_TO_PEDESTRIAN, OVERLOADING, DRIVING_AT_HIGH_SPEED, IMPROPER_PARKING, DRIVING_CARELESSLY, DRUNK_DRIVING та OTHER.

14. AccidentSeverity. Може набувати значень SLIGHT, SERIOUS, FATAL. Єдине поле аварії та передбачення, яке не може мати повільне значення.

Слід додати, що наведені вище значення перерахувань є суто програмними. Значення в інтерфейсі веб-застосунку мають більш дружній для користувача вигляд.

Розглянемо другу частину модулю взаємодії з базою даних – рівень репозиторію. Даний рівень майже повністю реалізований фреймворком Spring Data JPA. Для того, щоб робити прості запити до таблиці, необхідно створити інтерфейс, що наслідує JpaRepository та передає в якості параметру тип моделі та первинний ключ таблиці. Наприклад:

public interface AccidentRepository extends JpaRepository<Accident, Long>{}

де Accident – модель, над якою задано @Table(name = “accidents”), а Long – тип первинного ключа таблиці аварій

Для більш розширеного функціоналу роботи з базою даних в інтерфейси мають бути додані абстрактні методи. Їхня реалізація у форматі sql-запиту мовою JPQL чи PostgreSQL може знаходитись в анотації @Query над методом. Такі запити були написані для знаходження кількості аварій за датасетом, ступеню тяжкості аварії та інших атрибутів.

Для нескладних запитів, наприклад, знаходження користувача за полем username, запит може бути написаний в назві методу. Так, нижче наведено знаходження списку тренованих датасетів за ідентифікатором користувача.

List<Dataset> findByIsTrainedOnAndUserId(boolean isTrainedOn, Long userId);

Програма налічує шість запитів, що реалізовані шляхом іменування методу в інтерфейсі репозиторію.

4.3 Модуль аналізу даних

Модуль машинного навчання складається з двох класів, Model та MachineLearningService. Клас Model складається з 18 числових полів, що представляють собою нормалізовані дані для однієї аварії. Значення перерахувань переведені у відповідні числові значення.

MachineLearningService має два публічні методи, train та predict.

Метод train() бере на вхід номер набору даних, для якого має бути проведене тренування моделі. Спочатку сервіс знаходить всі дорожньо-транспортні пригоди класу, потім трансформує їх у список моделей. Наступним кроком є тренування двох моделей: перша передбачує тяжкість аварії, друга – кількість постраждалих. При цьому дані не відрізняються, лише цільова змінна є різною. Реалізацією методу наївного Байєса є клас NaiveBayes бібліотеки Weka.

Отримані класифікатори записуються до таблиці datasets бази даних, а саме до полів accident_severity_prediction_classifier та casualties_number_prediction_ classifier. Також поле is_trained_on задається як true.

Метод predict() бере на вхід Prediction, у якому не заповнені поля accidentSeverity та casualtiesNumber. Для їх заповнення програма завантажує з бази даних обидві моделі, перетворюючи їх на об’єкти типу Classifier, з реалізацією в класі NaiveBayes. Далі на кожній моделі викликається метод classify, що повертає числове значення.

Якщо модель передбачення тяжкості аварії повертає 2.0, це означає що ступінь тяжкості аварії фатальний, оскільки константа FATAL має номер 2 у перерахуванні AccidentSeverity. Оскільки поле casualtiesNumber є числовим значенням, то подальші перетворення для цього не потрібні.

Після заповнення об’єкту класу Prediction він повертається методом predict(), після чого модуль інтерфейсу зберігає його у базі даних.

4.4 Модуль імпорту та експорту

Модуль імпорту та експорту має два компоненти, ImportService та ExportService. Розглянемо кожен з сервісів.

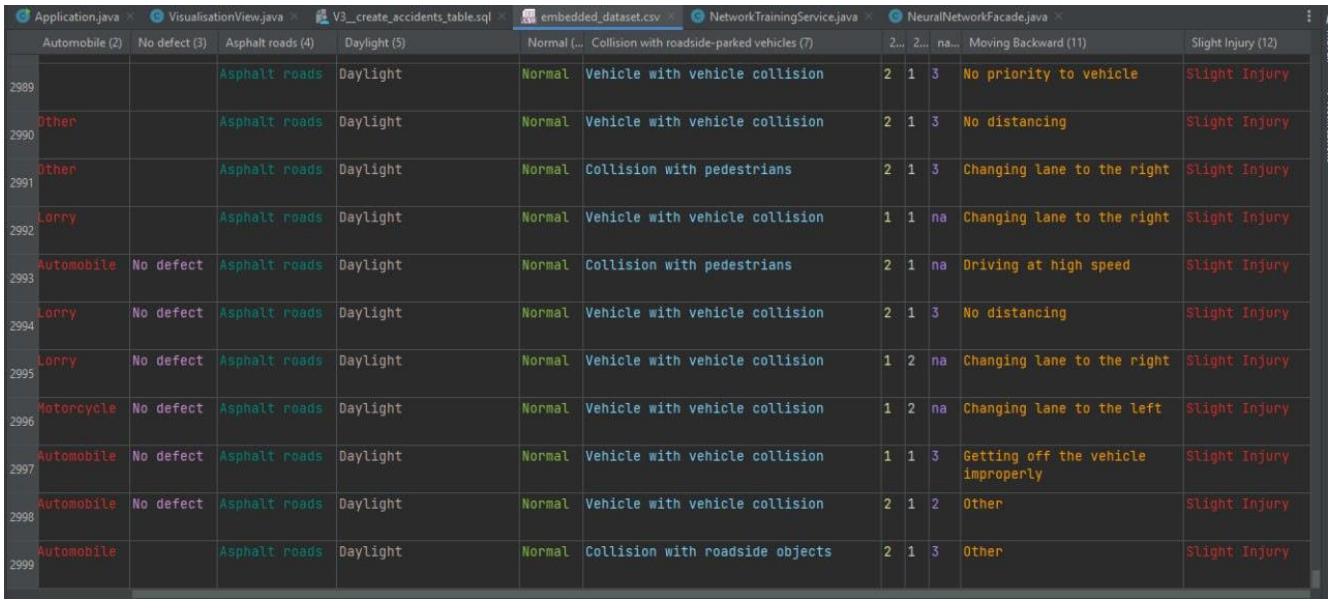
Клас ImportService має два методи: importAccidents та importDefaultDataset. Мета створення першого методу – завантаження наданих користувачем аварій у форматі .csv до бази даних. На вхід потрапляє id набору даних, у який необхідно завантажити аварії, та аварії у форматі InputStream. Метод знаходить датасет за ідентифікатором, після чого построково зчитує дані з InputStream. Отримані текстові дані передаються в конструктор класу Accident, де відбувається трансформація рядкових даних в дані моделі. Наприклад, наступний рядок знаходить необхідне значення для WeatherCondition:

weatherCondition = WeatherCondition.findByTitle(data[11]);

де weatherCondition – поле класу Accident, data[11] – дванадцятий елемент строки, findByTitle – метод перерахування, що повертає значення перерахування для введених користувачем даних.

Таким чином, після створення нового об’єкту класу Accident він додається до списку аварій, що були імпортовані. Далі, після завершення обробки кожної строки InputStream, список дорожньо-транспортних пригод завантажується до відповідної таблиці бази даних. Якщо під час обробки строки у програмі виникла помилка (некоректний формат строки, не задане значення для поля accidentSeverity, тощо), строка пропускається. Датасет може бути великим та при його створенні можливі помилки, отже система не має завершувати обробку через зайвий роздільник.

На рисунку 4.2 наведено приклад типового набору даних у форматі .csv, інтеграцією якого в програму займається даний модуль.



Application.java | VisualisationView.java | V3__create_accidents_table.sql | embedded_dataset.csv | NetworkTrainingService.java | NeuralNetworkFacade.java

	Automobile (2)	No defect (3)	Asphalt roads (4)	Daylight (5)	Normal (...	Collision with roadside-parked vehicles (7)	2...	2...	na...	Moving Backward (11)	Slight Injury (12)
2989			Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	2	1	3	No priority to vehicle	Slight Injury
2990	Other		Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	2	1	3	No distancing	Slight Injury
2991	Other		Asphalt roads	Daylight	Normal	Collision with pedestrians	2	1	3	Changing lane to the right	Slight Injury
2992	Lorry		Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	1	1	na	Changing lane to the right	Slight Injury
2993	Automobile	No defect	Asphalt roads	Daylight	Normal	Collision with pedestrians	2	1	na	Driving at high speed	Slight Injury
2994	Lorry	No defect	Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	2	1	3	No distancing	Slight Injury
2995	Lorry	No defect	Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	1	2	na	Changing lane to the right	Slight Injury
2996	Motorcycle	No defect	Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	1	2	na	Changing lane to the left	Slight Injury
2997	Automobile	No defect	Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	1	1	3	Getting off the vehicle improperly	Slight Injury
2998	Automobile	No defect	Asphalt roads	Daylight	Normal	Vehicle with vehicle collision	2	1	2	Other	Slight Injury
2999	Automobile		Asphalt roads	Daylight	Normal	Collision with roadside objects	2	1	3	Other	Slight Injury

Рисунок 4.2 – Типовий набір даних у .csv

Метод importDefaultDataset бере на вхід об’єкт класу User, до якого належатиме вбудований набір даних. Метод знаходить у папці /resources файл embedded_dataset.csv та створює InputStream на основі даних файлу. Далі відбувається створення та збереження датасету, після чого викликається функція importAccidents з ідентифікатором нового датасету та InputStream вбудованого датасету.

Клас ExportService містить два методи. Перший має назву export та бере на вхід ідентифікатор датасету. За цим ідентифікатором відбувається пошук передбачень, кожне з яких трансформується у строку значень, розділену комою.

Набір передбачень в свою чергу розділяється за допомогою символу переходу на нову строку. На виході методу - сформована строка, що у подальшому буде без змін записана у файл формату .csv.

Другий метод класу ExportService має назву exportAccidentImportRules та не бере параметрів на вхід. Задача даного методу – створення пам’ятки для користувача, з усіма даними про створення датасету через .csv файл. Строка на виході містить кількість змінних датасету, пояснення формату, у якому мають бути значення, а також перерахування всіх значень через кому.

Для оформлення завантаження файлів користувачем було використано Vaadin компонент Upload, а для завантаження файлів на його локальну машину – компоненти Button та FileDownloadWrapper. Максимальний розмір .csv файлу аварій, що може бути завантажений, становить 10 мегабайтів.

4.5 Модуль безпеки

Даний модуль побудовано на основі фреймворку Spring Security. Він містить три класи: SecurityConfiguration, UserDetailsServiceImpl та AuthenticatedUser. Розглянемо кожен з них.

Клас SecurityConfiguration наслідує клас VaadinWebSecurity та містить два методи. Перший метод (configure) задає клас сторінки логіну, яка стає точкою входу в систему.

Слід зазначити, що захист конкретних рутів у веб-застосунку розроблено за допомогою Vaadin анотацій. Наприклад, для того щоб дозволити доступ анонімному користувачеві над класом сторінки додається анотація @AnonymousAllowed, Для дозволу доступу до сторінки кожному авторизованому користувачеві додається анотація @PermitAll.

Другий метод класу SecurityConfiguration має назву passwordEncoder та повертає реалізацію однойменного інтерфейсу, а саме BCryptPasswordEncoder(). Тепер даний клас є керованим Spring Framework біном, що зашифровує строку та

порівнює введений пароль з уже зашифрованим. Використовується у LoginView, що є реалізацією сторінки логіну.

UserDetailsServiceImpl є реалізацією Spring Security інтерфейсу UserDetailsService. Даний клас буде використаний під капотом, для авторизації та аутентифікації у Spring-застосунку. UserDetailsServiceImpl має один метод, loadUserByUsername, що викликає UserRepository для знаходження користувача за його унікальним ім’ям. Якщо користувача не знайдено, метод викидає виключення, яке буде оброблено Spring з висновком, що користувач для якого викликаний метод є анонімним.

AuthenticatedUser є сервісом, через який відбувається взаємодія веб-застосунку з даними про поточного користувача. Клас має два методи, get(), що отримує з Spring контексту користувача, та logout(), що видаляє його з контексту.

4.6 Модуль інтерфейсу

На рисунку 4.3 зображена діаграма прецедентів для даного веб-застосунку.

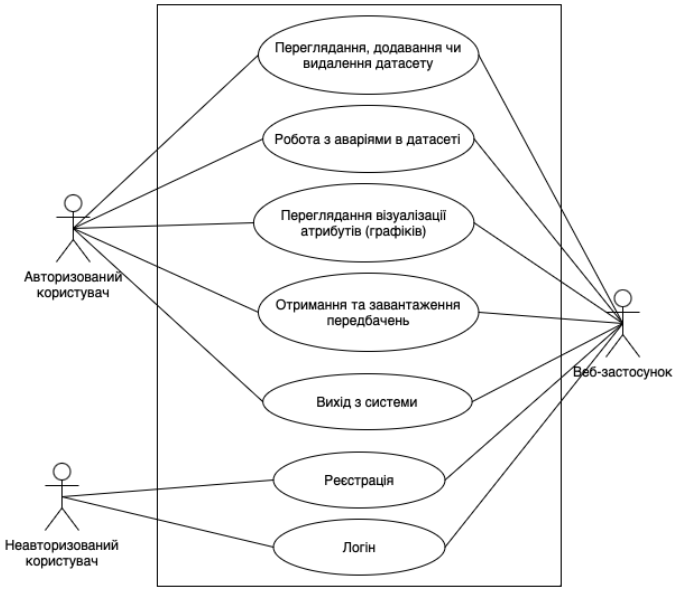


Рисунок 4.3 – Діаграма прецедентів

Як можна побачити з діаграми, окрім веб-застосунку система має двох акторів: неавторизований та авторизований користувач. Прецеденти для обох

акторів різні. Таким чином, неавторизований користувач може зробити запит на реєстрацію та логін, в той час як авторизований – безпосередньо працювати з системою, тобто переглядати, додавати та видаляти датасети, аварії, переглядати візуалізацію атрибутів у вигляді графіку, отримувати та завантажувати передбачення, виходити з системи.

Беручи до уваги даний функціонал, на основі фреймворку Vaadin було розроблено вісім сторінок. Розглянемо кожен клас створення сторінки інтерфейсу окремо.

Першою сторінкою, що бачить користувач при знайомстві з веб-застосунком, є сторінка логіну. Клас даної сторінки має назву LoginView та побудований на основі вбудованого компоненту LoginOverlay. Він містить просту реалізацію сторінки логіну (LoginI18n), що складається з назви сторінки, опису, а також полів вводу логіну та пароля. На компоненті логіна також наявне посилання на сторінку реєстрації.

Якщо авторизований користувач вже наявний в системі, то він автоматично переадресується на сторінку набору даних. Даний функціонал реалізовано за допомогою BeforeEnterObserver, Vaadin компоненту що виконує маршрутизацію до завантаження контенту сторінки. Ознакою наявності користувача в системі є наявність Spring Security біна типу AuthenticatedUser.

Наступною сторінкою є сторінка реєстрації, що містить свою реалізацію у класі RegistrationView, а також у допоміжних класах RegistrationForm та RegistrationFormBinder. Перший допоміжний клас є компонентом форми та відповідає за її зовнішню частину, тобто містить текстові поля та кнопку реєстрації. Після створення у класі RegistrationView він додається до сторінки, а також передається на вхід до RegistrationBinder. Задачею другого допоміжного класу є логічна частина реєстрації, тобто заповнення класу користувача введеними у текстові поля даними, перевірка коректності паролю та його підтвердження.

Аналогічно до сторінки логіну, у разі потрапляння авторизованого користувача на сторінку реєстрації, він буде автоматично перенаправлений на сторінку наборів даних.

Реалізація сторінки датасетів знаходиться у DatasetsView. Сторінка логічно поділена на дві частини, в першій знаходиться назва сторінки та кнопка створення нового датасету, а в другій – перераховані всі наявні.

Кожен елемент що представлений у списку датасетів є об'єктом допоміжного класу DatasetsViewCard. Один датасет на сторінці представлений у вигляді блоку з демонстраційною картинкою, назви, опису та набору кнопок. Картинка завантажується з файлової системи програми та не може бути змінена. Один блок-відображення датасету складається з чотирьох кнопок, а саме: подробиці датасету, його тренування та видалення, а також завантаження передбачень на його основі. Логіка кожної з цих кнопок прописана у ComponentEventListener. Зазвичай вона складається з виклику сервісу (наприклад, MachineLearningService для кнопки тренування датасету) та переходу на інше відображення.

Наступна сторінка є сторінкою створення датасету та реалізована у класі DatasetView. На ній знаходиться п'ять компонентів, а саме назва сторінки, два текстові поля та дві кнопки. У текстові поля мають бути введені назва та опис майбутнього набору даних. При натисканні на кнопку збереження, за допомогою Vaadin класу Binder відбувається створення та додавання до бази даних об'єкту класу Dataset.

Реалізація сторінки датасету знаходиться у SingleDatasetView. Сторінка розділена на два вертикальні блоки, що розроблені за допомогою компоненту SplitLayout та можуть змінювати свої розміри в залежності від перетягування. Ліва (основна) частина SplitLayout є списком аварій, а права що містить форму для створення та редагування аварій, а також посилання на завантаження датасету через .csv файл. Форма реалізована за допомогою текстових полів та полів вибору

(для типів даних, що є перерахуваннями). Як і в попередніх формах, створення аварії відбувається за допомогою класу Binder.

Сторінка завантаження дорожньо-транспортних пригод розроблена у класі ImportDatasetView та складається з компоненту завантаження файлу-пам’ятки з заповнення датасету і з поля, у яке csv файл має бути вставлений. Для створення файлу-пам’ятки та обробки отриманого від користувача файлу використовується модуль імпорту та експорту.

Реалізація сторінки передбачень знаходиться у PredictionsView. За своєю структурою дана сторінка не відрізняється від SingleDatasetView – використано компонент SplitLayout, основною частиною якого є список передбачень, а правою – форма створення передбачення. Сторінка викликає метод predict() модулю аналізу даних при збереженні передбачення.

Сторінка візуалізації реалізована у класі VisualisationView. Графік будується на основі обраного користувачем датасету та поля, для якого необхідно відобразити залежність з ступенем тяжкості аварії. При побудові графіку було використано бібліотеку ApexCharts для Vaadin. На рисунку 4.4 наведено залежність тяжкості аварії від погодних умов.

Якщо в програмі немає натренованих наборів даних, PredictionsView та VisualisationView не зможуть коректно відображати функціонал, а отже замість контенту дані сторінки відображають екран-заглушку.

Слід зазначити, що кожна сторінка системи (окрім логіну та реєстрації) знаходиться поруч з вертикальною панеллю навігації, що містить в собі вкладки «Датасети», «Передбачення» та «Візуалізація». Внизу панелі знаходиться кнопка виходу з системи.

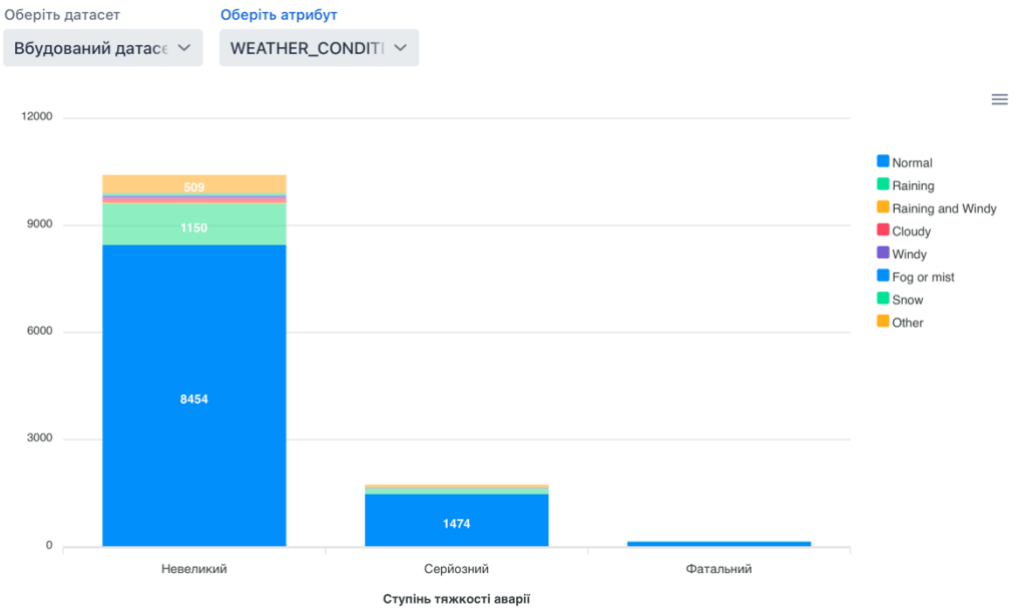


Рисунок 4.4 – Залежність тяжкості аварії від погодних умов

4.7 Опис бази даних

База даних складається з чотирьох таблиць, а саме таблиці користувачів, наборів даних, аварій та передбачень (рисунок 4.5).

Таблиця користувачів заповнюється під час реєстрації та більше не редагується. Пароль зберігається у зашифрованому вигляді, отже авторизація відбувається порівнянням двох зашифрованих строк, що робить програму безпечнішою. Інші поля зберігаються без програмних змін, лише з заданням максимальної кількості символів (наприклад, varchar(50) для імені користувача та електронної адреси).

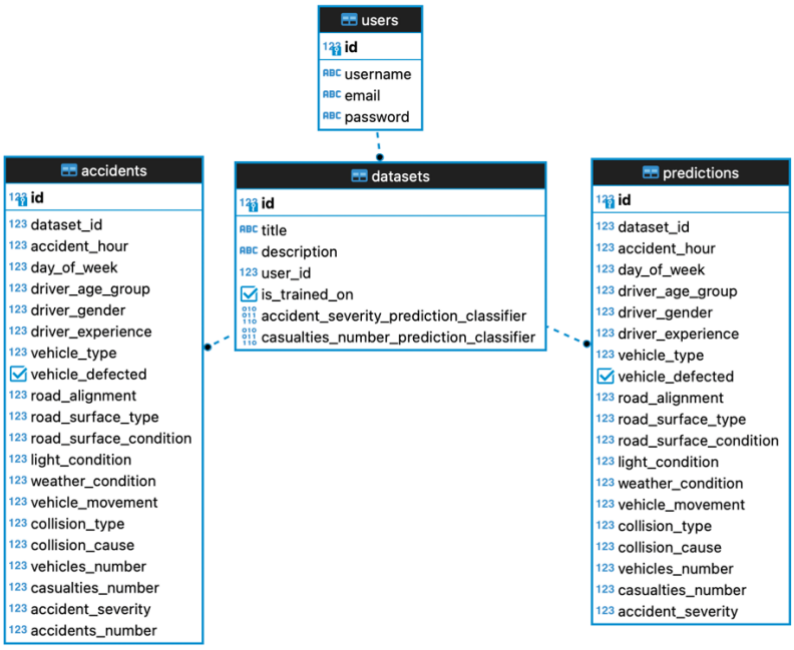


Рисунок 4.5 – Архітектура бази даних

Кожен користувач має певну кількість наборів даних. Набір даних складається з назви, опису, посилання на таблицю користувачів, інформації про натренованість датасету та двох натренованих моделей, одна з яких передбачає тяжкість аварії, а друга – кількість постраждалих. Обидві моделі зберігаються у форматі bytea (масив байтів).

Таблиця наборів даних відноситься як один к багатьом для двох схожих за наповненням таблиць – таблиці аварій та передбачень. Обидві таблиці мають аналогічну структуру, єдиною відмінністю є поле accidents_number.

На відміну від класів доменної моделі, таблиці містять не строкові, а лише числові або булеві значення. Так, поле accidents_severity містить значення 0, 1 або 2, в залежності від класу тяжкості аварії.

4.8 Алгоритм роботи програми

Робота веб-застосунку буде значно обмежена якщо користувач некоректно виконуватиме послідовність необхідних дій. Розглянемо кожний крок користувацької взаємодії з системою.

Заходячи на сайт, неавторизований користувач потрапляє на сторінку логіну. Оскільки він ще не має акаунту, відбувається перехід на сторінку реєстрації. Після вводу даних, а саме імені користувача, електронної адреси, паролю та підтвердження паролю, користувач потрапляє назад на сторінку логіну, після чого вводить дані для авторизації.

Сторінкою, на яку відбувається перехід після авторизації, є сторінка датасетів, куди вже було автоматично додано вбудований датасет з більш ніж 10000 дорожньо-транспортних пригод. Користувач може переглянути дані, тренувати, завантажувати, створювати чи видаляти набір даних.

Положимо, що користувач хоче додати дані до програми, але цих даних велика кількість та він не може використовувати інтерфейс веб-застосунку. У цьому випадку він має спочатку створити датасет, натиснувши на однойменну кнопку на сторінці датасетів. Після введення назви та опису набору даних користувач потрапляє на сторінку датасету, а звідти – на сторінку завантаження. Там він може спочатку переглянути інструкцію з заповнення набору даних, а потім завантажити свій датасет аварій, перероблений під вимоги. Після завантаження користувач може переглянути, відредагувати та видалити окремі аварії.

Обов’язковим наступним кроком, без якого не будуть відображатись передбачення та графіки, є тренування датасету. Після успішного тренування користувач може зробити передбачення, відвідавши відповідну вкладку та ввівши дані 16 змінних.

Отримав декілька передбачень, користувач може завантажити їх у форматі .csv зі сторінки датасетів. Формат та послідовність змінних у файлі є аналогічними до отриманих програмою аварій.

Необов’язковим, але важливим для розуміння набору даних кроком є переглядання складених з натренованого датасету графіків. Графік будується після того, як користувач вибере набір даних та змінну датасету.

Останнім кроком є вихід з веб-застосунку після натиснення на кнопку «Logout» у випадаючому списку внизу лівої панелі. Після даної дії користувач

переадресується на сторінку логіну, після чого алгоритм роботи з програмою може бути повторений.

Висновки до розділу 4

В даному розділі було детально розглянуто модулі веб-застосунку, наведено основні класи з яких складається кожен модуль, описано алгоритм роботи ключових методів. Також проведено аналіз взаємодії модулів між собою, та з користувачем, надано опис сторінок інтерфейсу та алгоритму взаємодії з функціоналом програми.

5 РОБОТА КОРИСТУВАЧА З ВЕБ-ЗАСТОСУНКОМ

5.1 Системні вимоги

Розроблений веб-застосунок є адаптованим як під великі монітори, так і під мобільні телефони. Однак якщо для розгортання застосунку використовується не автономний хостинг (AWS, Heroku, Google Cloud, Microsoft Azure), а локальна машина, то єдиним способом перегляданння сайту є комп'ютер з попередньо розгорнутим веб-застосунком.

5.2 Встановлення веб-застосунку на комп'ютер

Локальна машина користувача повинна мати наступне:

1) JRE (Java Runtime Environment) – програмне забезпечення, що відповідає за виконання jar-файлу.

2) Jar-файл, sql-файл міграцій та модуль фронтенду.

3) Docker.

4) Довільний браузер.

Для розгортання веб-застосунку необхідно спершу розгорнути базу даних. Для цього створимо Dockerfile, який задає пароль та назву майбутньої бази даних. Пароль за замовченням має бути порожнім, а строка назви має дорівнювати «diploma». Також скрипт має копіювати значення файлу migration.sql до внутрішньої папки контейнера.

Файл migration.sql є файлом міграцій, що створює усі таблиці в контейнері. Побудуємо та запустимо контейнер, після чого запустимо виконуваний jar файл.

Веб-застосунок буде розгорнутий на порту 8080. Для переходу до програми необхідно ввести в адресну строку браузера http://localhost:8080/login.

5.3 Робота з веб-застосунком

Заходячи на сайт вперше, користувач потрапляє на сторінку авторизації. (рисунок 5.1).

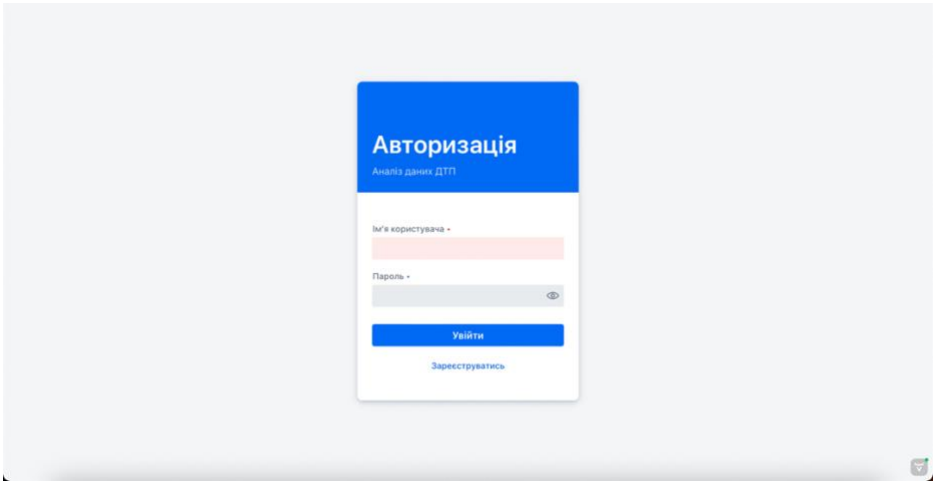


Рисунок 5.1 – Сторінка авторизації

Перейдемо до сторінки реєстрації (рисунок 5.2) та створимо акаунт.

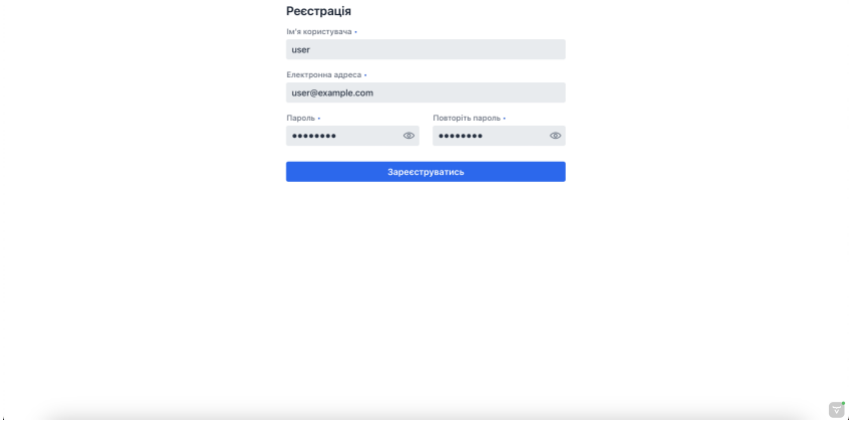


Рисунок 5.2 – Сторінка реєстрації

Натиснемо на кнопку «Зареєструватись» та введемо дані нового користувача у форму логіну. Авторизувавшись, ми потрапляємо на сторінку датасетів, де автоматично було створено вбудований датасет (рисунок 5.3).

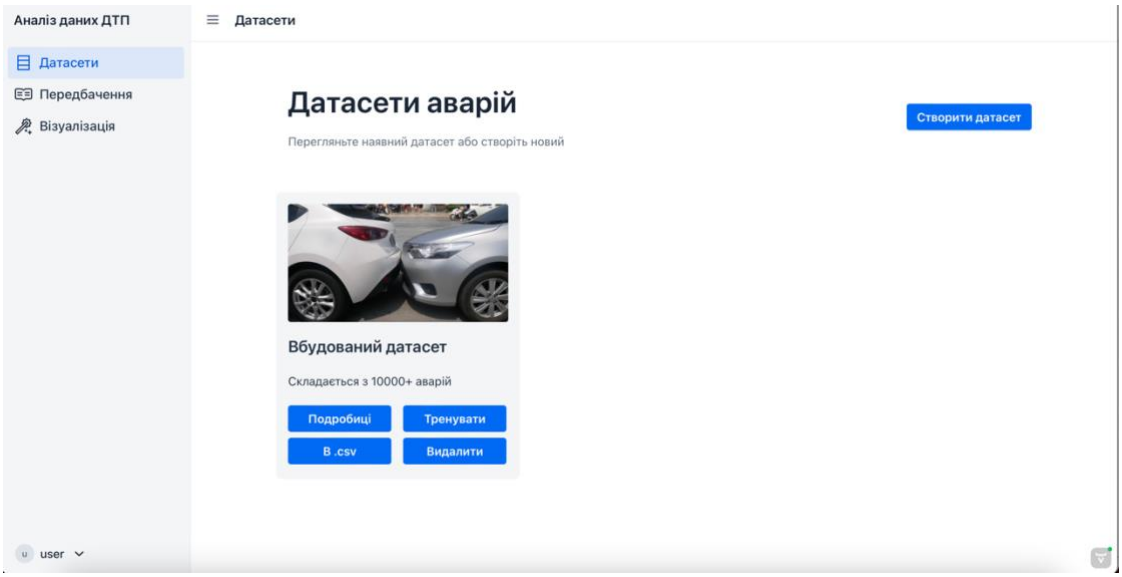


Рисунок 5.3 – Сторінка датасетів.

Натиснемо на кнопку «Створити датасет» у правому верхньому кутку та перейдемо до створення набору даних (рисунок 5.4).

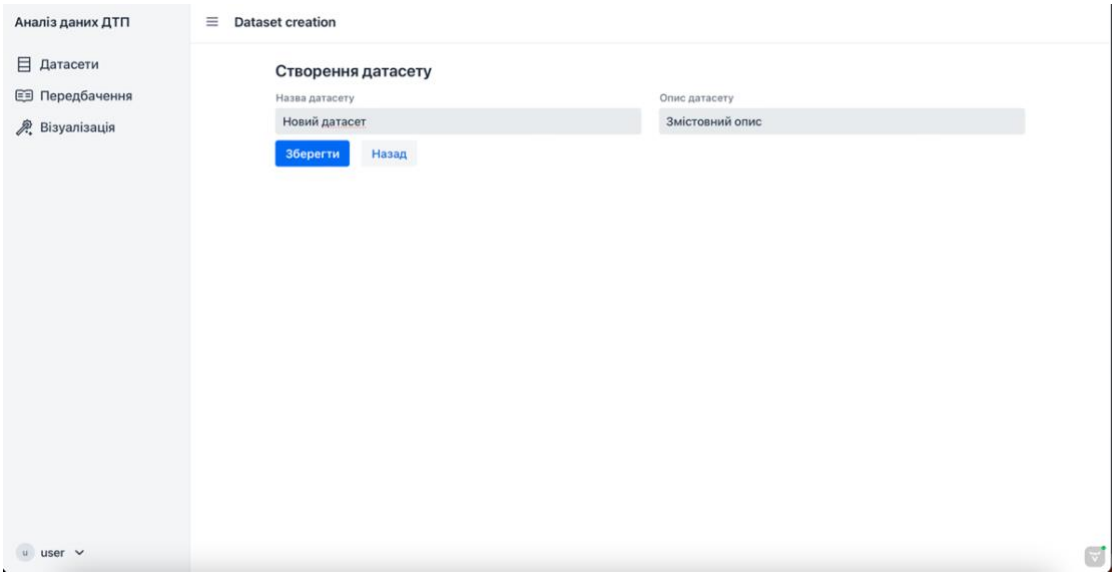


Рисунок 5.4 – Створення датасету

Введемо назву та опис датасету та натиснемо на «Зберегти». Наступна сторінка є переліком всіх аварій датасету (рисунок 5.5). Додамо декілька аварій.

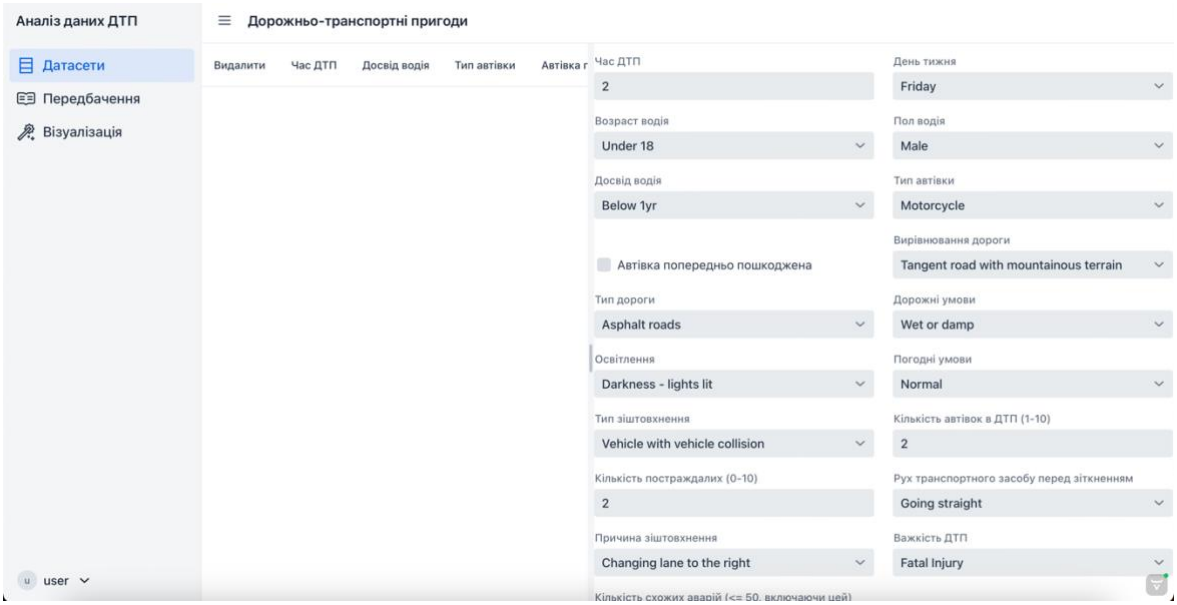


Рисунок 5.5 – Сторінка дорожньо-транспортної пригоди

Пролистаємо форму нижче (рисунок 5.6). Внизу знаходяться кнопки збереження та видалення змін, а також переходу до сторінки завантаження дорожньо-транспортних пригод у форматі .csv.

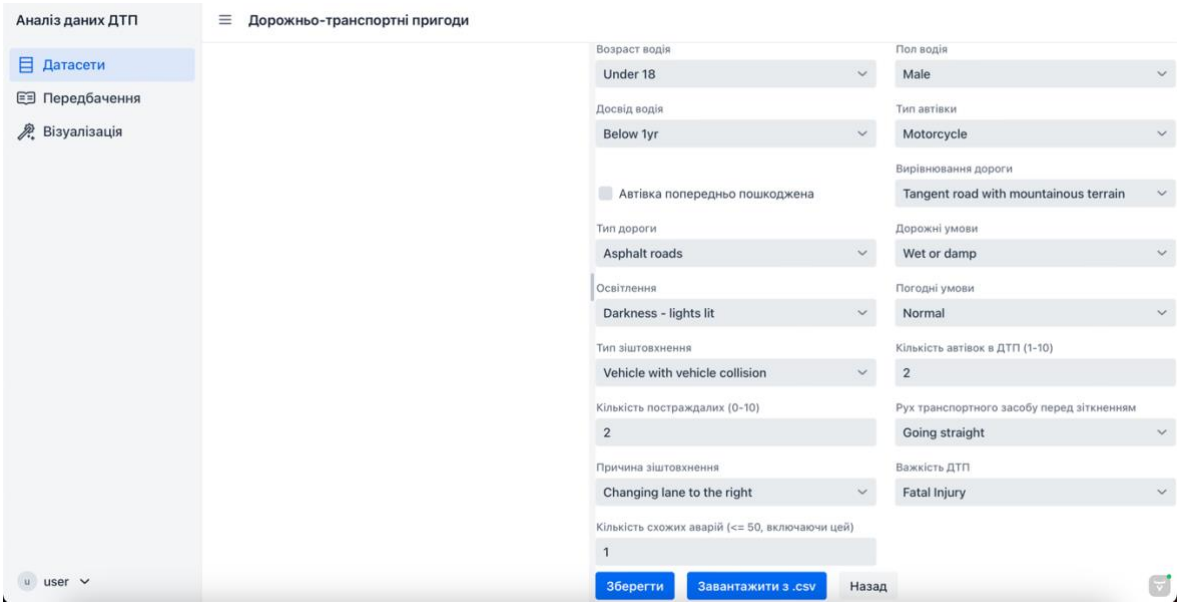


Рисунок 5.6 – Сторінка дорожньо-транспортної пригоди (продовження)

Збережемо введений набір даних та натиснемо на кнопку «Завантажити з .csv». Після натискання відбувається перехід на сторінку завантаження датасету (рисунок 5.7).

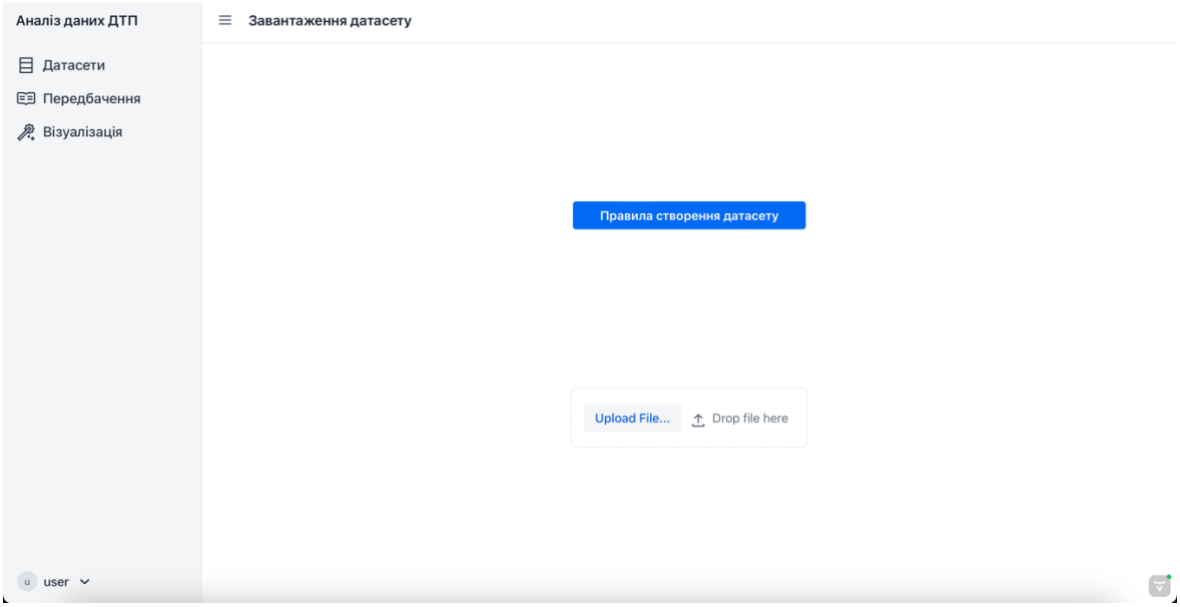


Рисунок 5.7 – Сторінка завантаження датасету

Завантажимо інструкцію, натиснувши на кнопку «Правила створення датасету». Відкриємо отриманий файл з назвою importRules.txt (рисунок 5.8).

```
1  Набір даних повинен мати 18 атрибутів, з яких 2 цільові змінні
2  Записувати значення для однієї аварії необхідно в строку, через кому
3  Нижче наведені змінні та можливі значення
4  accidentHour: числове поле, від 0 до 23
5  dayOfWeek: Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday,Other
6  driverAgeGroup: Under 18,18-30,31-50,Over 51,Unknown
7  driverGender: Male,Female,Other
8  driverExperience: Below 1yr,1-2yr,2-5yr,5-10yr,Above 10yr,No License,Other
9  vehicleType: Automobile,Lorry,Long lorry,Bus,Minibus,Motorcycle,Ridden horse,Other
10 vehicleDefected: строка, значеннями є 'no defect', 'defected'
11 roadAlignment: Tangent road with flat terrain,Tangent road with mountainous terrain,Tangent road with mild
12 roadSurfaceType: Asphalt roads,Asphalt roads with some distress,Earth roads,Gravel roads,Other
13 roadSurfaceCondition: Dry,Wet or damp,Other
14 lightCondition: Daylight,Darkness - lights lit,Darkness - no lighting,Other
15 weatherCondition: Normal,Raining,Raining and Windy,Cloudy,Windy,Fog or mist,Snow,Other
16 collisionType: Vehicle with vehicle collision,Collision with roadside-parked vehicles,Collision with pedest
17 vehiclesNumber: числове поле, від 1 до 10
18 casualtiesNumber: числове поле, від 0 до 10
19 vehicleMovement: Going straight,Moving Backward,U-Turn,Turnover,Waiting to go,Reversing,Stopping,Overtaking
20 collisionCause: Moving Backward,Overtaking,Changing lane to the right,Changing lane to the left,No distanci
21 accidentSeverity: Slight Injury,Serious Injury,Fatal Injury
```

Рисунок 5.8 – Інструкція зі створення датасету у форматі .csv

Файл містить пояснення формату, у якому повинні бути дані, а також можливі значення для кожного поля. На основі даного файлу відредагуємо та завантажимо наявний набір даних. Після завантаження веб-застосунок автоматично перенаправляє користувача до списку датасетів. Перейдемо назад до сторінки

аварій датасету та відредагуємо довільну дорожньо-транспортну пригоду (рисунок 5.9).

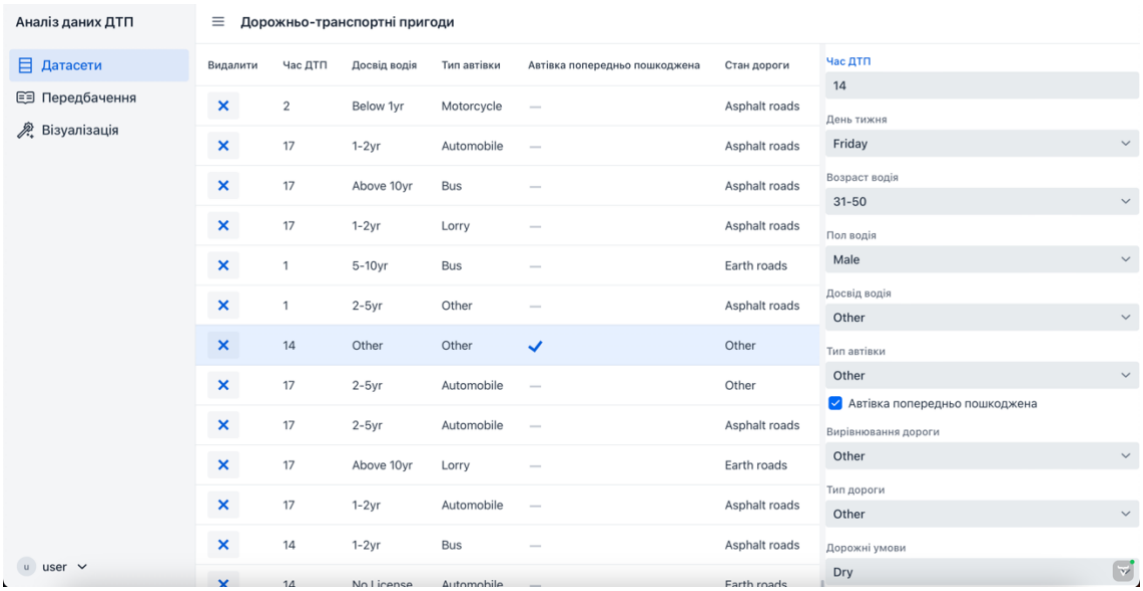


Рисунок 5.9 – Редагування дорожньо-транспортної пригоди

Далі перейдемо до сторінки передбачень (рисунок 5.10). Оскільки ми не натренували модель на жодному з датасетів, замість списку передбачень, схожого на список аварій, користувач побачить лише заглушку.

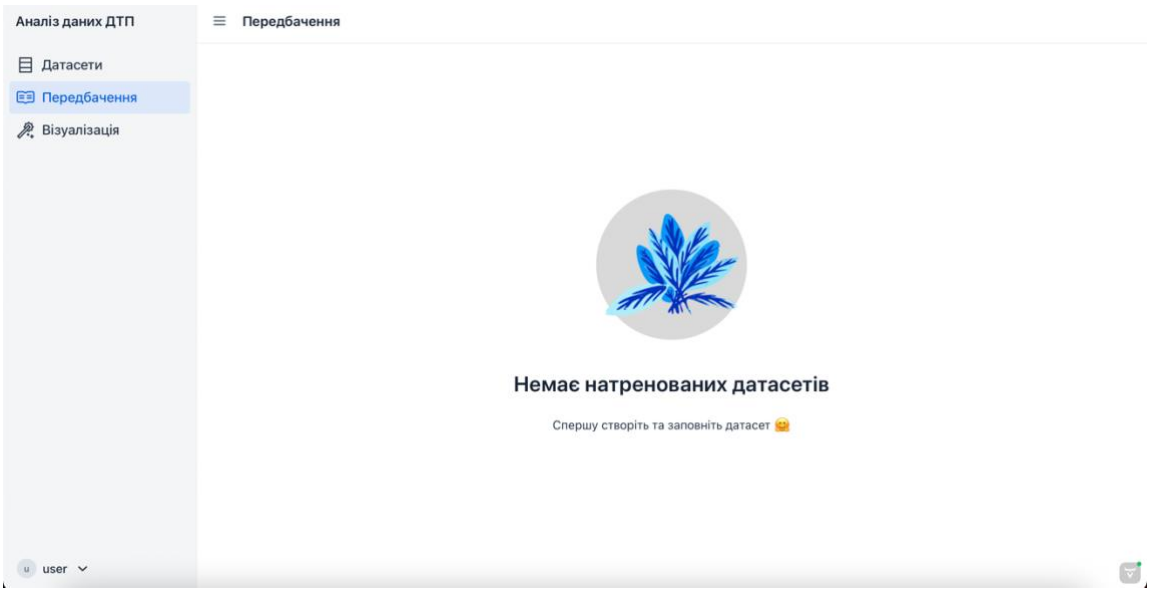


Рисунок 5.10 – Сторінка передбачень без наявності натренованого датасету

Повернемось на сторінку датасетів та натиснемо кнопку «Тренувати» на створеному наборі даних. Запит триватиме секунду-дві та у нижньому лівому кутку користувач зможе побачити сповіщення о успішності тренування (рисунок 5.11).

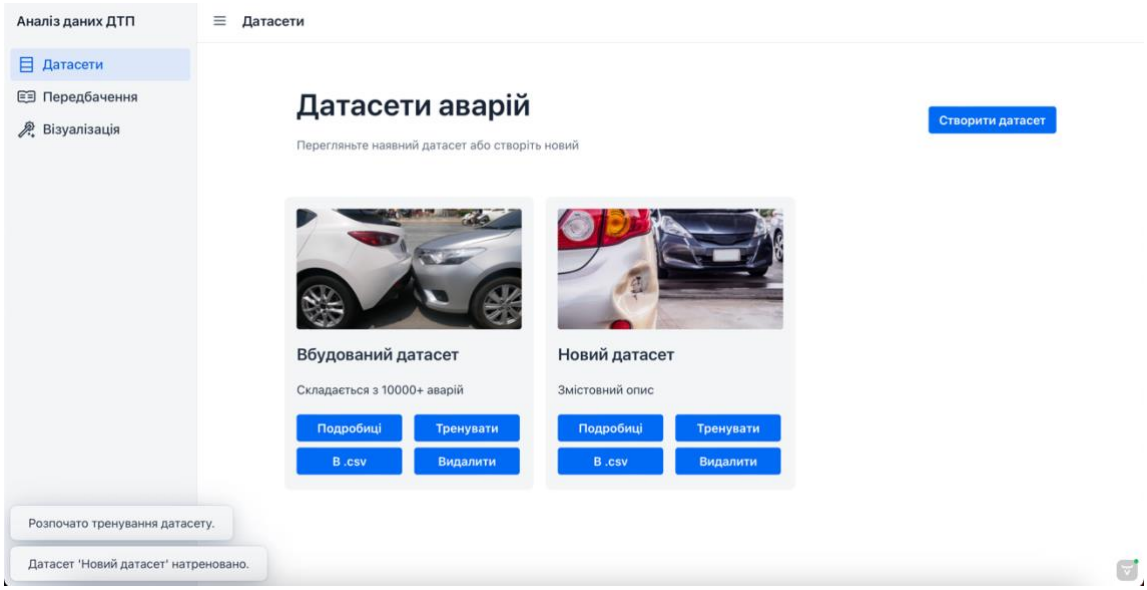


Рисунок 5.11 – Тестування створеного набору даних

Перейдемо назад до сторінки передбачень. Оскільки у поточному акаунті вже є натренований датасет, користувач побачить список та редактор. Введемо дані, для яких хочемо отримати передбачення (рисунок 5.12).

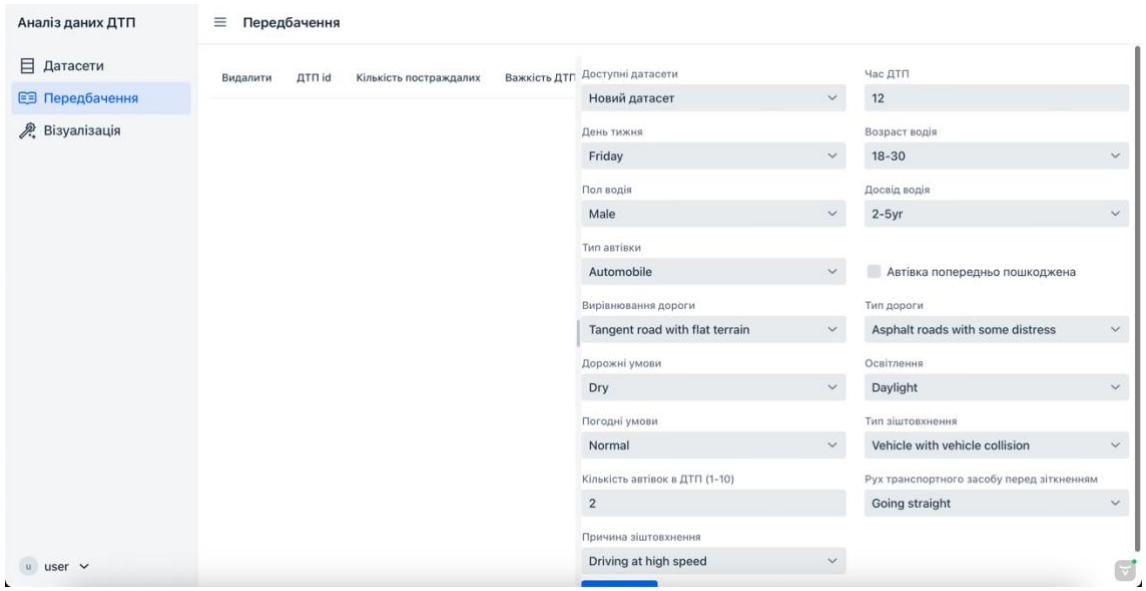


Рисунок 5.12 – Створення передбачення

Натиснемо на кнопку «Зберегти» внизу сторінки. У списку зліва з’явиться передбачення кількості постраждалих та важкості дорожньо-транспортної пригоди (рисунок 5.13).

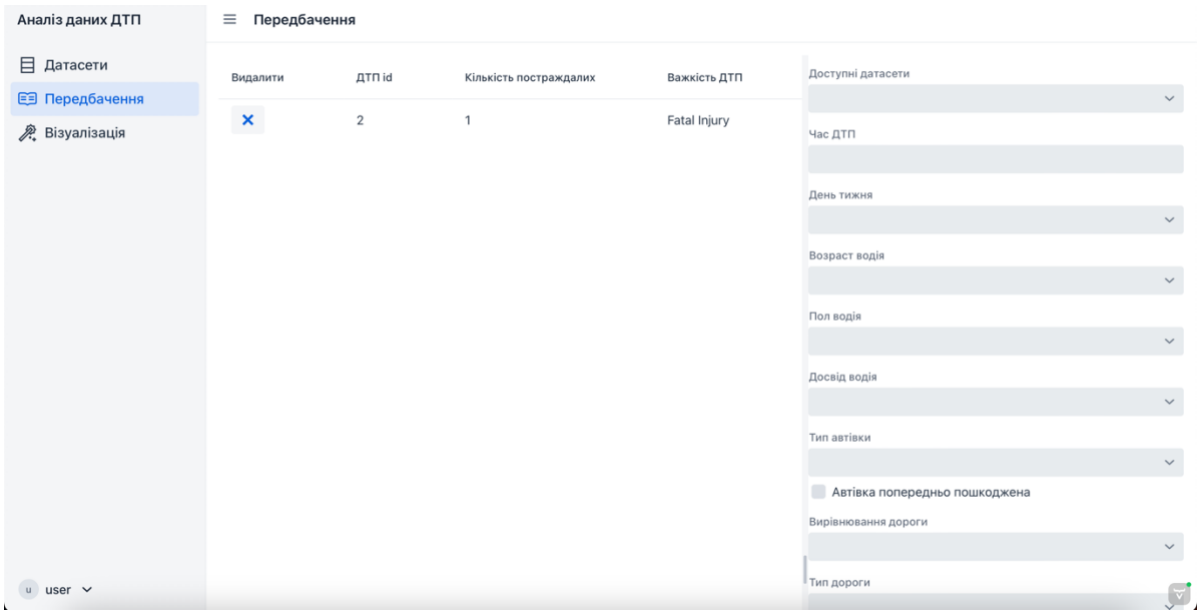


Рисунок 5.13 – Отримання передбачення

Для завантаження передбачень у форматі .csv повернемось до сторінки датасетів та натиснемо на відповідну кнопку на натренованому наборі даних. На рисунку 5.14 зображено наповнення файлу (для скріншоту було додано перехід на нову строку).

```
12,Friday,18-30,Male,2-5yr,Automobile,false,Tangent road with flat terrain,Asphalt roads with some distress,
Dry,Daylight,Normal,Vehicle with vehicle collision,2,1,Going straight,Driving at high speed,Fatal Injury
```

Рисунок 5.14 – Передбачення у форматі .csv

Перейдемо до вкладки з назвою «Візуалізація». Оскільки ми маємо натренований датасет у поточному акаунті, на сторінці відображається не заглушка, а поле вибору датасету (рисунок 5.15). Виберемо єдиний набір даних, після чого задаймо «WEATHER_CONDITION» в якості поля, залежність від якого буде будуватись на графіку.

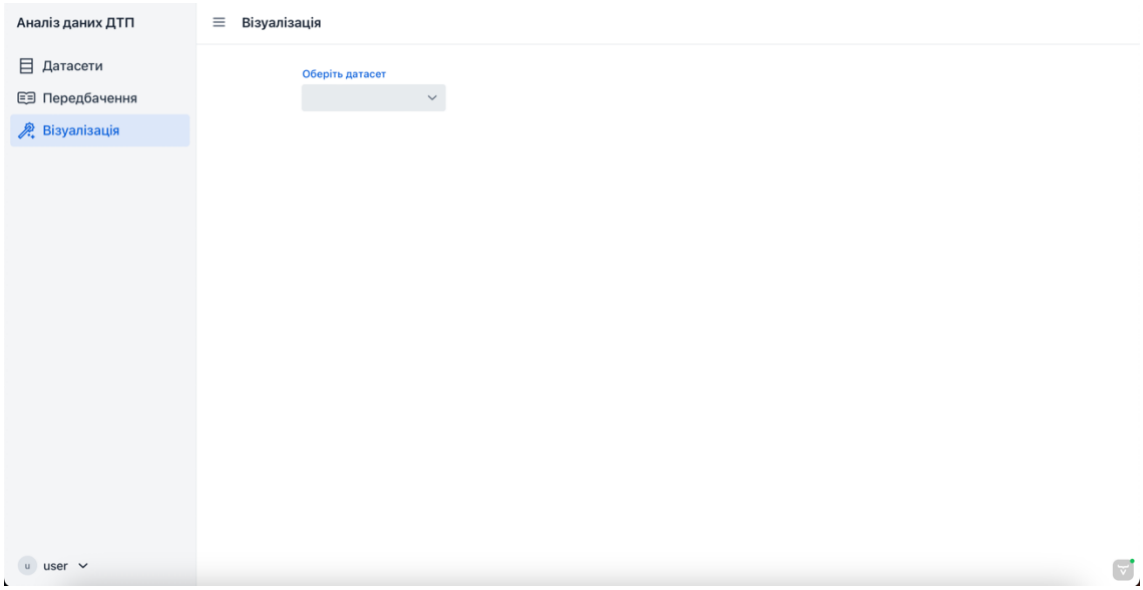


Рисунок 5.15 – Сторінка візуалізації до вибору датасету

На рисунку 5.16 побудовано графік, в якому вісь X відповідає за класи ступеню тяжкості аварії, а клас Y – за обрану змінну. Можливі типи погодних умов представлені як частини

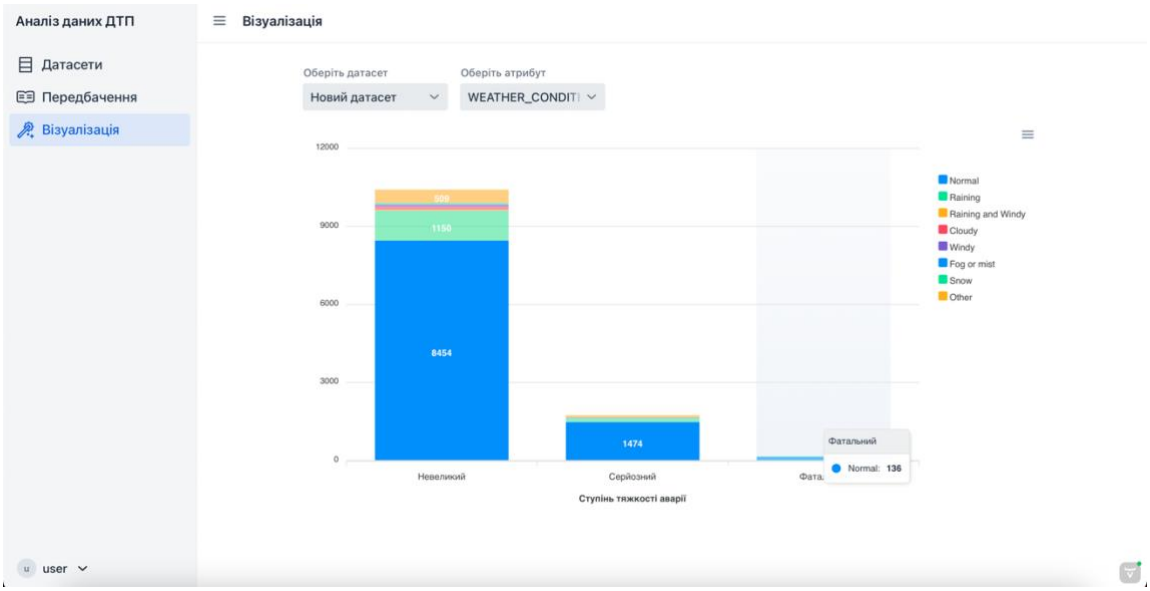


Рисунок 5.16 – Графік залежності ступеню тяжкості аварії від погодних умов

Можливе також завантаження даних у форматі .csv, .png або .svg. Для цього необхідно натиснути на три полоси у правому верхньому кутку сторінки. Завантажимо та переглянемо отриманий з графіку .csv файл (рисунок 5.17).

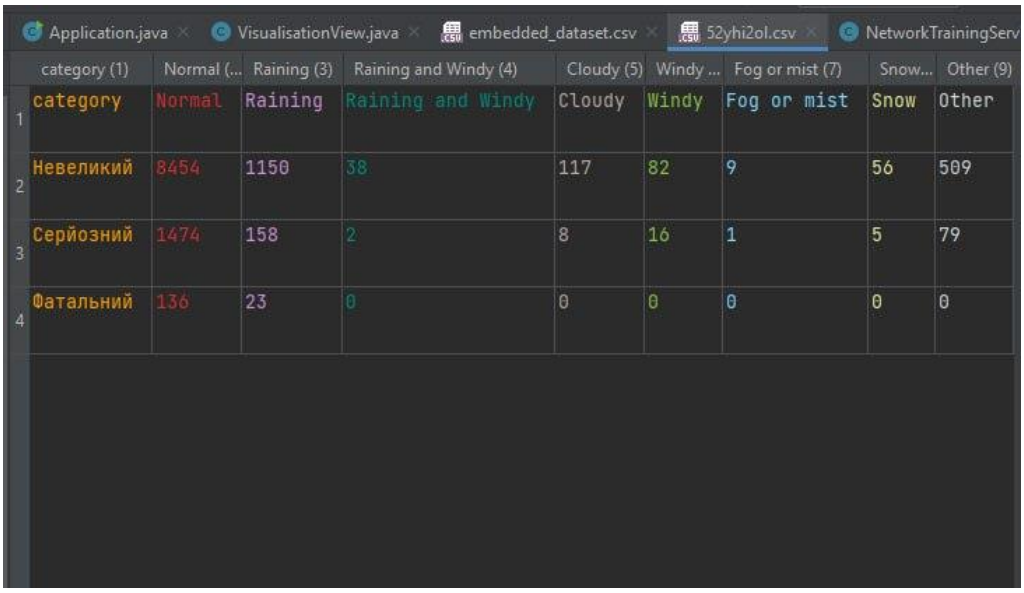


Рисунок 5.17 – Завантаження файлу з графіку

Оскільки система не має потреби у вбудованому датасеті, перейдемо на сторінку дорожньо-транспортних пригод та видалимо його (рисунок 5.18).

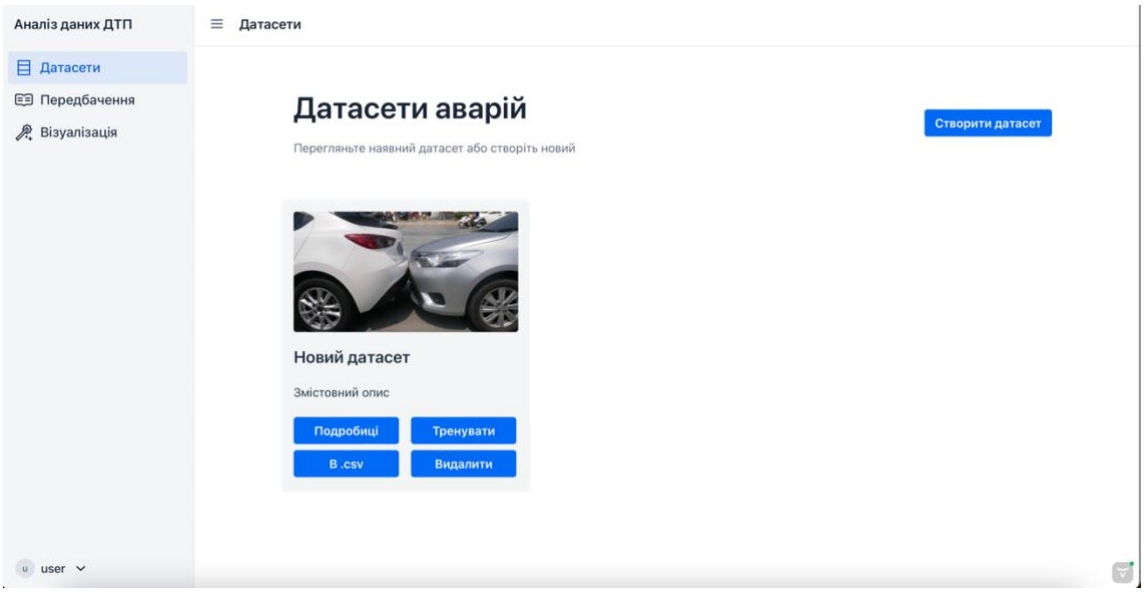


Рисунок 5.18 – Вбудований датасет видалено

Після завершення взаємодії з системою користувач може її покинути, натиснувши на блок з іменем користувача внизу панелі, де з’явиться посилання з назвою «Logout» (рисунок 5.19).

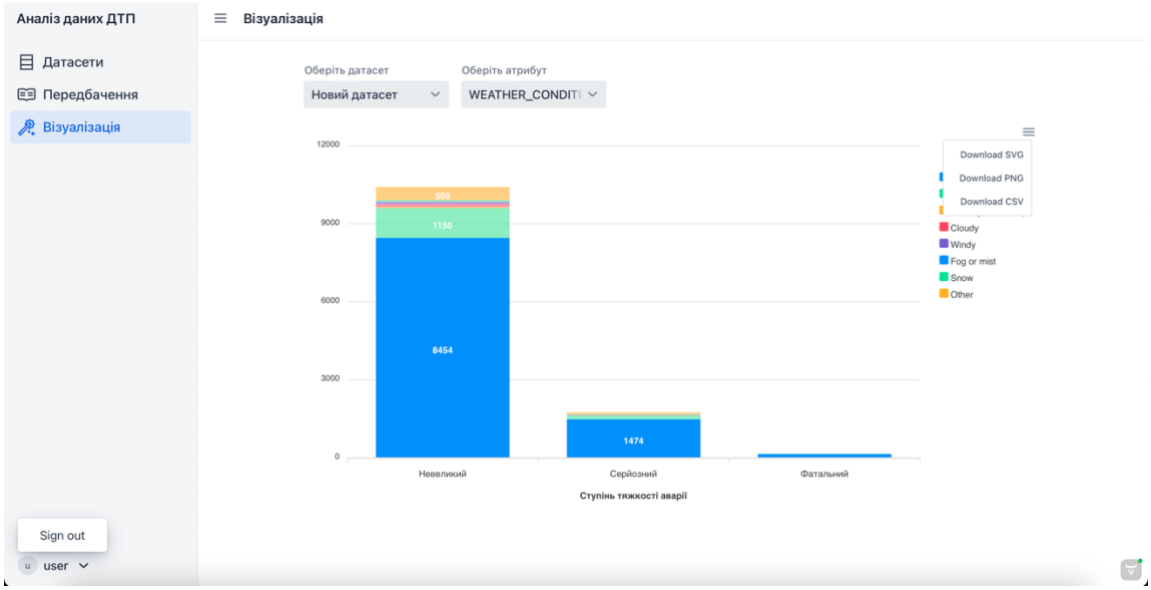


Рисунок 5.19 – Вихід з системи

Висновки до розділу 5

У даному розділі було детально описано взаємодію користувача з системою, а саме описані вимоги та кроки розгортання, наведено послідовність користування веб-застосунком, продемонстровано роботу ключового функціоналу, такого як переглядання, додавання, редагування та видалення наборів даних,

ВИСНОВКИ

У рамках проектування та розробки даної роботи було створено веб-застосунок для предиктивного аналізу даних дорожньо-транспортних пригод. Було проведено аналіз чинників, що впливають як на появу дорожньо-транспортної пригоди, так і на визначення її ступеню тяжкості та кількості постраждалих.

Було розроблено модулі програми, включаючи модуль відображення, аналізу даних, роботи з базою даних, безпеки, імпорту та експорту. В процесі розробки модулю машинного аналізу було обрано класифікаційний алгоритм машинного навчання, за яким було натреновано модель для знаходження ступеню тяжкості аварії та кількості постраждалих.

При проектуванні веб-застосунку була створена структура бази даних та обрана система управління базами даних що задовольняє поставленій задачі. Для забезпечення безпеки та приватності користувачів була додана авторизація.

Користувацький інтерфейс веб-застосунку складається з восьми відображень: авторизація, реєстрація, список наборів даних, список дорожньо-транспортних пригод за набором даних, створення та завантаження аварій у набір даних, переглядання передбачень та графіків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Road Traffic Injuries. *who.int*. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries(date of access: 02.05.2023).
2. Статистика ДТП в Україні. patrolpolice.gov.ua: вебсайт. URL: https://patrolpolice.gov.ua/statystyka/ (дата звернення 01.05.2023).
3. Road Accident Prediction And Classification. URL: https://github.com/theDefiBat/road-accidents-prediction-and-classification (date of access: 10.05.2023).
4. Christoforou Z., Cohen S., Karlaftis M. G. Vehicle occupant injury severity on highways: An empirical investigation. *Accident Analysis & Prevention*. 2010. Vol. 42, no. 6. P. 1606–1620.
5. Gray R. C., Quddus M. A., Evans A. Injury severity analysis of accidents involving young male drivers in Great Britain. *Journal of Safety Research*. 2008. Vol. 39, no. 5. P. 483–495.
6. Dataset. Road Traffic Accident Dataset Of Addis Ababa City. URL: https://www.narcis.nl/dataset/RecordID/oai:easy.dans.knaw.nl:easy-dataset:191591 (date of access: 10.05.2023).
7. Kelleher J., Mac Namee B., D'arcy A. Fundamentals of Machine Learning for Predictive Data Analytics, 2015. 624 p.
8. Suthaharan M. Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning (Integrated Series in Information Systems, 36), 2016. 378 p.
9. Schildt H. Java : the Complete Reference, Twelfth Edition: Comprehensive Coverage of the Java Language. McGraw-Hill Education, 2021. 1248 p.
10. IntelliJ IDEA Ultimate vs IntelliJ IDEA Community. Jetbrains. URL: https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce (date of access: 13.05.2023).
11. Walls C. Spring in Action. Manning Publications Co. LLC, 2018. 520 p.

12. Walls C. Spring Boot in Action. Manning Publications, 2016. 264 p.
13. Pollack M. Spring Data: Modern data access for enterprise Java. Sebastopol, CA : O'Reilly, 2013. 288 p.
14. Vaadin Team. Book of Vaadin: Vaadin 14 Edition. Vaadin Ltd., 2019. 817 p.
15. Obe R. O., Hsu L. S. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. O'Reilly Media, 2017. 314 p.
16. Eibe Frank, Mark A. Hall, Ian H. Witten. The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques. 2016, 128 p.

ДОДАТОК А

Веб-застосунок для предиктивного аналізу даних дорожньо-транспортних пригод

Текст програми

Аркушів 14

Київ — 2023

MachineLearningService.java

```
@Service
@AllArgsConstructor
public class MachineLearningService {

    private static final int ACCIDENT_SEVERITY_INDEX = 17;
    private static final int CASUALTIES_NUMBER_INDEX = 14;

    private AccidentRepository accidentRepository;
    private DatasetRepository datasetRepository;

    private static final String DATASET_LOCATION = System.getProperty("user.dir") +
"/output/datasets/";
    private static final String PREDICTION_LOCATION = System.getProperty("user.dir") +
"/output/predictions/";

    @SneakyThrows
    public void train(Long datasetId) {
        List<Accident> datasetAccidents = accidentRepository.findByDatasetId(datasetId);
        List<Model> datasetModels = MappingUtils.accidentsToModelList(datasetAccidents);

        byte [] accidentSecurityClassifierBytes = trainAndGetPredictionClassifier(datasetId,
datasetModels, ACCIDENT_SEVERITY_INDEX);
        byte [] casualtiesNumberClassifierBytes = trainAndGetPredictionClassifier(datasetId,
datasetModels, CASUALTIES_NUMBER_INDEX);

        Dataset dataset = datasetRepository.findById(datasetId).get();
        dataset.setTrainedOn(true);
        dataset.setAccidentSeverityPredictionClassifier(accidentSecurityClassifierBytes);
        dataset.setCasualtiesNumberPredictionClassifier(casualtiesNumberClassifierBytes);
        datasetRepository.save(dataset);
    }
```

```
    @SneakyThrows
    public Prediction predict(Prediction prediction) {
        Dataset dataset = prediction.getDataset();

        double casualtiesNumber = predictTargetClass(prediction, CASUALTIES_NUMBER_INDEX,
                dataset.getCasualtiesNumberPredictionClassifier());
        prediction.setCasualtiesNumber((int) Math.round(casualtiesNumber));
        double accidentSeverity = predictTargetClass(prediction, ACCIDENT_SEVERITY_INDEX,
                dataset.getAccidentSeverityPredictionClassifier());
        AccidentSeverity accidentSeverityClass = AccidentSeverity.values()[(int)
Math.round(accidentSeverity)];
        prediction.setAccidentSeverity(accidentSeverityClass);

        System.out.println("Casualties number: " + casualtiesNumber);
        System.out.println("Accident severity: " + accidentSeverity);
        System.out.println("Accident severity class: " + prediction.getAccidentSeverity());

        return prediction;
    }

    private byte [] trainAndGetPredictionClassifier(Long datasetId, List<Model> models, int
targetIndex) throws Exception {
        String arffFilename = targetIndex == ACCIDENT_SEVERITY_INDEX ?
                writeAccidentSeverityModelsToArffFile(DATASET_LOCATION, models, datasetId) :
                writeCasualtiesNumberModelsToArffFile(DATASET_LOCATION, models, datasetId);
        Instances dataSet = getDataSet(arffFilename, targetIndex);
        Classifier classifier = new NaiveBayes();
        classifier.buildClassifier(dataSet);

        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        SerializationHelper.write(outputStream, classifier);
        byte [] classifierByteArray = outputStream.toByteArray();
        outputStream.close();
```

```
        return classifierByteArray;
    }

    private double predictTargetClass(Prediction prediction, int targetIndex, byte [] classifierBytes)
throws Exception {
        Long datasetId = prediction.getDataset().getId();
        ByteArrayInputStream inputStream = new ByteArrayInputStream(classifierBytes);
        Classifier classifier = (Classifier) SerializationHelper.read(inputStream);

        Model model = MappingUtils.predictionToModel(prediction);
        String arffFilename = targetIndex == ACCIDENT_SEVERITY_INDEX ?
                FileUtils.writeAccidentSeverityModelsToArffFile(PREDICTION_LOCATION,
List.of(model), datasetId) :
                FileUtils.writeCasualtiesNumberModelsToArffFile(PREDICTION_LOCATION,
List.of(model), datasetId);
        Instance instance = getDataSet(arffFilename, targetIndex).firstInstance();
        return classifier.classifyInstance(instance);
    }

    private Instances getDataSet(String filename, int targetIndex) throws Exception {
        ArffLoader loader = new ArffLoader();
        loader.setSource(new File(filename));
        Instances dataSet = loader.getDataSet();
        dataSet.setClassIndex(targetIndex);
        return dataSet;
    }
}
```

ExportService.java

```
@Service
@AllArgsConstructor
public class ExportService {
```

```
private PredictionRepository predictionRepository;

public String export(Long datasetId) {
    List<Prediction> predictions = predictionRepository.findByDatasetId(datasetId);
    return predictions.stream()
            .map(Prediction::toString)
            .collect(Collectors.joining(System.lineSeparator()));
}

public String exportAccidentImportRules() {
    var lines = List.of("Набір даних повинен мати 18 атрибутів, з яких 2 цільові змінні",
            "Записувати значення для однієї аварії необхідно в строку, через кому",
            "Нижче наведені змінні та можливі значення",
            "accidentHour: числове поле, від 0 до 23",
            createStringFromEnum("dayOfWeek", DayOfWeek.values()),
            createStringFromEnum("driverAgeGroup", DriverAgeGroup.values()),
            createStringFromEnum("driverGender", DriverGender.values()),
            createStringFromEnum("driverExperience", DriverExperience.values()),
            createStringFromEnum("vehicleType", VehicleType.values()),
            "vehicleDefected: строка, значеннями є 'no defect', 'defected'",
            createStringFromEnum("roadAlignment", RoadAlignment.values()),
            createStringFromEnum("roadSurfaceType", RoadSurfaceType.values()),
            createStringFromEnum("roadSurfaceCondition", RoadSurfaceCondition.values()),
            createStringFromEnum("lightCondition", LightCondition.values()),
            createStringFromEnum("weatherCondition", WeatherCondition.values()),
            createStringFromEnum("collisionType", CollisionType.values()),
            "vehiclesNumber: числове поле, від 1 до 10",
            "casualtiesNumber: числове поле, від 0 до 10",
            createStringFromEnum("vehicleMovement", VehicleMovement.values()),
            createStringFromEnum("collisionCause", CollisionCause.values()),
            createStringFromEnum("accidentSeverity", AccidentSeverity.values()));
    return lines.stream()
            .collect(Collectors.joining(System.lineSeparator()));
}
```

```
    private String createStringFromEnum(String name, Enum [] array) {
        String enumString = Arrays.stream(array)
                .map(Enum::toString)
                .collect(Collectors.joining(","));
        return name + ": " + enumString;
    }
}
```

ImportService.java

```
@Service
@AllArgsConstructor
public class ImportService {

    private final AccidentRepository accidentRepository;
    private final DatasetRepository datasetRepository;

    public void importAccidents(Long datasetId, InputStream fileData) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(fileData));
        List<Accident> accidents = new LinkedList<>();
        var dataset = datasetRepository.findById(datasetId).get();
        int rowsSkipped = 0;
        while(reader.ready()) {
            try {
                String accidentString = reader.readLine();
                var accident = new Accident(accidentString);
                accident.setDataset(dataset);
                accidents.add(accident);
            } catch (NoTargetVariableProvidedException e) {
                rowsSkipped++;
            } catch (RuntimeException e) {
                e.printStackTrace();
            }
```

```
        }
        System.out.println("Rows skipped: " + rowsSkipped);
        reader.close();
        accidentRepository.saveAll(accidents);
    }

    public void importDefaultDataset(User user) throws IOException {
        ClassLoader classloader = Thread.currentThread().getContextClassLoader();
        try (InputStream is = classloader.getResourceAsStream("embedded_dataset.csv")) {
            Dataset dataset = Dataset.builder()
                    .title("Вбудований датасет")
                    .description("Складається з 10000+ аварій")
                    .user(user)
                    .build();
            Dataset savedDataset = datasetRepository.save(dataset);
            importAccidents(savedDataset.getId(), is);
        }
    }
}
```

User.java

```
@Entity
@Table(name = "users")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString(exclude = "datasets")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    private String username;
    private String email;
    @JsonIgnore
    private String password;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Dataset> datasets;
}
```

Dataset.java

```
@Entity
@Table(name = "datasets")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Dataset {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;
    private boolean isTrainedOn;
    private byte [] accidentSeverityPredictionClassifier;
    private byte [] casualtiesNumberPredictionClassifier;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

```
    @OneToMany(mappedBy = "dataset", cascade = CascadeType.ALL)
    private List<Prediction> predictions;

    @OneToMany(mappedBy = "dataset", cascade = CascadeType.ALL)
    private List<Accident> accidents;

    public Dataset(Long id) {
        this.id = id;
    }
    @Override
    public String toString() {
        return title;
    }
}
```

Accident.java

```
@Entity
@Table(name = "accidents")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Accident {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; //is not included in prediction
    private int accidentHour;
    @Enumerated
    private DayOfWeek dayOfWeek;
    @Enumerated
    private DriverAgeGroup driverAgeGroup;
    @Enumerated
```

```
private DriverGender driverGender;
@Enumerated
private DriverExperience driverExperience;
@Enumerated
private VehicleType vehicleType;
private boolean vehicleDefected;
@Enumerated
private RoadAlignment roadAlignment;
@Enumerated
private RoadSurfaceType roadSurfaceType;
@Enumerated
private RoadSurfaceCondition roadSurfaceCondition;
@Enumerated
private LightCondition lightCondition;
@Enumerated
private WeatherCondition weatherCondition;
@Enumerated
private CollisionType collisionType;
private Integer vehiclesNumber;
private Integer casualtiesNumber;
@Enumerated
private VehicleMovement vehicleMovement;
@Enumerated
private CollisionCause collisionCause;
@Enumerated
private AccidentSeverity accidentSeverity;
private Integer accidentsNumber;

@ManyToOne
@JoinColumn(name = "dataset_id")
private Dataset dataset;

public Accident(String accidentString) {
    String [] data = accidentString.split(",");
```

```
        if (data[14].isBlank()) {
            throw new NoTargetVariableProvidedException();
        }
        accidentHour = data[0].isBlank() ? 0 : Integer.parseInt(data[0].split(":")[0]); //парсит из строки формата 16:00:00
        dayOfWeek = DayOfWeek.findByTitle(data[1]);
        driverAgeGroup = DriverAgeGroup.findByTitle(data[2]);
        driverGender = DriverGender.findByTitle(data[3]);
        driverExperience = DriverExperience.findByTitle(data[4]);
        vehicleType = VehicleType.findByTitle(data[5]);
        vehicleDefected = !data[6].toLowerCase().contains("no defect");
        roadAlignment = RoadAlignment.findByTitle(data[7]);
        roadSurfaceType = RoadSurfaceType.findByTitle(data[8]);
        roadSurfaceCondition = RoadSurfaceCondition.findByTitle(data[9]);
        lightCondition = LightCondition.findByTitle(data[10]);
        weatherCondition = WeatherCondition.findByTitle(data[11]);
        collisionType = CollisionType.findByTitle(data[12]);
        vehiclesNumber = data[13].isBlank() ? 1 : Integer.parseInt(data[13]);
        casualtiesNumber = Integer.parseInt(data[14]);
        vehicleMovement = VehicleMovement.findByTitle(data[15]);
        collisionCause = CollisionCause.findByTitle(data[16]);
        accidentSeverity = AccidentSeverity.findByTitle(data[17]);
        accidentsNumber = 1;
    }
}
```

UserRepository.java

```
public interface UserRepository extends JpaRepository<User, Long>,
JpaSpecificationExecutor<User> {
    User findByUsername(String username);
}
```

DatasetRepository.java

```
public interface DatasetRepository extends JpaRepository<Dataset, Long> {
    List<Dataset> findByUserId(Long userId);
    List<Dataset> findByIsTrainedOnAndUserId(boolean isTrainedOn, Long userId);
}
```

AccidentRepository.java

```
public interface AccidentRepository extends JpaRepository<Accident, Long> {

    String FIND_BY_FIELD_TEMPLATE = "select count(id) from accidents where dataset_id = ?1 and accident_severity = ?2 ";

    List<Accident> findByDatasetId(Long datasetId, Pageable pageable);
    List<Accident> findByDatasetId(Long datasetId);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and vehicle_type = ?3", nativeQuery = true)
    Integer countByVehicleType(long datasetId, int accidentSeverity, int vehicleType);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and weather_condition = ?3", nativeQuery = true)
    Integer countByWeatherCondition(long datasetId, int accidentSeverity, int weatherCondition);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and light_condition = ?3", nativeQuery = true)
    Integer countByLightCondition(long datasetId, int accidentSeverity, int lightCondition);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and road_condition = ?3", nativeQuery = true)
    Integer countByRoadCondition(long datasetId, int accidentSeverity, int roadCondition);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and collision_type = ?3", nativeQuery = true)
    Integer countByCollisionType(long datasetId, int accidentSeverity, int collisionType);

    @Query(value = FIND_BY_FIELD_TEMPLATE + "and collision_cause = ?3", nativeQuery = true)
    Integer countByCollisionCause(long datasetId, int accidentSeverity, int collisionCause);
}
```

PredictionRepository.java

```
public interface PredictionRepository extends JpaRepository<Prediction, Long> {
   List<Prediction> findByDatasetId(Long datasetId);
}
```

SecurityConfiguration.java

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends VaadinWebSecurity {

   @Bean
   public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      setLoginView(http, LoginView.class);
      super.configure(http);
   }
}
```

AuthenticatedUser.java

```
@Component
public class AuthenticatedUser {

   private final UserRepository userRepository;
   private final AuthenticationContext authenticationContext;

   public AuthenticatedUser(AuthenticationContext authenticationContext, UserRepository
userRepository) {
      this.userRepository = userRepository;
```

```
        this.authenticationContext = authenticationContext;
    }

    public Optional<User> get() {
        return authenticationContext.getAuthenticatedUser(UserDetails.class)
                .map(userDetails -> userRepository.findByUsername(userDetails.getUsername()));
    }

    public void logout() {
        authenticationContext.logout();
    }
}
```

UserDetailsService.java

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;

    public UserDetailsServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("No user present with username: " + username);
        } else {
            return new org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(),
                List.of(new SimpleGrantedAuthority("USER")));
        }
```

```
    }
}
```

AccidentSeverity.java

```
@Getter
public enum AccidentSeverity {
    SLIGHT("Slight Injury"),
    SERIOUS("Serious Injury"),
    FATAL("Fatal Injury");

    private final String title;

    AccidentSeverity(String title) {
        this.title = title;
    }

    @Override
    public String toString() {
        return title;
    }

    public static AccidentSeverity findByTitle(String title) {
        return Arrays.stream(AccidentSeverity.values())
                .filter(e -> Objects.equals(e.getTitle(), title))
                .findFirst()
                .orElseThrow(NoTargetVariableProvidedException::new);
    }
}
```